DIPLOMARBEIT

A Multi-Purpose Virtual Model of the Solar System (VRMoSS)

ausgeführt am

Institut für Computergraphik und Algorithmen
der Technischen Universität Wien

unter der Anleitung von

Prof. Dr. Michael Gervautz

durch

Georg Zotti
Ghelengasse 13a
A-1130 Wien

Wien, 21. Oktober 2001 Georg Zotti

Abstract

For centuries, astronomers have presented their research results about the structure of the Solar System with models. Most of them were small and provided an "outside look" onto the Solar System. Developments in Computer Graphics, Virtual Reality environments, and space exploration make it possible for a user to see the Solar System as though he was flying through space in an — admittedly physically impossible — super-lightspeed spacecraft and visit the planets close-up, or, see the components of the Solar System (the Sun, planets, moons, asteroids, comets) from the distance to get a feeling of the structure at a glance. Depending on hardware setup, this can be done on a single high-end PC desktop with keyboard control or may include a stereo projection, or head mounted displays and head tracking with interaction via specially designed user interface tools.

The author has implemented such a model using the STUDIERSTUBE framework of the Institute of Computer Graphics and Algorithms of the University for Technology, Vienna, as part of the ASH project (Access to Scientific Space Heritage), one of several educational projects sponsored by the IST (Information – Society – Technology) initiative of the European Union.

For integration into a larger system, data and commands can be exchanged via a network socket connection using XML formatted messages.

Deutsche Zusammenfassung

Seit Jahrhunderten präsentieren Astronomen ihre Erkenntnisse über das Sonnensystem anhand von Modellen. Die Entwicklung der Computergraphik sowie Ergebnisse von Weltraummissionen zu den Planeten gestatten es nun, animierte Modelle des Sonnensystems zu bauen, die nicht nur (wie die Tischplanetarien früherer Zeiten) von außen zu betrachten sind, sondern den Betrachter in das Modell hineinversetzen können. Durch entsprechende Skalierung können Übersichts- oder Detailansichten präsentiert werden.

Der Autor präsentiert ein solches Modell für das STUDIERSTUBE-System des Instituts für Computergraphik und Algorithmen, das er im Rahmen des IST (Information – Society – Technologies) Bildungsprogramms der Europäischen Union als Teil des Projektes ASH (Access to Scientific Space Heritage) entwickelt hat.

Das Modell wird mittels Stereo-Projektion präsentiert, der Besucher kann über spezielle Eingabegeräte mit der Szene interagieren und sich das Sonnensystem und seine Teile (Sonne, Planeten, Monde, Kleinplaneten (einige detailliert, tausende schematisch als Asteroidengürtel) und Kometen sowie die jeweiligen Umlaufbahnen) in unterschiedlichen Größen betrachten. Ebenso kann der Flug der Raumsonde Rosetta (ESA) zum Kometen P46/WIRTANEN verfolgt werden. Die Applikation kann mit anderen Programmen über eine Socket-Verbindung Daten und Kommandos im XML-Format austauschen.

VRMoSS

A Multi-Purpose Virtual Model of the Solar System

Georg Zotti
Institute of Computer Graphics
Vienna University of Technology

ASH – Access to Scientific Space Heritage

Contents

Chapter 1

ASH – Access to Scientific Space Heritage

1.1 Introduction

To advance Europe in the fields of science and technology, the European Commission offers the IST programme (Information Society Technologies). One of its key actions is [COR01]

> to confirm Europe as a leading force in [the field of multimedia content and tools] and enable it to realise the potential of its creativity and culture. It will address issues such as interactive electronic publishing, digital heritage and cultural content, education and training, human language technologies and information access, filtering and handling.

The ASH project (Access to Scientific Space Heritage), proposed in late 1999, has been accepted as part of the IST programme under European Commission contract number IST-1999-10859.

The prime objective of this project is to set up a prototype of a collaborative learning environment called "Virtual Control Room" (VCR), which allows students to participate in a simulated scientific space mission.

The Virtual Control Room will allow European planetariums and science theatres to offer a unique experience to audiences in the European Union, mediating knowledge of science, astronomy and space to the European public.

The VCR is an edutainment facility, bridging the gap between education and entertainment. New media are highly exploited and cut-of-the-edge 3D interactive systems are provided.

The ASH VCR should get young people interested in astronomy, space science, and generally in technology and nature sciences, so that more students attend technical or scientific studies.

Project Partners The following companies and institutions are participating in the ASH project:

DELTA Danish Electronics, Light and Acoustics, Denmark. Main contractor. Experienced in collaborative software development. Responsible for project management and the Mission Server.

SAS Space Applications Services, Belgium. Experience with real-world space missions. Responsible for the Simulation Server.

TUV Institute of Computer Graphics and Algorithms, Technical University Vienna, Austria. Experts in computer graphics, virtual and augmented reality, responsible for the graphical content and user interface.

RoB Royal Observatory, Brussels, Belgium. Experience with teaching astronomy. Head of the the storyboard team.

EuroP Europlanetarium Genk, Belgium. Experience with teaching astronomy. Member of the storyboard team.

Tycho Tycho Brahe Planetarium, Copenhagen, Denmark. Experience with a "Control Room" used for school classes. Member of the storyboard team.

The following introduction to the ASH project is based on the ASH System Specification Document (SSD). [TZW+01]

For budget reasons, the goal of the project was to develop a prototype of the VCR, designated P-VCR. A commercial version is termed C-VCR.

1.2 Hardware Architecture

1.2.1 The P-VCR Configuration

The VCR is a multi user environment based on client-server architecture with special features like a stereo projection, wide screen displays, and 3D interaction technology. (Figures 1.1, 1.2)

The hardware architecture considers scalability. Therefore the concept of islands has been introduced. An island comprises of four workplaces that are all physically embedded in a designed piece of furniture. There are three individual workplaces and one for collaborative aims. These are the *client machines*. The minimal VCR consists of just one island, the P-VCR will have two of them, and a C-VCR might have enough places for a whole school class. However, the configuration also depends on the mission, since their collaborative aspects mostly fit to a specific number of students. Therefore, missions development must take the number of islands into account.

The individual workplaces are standard desktop PCs hidden in the furniture (Figure 1.3). Touch screens are embedded into the top panel. If a

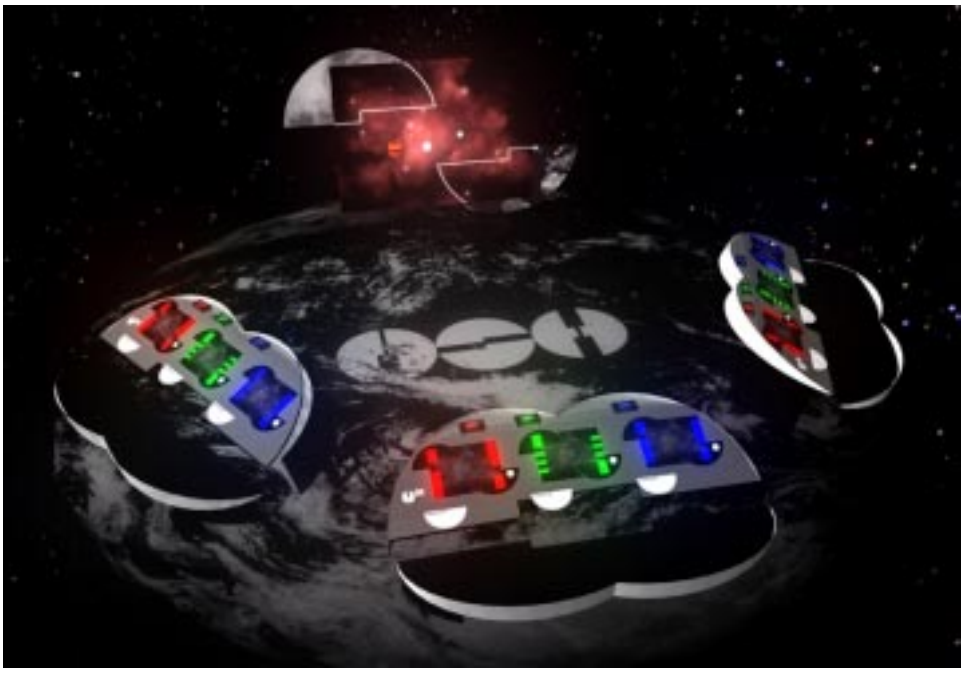


Figure 1.1: Artist's Concept of the ASH Virtual Control Room

Figure 1.2: Students in the VCR

Figure 1.3: An Island in the ASH Virtual Control Room

keyboard is needed, a software keyboard is displayed on the touch screen. Each island has one local sound system consisting of two small, embedded speakers for short notification sounds. There is an extra, collaborative workplace on each island. It also consists of a standard desktop PC with a 3D graphics accelerator and uses a wide screen display. The collaborative results of tasks are visualized on this display.

As central attraction, there is a large passive stereo projection collaboratively used by all people in the P-VCR, termed Bigscreen (Figure 1.4). It consists of two LCD projectors with differently oriented polarization filters in front of the lenses. The visitors wear special eyeglasses, also with polarization filters which are oriented so that they can separate the images, so that each eye receives only the one image intended for it. A special screen is necessary that preserves the polarization of the light, especially for the recommended back projection setup that avoids problems with shadows. One advantage of passive stereo is that the users do not need to wear high tech glasses but cheap one-way cardboard goggles with polarization filters, therefore avoiding hygienic problems.

The two image streams for the stereo projection come from an extra client (Bigscreen Client) with powerful graphics hardware. The Bigscreen is used for special tasks and provides a 3D input device called Personal Interaction Panel (PIP). It has been developed by the Institute of Computer Graphics and Algorithms of the Technical University of Vienna and was evaluated during various research projects. The PIP consists of a plastic panel and a pen with magnet field sensors attached to both of them. These sensors deliver spatial coordinates and orientation data relative to a magnet field

Figure 1.4: Students working on the Bigscreen

emitter to a Tracker Server. The Tracker Server transmits these data to the Bigscreen Client via TCP/IP.

The sensors, emitter and server are termed Tracker System. It can be conceived as a black box that delivers a stream of spatial data to be used in 3D applications.

The PIP allows direct manipulation of objects within a virtual environment in a quite natural way. The 3D representation of the panel holds objects, menus and standard GUI elements like buttons and sliders. With the pen a user can pick objects, manipulate them and deposit them somewhere in space. To learn more about the PIP please see [SG97a, SG97b].

In alternation with this exciting experience, the Bigscreen it is also used to display passive media, like videos, usually in mono.

The heart of the VCR is the Mission Server (MSV), which is a multi-user server, supplying all the multimedia data and managing the communication between clients. For that purpose it has to run on a more powerful machine, e.g. a dual processor unit. The Mission Server has a mass storage device attached to it (RAID) that holds all the multimedia data and the Mission Data Space.

Beside the Mission Server there is an extra Simulation Server (SSV). It is a software application to run different simulations, which also runs on the machine running the MSV. It gets prompted by clients via the MSV and

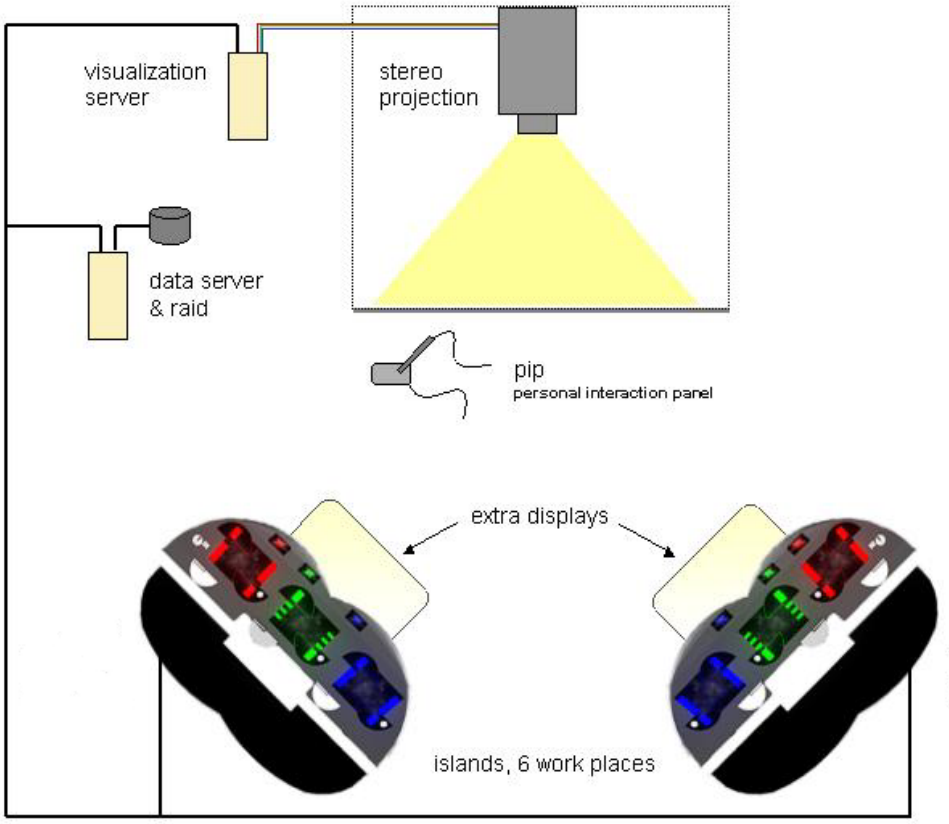


Figure 1.5: Hardware architecture of the ASH VCR

returns streams of simulation results to ongoing tasks at the clients. The simulation server has been implemented as recorder, which can record data during mission preparation and can then stream those pre-recorded/pre-calculated data to the clients.

To complete the hardware of the P-VCR we need network devices, a surround sound system mainly used in accordance with the stereo projection, the special designed furniture, nice lighting system and lots of cables. See Figure 1.5 for a diagram of the hardware architecture of the P-VCR.

Scaling the hardware is simply done by producing more islands. However there are some restrictions, especially concerning shared resources. The following list addresses problems that may arise when extending the hardware architecture of the P-VCR to those of the C-VCR from the present point of view. Advances in technology until the end of the project might solve most of these problems automatically.

1.3 Software Architecture

1.3.1 Introduction

The software of the VCR has heterogeneous client-server architecture. It consists of several distributed, self-contained components that interact via clearly defined interfaces. In addition to components that have been and will be developed within this project, there are also third party and/or open source products.

1.3.2 Mission, Episodes and Tasks

The VCR is a multi-user learning environment exploiting principles of edutainment — learning through entertaining. Knowledge about space is mediated by working together as a team to accomplish a space mission.

A mission consists of a network of episodes. For the P-VCR this is just a succession of tasks, whereas for the C-VCR also branches will be allowed. In that case it is like an interactive movie where the results achieved in an episode determines which episode follows.

Episodes can be reused in other missions. They are characterized by well-defined pedagogical objectives, i.e., are associated with student profiles like age or education. Episodes contain a schedule of interactive media to affect the mission state. It can be mixed with passive media, for example watching a video or reading an HTML page.

For each episode there exists one transition. A transition is also a self-contained entity that contains only passive media. Transitions are used when an episode is not available, has been interrupted or skipped. With transitions the mission time line is kept continuous. Students have to know what happens in an unavailable episode to keep pace and have the correct context to proceed. So a transition is a summary that shows what would have to be done in the corresponding episode. It is mostly a passive presentation (like a video). A transition can also be used in case a student completely fails to accomplish an episode. In that case it documents what the student should have done to succeed. Furthermore, transitions might be a regular part of a mission network, for example to show a video to relax students between two highly interactive episodes. See Figure 1.6 for how episodes and transitions make up a mission.

The interactive media entities of an episode are called tasks. Beside tasks there are also presentations, which are passive media entities, like a video or an HTML page. Both types of entities are ordered in a schedule. They are performed at clients only. In the P-VCR we have nine clients (six individual workplaces and three collaborative workplaces). The storyboard, however, already considers four islands, thus twelve individual and five collaborative workplaces. A storyboard for a C-VCR will have to consider the setup of the respective C-VCR.

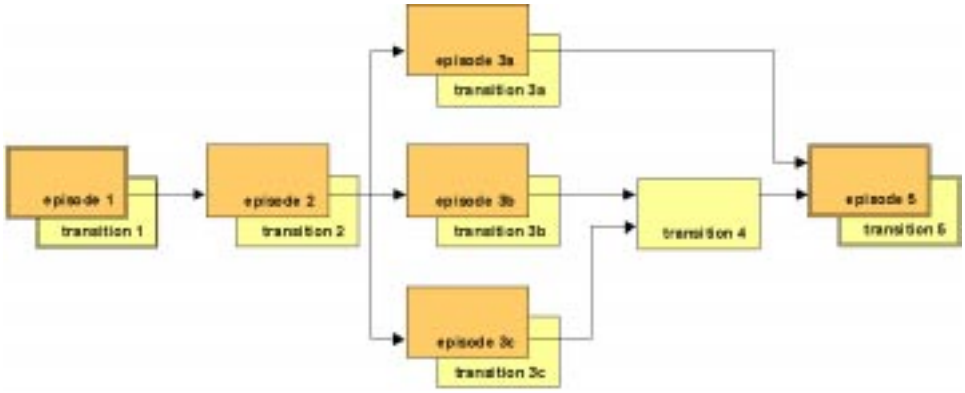


Figure 1.6: A mission is a network of episodes and transitions

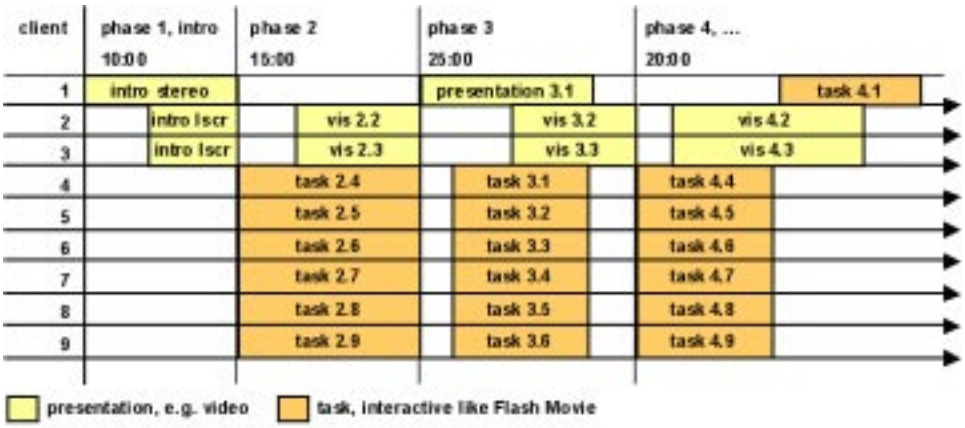


Figure 1.7: Tasks are scheduled in nine time lines to make up an episode for the P-VCR

So there are seventeen time lines in the schedule. That means at most seventeen tasks can run in parallel. Tasks are associated with roles and screen qualifiers that define on which client they can run. Students interact with tasks to achieve a goal, i.e., they change the mission state. Examples for media formats are FLASH Movies, SWING applications, or VRML worlds.

There are synchronization points, where the system waits for all ongoing tasks to finish until the episode is continued. The time frame between two synchronization points is called a phase. Phases have a predefined duration and exactly one task for each client. Figure 1.7 shows how tasks are arranged within an episode.

1.4 Multi-User Server-Client Architecture

The heart of the system is the Mission Server (MSV). It provides all the media organized in tasks and episodes by accessing a file system stored on a RAID, the Mission Data Space (MDS). The MDS works like a web-server, and the P-VCR indeed uses a standard APACHE web server. Dynamic data like the Mission State Information (MSI) or user data is stored in the Shared Data Space (SDS).

The Mission State Information (MSI) is the collection of all data resulting by finishing a task and is important to manage the mission time flow. User data includes personal information like the role within the mission scenario, performance during tasks, or results from tasks which can be used in a later task. It can be used for evaluation and user statistics afterwards.

The MSV has a special component called Mission Manager (MMG). It controls the flow of episodes and MDS consistency. The MMG knows which episode is currently running. It knows when it is finished and which one to start next. The MMG also recognizes when an episode is missing, skipped or interrupted and automatically starts the corresponding transition. Besides that it also maintains consistent Mission State Information. Each episode has clearly defined initial states and default results. When an episode is missing, skipped or interrupted, the MMG establishes a usable MSI by copying the appropriate default results from a mirror into the data space. In this way a reliable, fail-operational system is achieved.

Another component of the MSV is the Episode Manager (EMG). It basically does the same for tasks as the MMG does for episodes. It knows the schedule, triggers tasks and synchronizes at the end of phases. It also checks for consistency concerning tasks. The completion of a task changes the MSI. Like for episodes' default results there are default results of each task usable to advance to the next task stored in a data mirror that are copied by the EMG in case the task is interrupted.

This fail-operational attribute of both the MMG and EMG allows the operator to skip or interrupt single tasks or whole episodes anytime. In fact the system can run in "autopilot" mode. It can thus substitute results for any student, and there is no problem if some work places stay empty. This feature allows the VCR to operate with any number of visitors less or equal its maximum capacity. Furthermore, it is possible to run a four-island mission on a two-island VCR because the system takes over the tasks of missing workplaces.

Beside the MSV there is also a Simulation Server (SSV). For the P-VCR, it provides only pre-recorded data, but for a C-VCR it could be enhanced to also run computationally demanding applications. Its services are launched due to a request by the MSV. The Simulation Server then gets a stream of input data, and streams back pre-recorded data as simulation results. Examples for simulations are orbiting a planet or asteroid, driving a rover,

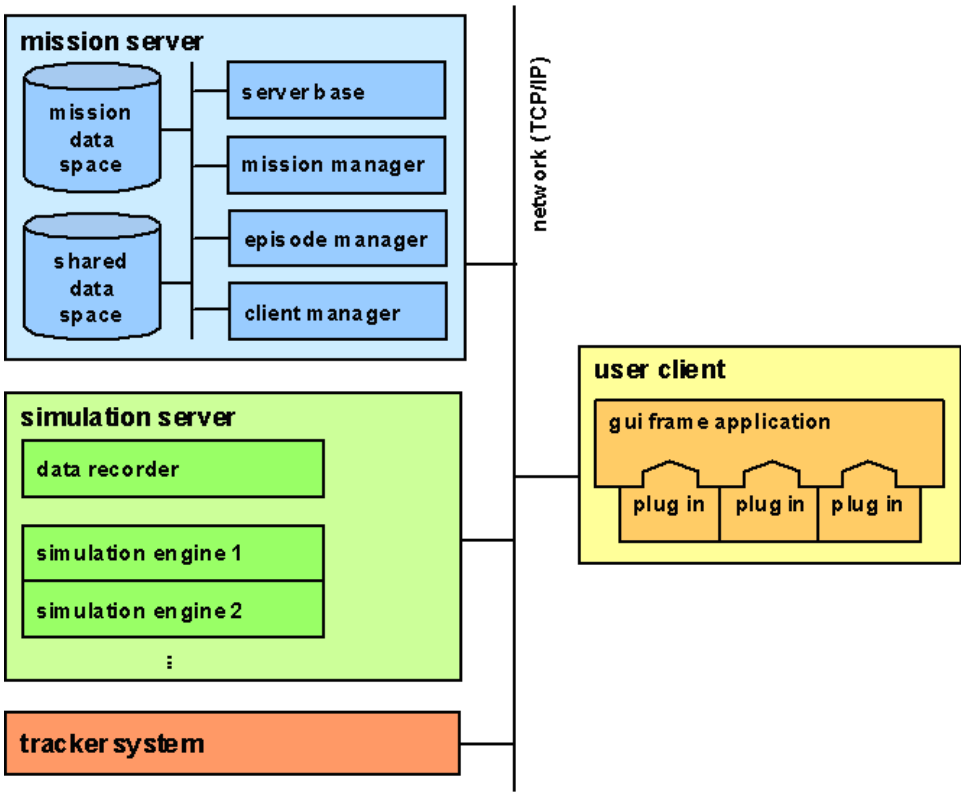


Figure 1.8: The top level view of the P-VCR software architecture

celestial mechanics and more. Detailed information on the SSV is given in [TZW+01].

Finally there is the User Client (UCL). The UCL is responsible for communication with the MSV. It does so by a corresponding instance of a Client Manager (CMG). It interprets packages and sends requests. It can launch sub-applications if necessary. It launches the tasks received by the MSV via CMG and checks when tasks are finished or interrupted, and communicates this to the CMG.

The UCL also establishes and maintains all communication channels and other local resources. It has a GUI consisting of a full screen frame and panels for tools that can be used during the whole mission. This encompasses, e.g., a repository browser to access optional material, or a chatting tool. The UCL for the P-VCR has been implemented in Macromedia Flash 5.

Figure 1.8 summarizes the top-level view of the software architecture.

1.5 Collaboration and Communication

Very complex CSCW software is not necessary, because we do not have a tight and sensitive collaboration scenario. The collaborative aspects of a mission are entirely designed by the storyboard, which defines the schedule for tasks. The system does not have to resolve any problems occurring from interdependencies.

If a student fails to produce results of a task needed by other students or fails to finish a phase the system goes into a wait position. After a certain timeout the EMG can suggest to skip the task and continue. Such problems are resolved in a quite natural manner by people communicating with each other and deciding how to proceed. The operator can always skip tasks or whole episodes without causing inconsistencies. As mentioned above, each task can also be done by the system and each episode can be replaced by a transition. Default results will be used in that case.

Collaboration implies exchange of data. The large screens on each island are mostly used to visualize collaborative aspects. The results obtained by working together are presented there, and students are aware of the result of their teamwork. They see how their individual performance contributes to the larger context. Another collaboration feature is chatting, for example to ask colleagues who play other roles for help.

In addition to collaboration, data exchange is essential for other scenarios as well. For example user input might be streamed to the Simulation Server (SSV), which takes the data as parameters for a simulation. Simulation results my be streamed from the SSV to the Bigscreen Client (stereo projection) to affect a space vehicle in real time.

Communication is based on TCP/IP. Servers produce data streams and packages and deliver them to the appropriate clients asking for these data. From a software point of view there are four main channels:

Mission Data Base Channel download tasks, stream media like Real Video

Shared Data Space Channel query results of tasks and mission state

Simulation Channel trigger simulation, stream simulation results

Collaboration Channel client-to-client communication, chatting, mailing

The Mission Data Base channel is implemented by HTTP communication. Clients requests data from a web server, which returns a stream of data to be played by the client.

The Simulation channel is also based on an ASCII-protocol, running through a socket communication channel - also upon request, providing a stream of data. This custom protocol is described in detail in [TZW+01].

The Shared Data Space channel and the Collaboration channel are different paradigms. The data here is more dynamic in nature - and is also shared by multiple clients, all updating and retrieving at the same time. For this, a repository of data objects (for instance, containing a set of strings in a chat or the selected orbit from an work-isle) will be available. The Shared Data Space (SDS) is thus not a collection of streams but a storage of named data objects, where all components of the system can save, share, edit and retrieve data.

1.6 The Prototype Mission: ROOTS

Discussion about the mission for the P-VCR crystallized a few key issues that should be observed:

- it should be a deep space mission of a spacecraft exploring the Solar System
- Europe's special interest and knowledge in space exploration should be expressed
- the mission should fit into school curricula

Currently, one key research topic in space exploration is the question whether there is life outside Earth. Recently speculated finds of traces of extraterrestrial life in structures in the Mars meteorite ALH 84001 made a mission to Mars a possible candidate.

However, Mars has mainly been target for American and Russian (former Soviet) space missions. Europe, on the other hand, performed a very successful space mission to HALLEY's Comet in 1986, when the GIOTTO spacecraft passed the comet's core in a close fly-by.

Moreover, the European Space Agency (ESA) is currently preparing a new mission, ROSETTA, for launch in early 2003. This spacecraft will first orbit the Sun almost twice, then use fly-by maneuvers with Mars (late August, 2005), Earth (late November 2005) to gain speed, will enter the asteroid belt between Mars and Jupiter to visit asteroid 4979 OTAWARA, fly-by Earth once more for a final speed-up (late November, 2007) and, after a second asteroid fly-by (140 SIWA, late July, 2008), will finally, in late November, 2011, reach comet 46P/WIRTANEN and enter an orbit around the comet. It will map the core and finally release a lander that further analyzes the comet soil. [ESA]

Comets are believed to be the oldest remnants from the time the Solar System formed about 4.5 billion years ago. By observing a comet in detail, ROSETTA shall give new insights into the origin, composition, formation and development of the Solar System.

The ASH consortium agreed on developing a mission scenario that closely follows the ROSETTA mission. By following the spacecraft through the Solar System, the students can learn about its structure and contents. Many aspects of course material (physics, chemistry, astronomy, mathematics) can be built into the mission material ("Stealth Learning"). The students will get an impression how the sciences work together, and may be encouraged to start a scientific career.

The mission was given the descriptive acronym ROOTS for *ROsetta Observing The Solar system.*

1.6.1 The Solar System Model inside ROOTS

The ASH consortium decided on developing a virtual model of the Solar System for the ROOTS mission for several reasons.

- Presenting the Solar System as VR model immediately gives a good impression about its parts and structure.
- The journey of the student-controlled spacecraft through the Solar System can be impressively visualized.
- Algorithms and data for modeling the Solar System are readily available in the literature and reasonably well understood.
- A model of the Solar System can be reused in many missions.

Thus, it was required to develop a visually appealing model of the Solar System which could show as many aspects as possible of its contents and structure. The program should be a fully integrated application in the ASH VCR. It should allow active control with PIP interaction as well as remote control for passive presentations.

Chapter 2

Modeling the Solar System

For centuries, the structure of the universe, and especially the Solar System, has been the object of study by countless scientists. To show and explain their results to colleagues and the general public, various forms of models have been produced, from simple drawings on paper to delicate artwork with clock drives of astounding accuracy in brass and gold.

Most of these models are of very limited size, e.g., printed in a book or built to rest on a desktop, and show the Solar System as seen from the outside.

But humans live on Earth, the third planet (counted from the Sun outward) of the Solar System, thus raising demand for a model that gives an "inside" look. Creating a correctly scaled model that can be walked through by visitors was always much more demanding and costly, and faced the problem of having to model vast distances between the planets.

This chapter gives a short historical overview of the development of sky simulations and gives some insight into the difficulties faced by the prospective model maker.

2.1 Historical Notes

The prime purpose for models of the Solar System is to demonstrate its structure and the relative distances between the planets, which, until the invention of the telescope (about 1608), could only be observed as moving starlike points with unknown size.

When in 212 B.C. Marcus Claudius Marcellus conquered Syracuse, one of the trophies he kept for himself was the "Sphere" of Archimedes, a bronze device that showed the spheres of the planets in a framework of moving circles which were probably even moved automatically by water-driven engines and showed the planets in motion around the Earth. [Mei92] This old conception of the world saw the cosmos as system of interlocked

crystalline spheres surrounding Earth and carrying one planet each. The outermost sphere was decorated by the stars.

The SYNTAXIS (better known by the title of its arabic translations, ALMAGEST) of Alexandrine astronomer CLAUDIUS PTOLEMÆUS (ca. A.D. 140) describes Earth as immovable sphere in the center of the universe, with circular orbits carrying the planets surrounding Earth. The innermost circle carries the Moon, then came Mercury, Venus, the Sun, Mars, Jupiter, Saturn, and the outermost sphere of the fixed stars. To explain the observed strange looping motions of the planets, a system of interlocked circles is used. In this system, the planet's orbit (deferent) only carries the center of another circle, the epicycle, on which the planet moves. All motions were circular motions of constant speed, which was regarded the only possible form of motion for celestial objects. Putting the center of the deferent outside Earth and adding even more epicycles where necessary helped to explain remaining differences between observation and computation. [Dre53]

The armillary sphere, an instrument with interlocked movable rings representing the celestial equator and the ecliptic, could be used as observing tool and also to demonstrate the apparent daily motion of the sky and apparent yearly motion of the Sun. Armillar spheres have a long history dating from several centuries B.C. up to the 17th century. [Mei92]

Starting in the late 14th century, clockworks were constructed that could not only show time and calendar dates, but also Moon phases and the positions of the planets. These clockworks made use of the best mechanical models of the geocentric planetary system, which had been refined over the past millennium, with excentric epicycles, oval gear wheels, *etc.*, and reached remarkable accuracy. [Mei92]

The year 1543 marks a milestone in the history of astronomy: NICOLAUS COPERNICUS published his DE REVOLUTIONIBUS ORBIUM COELESTIUM, describing the heliocentric system, with the Sun in the center of the world and the planets moving around the Sun[1]. However, because regular circular motion was then still regarded as the only motion possible for celestial objects, COPERNICUS again had to use epicycles and excentric deferents to fit the theory to the observations. [Dre53]

In 1596, JOHANNES KEPLER, in his MYSTERIUM COSMOGRAPHICUM, carefully tried to fit the spheres of the planet orbits into a framework of the five Platonic bodies, reasoning the order by the respective body's "dignity" (Figure 2.1).

Later, based on TYCHO BRAHE's exact observations of the planet Mars, he had to give up circular orbits and found his famous three laws of planetary motion, finally abandoning the need for epicycles:

[1] COPERNICUS was not the first astronomer describing this arrangement of the universe. ARISTARCHUS OF SAMOS had proposed it already in the 3rd century B.C.

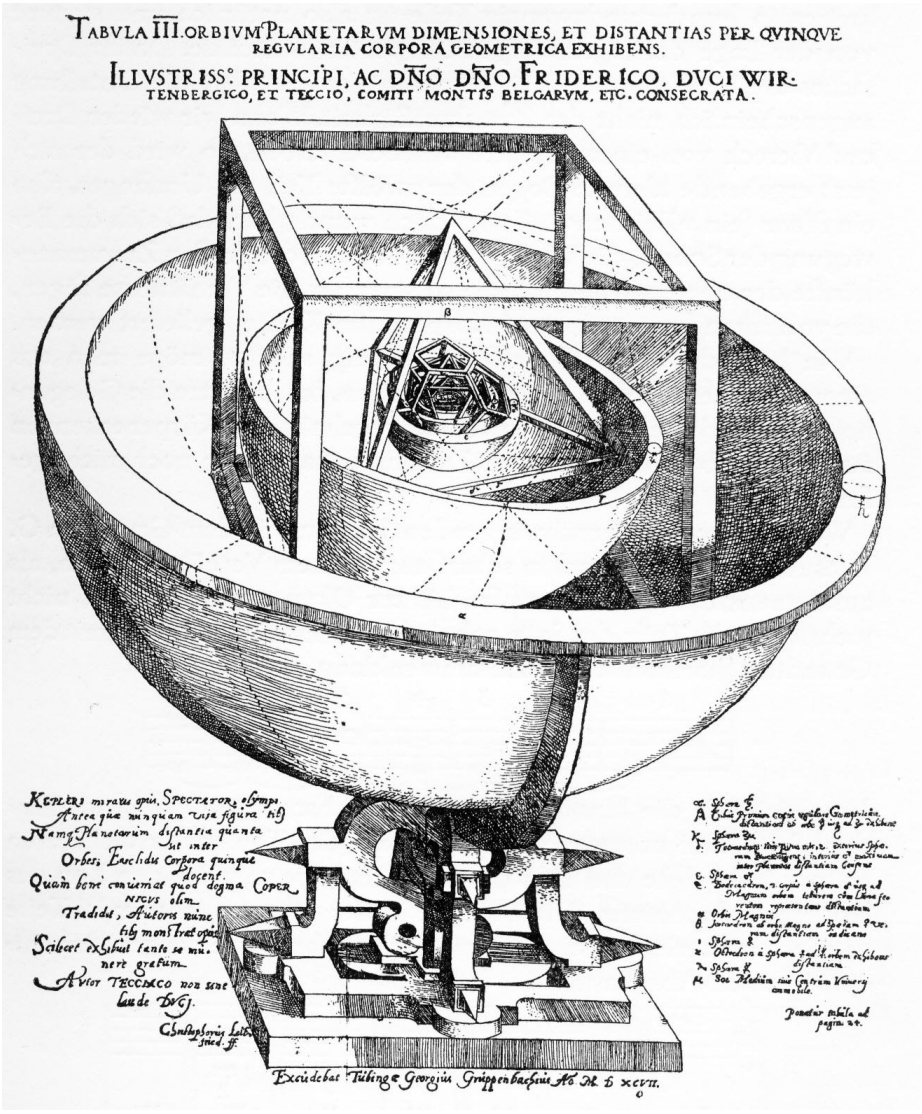


Figure 2.1: JOHANNES KEPLER tried to use the five Platonic Bodies to model the planets' relative distances. [Bry18]

1. The planets orbit the Sun on almost circular elliptic orbits. The Sun resides in one focus of each ellipse.

2. The radius vector from Sun to planet covers equal areas in equal time spans. This means, when the planet is nearest to the Sun, it is also fastest on its way.

3. The squared orbit periods of two planets relate to each other like the cubed great semiaxes. $p_1^2 : p_2^2 = a_1^3 : a_2^3$

Figure 2.2: Early Copernican Tellurium, completed around 1634 by Willem Janszoon Blaeu, Netherlands. The Sun is represented as spike, an Earth globe, surrounded by circles of an armillary sphere, rotates around the Sun. Earth's axis keeps orientation, allowing a demonstration of the seasons. [Mei92]

The third law, found much later than the first two, finally allowed to deduce accurate relative distances between the planets from observations of orbital periods.

Mechanic models of that time (Figure 2.2) concentrated primarily on the motion of the Earth around the Sun and the explanation of the seasons. The models could be set into motion by hand cranks or clock drives.

Later models (Figure 2.3) included the Moon's motion around the Earth and, finally, the other planets and even moons of other planets, usually as separate models (Lunarium, Jovilabe of Ole Rømer 1677), but sometimes integrated in a full model of the Solar System. The Sun was sometimes

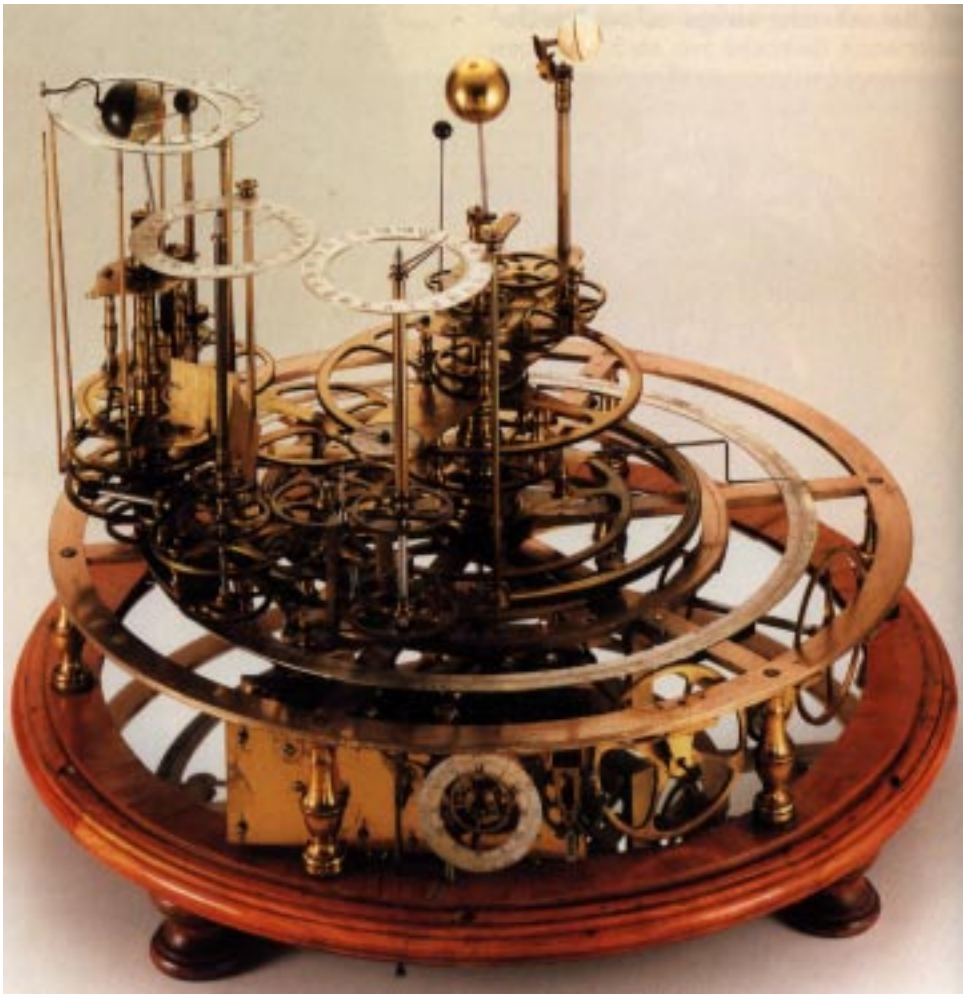


Figure 2.3: JAMES FERGUSON's clock-driven Planetarium, London. Diameter: 63cm [Mei92]

represented as lamp, allowing to see the illuminated sides of the planets and explaining the phases of Mercury and Venus. [Mei92]

JOHN ROWLEY's model, demonstrating the daily motion, seasons, lunar phases and eclipses, built for the Earl of Orrery, is the name patron for these machines in the Anglo-American world, which have been since then called Orrery. [Mei92]

Since antiquity, models of the outermost sphere carrying the stars were made from marble, metal or cloth covered wood. These celestial globes show the stars as seen "from outside", thus all constellations are depicted reversed as seen from Earth. Interestingly, most sky atlases until about 1700 also were drawn in this reversed style.

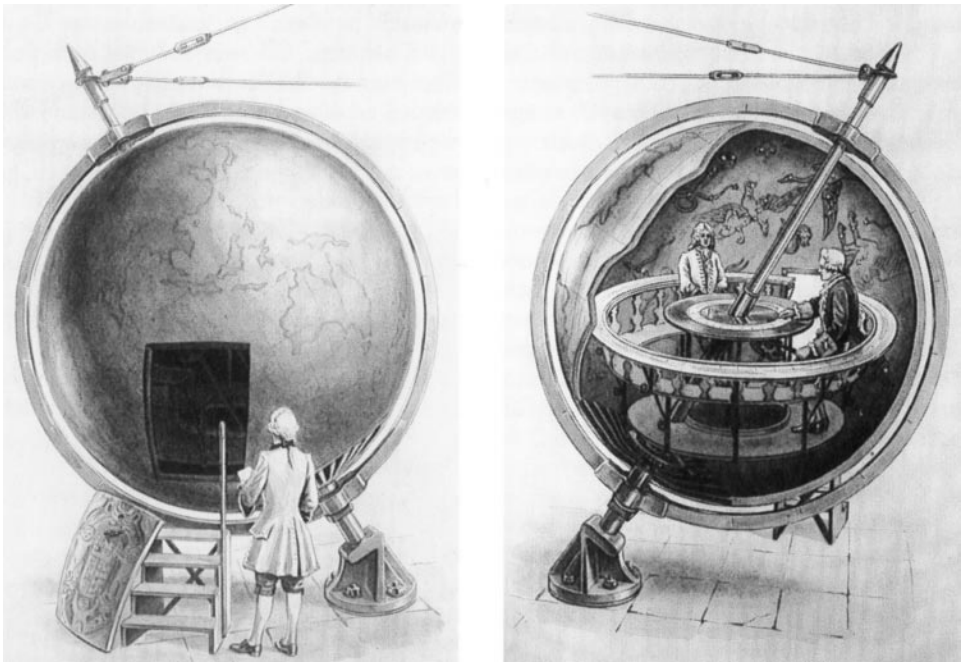


Figure 2.4: The Gottorp Globe allowed up to 12 visitors an impression of the night sky. The globe could be rotated to simulate the daily motions. [Mei92]

To model the view "from the inside", such a globe must be built large enough for humans to enter it. In 1664, such a globe was completed for Duke Friedrich III of Holstein and erected in Gottorp castle (Figure 2.4). The constellations were painted on the inside wall, the stars were small silver or gold balls. Similar globes were built until the early 20th century. [Mei92]

The invention of the projecting planetarium by Walther Bauersfeld of Carl Zeiss in 1919 finally combines the impression of the star globe with a mechanic clockwork for planetary motion, showing all elements projected on the inner wall of a dome and thus allowing to simulate the view of the sky from every place on Earth for every historic date. The stars are projected using holes of different sizes in a metal sheet, producing light patches of according size on the wall. To avoid unnaturalistic impressions by too large patches, the brightest stars are projected with an extra projector. [Mei92]

The development of space exploration and space flight also initiated further development for simulations of celestial views. Starting in the early 1970s, computer controlled projection systems were developed which allowed simulation of sky panoramas including the planets as seen from other locations in space.

The new Hayden Planetarium in New York's Rose Center for Earth and Space presents the current masterpiece in sky simulation [Mar00]. It combines the latest model of the ZEISS IX planetarium projector, which uses fiber optic to produce pinpoint-sized star images as seen from Earth, with a SILICON GRAPHICS ONYX2 computer-controlled Digital Dome system of video projectors to produce views from practically any location in the known universe.

For the average user of personal computers, countless desktop planetarium programs have been developed in the last years. Many of them are limited to earth-based views, but some also allow to move in the Solar System and even beyond. Development of affordable powerful graphics hardware for the consumer market allowed ever more realism. Images, prepared by NASA and others and obtainable from JPL's World Wide Web services, allow a presentation of the planets and many of their moons with realistic surface markings.

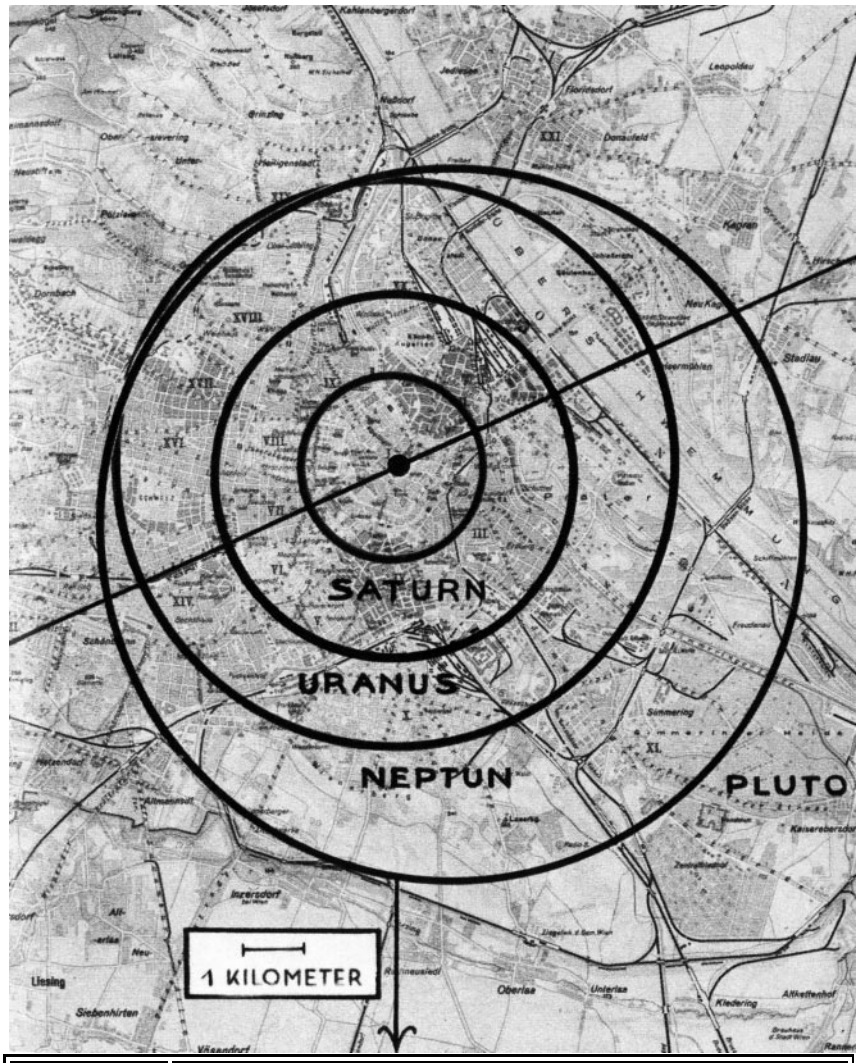
There are now even some sites on the World Wide Web which offer online information about the planets and provide simulated images as seen from other points in space.

2.2 Modeling Distances and Sizes

Compared to the vast distances between the planets, the sizes of their bodies are very small. A model of the Solar System scaled $1 : 1.000.000.000$ would cover the area of the city of Vienna, as presented by OSWALD THOMAS (Figure 2.5) [Tho29, Tho56]. Using equal scale for distances and planet diameters, Earth would be represented as only hazelnut-sized marble, offset $150m$ from a model of the Sun with $1.4m$ diameter. Usually, therefore, planets are modelled much larger in relation to their distances, so that details remain visible at all.

Most planets also have moons, which should of course be included in the model. This raises another problem: If we model the planet and moon bodies with greatly exaggerated size but leave the distances to the moons in the same distance scale as the rest of the system, the moons lie obviously notably too near to, or even inside, their (enlarged) planets. If we use the exaggerated scale also for the moons' distances, the moon orbits may reach far out to the other planets! Thus, we either have to take a third, intermediate scale or really carefully select the exaggerated scale to give at least some impression of the bodies.

For example, the Copernican Planetarium, a detailed automated mechanical model of the Solar System, finished in 1924 by CARL ZEISS for the German Museum in Munich, allowed the explanation of planetary movements in a room of $12m$ diameter. In this model, Earth orbits the Sun in a circle of $4.43m$ diameter. Saturn's orbit would have a diameter of $42.45m$



Object	Distance (m)	Size (cm)	Comparable Body
Sun	—	139	Tractor wheel
Mercury	58	0.5	Small pea
Venus	108	1.2	Hazelnut
Earth	150	1.3	Hazelnut
Moon	38 cm fr. Earth	3.5 mm	Tiny Pea
Mars	228	0.7	Pea
Asteroid	420	0.8 mm	Millet-seed
Jupiter	778	14	Coconut
Saturn	1400	12 (Ring: 28)	Coconut with thin LP record
Uranus	2900	5	Billiard ball
Neptune	4500	5.5	Billiard ball
Comet (tail)	variable	16–160 m	Room . . . House

Figure 2.5: A model of the Solar System, scaled 1 : 1.000.000.000, covering the city of Vienna [Tho56]. The table gives sizes and mean distances of the planets as modeled in the same scale, and an expressive comparison with some well-known object. [Tho29]

in this scale, but was modeled with only $11.25m$ diameter. Also, the globes of Jupiter and Saturn are built similar in size to Earth's $12cm$ globe (which itself is far too large in relation to its distance from center), instead of being ten times larger! [Mei92]

For the naked eye looking into the sky, all planets just look like stars added to the stellar background. Telescopes are required to see surface markings or phase figures. In planetarium systems, the planets are usually projected as small planet images as seen through a small telescope to help the visitor to distinguish them from the background stars.

Current desktop planetarium programs give the user some options how planets shall be presented. Either in natural scale (they will appear as star-like dots), as color-coded dot, with a symbol, or fully-featured, and thus highly exaggerated in scale.

Most programs showing the Solar System "from the outside" give an impression of the relative sizes of the orbits. The planet sizes are widely exaggerated to be able to see them, and usually the exaggeration is not equal for the planets, given, e.g., the $62 : 1$ size relation between Jupiter and Pluto. Sometimes, the orbits are plotted in a logarithmic scale, e.g., in the NATIONAL GEOGRAPHIC SOCIETY's online VIRTUAL SOLAR SYSTEM (see `http://www.nationalgeographic.com/solarsystem/`). However, while this helps getting all orbits into a small image, it gives a very unnatural impression of the Solar System's structure for the casual visitor.

Using a scalable VR model may free the creator of such a model from some aspects of the problem. Getting a close-up look of the surface of an interesting object does not require a microscope, we would just put that object in focus and turn up the scale. To see surface features in an overview with many objects that are physically too far apart to see details, selective scaling can be used to increase the planets' sizes. The model could be built, in principle, to appear very large, as long as the visitor can easily move around so reach every point.

Of course, practically every form of fast "intuitive" movement will break the laws of physics: moving from one planet to the next within seconds would require travels faster than light. These problems, and the problem of image delays due to signal time, are usually silently ignored in such models.

Chapter 3

Astronomical Aspects

The author has been interested in astronomy for many years. A few of the astronomical aspects which went into the development of VRMoSS shall be described in this chapter.

3.1 Astronomical Computing

3.1.1 Books and Data Sources for Astronomical Computing

In 1992, Seidelmann et al. published a revised edition of the 1961 edition of the Explanatory Supplement to The Astronomical Ephemeris and The Americal Ephemeris and Nautical Almanac, using new astronomical theories and presenting new techniques for rigorous calculations with computers. The astronomical constants used in VRMoSS were taken from this source. [Sei92]

The current world authority for computation of planetary ephemerides (tables of positions of astronomical objects) lies at NASA's Jet Propulsion Laboratory (JPL). Their solutions to the positions of the planets, performed by large-scale numerical integration and data-fitting to tens of thousands of observations, provide positions to a few tens of meters' accuracy. A description of the details behind JPL's DE-200 theory can be found in Seidelmann. [Sei92]

Belgian meteorologist Jean Meeus, who has been actively involved in the field of astronomical computing since the early 1960s, presents with his book Astronomical Algorithms [Mee91, Mee98] probably the best currently available compilation of frequently used astronomical computations, including coordinate transformations, calculations around time and date, planetary positions, *etc.* Meeus presents an analytic solution for the positions of the planets, the VSOP87 theory by P. Bretagnon and G. Francou. Its accuracy is not as high as that of the JPL ephemerides, but it can be computed very fast and is sufficient for most astronomical applications, including VRMoSS.

In 1999, HEAFNER [Hea99] presented a collection of algorithms with program code in BASIC and C for various astronomical applications. Most notably, he explains how to use the JPL ephemerides and provides commented source code to access the files in languages other than FORTRAN. The sections most important for VRMOSS describe finding an object's position and speed from orbital elements and time inputs, and especially also for the reverse operation: finding an object's orbital elements, given its position and speed and the time of observation.

3.1.2 Time

Civil time as it is used today is based on Universal Time (UT) or Greenwich Civil Time[1], which is coupled to the rotation of the Earth. The speed of this rotation was considered invariable until the late 19th century. Research on solar eclipses then showed an apparent acceleration in the Moon's angular velocity around the Earth, which was compensated by empirical corrections in the Lunar theories of that time. However, the invention of clocks with ever higher precision have since shown that Earth itself does not rotate uniformly, but does slow down, due to tidal forces caused by the Moon and, to a lesser extent, by the Sun. To make matters even more complicated, this deceleration is irregular and unpredictable. Thus, UT is not a uniform time scale.

For astronomical computations in the field of celestial mechanics, however, a uniform time scale is essential. The Ephemeris Time (ET), based on planetary motions, which was used 1960–1983, was replaced in 1984 by Dynamical Time (TD, sometimes DT), which is kept by atomic clocks.

Dynamical Time is the uniform time that appears in an object's equation of motion. It always passes at the same rate, but, due to the effect of general relativity, this rate is not equal for all observers.

We discern a Barycentric Dynamical Time (TDB) and a Terrestrial Dynamical Time (TDT, later shortened to TT). However, the difference between these times, always less than 0.0017 seconds, can be safely neglected for most practical purposes. Details can be found in [Mee98] and [Hea99].

The exact value of the difference $\Delta T = TD - UT$ cannot be predicted, but only derived by observations. MEEUS [Mee98] gives a table for the years 1620–1998, approximation formulae for the past and estimations for the (near) future. So, to compute Earth's rotation with respect to the Sun, one has to find ΔT for the given TD and adjust the rotation accordingly. Currently, ΔT grows about one second per year.

[1] UT is frequently called Greenwich Mean Time (GMT). By definition, however, *mean* time is measured from the superior transit of the *mean* Sun, which occurs at *mean noon*. Thus, GMT and UT differ by 12 hours!

3.1.3 Julian Day Number

The traditional civil way of specifying a instant of time, using a calendar that breaks time into years (some of them leap years), months of unequal length (and sometimes leap days inserted!), days, hours, minutes, and seconds, is rather complicated to handle. For example, there is no year 0 in the Julian Calendar used in common western historical literature[2]. However, for astronomical purposes, a year 0 is accepted practice, and is equal to the year 1 B.C., year -1 is 2 B.C., *etc.* Errors with handling these matters are frequent and sometimes hard to find.

In the year 1582, JOHANN JUSTUS SCALIGER, to simplify date conversions between calendars, most notably between the Julian Calendar and the then new Gregorian Calendar, introduced a simple day count, which he named after his father JULIUS: the Julian Day Number JD[3]. Using JD, an instant of time can be specified using a single number.

JD, as gladly accepted and used in astronomy, is the number of days that have passed since noon of January 1st, 4713 B.C. = -4712 01 01 (Julian), 12:00 h UT [4]. For instants of time given in TD, Julian Ephemeris Day (JDE, sometimes JED) is used. MEEUS [Mee98] gives simple algorithms for conversion between JD and Gregorian and Julian Calendar dates.

3.1.4 Earth and Sun

The common use of time relates the day to the apparent daily motion of the Sun over the sky, mirroring Earth's rotation around its axis. 24 hours after the Mean Sun has crossed the meridian (stood highest in the sky, defining the instant of Mean Noon), it again crosses the meridian.

However, Earth also has moved in space, and, reflecting this motion, the Sun has moved about one degree eastward in relation to the stars. This means that it does not take full 24 hours for the stars to be in the same position in the sky, but about 4 minutes less, or $23^h56^m4.1^s$.

Earth, by its rotation, defines a great circle on the celestial sphere: the celestial equator. It is the projection of Earth's equator on the celestial sphere.

By its movement around the Sun, it defines another great circle: the ecliptic, or zodiac. Mirroring this motion, this is the apparent path of the Sun amongst the stars.

Earth's rotational axis is tilted from the normal to its orbital plane by about 23.5 degrees. This angle is not perfectly constant, but changes very

[2]The Romans had no idea of the number zero, thus they always started counting with one. Zero, and the indian-arabic way to write numbers, were only slowly introduced in Europe starting in the 11th century.

[3]The term "Julian Date" should be avoided to prevent confusion with a date given in the Julian Calendar, which is named after JULIUS CÆSAR.

[4]Starting the count from noon helped avoiding errors with observations performed during night times in Europe.

slowly. Celestial Equator and ecliptic, being great circles, necessarily intersect each other at this angle at opposite points in the sky. Thus the Sun, moving along the ecliptic, crosses the celestial equator twice during a year.

When the Sun reaches its southernmost point in the ecliptic around December 21st each year, this defines the beginning of winter in the northern hemisphere (Winter Solstice). Crossing the celestial equator from south towards north (around March 21st) defines the beginning of spring (Vernal Equinox). Its arrival at the northernmost point (Summer Solstice) defines the beginning of summer for the northern hemisphere, and the crossing into the southern celestial hemisphere (around September 23rd), called Autumnal Equinox, marks the beginning of autumn.

During the (north) summer half of the year, Earth's north pole is tilted towards the Sun, thus the northern hemisphere receives more of Sun's radiation. For a terrestrial observer on the northern hemisphere, the Sun stands higher in the sky and is longer above the horizon. The north pole even is in permanent sunlight.

In the (north) winter half of the year, Earth's south pole is tilted towards the sun, the southern hemisphere receives more sunlight. The Sun's rays hit the northern hemisphere at a very flat angle. For an observer on the northern hemisphere, the Sun rises late, moves low in the sky, and sets early. The lack of sunlight leads to a drop in surface temperature. The north pole receives no sunlight at all.

A popular misbelief is that the seasons are related to the distance of the Earth from the Sun. The interesting fact is that Earth is closest to the Sun in early January, and farthest from the Sun in early July each year. However, the eccentricity of Earth's orbit is so small that the difference in radiation is insignificant and can safely be neglected.

3.1.5 Coordinate Systems

For positional astronomy, there are several spherical coordinate systems in use, serving different purposes. Earth is always seen as lying in the center of a sphere of infinite radius.

Horizontal Coordinates

An observer standing on Earth defines his own coordinate system: The horizon defines a great circle. The angle between the north point on the horizon and an observed object, as counted from north towards east, is called its azimuth[5]. The angular distance between horizon and an object on a great circle perpendicular to the horizon is called altitude. The vertical axis, defined by gravity, points up to the zenith, the point opposite the zenith, towards the observer's feet, is called nadir.

[5]Some authors count azimuth from south. Care must be taken to avoid confusion!

Azimuth and altitude of a celestial object depends on the observer's location (geographical coordinates), its real position in the sky and the current rotational state of the Earth (which is, in fact, a function of time TD).

Sidereal Time

As described above, civil noon occurs (approximately) when the Sun crosses the meridian. However, when we observe a single star, it will not return to its apparent place after 24 hours, but it will be early by almost four minutes, due to the fact that Earth has moved on its path around the Sun.

The interval it takes a star from one upper culmination (crossing the meridian at highest altitude) to the next is termed one sidereal day. It is exactly the time it takes Earth to rotate once around its axis with respect to the stars, 23 hours, 56 minutes and 4.1 seconds.

If we define a certain point on the celestial sphere as origin of a spherical longitude system, each time this point reaches upper culmination can mark the begin of a new sidereal day. Such a point has been used since antiquity: the intersection between ecliptic and celestial equator, where the Sun on its apparent yearly path crosses the celestial equator from south towards north. The point is called, for historic reasons, First Point of Aries or Vernal Equinox, and labeled with the symbol Υ, and is used as origin for both the equatorial and ecliptical coordinate systems.

When the First Point of Aries crosses the local meridian, sidereal time is zero hours. Sidereal time, due to its similarity with civil time, usually is counted in hours, not degrees. Slightly less than a (civil) hour later, we have one o'clock sidereal time, and so on.

Note that the year has about 365.25 solar days, but 366.25 sidereal days. By going around the Sun, Earth loses a solar day each year.

Hour Angle

The Hour Angle of a celestial object is the interval on the sidereal time scale that has passed since the upper culmination of its position. This makes sidereal time equal to the hour angle of the First Point of Aries.

Equatorial Coordinates

The rotation of the Earth defines the Equatorial Coordinate System. The projection of Earth's equator in the sky is called Celestial Equator. Objects lying on the celestial equator have a declination δ of zero degrees. Declination is counted in degrees, positive towards the North Celestial Pole ($\delta = 90°$), negative towards the South Celestial Pole ($\delta = -90°$).

The longitude coordinate is counted in hours, minutes and seconds from the first point of Aries Υ: it equals exactly the sidereal time of the instant of its upper culmination and is termed Right Ascension α.

The equatorial system is the most widely used system for stellar coordinates. It is independent of the observer's location, and the computation of its position for pointing a telescope towards the object is straightforward: The object's hour angle is found by just subtracting its right ascension from the current sidereal time. Aligning a telescope parallel to Earth's axis allows to set it to the object's declination directly, and use the object's hour angle as longitude coordinate! It was only required to adjust one telescope axis to keep the object centered in the telescope's field of view, a process usually performed by a relatively simple mechanical clock running at a constant speed. This has been astronomical practice for practically all astronomical telescopes since the early 17th century. Of course, the necessity of mounting ever growing telescopes parallel to Earth's axis, with no flexion in the mechanical parts of the telescope, required the construction of very heavy mounts, which made large instruments exceedingly expensive. Only starting in the late 1980, the development of computer controls for very large instruments allowed the construction of telescopes with the simpler altazimuth mount. Nowadays, practically all new large telescopes are built with a computer controlled altazimutal mount. Azimuth and altitude (for pointing a horizontally mounted telescope) can be found by simple operations [Mee98].

Ecliptical Coordinates

The Ecliptic Coordinate System is defined by Earth's motion around the Sun. The "equator" of this system is the ecliptic. Ecliptical latitudes β are counted from the ecliptic ($\beta = 0°$) towards North Ecliptical Pole ($\beta = 90°$) and South Ecliptical Pole ($\beta = -90°$).

Ecliptical Longitude λ is counted in degrees along the ecliptic, starting, like the equatorial coordinates, from the vernal equinox Υ, and going eastward. Thus, the Sun's ecliptical longitude increases by about one degree per day, and its passage through $360° = 0°$ marks the beginning of spring for the northern hemisphere.

The ecliptical coordinate system is used mainly with objects of our solar system. When dealing with coordinates in our solar system, however, the spherical coordinates can be enriched with a radial distance r, usually given in Astronomical Units AU[6].

[6]The astronomical unit has been defined as the semi-major axis of Earth's orbit around the Sun. It was a very convenient unit: astronomers could calculate distances without actually knowing the real distances, using KEPLER's third law (See page 16). In the 18th and 19th centuries, several excellent estimates were performed by careful observations of planet parallaxes (triangulation method). In the 20th century, radar allowed to make measurements of unprecedented accuracy. The value has now been frozen to $1AU \approx 149.6 \times 10^6 km$.

These values can be easily transformed into right handed cartesian coordinates, with the x coordinate pointing towards the Vernal Equinox Υ, y pointing towards $\lambda = 90°$, and z pointing towards ecliptical north:

$$\begin{aligned} x &= r\cos\beta\cos\lambda \\ y &= r\cos\beta\sin\lambda \\ z &= r\sin\beta \end{aligned} \tag{3.1}$$

Object coordinates in cartesian coordinates must always be given with respect to a center. For Earth based observations, this must be Earth itself, but the usual application for cartesian coordinates are values with respect to the Sun. When building a model of the Solar System, heliocentric cartesian coordinates are the natural choice.

Galactic Coordinates

The Sun is part of a huge disk, about 100.000 light years[7] in diameter, consisting of about a hundred billion stars: the Milky Way galaxy. The Solar System lies tilted arbitrarily to the galactic equator. It is not easy to determine this equator, as it is heavily obscured by dark clouds. A first system of galactic coordinates, defined in the early 20th century, used the intersection of what was the best guess on the galactic equator with the celestial equator as starting point for galactic longitude l.

In 1959, a new system (System II) was introduced, based on a new determination of the galactic equator by radio astronomy. The galactic center is identified by radio source Sagittarius A.

Galactic longitude l^{II} is measured in degrees from the direction towards the galactic center, in the same sense as right ascension. Galactic latitude b^{II} is measured in degrees as well.

This coordinate system is mainly used for applications of stellar statistics, concerning the structure of the Milky Way. In the wider context, the distribution of galaxies, quasars, *etc.*, in the universe is also usually given in galactic coordinates.

Precession

The orientation of Earth's axis seems to define very neatly the equatorial and ecliptical coordinate systems. However, Earth is not a sphere, but approximately a rotational ellipsoid. The Moon, on its orbit around Earth, as well as the Sun, tugs on the equatorial bulge and tries to pull the Earth upright from its 23.5° axis tilt. Like any other spinning top, Earth reacts by a sideways movement of its axis: Precession. Within about 26.000 years, Earth's

[7] One light year is the distance light travels during one Earth year, which is about $9.46 \times 10^{12} km$.

axis describes a complete conic rotation. This motion is reflected in stellar coordinates: all stars shift their positions by about one degree in 72 years, or $50''$ annually, in positive sense along the ecliptic, hence *pre-cession*.

Moreover, in addition to this lunisolar precession, the planets influence Earth's orbit (planetary precession), therefore the ecliptic plane slowly rotates around a line of nodes, currently at $47''$ per century.

Because of this, all star maps and coordinates in general have to be given with respect to a specified date, the epoch. The current standard epoch is designated $J2000.0$, equivalent to $JD2451545.0$ or January 1st, 2000, 12:00 TD.

3.1.6 Kepler Orbits

KEPLER's laws of planetary motion (page 16) describe a special case of an orbit under the influence of gravity: a large central mass, and a practically massless body moving on an elliptic orbit, in one focus of which the large mass resides.

The motion of the planets (and smaller bodies like asteroids and comets, and also spacecraft when their thrusters are inactive) can indeed be approximated with such orbits, as long as no great accuracy is needed.

The undisturbed orbit of a (massless) body around a central, massive body can be described by six constants. The classical set of orbital elements (Figure 3.1) consists of:

q pericenter distance, given in AU

e numerical eccentricity

i inclination with respect to some reference plane

ω argument of pericenter

Ω longitude of ascending node

T time of passage through pericenter

When we describe the orbit of objects around the Sun, the reference plane is always the ecliptic, the orbital plane of Earth. The pericenter, the point of closest approach to the central body, is then called perihelion. The x-axis in the cartesian heliocentric coordinate system points towards the vernal equinox Υ, the y-axis points towards $90°$ ecliptical longitude, and the z-axis points to the ecliptical north pole.

The ascending node Ω is the ecliptical longitude of the point where the body on its way around the Sun intersects the ecliptic plane from ecliptical south towards north.

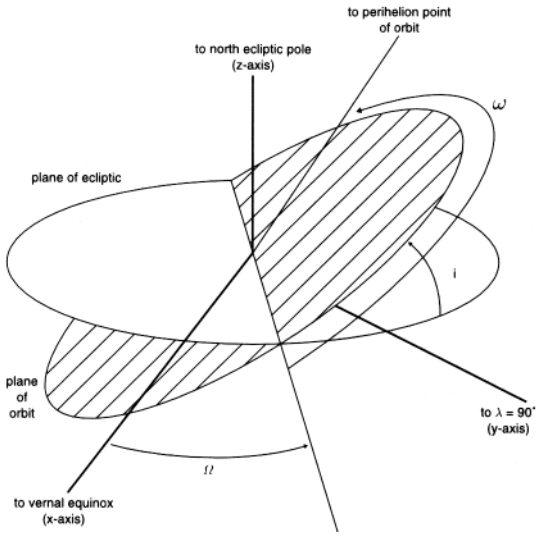


Figure 3.1: Illustration of the orbital elements that determine the orbit's orientation in space. [Hea99]

The inclination i is the angle between the orbital and the ecliptic planes, and is a value in the range $[0° \ldots 180°]$. If the object moves around the Sun in the same sense as the planets (direct or prograde motion; increasing ecliptical longitude), inclination is smaller than 90°. If it appears to be in a retrograde orbit around the Sun, this is expressed in the inclination lying between 90° and 180°.

The argument of perihelion ω is the angle between the ascending node and the point of perihelion, measured along the object's orbit. The name *argument* is used to discern it from the term longitude of perihelion $\tilde{\omega} = \omega + \Omega$, which is sometimes given instead of ω, especially if i is very small, and thus difficult to determine. For retrograde orbits, where i is near 180°, the retrograde longitude of perihelion $\tilde{\omega}_r = \omega - \Omega$ is then given.

These elements are, strictly speaking, only valid for the epoch (date) which must therefore also be given. They describe the orientation of the orbit in relation to the ecliptical system of some standard equinox, which is currently $J2000.0$. The effects of mutual gravitational perturbations by the planets become visible in changes in the orbit's shape and orientation. Thus these elements, which are called osculating elements, describe the orbit only at epoch time, and should not be used outside a limited range of dates, depending on accuracy requirements. Changes in the elements can only be computed by numerical simulation of the forces which act on the bodies. Algorithms for the much simpler transformation of orbital elements between equinoxes can be found, e.g., in [Mee98].

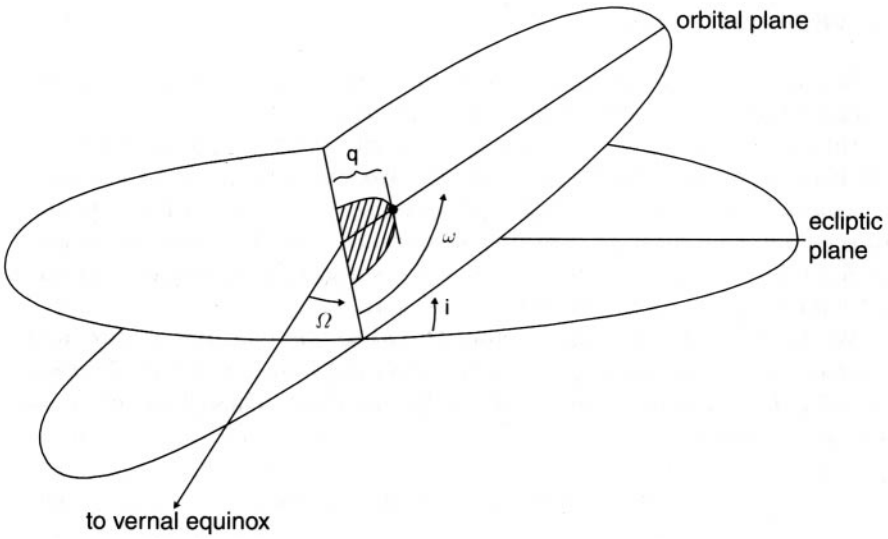


Figure 3.2: Shaded portion indicates actual orbit. The object is shown at perihelion, $\tilde{\omega} = \Omega + \omega$. [Hea99]

These six elements are all that is needed for an object on an orbit, be it elliptic ($e < 1$), parabolic ($e = 1$) or even hyperbolic ($e > 1$). This generality is useful, because comets often move on orbits with these extreme eccentricities. However, for asteroids, which orbit the Sun on orbits of usually small eccentricity, frequently the orbit's large semimajor axis a is given instead of q, and perihel passage T is replaced by giving a value for the object's mean anomaly M for the epoch. $M = 0$ at epoch T.

The perihelion distance q can be calculated from a as

$$q = a(1 - e) \tag{3.2}$$

M is a fictitious angle of the object measured in the plane of the orbit, from the perihelion point to the body. M grows with a constant rate, the mean motion n, which can be determined from a as

$$n = \sqrt{\frac{\mu}{|a|^3}} \qquad \text{radians/day} \tag{3.3}$$

Here, $a = q/(1 - e)$
$\mu = k^2(1 + m)$
$k = 0.01720209895$ (Gaussian constant of gravitation)
$m =$ Body's mass in fractions of mass of central body

Neglecting the body's mass in relation to the Sun's is a common safe approximation.

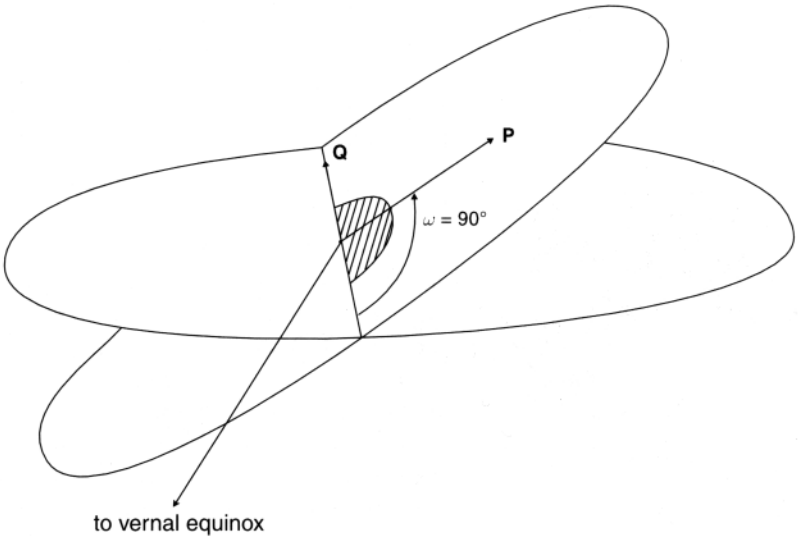


Figure 3.3: Vector **P** points toward the object's perihelion point. Vector **Q** points 90° from **P** in the orbital plane. Here, because $\omega = 90°$, **Q** also coincides with the orbit's descending node. **P** and **Q** form an inertial reference frame. [Hea99]

We begin the calculation of an object's position by computing two unit vectors **P** and **Q**. **P** points from the Sun towards the perihelion point of the orbit, and **Q** points 90° from **P** in the direction of the object's motion (Figure 3.3). To avoid numerical problems with i near 0° or 180°, we use the elements $\tilde{\omega} = \omega + \Omega$ and $\tilde{\omega}_r = \omega - \Omega$.

$$\begin{aligned}
\mathbf{P}_x &= \frac{1}{2}\Big((1+\cos i)\cos\tilde{\omega} + (1-\cos i)\cos\tilde{\omega}_r\Big) \\
\mathbf{P}_y &= \frac{1}{2}\Big((1+\cos i)\sin\tilde{\omega} - (1-\cos i)\sin\tilde{\omega}_r\Big) \qquad (3.4) \\
\mathbf{P}_z &= \sin\omega\sin i \\
\mathbf{Q}_x &= -\frac{1}{2}\Big((1+\cos i)\sin\tilde{\omega} + (1-\cos i)\sin\tilde{\omega}_r\Big) \\
\mathbf{Q}_y &= \frac{1}{2}\Big((1+\cos i)\cos\tilde{\omega} - (1-\cos i)\cos\tilde{\omega}_r\Big) \qquad (3.5) \\
\mathbf{Q}_z &= \cos\omega\sin i
\end{aligned}$$

Next, we need the quantities $r\sin\nu$ and $r\cos\nu$, where r is the object's distance from Sun and ν is the true anomaly. The method of finding those values depends on the excentricity of the object's orbit.

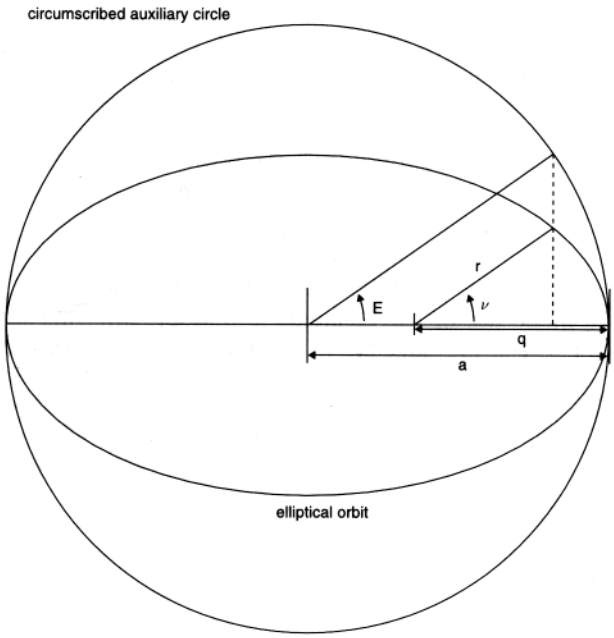


Figure 3.4: Illustration of perihelion distance, true anomaly, semi-major axis, and eccentric anomaly for an elliptic orbit. [Hea99]

Elliptical Orbits — Solving the Equation of Kepler

The mean anomaly M can be easily computed as given above. This angle is zero at perihelion passage and grows by the angle of n each day. We now need to determine the true anomaly ν of the object. For this, we introduce another auxiliary angle, the eccentric anomaly E (Figure 3.4). The Equation of Kepler is then

$$E = M + e \sin E \tag{3.6}$$

This equation must be solved for E. Many iterative methods of solution have been worked out since KEPLER's time for this transcendental function. The LAGUERRE-CONWAY method presented in [Hea99] apparently does not always converge as fast as promised. Therefore we follow the method given in [Mee91].

Starting with $E_0 = M$, we compute

$$E_{n+1} = E_n + \frac{M + e \sin E_n - E_n}{1 - e \cos E_n} \tag{3.7}$$

until $E_{n+1} \approx E_n$ to sufficient accuracy. All angles are expressed in radians. MEEUS demonstrates in [Mee91] that this solution is numerically unstable for

$e > 0.975$ and $|M| < 30°$. In these cases, instead of starting with $E_0 = M$, we need a better initial value for E_0, which can be found from:

$$\alpha = \frac{1-e}{4e+0.5} \tag{3.8}$$

$$\beta = \frac{M}{8e+1} \qquad (-\pi < M < \pi) \tag{3.9}$$

$$z = \sqrt[3]{\beta + \mathrm{sign}(\beta)\sqrt{\beta^2+\alpha^3}} \tag{3.10}$$

$$s_0 = z - \frac{\alpha}{2} \tag{3.11}$$

$$s = s_0 - \frac{0.078 s_0^5}{1+e} \tag{3.12}$$

$$E_0 = M + e(3s - 4s^3) \tag{3.13}$$

The values r and ν, or directly the required combinations can now be obtained from

$$\tan\frac{\nu}{2} = \sqrt{\frac{1+e}{1-e}}\tan\frac{E}{2} \tag{3.14}$$

$$r = a(1 - e\cos E) \tag{3.15}$$

$$r\sin\nu = a\sqrt{1-e^2}\sin E \tag{3.16}$$

$$r\cos\nu = a(\cos E - e) \tag{3.17}$$

Parabolic Orbits

In case of a parabolic orbit, the procedure is simpler. One way to proceed [Mee98] starts with

$$W = \frac{3k/\sqrt{2}}{q\sqrt{q}}(t-T) \tag{3.18}$$

where $k = 0.01720209895$ Gaussian gravitational constant
$(t-T)$ time from perihel, days
and needs the solution of BARKER's Equation

$$s^3 + 3s - W = 0 \tag{3.19}$$

We find

$$G = \frac{W}{2} \tag{3.20}$$

$$Y = \sqrt[3]{G + \sqrt{G^2+1}} \tag{3.21}$$

$$s = Y - \frac{1}{Y} \tag{3.22}$$

and the required final values by

$$\nu = 2\,\mathrm{atan}\,s \tag{3.23}$$

$$r = q(1+s^2) \tag{3.24}$$

Near-parabolic Hyperbolic Orbit

Very rarely comets on orbits with eccentricities $e > 1$ appear: they move on hyperbolic orbits. MEEUS [Mee98] presents LANDGRAF's method, which is based on STUMPFF's HIMMELSMECHANIK.

Start with computing $s_0 = \tan \frac{\nu}{2}$ with the algorithm for parabolic orbits (3.22), and calculate

$$Q = \frac{k}{2q}\sqrt{\frac{1+e}{q}} \tag{3.25}$$

$$\gamma = \frac{1-e}{1+e} \tag{3.26}$$

Then, with t the time from perihelion (days), iterate

$$s_{n+1} = Qt - (1-2\gamma)\frac{s_n^3}{3} + (2-3\gamma)\frac{s_n^5}{5} - (3-4\gamma)\frac{s_n^7}{7} + \cdots \tag{3.27}$$

until $s_{n+1} \approx s_n$. The true anomaly ν and distance to the Sun are then

$$\nu = 2 \operatorname{atan} s \tag{3.28}$$

$$r = \frac{q(1+e)}{1+e\cos\nu} \tag{3.29}$$

The State Vector

With values $r\sin\nu$ and $r\cos\nu$ determined, the position vector $\mathbf{r}$ and velocity vector $\dot{\mathbf{r}}$, which together are termed the state vector, can be calculated as

$$\mathbf{r} = r\cos\nu\mathbf{P} + r\sin\nu\mathbf{Q} \qquad AU \tag{3.30}$$

$$\dot{\mathbf{r}} = \sqrt{\frac{\mu}{p}}\Big((e+\cos\nu)\mathbf{Q} - \sin\nu\mathbf{P}\Big) \qquad AU/\text{day} \tag{3.31}$$

where $\mu = k^2(1+m) \approx k^2$ (neglecting the object's mass)
$p = q(1+e)$ semilatus rectum

3.1.7 Finding the Orbital Elements from the State Vector

HEAFNER [Hea99] presents a method to calculate the orbital elements from the state vector $[\mathbf{r}\dot{\mathbf{r}}]$, which, together with the state vector's epoch t, describe the object's instantaneous position and motion around the Sun, but does not describe any disturbations. The orbit determined by this process may be termed instantaneous Kepler orbit.

Stumpff Functions The following method makes use of the Stumpff function $c_n(x)$, named after German astronomer KARL STUMPFF.

$$c_n(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^k}{(2k+n)!} \qquad k = 0, 1, 2, \ldots \tag{3.32}$$

Note that, e.g.,

$$c_3(x^2) = \frac{x - \sin x}{x^3} \qquad c_3(-x^2) = \frac{\sinh x - x}{x^3} \tag{3.33}$$

We start by calculating the angular momentum vector:

$$\mathbf{L} = \mathbf{r} \times \dot{\mathbf{r}} \tag{3.34}$$

Next, let

$$r = |\mathbf{r}| \tag{3.35}$$
$$L = |\mathbf{L}| \tag{3.36}$$
$$\mathbf{W} = \frac{\mathbf{L}}{L} \qquad \text{(Unit vector)} \tag{3.37}$$
$$\mathbf{U} = \frac{\mathbf{r}}{r} \qquad \text{(Unit vector)} \tag{3.38}$$
$$\mathbf{V} = \mathbf{W} \times \mathbf{U} \qquad \text{(Unit vector)} \tag{3.39}$$
$$p = \frac{L^2}{\mu} \qquad \text{where} \quad \mu = k^2(1+m) \quad \text{(see Eq.3.3)} \tag{3.40}$$
$$\alpha = \frac{2\mu}{r} - \dot{\mathbf{r}} \cdot \dot{\mathbf{r}} \tag{3.41}$$

Now, e and q can be determined directly from

$$e = \sqrt{1 - \alpha \frac{L^2}{\mu^2}} = \sqrt{1 - \alpha \frac{p}{\mu}} \tag{3.42}$$
$$q = \frac{p}{1+e} \tag{3.43}$$

For i and Ω, we need

$$\chi = \frac{\sin i \sin \Omega}{1 + \cos i} = \frac{\mathbf{W}_1}{1 + \mathbf{W}_3} \tag{3.44}$$
$$\psi = \frac{\sin i \cos \Omega}{1 + \cos i} = \frac{-\mathbf{W}_2}{1 + \mathbf{W}_3} \tag{3.45}$$

Then, with a function atan2(a, b), which gives the angle of $\tan \frac{a}{b}$ in the correct quadrant,

$$\sin i = \frac{2\sqrt{\chi^2 + \psi^2}}{1 + \chi^2 + \psi^2} \tag{3.46}$$

$$\cos i = \frac{1 - (\chi^2 + \psi^2)}{1 + \chi^2 + \psi^2} \tag{3.47}$$

$$\sin \Omega = \frac{\chi}{\sqrt{\chi^2 + \psi^2}} \tag{3.48}$$

$$\cos \Omega = \frac{\psi}{\sqrt{\chi^2 + \psi^2}} \tag{3.49}$$

$$i = \text{atan2}(\sin i, \cos i) \tag{3.50}$$

$$\Omega = \text{atan2}(\sin \Omega, \cos \Omega) \tag{3.51}$$

Next, with $\dot{r} = \dot{\mathbf{r}} \cdot \mathbf{U}$ and $r\dot{\nu} = \dot{\mathbf{r}} \cdot \mathbf{V}$, we find

$$e \sin \nu = \dot{r} \sqrt{\frac{p}{\mu}} \tag{3.52}$$

$$e \cos \nu = r\dot{\nu} \sqrt{\frac{p}{\mu}} - 1 \tag{3.53}$$

from which we could find $\nu = \text{atan2}(e \sin \nu, e \cos \nu)$, if this value was directly required. Instead, we proceed with

$$z = \tan \frac{\nu}{2} = \frac{e \sin \nu}{e + e \cos \nu} \tag{3.54}$$

$$\cos \nu = \frac{1 - z^2}{1 + z^2} \tag{3.55}$$

$$\sin \nu = \frac{2z}{1 + z^2} \tag{3.56}$$

$$\mathbf{P} = \mathbf{U} \cos \nu - \mathbf{V} \sin \nu \tag{3.57}$$

$$\mathbf{Q} = \mathbf{U} \sin \nu + \mathbf{V} \cos \nu \tag{3.58}$$

$$\omega = \text{atan2}(\mathbf{P}_z, \mathbf{Q}_z) \tag{3.59}$$

To find the time of perihelion passage T, we define

$$w = z \sqrt{\frac{q}{\mu(1 + e)}} \tag{3.60}$$

and approximate s with the first terms ($k = [0 \ldots 7]$ is enough) of

$$s = 2w \sum_{k=0}^{\infty} (-1)^k \frac{\alpha^k w^{2k}}{2k + 1} \tag{3.61}$$

Now we can use the Universal Kepler Equation

$$t - T = qs + \mu e s^3 c_3(\alpha s^2) \tag{3.62}$$

and, using $c_3(.)$ (Equ. 3.33) and α (Equ. 3.41), solve it for T:

$$T = t - qs - \mu e s^3 c_3(\alpha s^2) \tag{3.63}$$

3.2 Star Data

NASA's Astronomical Data Center (ADC) published several CD-ROMs with astronomical catalogs.

A popular star catalog used in creation of planetarium skies is the YALE BRIGHT STAR CATALOGUE, 5th edition, found on ADC's SELECTED ASTRONOMICAL CATALOGS, VOL. 1 [BG91].

Of the 9110 entries in the first edition (1908), 14 turned out to be novae (explosive outbursts of much dimmer stars) or nonstellar objects, leaving 9096 stars which can be seen with the naked eye, making it an excellent source for simulations of the night sky as seen from Earth (or from any other location within the Solar System) without optical aids like binoculars or telescopes.

The catalog provides, among other data, position data in equatorial and galactic coordinates, data about proper motion, spectral classification, color indices $(B-V)$, $(U-B)$, and $(R-I)$, where available, double/multiple star information, cross references to other catalogs, and lots of annotations.

3.2.1 Astronomical Magnitudes

Since the times of HIPPARCHOS (2nd century B.C.), the stars' brightnesses have been described with magnitude values. The first stars that appeared after sunset were said to be of "first magnitude", then appeared stars of "second magnitude", and so on. The dimmest stars commonly visible with the naked eye were classified with magnitude 5.

In 1857, the English astronomer POGSON formulated a mathematical description of the stellar magnitude scale [WZ71], which was fitted to the classical system: The brightness difference between stars with radiation intensities I_1 and I_2, which could be measured, e.g., with a photometer, is given as

$$m_1 - m_2 = -2.5 \log \frac{I_1}{I_2} = 2.5 \log \frac{I_2}{I_1} \tag{3.64}$$

This logarithmic scale has been defined to fit the human eye's logarithmic perception of brightness differences.

Solved for intensity ratio $\frac{I_1}{I_2}$, this gives

$$\frac{I_1}{I_2} = 10^{0.4(m_2 - m_1)} \tag{3.65}$$

Thus, two stars with a brightness difference of one magnitudo (mag) differ in (linear) intensity by $10^{0.4} \approx 2.512$, 2.5 mag difference correspond to a brightness ratio of $1:10$, and 5 mag difference describes a ratio of $1:100$. The brightest stars are now given magnitudes of zero and even negative values. So, a star of first magnitude has only about 40% the brightness of a star of zero magnitude.

3.2.2 Spectral Classification

An attentive observer of the night sky can see a star not only as white pinpoint, but often with a slight tint of color. Some stars are bluish-white, some shine golden-orange, and a few show a distinct red.

In the late 19th century, this mystery was solved: the color represents the wavelength of maximum intensity in the black-body radiation curve of the star, depending on the star's surface temperature T (in Kelvin), as described by WIEN's Displacement Law:

$$\lambda_{max} = \frac{0.002897756[m \cdot K]}{T[K]} \tag{3.66}$$

The stars were classified according to this maximum wavelength and by other characteristical features in their spectra. The Harvard scheme classifies stars into spectral classes designated by letters O, B, A, F, G, K, M[8]. Each class is further split into subclasses $0 \ldots 9$, so that, e.g., $F9$ is followed by $G0$. Table 3.1 [Bur78] gives some details.

Class	Temperature (K)	Color
O	35000	blue-white
B	20000	bluish-white
A	10000	white
F	7000	yellow-white
G	6000	yellow
K	4000-4700	orange
M	2500-3000	red

Table 3.1: The most common spectral classes of the Harvard scheme

A roman letter (see Table 3.2) adds the luminosity class.

In this system, our Sun, with its surface temperature of about $6000K$, is classified as $G2V$.

[8]This letter system started just by classifying the spectra after their characteristic lines into groups A, B, C, *etc.* Later, some groups were joined, and the order was changed to put the blue-white, hot stars first and the red, cool stars last. There are more classes, W, R, N, S, which are used for stars with certain special characteristics. See general astronomical literature for details.

Class	Name
Ia	Most luminous supergiants
Ib	Less luminous supergiants
II	Bright giants
III	Normal giants
IV	Subgiants
V	Main sequence
VI	Subdwarfs

Table 3.2: Stellar Luminosity Classes [Bur78]

3.3 Constellations

Since ancient times, people have ordered patterns in the night sky into figures: the constellations. Many of these constellations, mostly depicting the animals of the zodiac and heroes of Greek mythology, are still in use today.

Until about 1800, these figures were always included in stellar maps, sometimes they were painted so colorful that they even obscured the stars [Ses91].

The atlases of the 19th century usually still show figures, but only slightly sketched. Between the figures, curved lines are used to separate the area of sky that belongs to each constellation.

Today's atlases no longer show figures. In 1930, the International Astronomical Union (IAU) fixed the number of constellations to 88 and defined the constellations' areas in the sky with rectilinear borders in the equatorial coordinate system of equinox 1875. These borders are fixed in relation to the stars[9], thus they change with precession, so that they no longer are parallel to the equatorial coordinate grid. These borders are now plotted in modern sky maps.

Many popular maps also use simple straight line drawings which connect the main stars into a figure. There is no standard for these figures, and great care should be taken when inventing such a set of lines. H. A. Rey [Rey76] shows the obvious difference between selections that help identifying the constellation and selections that makes identifying the figure practically impossible.

[9]It is possible for a star to change constellation: by its own proper motion in the sky.

Chapter 4

Tools

VRMoSS has been implemented using several tools, which are presented in this chapter.

4.1 Open Inventor

4.1.1 Introduction

Open Inventor, originally developed in the early 1990s by Silicon Graphics (SGI), is one of the de-facto standard toolkits for interactive 3D graphics applications. It is an extensive C++ class library that allows object-oriented graphics development [Wer94a].

Open Inventor is a set of building blocks that enables the programmer to use the powerful features of modern graphics hardware without the need of direct OpenGL programming. The library provides database primitives, such as shape, transformation, property, group or engine objects, interactive manipulators like trackball and handle box, and components like viewers, a material editor, or a light editor, which can help in modeling scenes. There is the Inventor 3D file format, which is very similar to VRML 1.0, allowing data exchange with other applications.

Open Inventor is independent of the used window system and operating system. Component libraries exist for the X Window System on UNIX platforms and for Microsoft® Win32 platforms. Open Inventor has been written in C++ and is most used with C++, but also includes C bindings. In 2000, SGI decided to distribute their version under the open source license. Recently TGS, who had further developed Open Inventor and added a few extensions esp. for Win32 platforms, introduced Open Inventor Version 3.0, also supporting Java.

Open Inventor allows the developer of a 3D graphics appplication to think on a high level of abstraction. The scene objects (shape, transformation, light, color, texture, ...) are kept in a scene database, and various

actions, such as drawing, bounding box calculation, or interaction, e.g., picking an object with the mouse, moving or rotating an object, or animation, can be performed on the objects. Rendering (image generation) is performed by OpenGL calls, thus possibly using fast graphics functions implemented in the graphics hardware. However, Open Inventor encapsulates the details, such as OpenGL function calls, inside the objects, so the developer usually need not trace geometric transformation matrices or dive into details of OpenGL like status bits, thus allowing him or her to concentrate on the application.

A great number of classes for many different general graphics applications is included with Open Inventor. However, where necessary, it is possible to extend Open Inventor with derived classes, as documented in the Inventor Toolmaker [Wer94b].

4.1.2 Concepts

Open Inventor is an object oriented toolkit. It provides a hierarchical class tree with a few base classes and many derived classes. Some of the most important classes shall be given here, details can be found in [Wer94a].

Scene Basic Types

Three dimensional computer graphics requires a few standard data types and operations not directly supported in the C++ programming language. However, by defining appropriate classes and operators, Open Inventor adds Scene Basic Types, all named `Sb...`, like *n*- (*n* = 2, 3, 4) dimensional vectors `SbVec`*n*`x` of type float (*x*=`f`), double (*x*=`d`) or short (*x*=`s`), 4×4 matrices `SbMatrix`, or "smart" `SbString` objects with numerous convenience methods.

Scene Objects Class Hierarchy

Figures 4.1, 4.2 and 4.3 present a graphical representation of the class hierarchy [Wer94a].

`SoBase` is the base class for `SoFieldContainer`, from which `SoNode`s and `SoEngine`s are derived. `SoNode`s include, e.g., camera, light, material, texture, text, transformation, coordinate, and shape nodes.

Fields are value containers for public access. Values of the appropriate type (integer, float, strings, ...) can be written into and read from the fields. It is possible to connect fields so that if a field value is set, the connected fields take the same value.

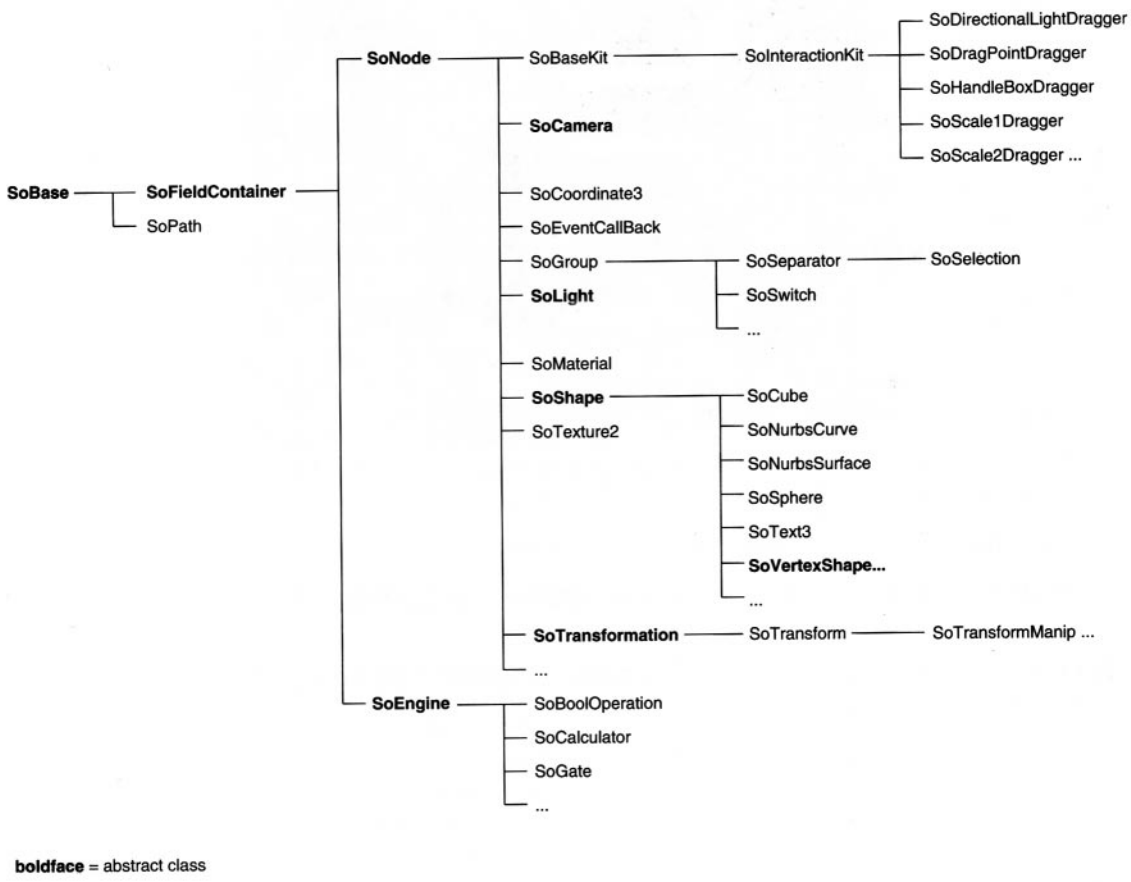


Figure 4.1: Open Inventor Class Tree Summary (1)

Node objects and the Scene Graph

Building a 3D scene in Open Inventor, as in many 3D graphic applications, means designing a scene graph. The node objects are kept in a tree-like structure, more precisely in a Directed Acyclic Graph (DAG). This means that, searching from the root node, a single node may be reached on more than one path from the root node while traversing in a left-to-right, depth-first search, but loops in a path are not allowed.

There are three basic categories for the node classes:

Shape Nodes These define an actual shape (e.g., `SoCube`, `SoSphere`, `SoTriangleStripSet`), `SoNurbsSurface`, possibly using coordinates or other values like colors or textures specified earlier in the scene graph.

Property Nodes define properties like vertex coordinates (`SoCoordinate3`), object appearance (e.g. `SoMaterial`, which defines, e.g., color or trans-

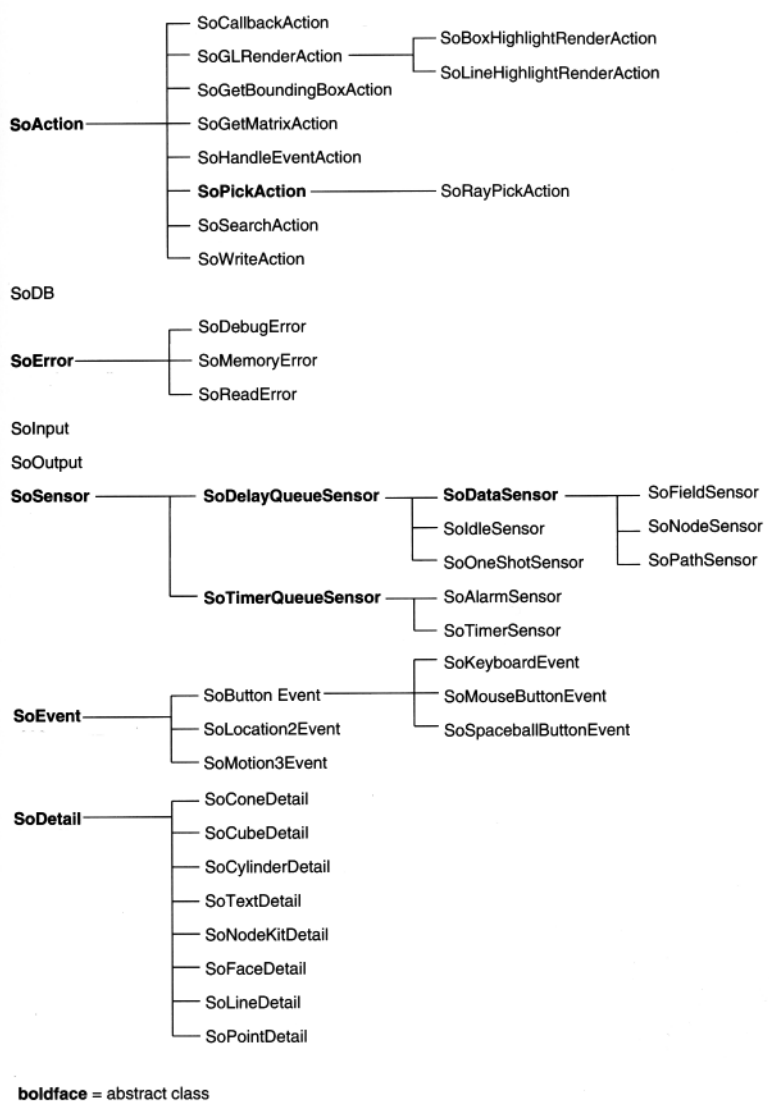


Figure 4.2: Open Inventor Class Tree Summary (2)

parency values, or `SoTexture2`, which specifies a 2-dimensional texture map), or specify a geometric operation (`SoTransformation` classes). Other nodes can influence various rendering settings, like `SoDrawStyle` or `SoComplexity`.

Group Nodes are containers that collect other nodes into subgraphs.

The scene graph is used by applying Actions on it. An `SoAction` traverses the tree in left-to-right, depth-first order. Each node has its own action behavior for the different subclasses of `SoAction`.

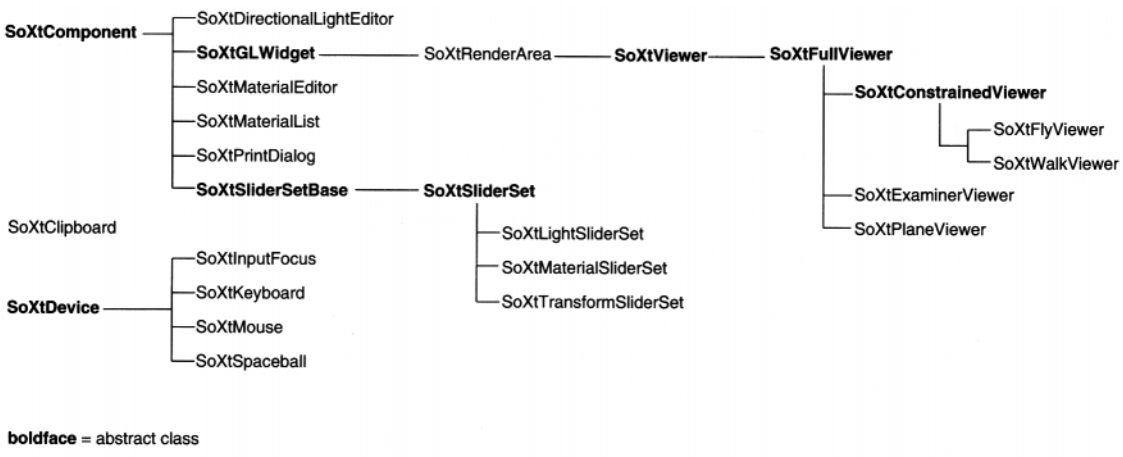


Figure 4.3: Open Inventor Class Tree Summary (3)

Nodes can be searched by type or name with a `SoSearchAction`, or a mouse click onto the object performed by the user, which is translated into a `SoPickAction`. Both return a `SoPath` to the object which specifies exactly the path from the scene graph root to the found node, even in case of Shared Instancing (reuse of a subgraph). Direct access to a node is also available via the `SoNode::getByName()` method.

For rendering (image generation), the scene graph is traversed by a `SoGLRenderAction`. Here, the traversal of property nodes may trigger a change in the current OpenGL state (like `SoMaterial`, which sets, e.g., color or transparency values), or a geometric transformation, or is just the definition of coordinate values (`SoCoordinate3`) for nodes that come after the node which is just traversed.

`SoCamera` nodes define cameras which can be used to render all objects that come after them in the scene graph. The image is then usually displayed within a `SoXtRenderArea` on the screen, or may be stored in a file if created with an `SoOffScreenRenderer`.

A `SoSeparator` (derived from `SoGroup`) encapsulates the state changes performed by its children, so that nodes to the right of the `SoSeparator` do not see the effects of, e.g., transformations or color changes done by child nodes of the `SoSeparator`. Thus, a `SoSeparator` can be used to contain all the properties for the shapes below it, and the whole sub-tree can be seen as an entity. However, if an application designer frequently uses objects of similar structure (say, all having a location in space, color, texture and shape nodes), it is recommended to use a node kit.

References and Deletion Inventor nodes are created in the usual C++ fashion (`new SoNode`). However, deletion is performed automatically to sim-

plify object management. Every node keeps a Reference Count. On object creation, the reference count is zero. Adding a node `N` as child to a group node `G` increments the reference count of `N` by one. Removing `N` from `G` decrements `N`'s reference count to zero and thus causes its deletion. (An object is deleted only if the counter *goes down to zero*, or newly created objects would immediately be deleted.) Nodes are also referenced if they are included in a `SoPath`, and dereferenced when the `SoPath` is deleted.

If `G` is deleted, all reference counts of its children are decremented, and former children with a new reference count of zero are deleted as well.

By manually referencing or unreferencing nodes, the developer can influence this scheme and prevent unwanted deletions. The scene graph's root node has to be referenced in this way.

Node Kits

To help in defining 3D-objects with all their properties without having to build every object by inserting appropriate nodes at the right places, Open Inventor provides Node Kits. These are special nodes with fields defining their children. The node layout (structure) of the object is defined in the class constructor. Not all nodes defined here need to exist, but if they are inserted, they are always inserted at the right places.

Sensors

Interactive graphics require response to events. Inventor Sensors can be used to watch for various types of events and invoke a user-supplied callback function when that event occurs.

Data Sensors respond to changes in the data of a node's fields, in a node's children, or in a path. For example, whenever the value in a field changes, all attached `SoFieldSensor`'s callback functions are scheduled for execution.

Timer Sensors respond to certain time conditions, i.e., their callback functions are scheduled when a certain time or intervall has passed.

Engines

To allow calculations and combinations of field values as well as contiuous changes and complex animation in the scene graph, Open Inventor allows to interlink node fields with Engines. There are many predefined Arithmetic Engines for creating vectors and matrices from scalar values and *vice versa*, linear interpolation between two scalars, vectors and matrices, and a general `SoCalculator` engine which can perform simple calculations by specification of formulae via character strings.

Animation Engines can drive connected field values according to some elapsed real time.

Triggered Engines use events in special input fields (touching of a dataless input field) to switch the values in their outputs.

There are also nodes with built-in engines. For example, an `SoRotor` is a combination of an `SoRotation` node with a built-in `SoElapsedTime` engine. It can be used for circular motion of, e.g., a wheel, where its `speed` field defines the rotation speed. The `rotation` field will change its angular value.

Connections Every field can be connected to an engine output of appropriate type. If a connection is tried between inappropriate types, e.g., a `SoSFString` is connected from an `SoEngineOutput` of type `SoSFFloat`, the Inventor database tries to add a Field Converter which will, in this case, set the string value to a string representation of the float value.

An engine output can be connected to any number of fields. However, a field can only be connected to a single engine output or other field. Its value can still be set by other means. Whichever way the field's value was set, the last entry "wins". The collection of engines and fields that are "wired together" in the scene graph is termed Engine Network.

Connecting a field to an engine output increases the reference count of the engine. Field-to-field connections do not change any reference counts. An unreferenced engine with no connected outputs is deleted.

Evaluation For efficiency reasons, engines do not perform their function and set the values in the fields connected to their outputs whenever the inputs change. They only send an invalidation message to the connected fields. When a field value has to be used, but the field is marked invalid, it will call the engine that feeds the field, which is then evaluated, thus setting and revalidating the field.

4.2 Studierstube

Since 1996, the research group for Virtual Reality of the Institute for Computer Graphics and Algorithms at Vienna University of Technology, in close collaboration with Fraunhofer CRCG, Inc., in Providence, Rhode Island, USA, has created an experimental environment for Augmented Reality: the STUDIERSTUBE (German for "study room", where GOETHE's famous character, FAUST, tried to acquire knowledge and enlightenment). [SFH+00b]

Augmented Reality Virtual Reality (VR) in its most immersive form replaces the real world by artificial, virtual objects and lets the user interact with them. Many users of VR systems feel uneasy when they exchange the

natural environment which surrounds them in reality for a virtual environment with objects that only appear visually, but not physically.[1] VR's less obtrusive sibling, Augmented Reality (AR), adds artificial objects to the real world. The user can see the real objects together with virtual additions.

STUDIERSTUBE is a development environment for experiments with many aspects of augmented reality, from AR for single users to collaborative, multi-user setups, experimenting with new user interfaces, interaction devices, and different output devices (HMDs, mono-/stereoscopic projection screens, hand-held displays, *etc.*). While most VR and AR projects concentrate on a single application context, STUDIERSTUBE provides a general environment in which several AR applications can run in parallel, each in its own 3D window. Data exchange between windows is possible, as is collaboration between several users. Computation load can be distributed over several workstations.

STUDIERSTUBE is based on OPEN INVENTOR and provides a toolkit comprising numerous extension classes, mainly for 3D event handling and interaction, and a runtime framework, providing the necessary environment for execution of STUDIERSTUBE applications.

The following sections describe the parts of STUDIERSTUBE which are relevant for the development of VRMOSS. More details can be found in [SFH00a, SFH+00b, SG97a, SG97b] and other STUDIERSTUBE research papers available at `http://www.studierstube.org`.

4.2.1 The Original STUDIERSTUBE

The original STUDIERSTUBE setup is a collaborative augmented reality system that allows several users, gathered in a laboratory room and equipped with head-tracked see-through head-mounted displays (HMDs), to share a virtual space that can be filled with three-dimensional data. All visitors see the data in the same spatial area, and can point out interesting areas and discuss them in a natural way while still seeing each other. By walking around, the data can be looked at from all sides (Figure 4.4).

4.2.2 Tracking

Tracking was originally implemented with a magnet field tracking system. It consists of one magnet field emitter for the whole room and one magnet field sensor per tracked unit. Each sensor can measure position and orientation relative to the emitter. This technique allows to produce graphics that can be

[1] For example, when walking around in a room, wearing a helmet with a set of built-in stereo eyeglasses (Head Mounted Display, HMD) and audio headset, which give the user the impression of being in a different location, the user's motions could be translated into motion in that virtual world. However, the virtual world will very likely show walls or other obstacles on places different from the physical room the user walks in, and the user may feel to be in constant danger of tripping over a real chair or bumping into a real wall.

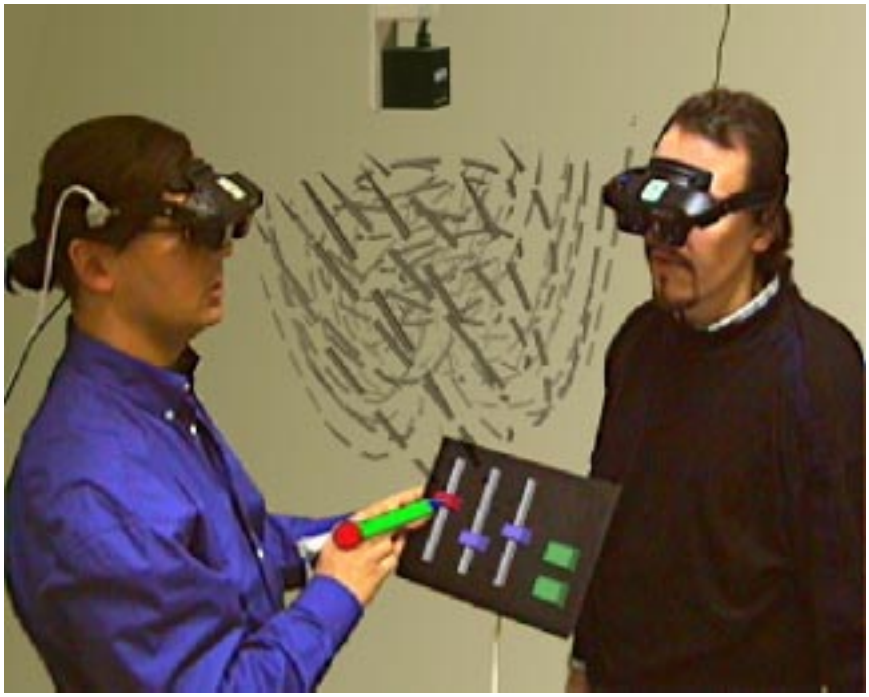


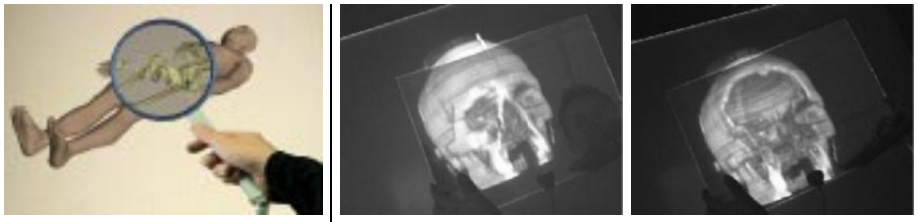
Figure 4.4: Two collaborators wearing see-through displays are examining a flow visualization data set. [SFH+00b]

presented in a user's HMD in the correct position and orientation. However, magnetic tracking faces the problem of magnet field distortion near metallic objects, and generally limited field size. Further and ongoing development includes setups with optical tracking, where the HMDs (or other tracked devices) are equipped with markers. The system uses video cameras and filters the markers' positions from the video images to deduce position and orientation.

The raw tracker data are processed by an auxiliary system component, the tracking system, usually running on a separate machine, and output in multicast packets on a TCP/IP network, where all connected stations can read and process the positional data to find camera positions and orientations, *etc.*

4.2.3 Interaction

The application is controlled with a novel two-handed interface device, the Personal Interaction Panel (PIP). It consists of a light flat panel, made from non-magnetic materials like wood, plastic or acrylic glass, and a pen with one or two buttons. Both devices are also equipped with magnet field sensors



(a) (b)

Figure 4.5: (a) X-ray lens — (b) The panel is used to position a clipping plane that cuts away a portion from the volumetric scan of a human skull. [SFH+00b]

which measure position coordinates and orientation in the same magnetic field as the head-tracked devices, or markers in optical tracking setups.

The system includes graphical representations of pen and panel in the augmented world, which are overlaid in the same position as the real objects when seen through the HMDs. While the pen usually is just represented with a similar pen, the panel is typically augmented with user interface elements known from two-dimensional graphical user interfaces, like buttons, sliders, or rotating dials, or novel 3D interaction widgets. These control widgets can be activated with the pen by either just touching them, or by touching them and pressing one of the buttons mounted on the pen. The overlaid graphics, termed sheets, can be designed to fit the needs of the application, and several switchable sheets can be used if necessary.

The haptic feedback from the physical PIP greatly helps the user when interacting with the controls in the graphical overlay. Having a panel with sliders in one hand, operating these sliders with the pen in the other hand is regarded much easier than finding a similar slider floating somewhere in 3D space.

Besides interacting with the controls on the PIP, direct manipulation of the scene is possible. The pen can be used, e.g., to point out interesting locations and also to put marks in the scene, or to pick an object and rotate or drag it to a different location.

Replacing the pen with other artifacts opens the doors for numerous applications. A brush can be used to paint scene objects, or a tool shaped as "magnifying lens" can be used to provide a different look on the scene, e.g., in a medical application, the lens may represent an X-ray viewer where the user can look into a virtual body and see the skeleton (Figure 4.5 (a)).

The panel, on the other hand, can also be used for novel ways of interaction. It can be used not only as carrier for conventional interaction widgets, but also as separate hand-held display area, e.g., for "photos" taken with a virtual camera mounted on the pen. One medical application working with

Figure 4.6: Personal displays secure privacy when playing Mahjongg: the left player cannot see his opponent's tile labels and *vice versa*. [SFH+00b]

data from computer tomography uses the panel even as slicing plane tool to grab an image of a cross section through a patient's body, and the pen can then be used to set marks on the image presented on the PIP (Figure 4.5 (b)).

Depending on the application, there may be either just one PIP set that is physically transferred between users, or each user can have one, controlling the application collaboratively. It is also possible to present the PIP sheets depending on the user's role, e.g., in a collaborative game, the opponent's PIP can be shown with details hidden to avoid cheating (Figure 4.6).

4.2.4 STUDIERSTUBE Applications

STUDIERSTUBE provides an application framework where applications can be dynamically loaded for execution. Each application can have one or several 3D windows, similar in concept to the multiple document interface on common desktop systems. This feature has not been exploited in VRMOSS, details can be found in the references mentioned above.

4.2.5 Setup Variants

Besides using head-tracked HMDs in a collaborative setup, which certainly provide the most advanced setup for STUDIERSTUBE applications, there are other setups that can be used with the OPEN INVENTOR viewers that have been developed in the STUDIERSTUBE project.

Virtual Table The 3D content can be presented on a large horizontal or tilted table with integrated video projector. Simple shutter glasses allow collaborative operation with up to two users. Two images per user (one per eye) are created and delivered to the screen in Field Sequential mode. The shutter glasses are synchronized with the frame rate on the virtual table and allow only the eye for which the image was rendered to see it. The other eye, and both eyes of the possible second user, are blocked from view. A PIP made from acrylic glass can be used in this setup, which allows even slightly different ways of interaction appropriate to the application.

Stereo Projection For a single head-tracked user or multiple non-tracked users, a passive stereo projection can be used. Two images are generated and displayed with two video projectors through polarizing filters, where the polarization axes are set 90° apart. The users wear simple cardboard or plastic goggles with polarizing filters in appropriate orientation, so that the images, which are overlaid on the screen, are filtered and each eye gets only one image.

Desktop VR If only CRT monitors are available, a "Fishtank VR" setup can be used, where active stereo can be presented with shutter glasses.

Monoscopic setups The simplest setups, mainly used during application development, consists of a monoscopic view inside one of OPEN INVENTOR's standard viewers.

Passive stereo on a single monitor? Using a stereo STUDIERSTUBE viewer in vertical split mode on a single monitor allows stereoscopic viewing by making use of a special viewing technique: The left eye's image is presented on the right half of the display, and the right eye's image on the left half. Using the crossed-eyes technique, the user may be able to visually overlay the half images. However, this technique quickly leads to eye fatigue and nausea for most users, and many people cannot align the images properly, so this method cannot be recommended.

4.3 XML

Over the last years, XML (Extensible Markup Language) has evolved as versatile and flexible standard for text document markup usable for a large number of applications. Developed 1996–1998 as "light" version of SGML (Standard Generalized Markup Language), it appears similar in structure to HTML, but is not limited to a fixed, predefined set of tags describing text formatting. Within a short period there appeared support for many popular programming languages, especially Java and C++.

Development of XML and related standards is still in progress, with many features to be developed. The following is a short description of the relevant parts of XML used in the ASH project. More details can be found in [HM01].

4.3.1 Definitions

XML is a meta-markup language, which means that the tags are not predefined, but can be defined by the application designer. This gives him or her all freedom to give the tags meaningful names which fit in the context of

the application, be it chemistry, second-hand cars, cooking recipes, music, or poetry.

An XML document that conforms to the grammar markup rules of XML is called well-formed. The grammar rules define where tags may appear, which names are allowed, how the markup has to be structured, *etc.* According to the XML standard, an XML processor must stop and report an error as soon as it finds a well-formedness error, it is not allowed to try any form of automatic correction.

An XML application is not a software application which can use XML, like a web browser, it is an XML tag set which is used in a particular domain, such as vector graphics, biographies, or HTML pages.

A Document Type Definition (DTD) can be used to declare the tags permitted in a certain XML application. XML documents that match the DTD are called valid, those that don't match are invalid. The DTD syntax is limited to describe the document structure, names of tags and attributes and the format of attribute values. Many other aspects, like range of allowed values, cannot be specified inside a DTD. The XML processor has to verify those limitations in code. On the other hand, an XML document may not need to be valid. Frequently, well-formedness is enough for being usable.

4.3.2 Document Structure

An XML document contains text, never binary data. Thus it can be read by any program that can open a text file, e.g., a text editor, and does not require special XML editors.

An XML document consists of elements with optional attributes. The designer of the XML application may select element and attribute names according to the needs of the application.

An XML document should, but does not have to, start with an XML header as very first line:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
```

This header serves two purposes: First, it declares the document to consist of XML and declares the document to be coded in one of numerous encoding systems. Second, programs capable of processing XML will look at the first five characters (`<?xml`) of the document and will be able to discern in which of the supported Unicode formats (variable-length UTF-8 or two-byte UTF-16) the document is coded, and will also find the byte order for the latter case. If no encoding is given, UTF-8, which is a strict superset of pure ASCII, is taken by default.

The `standalone` tag indicates whether validation of the document requires reading an external DTD file or can be performed with an Internal DTD subset given inside the XML document.

Next, a valid document includes a Document Type Declaration, which is a reference to a DTD to which the document should be compared. This declaration can either point to an external DTD file or can include the DTD verbatim in square brackets. For example, the document type declaration for an ASH mission looks like:

```
<!DOCTYPE mission SYSTEM "http://missionserver/ash/Mission.dtd">
```

Element and attribute names may contain any alphanumeric character (even non-English letters such as ö or ψ), underscore, hyphen and period characters. Colons are allowed, but reserved to use as namespace separators. A name may not start with a number, hyphen, or period. Names are case sensitive.

An element starts with an angle bracket (<) and its name. Then follow the attributes in `name="value"` pairs. Element and attribute names are strings starting with a letter, optionally followed by more (lowercase) alphanumerical and underscore characters. The attribute value is a string always enclosed by single (') or double (") quotes. String delimiters within the value string should be dealt with by using the according other delimiter form (by enclosing a string containing a double quote with single quotes, and *vice versa*) or using the escape sequences `&quot;` for " and `&apos;` for '. The characters < and & must be escaped by `<` and `&`, respectively. The element declaration ends with a closing angular bracket (>). Then follow the enclosed elements, which may consist of other structural elements or just character strings. The element closes with the closing statement of the form `</element>`. Thus, a complete element may look like

```
<element attribute1="value1" attribute2='The number "2"'>
  <subelement1 att1="val1">
  This text is part of subelement1.
    <subsubelement attribute="value">
    This text is part of the subsubelement.
    </subsubelement>
  More text in subelement1.
  </subelement1>
  <subelement2 attribute="value" /> <-- element is "empty" -->
</element>
```

If an element has no enclosed elements, it is called "empty" and needs just one declaration of the form

```
<emptyelement attribute1="value1" attribute2="value2" />
```

A well-formed document has one root element, which may contain other elements. The structure of an XML document allows it to be seen as one of computer science's most common data structure: a tree.

For database applications, configuration files and also for structured messages, an XML format with elements only having attributes and optionally containing other elements, but without unstructured text inside them clearly are most suitable.

4.3.3 Using XML in programs

A program using XML documents makes use of an XML parser. Instead of implementing a parser from scratch, a developer may use one of several existing implementations of XML parsers. For the most common programming languages, especially the object-oriented languages Java and C++, there are a few implementations available, which are mostly free to use. Most of them use one of two API models: The W3C Document Object Model (DOM) or the Simple API for XML (SAX).

Document Object Model (DOM)

A program using the DOM (Document Object Model) will let its parser read in a complete XML document. The parser builds a document object in memory which consists of a tree-like structure of element objects containing the same structure and information as the XML document, and which then can easily be acccessed to get information about elements and their attributes.

While very simple to use, problems can arise with very large documents which have to reside in memory. Some DOM implementations require even twice the amount of memory while the parser is running.

Simple API for XML (SAX)

The SAX (Simple API for XML) defines an event-based handling of XML documents.

Originally developed for Java, SAX consists of a number of interfaces for document handling. The interface defines callback methods which will be called by the parser whenever an element or attribute has been read. The application designer has to implement the methods defined in the callback interface according to the needs of the application. Using this model, very large documents can be handled without having to keep them in memory.

4.3.4 XML in ASH

XML is used within the ASH project in several places.

- The Missions are defined in AML, an XML application (see below).
- The ASH User Clients, implemented in MacroMedia Flash 5, read configuration files on startup which specify configuration data in XML format.

- Data exchange between the User Clients and the Mission Server (written in Java) uses messages in XML format.
- VRMoSS uses similar messages for receiving commands and sending status messages to the Mission Server Base. This would also allow direct communication between the VRMoSS and a special Flash GUI client, or some other client.

ASH Mission Language (AML)

Missions are defined in AML, the ASH Mission Language, an XML application, which is used by the Mission Object's DOM parser during mission startup. As practical example of the above short introduction to XML, the DTD specification of AML, `Mission.dtd`, shall be presented here:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Document Type Declaration (DTD) for an ASH mission.

This specifies AML, the ASH Mission Language, an XML application.
AML files are processed by the Mission load() method.

Author: GZ Georg Zotti, TU Vienna, Institute for Computer Graphics

Version 0.0 April 10, 2001 GZ Initial version.
Version 0.1 April 12, 2001 GZ added defaultResultUrl to transition,
                              changed attribute order a bit.
Version 0.2 April 19, 2001 GZ added version, description.
                              Put DefaultResultsUrl in episode, not transition.
Version 0.3 April 20, 2001 GZ added task name. Selection Phases cancelled.
Version 0.4 April 23, 2001 GZ Task needs duration. Phase duration removed.
                              Only one Task/Phase allowed
Version 0.5 Sept. 26, 2001 GZ Added time, ttime, speed attributes for phases

Use:
<!DOCTYPE mission SYSTEM "url-path-to/Mission.dtd" >
-->

<!-- mission is declared with default namespace for all its elements -->
<!ELEMENT mission (description?,episode+)>
<!ATTLIST mission xmlns   CDATA #FIXED "http://www.ashproject.org/ASH"
                  name    CDATA #REQUIRED
                  baseUrl CDATA #REQUIRED
                  version CDATA #IMPLIED
>

<!-- A description can be given about the mission.
     This description could be presented in a mission selection menu.
-->
<!ELEMENT description (#PCDATA)>

<!-- The episode defaultResultUrl should contain the sum of all task defaultResults. -->
<!ELEMENT episode (transition, phase*)>
<!ATTLIST episode name CDATA #REQUIRED
                  defaultResultsUrl CDATA #IMPLIED
>

<!-- Each episode has exactly one transition, presented for all visitors at
```

```
     the screen indicated by station. All other screens will show e.g. mission logo.
     An episode should also have phases. If not, only the transition will be available.
-->
<!ELEMENT transition EMPTY>
<!ATTLIST transition name CDATA #REQUIRED
                     station ( bigscreen | islandscreen | localscreen ) #REQUIRED
                     url   CDATA #REQUIRED
>

<!-- phase. Duration has to be calculated as max(Task.duration+Task.delay)
     A phase is only a container for all tasks to be presented at the stations.
     time shall be a date string "YYYY:MM:DD:HH:MM:SS" representing UT of phase start
     ttime is a similar string representing the date of the nearest important event,
           say launch, thruster firings maneuvers, etc.,
           to allow the famous "T-minus" notation.
     speed indicates time lapse. 1=realtime, 86400=1day/second, etc.
-->
<!ELEMENT phase (task+)>
<!ATTLIST phase name     CDATA #REQUIRED
                time     CDATA #IMPLIED
                ttime    CDATA #IMPLIED
                speed    CDATA #IMPLIED>

<!-- task: duration: seconds. Duration of the media content.
           station: indicates where media output goes:
                    bigscreen, islandscreen[12], island[12]wp[123]
           url: points to the medium (Flash file, MPG movie, ...)
           defaultResultsUrl: URL that points to an XML file with default results.
                              may be used in case of automatic evaluation or
                              if student cowardly escapes from task
                              Meaningful obviously only for workplace stations.
           delay: time delay from phase begin.
-->
<!ELEMENT task EMPTY>
<!ATTLIST task name     CDATA #IMPLIED
               duration CDATA #REQUIRED
               station  ( bigscreen | islandscreen1 | islandscreen2 |
                          island1wp1 | island1wp2 | island1wp3 |
                          island2wp1 | island2wp2 | island2wp3 )       #REQUIRED
               url               CDATA #REQUIRED
               defaultResultsUrl CDATA #IMPLIED
               delay             CDATA #IMPLIED
>
```

Mission Files

`ROOTSMission.aml` shows the layout of the mission that is developed for the P-VCR. It references `Mission.dtd` in the `<!DOCTYPE` statement, so that a verifying XML parser can check if the structure of the AML file conforms to the AML rules:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>

<!-- This is the prototype layout for the ROOTS mission

Authors:
  GZ Georg Zotti, TU Vienna, Institute for Computer Graphics

History:
```

```
  V0.0 April 11, 2001 GZ  Based on the January storyboard draft.
  V0.1 April 12, 2001 GZ  adapted to Mission.dtd V0.1
  V0.2 April 19, 2001 GZ  adapted to Mission.dtd V0.2. Added description.
  V0.3 April 23, 2001 GZ  adapted to Mission.dtd V0.4. Added task duration.
  V0.4 April 25, 2001 GZ  completed conforming to spec 4.7.0,
                          added baseUrl, filled in dummy URLs
  V0.5 Sept. 03, 2001 GZ  Phase needs date and speed to synchronize solar system

Make sure to configure the first 2 lines correctly!
-->

<!-- !DOCTYPE mission SYSTEM "http://missionserver/ASH/Missions/Mission.dtd" -->
<!DOCTYPE mission SYSTEM "Mission.dtd" >

<mission name="ROOTS" baseUrl="http://missionserver/ASH/Missions/ROOTS/">
<description>
ROsetta Observing The Solar System.

Prototype mission for the ASH Project.

This mission is related to the ESA Rosetta mission. Driven by the
search for pristine material to study for traces of life in space, the
mission includes selecting observation targets, preparing and
launching a spacecraft, controlling its trajectory, deploy landers on
Mars and an asteroid during fly-by maneuvers, and rendez-vous with a
comet plus a final landing there.
</description>

<episode name="Episode 1 -- Introduction: Formation of the Solar System"
         defaultResultsUrl="ep01/dResults.xml" >
 <transition name="Transition1" station="bigscreen" url="ep01/transition.html" />
 <phase name="FindPristineMaterial" >
  <task duration="120" station="bigscreen" url="ep01/Intro01.mpg" />
  <task duration="150" station="islandscreen1" url="ep01/Intro-i1.swf" delay="120"/>
  <task duration="150" station="islandscreen2" url="ep01/Intro-i2.swf" delay="120"/>
  <task duration="240" station="island1wp1" url="ep01/taskPristineMaterial.swf"
        defaultResultsUrl="ep01/taskPristineMaterial-dResults.xml" delay="120" />
  <task duration="240" station="island1wp2" url="ep01/taskPristineMaterial.swf"
        defaultResultsUrl="ep01/taskPristineMaterial-dResults.xml" delay="120" />
  <task duration="240" station="island1wp3" url="ep01/taskPristineMaterial.swf"
        defaultResultsUrl="ep01/taskPristineMaterial-dResults.xml" delay="120" />
  <task duration="240" station="island2wp1" url="ep01/taskPristineMaterial.swf"
        defaultResultsUrl="ep01/taskPristineMaterial-dResults.xml" delay="120" />
  <task duration="240" station="island2wp2" url="ep01/taskPristineMaterial.swf"
        defaultResultsUrl="ep01/taskPristineMaterial-dResults.xml" delay="120" />
  <task duration="240" station="island2wp3" url="ep01/taskPristineMaterial.swf"
        defaultResultsUrl="ep01/taskPristineMaterial-dResults.xml" delay="120" />
 </phase>
</episode>

<episode name="Episode 2 -- Selection of target"
         defaultResultsUrl="ep02/dResults.xml">
<!-- Only a transition will be implemented for the prototype ASH VCR -->
 <transition name="Ground Based Observation of Comets" station="bigscreen"
             url="ep02/transition.html"/>
</episode>

<!-- OTHER EPISODES OMITTED -->

<episode name="Episode 9 -- The Comet" defaultResultsUrl="ep09/dResults.xml">
 <transition name="The Comet" station="bigscreen" url="ep09/transition.html"/>
 <phase name="Orbit Insertion"
```

```
       gdate="2011_10_12 12:00:00" speed="1"> <!-- real time speed! -->
  <task duration="300" station="bigscreen"  url="ep09/orbitIntro.mpg" />
  <task duration="300" station="islandscreen1" url="ep09/orbIntroIs.mpg" delay="200"/>
  <task duration="300" station="islandscreen2" url="ep09/orbIntroIs.mpg" delay="200"/>
  <task duration="240" station="island1wp1" url="ep09/orbit.swf"
       defaultResultsUrl="ep09/orbit-dResult.xml" delay="500" />
  <task duration="240" station="island1wp2" url="ep09/orbit.swf"
       defaultResultsUrl="ep09/orbit-dResult.xml" delay="500" />
  <task duration="240" station="island1wp3" url="ep09/orbit.swf"
       defaultResultsUrl="ep09/orbit-dResult.xml" delay="500" />
  <task duration="240" station="island2wp1" url="ep09/orbit.swf"
       defaultResultsUrl="ep09/orbit-dResult.xml" delay="500" />
  <task duration="240" station="island2wp2" url="ep09/orbit.swf"
       defaultResultsUrl="ep09/orbit-dResult.xml" delay="500" />
  <task duration="240" station="island2wp3" url="ep09/orbit.swf"
       defaultResultsUrl="ep09/orbit-dResult.xml" delay="500" />
  <task duration="45" station="islandscreen1" url="ep09/orbResult.swf" delay="740"/>
  <task duration="45" station="islandscreen2" url="ep09/orbResult.swf" delay="740"/>
 </phase>
 <phase name="Land Probe" >
  <task duration="240" station="bigscreen" url="ep09/lnd3IntroBig.mpg" />
  <task duration="240" station="islandscreen1" url="ep09/lnd3IntroI.mpg" delay="150"/>
  <task duration="240" station="islandscreen2" url="ep09/lnd3IntroI.mpg" delay="150"/>
<!-- OTHER TASKS OMITTED -->
 </phase>

<!-- OTHER PHASES OMITTED -->

</episode>

<episode name="Episode 10 -- Summary and Discussion">
<!-- This episode will not be implemented for the prototype ASH VCR -->
 <transition name="What we have learned" station="bigscreen"
             url="ep10/transition.swf"/>
</episode>

</mission>
```

4.4 Advanced Rendering Toolkit (ART)

Software libraries from the Advanced Rendering Toolkit (ART) [TW+01], developed by the research group for photorealistic rendering at the Institute of Computer Graphics and Algorithms, were used in early experiments to create a natural looking star map which could be used as texture map on a sphere. Later, the stellar background was changed to an Open Inventor `SoPointSet`. Commands from the ART libraries were again used to calculate the color values for the stars.

Chapter 5

Implementation

5.1 The Solar System Model inside the ASH VCR

A virtual model of the Solar System has been chosen by the ASH consortium for several reasons:

- Presenting a model of the Solar System immediately gives a good impression over the structure, parts and relative positions of the objects inside: the Sun, nine planets, their moons, the asteroid belt, and selected asteroids and comets. Therefore, it can serve as a common introduction episode for every mission scenario, where the students will be presented basic facts of general interest.

- Most space science missions are satellite observatories orbiting Earth. Every space exploration mission leaving Earth orbit currently (and, it should be safe to say, for long times to follow) is reasonably confined to a target lying within the Solar System[1]. So, to understand what scientists are doing in space exploration, we have to see where we go.

- A model of the Solar System can be used to visualize the spacecraft's flight path through the planets' realm, and can help to explain why spacecrafts often take many years to reach their goals, and why there are "launch windows", i.e., sometimes rather short time spans where a space mission must be launched to be able to fly by intermediate targets (planets, asteroids) to gain speed, change the spacecraft's direction towards the final targets, and increase scientific output by observation of that intermediate target.

[1] The missions Pioneer 10, Pioneer 11, Voyager 1 and Voyager 2, all have left the area of the Solar System where the large planets reside. However, their primary targets were the outer planets of our Solar System, and they will have long stopped operation when they reach neighboring stars.

5.1.1 Requirements for ASH

VRMoSS will represent the central Bigscreen application for the ASH Virtual Control Room, shown on a large (about $3 \times 2m$) screen in a back projection setup.

It should give a vivid impression of the Solar System, allowing to get an impression of its structure (planet orbits, asteroid belt) as well as getting closeup views of single objects. For the ROOTS mission, the flight of the Rosetta spacecraft had to be included, and a realistic comet model had to be presented.

The model should support intuitive exploration and looking for task solutions as well as passive presentations. Therefore, it is used in two operation modes:

Exploration Mode Interactive control of the presented scene with the Personal Interaction Panel (PIP)

Demonstration Mode Automated demonstrations by commands sent via the Mission Server

5.1.2 Studierstube for VRMoSS

Given the extensive set of components, Open Inventor and the Studierstube API were the natural choice for the development of VRMoSS.

VRMoSS Setup

The ASH VCR is an environment for dozens of temporary visitors. For hygienic reasons, it was early decided not to use head-tracked HMDs, but only a stereo projection setup with simple discardable cardboard goggles. This also has the advantage that all visitors can share the stereo impression at the same time.

Mainly for cost reasons, at least the P-VCR uses only a single PIP set. This of course might pose a problem when all students of a school class want to take over control. The storyboard should come up with a solution for this, maybe by rewarding students with some time at the PIP. It has yet to be decided whether a C-VCR could possibly provide a PIP per island.

The program runs on the ASH VCR Bigscreen client, a high-end dual-processor PC running the Microsoft® Windows 2000 operating system. An NVidia GeForce 2MX Twin View graphics board drives two bright LCD projectors in a passive (polarization) stereo back-projection setup. The screen consists of a special material which does not destroy the polarization.

The ASH PIP

The ASH PIP was cut from orange acrylic glass in a form to fit the left half of the letter A in the ASH logo. The screen representation reflects the outline of the form with a thin line in the ASH design's orange, and is transparent elsewhere.

The pen is similar to the default STUDIERSTUBE pen, but colored to fit the ASH design. The physical pen comes with the POLHEMUS 3SPACE ISOTRAK II magnet tracker system used in ASH and has one button.

5.1.3 XML for VRMoSS

Because XML is used in other parts of ASH (see section 4.3.4), the communication messages between VRMoSS and the ASH Mission Server should be formatted in XML.

VRMoSS makes use of the XERCES-C++ libraries, version 1.5.0, provided by the APACHE project (`http://xml.apache.org`). The messages, which consist of short strings with single empty XML elements, are read from a network socket into a character buffer, fed to a DOM parser, and the returned DOM document object is analyzed to access the message contents.

5.2 Look and Feel

The ASH VCR setup with passive stereo on the Bigscreen is obviously less immersive than a STUDIERSTUBE setup with HMDs. All visitors of the VCR see the same image, and best stereo impression will be along the symmetry axis of the Bigscreen. The screen shows a full screen image without a frame or menu (Figure 5.1).

The PIP user is positioned in front of the screen, near the optical axis. The scene on the screen represents a window into the Solar System. The stereo setup is adjusted so that the object which is presented in focus seems to hover at about eye level height in front of the screen. The background, like stars, constellations, the Milky Way, appears behind the plane of the screen.

The point of interest lies about halfway between the user and the screen. Interesting objects of the Solar System are brought into this point by selecting the respective object with a (virtual) button mounted on the PIP. The object will stay in the point of interest even if it is moving in the scene, e.g., a planet moving around the Sun. The whole scene will be shifted accordingly.

The pen can be used to rotate and slightly shift the scene. Pointing the pen into the scene (keeping it off the PIP) and pressing the pen button locks the scene to the pen. When the user turns and moves the pen, the scene turns with it.

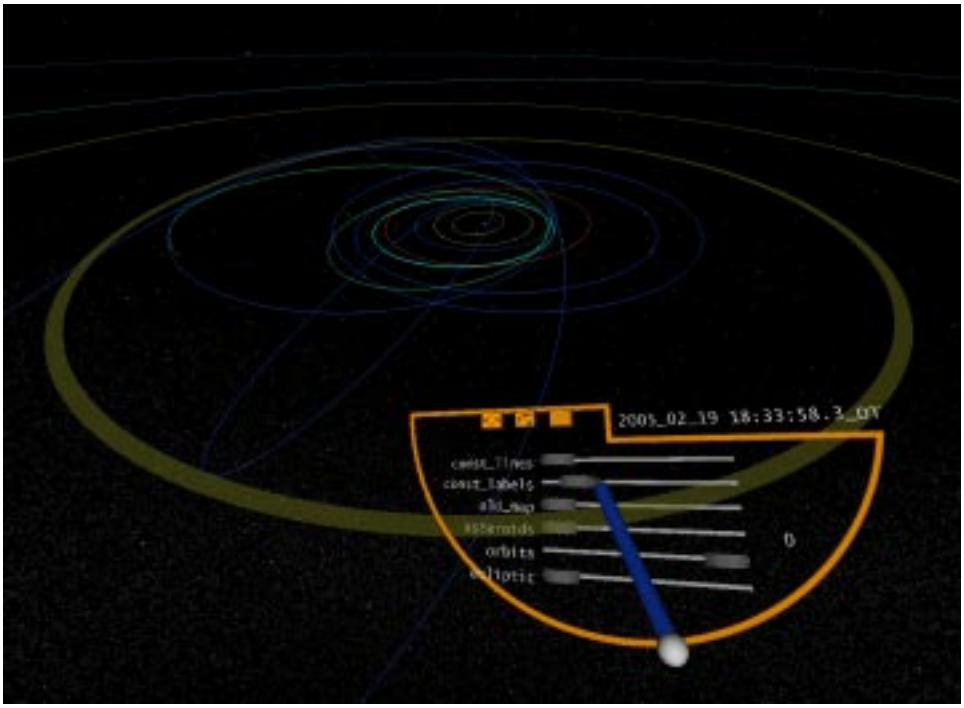


Figure 5.1: The Solar System. This overview shows orbits of the planets, several asteroids and comets (blue), and the path of the Rosetta spacecraft (green)

The calendar date and time (*DT*, see section 3.1.2) is shown on the PIP. The PIP is the only place where artificial objects like sliders or buttons, or text not directly used as object labels, may appear. The user interface presented on the PIP can be used to switch between objects to display, and to influence other aspects of the presentation. The PIP allows selection of several sheets with different controls. The first PIP sheet contains the main controls:

- The Sun, planets, comet and spacecraft are presented in a row of buttons of the respective shapes. Larger planets are represented by slightly larger buttons.

  The user can use the pen to point into a planet button, which is lighting up. Pressing the pen button will bring the selected object into the point of interest.

- The speed slider is used to influence simulation speed. It allows to set simulation speeds between -1000 and $+1000$ calendar days per animation second. The settings are translated in a logarithmic scale,

therefore slow speeds can be controlled with very fine resolution: if needed, speed can be set to less than one calendar second per animation second.

This allows presentation of fast processes like rotation of the planets and very slow processes like the orbit motion of the outer planets, which take up to many decades for a single tour around the Sun.

- The scale slider provides global scaling of the whole Solar System model. It allows to show the Solar System in overview as well as zooming into a small object like a comet.
- The planet scale slider allows to enlarge the planets when the Solar System is shown in overview. The planets are so small in relation to the distances between them that this feature seemed to be necessary to see more than single pixels of the planet in overview.

The second PIP sheet shows additional setting options:

- Constellation lines (stick figures) can be shown to help identifying the constellations.
- The constellations can be labelled in several styles: The IAU 3-letter abbreviations, full names in Latin, or full names in English.
- A painted celestial map from the 17th century, with the classical mythological characters of the constellations, can be shown.
- Up to 10000 asteroids (small planets, planetoids) can be added. These are represented by simple points at correct positions. The asteroid belt appears as toroidal region filled with objects, and the Troian groups, asteroids gravitationally locked in Jupiter's orbit, can be seen. With greater simulation speeds, differences in orbital speed become obvious.

  The asteroids usually should be shown only temporarily, because simulation of many asteroids can severely affect performance.
- The transparency of the planets' orbits can be controlled from invisible to opaque.
- The ecliptic can be presented as red dotted line within the stellar background.

Other PIP sheets may be added depending on the mission scenario.

5.3 The Software Components of VRMoSS

The VRMoSS is the only interactive stereo component in the ASH VCR, thus it was not necessary to exploit the capabilities of dynamical loading of several applications. VRMoSS has been developed as standalone application making use of some elements of the Studierstube API, namely a `UserKit`, providing pen, PIP and display definitions for a single user, and a customized `DragKit` which contains the whole scene content.

VRMoSS has two main scene parts: the Solar System and a star sphere around it. All objects are represented as specialized node kits, which allows to think in higher level building blocks and simplifies object instantiation. Most node kits in VRMoSS have matching private engines which, once connected to a application-central timekeeping node (a field in the `MainController`), automatically put their moving objects into the right places.

Following is a detailed description of the components of VRMoSS. Before explaining the implementation of the visible parts of the application, some basic items for timekeeping and application control have to be described.

5.3.1 Classes for Double Precision Computations

Open Inventor has no native support for double precision computations. For astronomical computations, especially for time keeping with the simple *JD* count (see section 3.1.3), it was necessary to create a Field which can contain double values.

The Inventor Toolmaker [Wer94b] delivers an example implementation for such a field, `SFDouble`, which was adequate for VRMoSS.

To allow automatic conversion betweens field values, e.g., to connect a numeric field to a text output field, Field Converters had to be implemented. `SFDoubleConverter` connects a field or engine output of class `SFDouble` to an `SoSFFloat` or an `SoSFString` field, while `SoSFFloatSFDoubleConverter` can feed an `SFDouble` field from an `SoSFFloat` field or engine output.

5.3.2 The Main Controller

The `MainController` is the central control node of VRMoSS. It is the first node of the scene graph, supplies the moving scene objects with a central *DT* time value (see sections 3.1.2 and 5.3.3) via its `jde` field and provides numerous other fields which, by connection to other parts of the application, affect many issues of presentation. The control fields are connected to the interface widgets on the PIP, with a possible conversion engine in between. The scene is influenced by direct field connections, forwarding the settings performed on the PIP, by commands called by a PIP button callback (or

keypress callback, when keyboard interaction is used), or by commands called by the XML processor (see section 5.3.10).

5.3.3 Keeping Time

As is common in astronomical applications, time is internally handled by the `MainController` by using the Julian Day Number JD (see section 3.1.3).

The time in VRMoSS is provided by the `MainController` by adding the time which has elapsed since the last time update, multiplied by the value in the `jdeSpeed` field, to the value in the `jde` field. This field is of type `SFDouble` to allow sub-second time resolution. This process is performed as sensor callback on the elapsed time field, so it is called whenever the Open Inventor scheduler processes callbacks pending in the idle queue.

To show the current instant of time on the PIP in common date format with year, month, day, hour, minute and second, a custom engine, `DecomposeJDEngine`, was created. It has an input, `jdIn`, and separate outputs `year`, `month`, `day`, `hour`, `minute` (all of type `SoSFShort`), `second` (`SoSFFloat`), and `dateString` (`SoSFString`), which is the output for a printed date representation which fits the ASH design. Two more inputs can be used to fine-tune the string: `monthStrings` (`SoSFBool`) can be used to decide whether the months should be printed as numbers (`FALSE`) or 3-letter abbreviations (`TRUE`). The input `timeLabel` (`SoSFString`) contains the time system in use, which will usually be `DT`. If UT is provided (see section 3.1.2) and/or time zone corrections are made before feeding the JD into `jdIn`, `timeLabel` should be set to `UT` or the respective zone designation. Default is `DT`.

5.3.4 Switching Visibility

Many slider controls on the PIP are used to show various parts of the scene, like orbits or constellation lines, in different "brightnesses", which is actually implemented by changing the part's material transparency. When such a part's is set fully transparent, its data would still be part of the scene graph. To increase performance (thus improve the frame rate), a `FloatSwitcherEngine` has been created.

Its `inverseOutput` is connected to the `transparency` field in a `SoMaterial` node below a `SoSwitch` containing a graphically demanding part of the scene. When the engine's `input` value falls below its `threshold` value, the `switchOutput` changes so that a `SoSwitch` hides its single child or switches between two children.

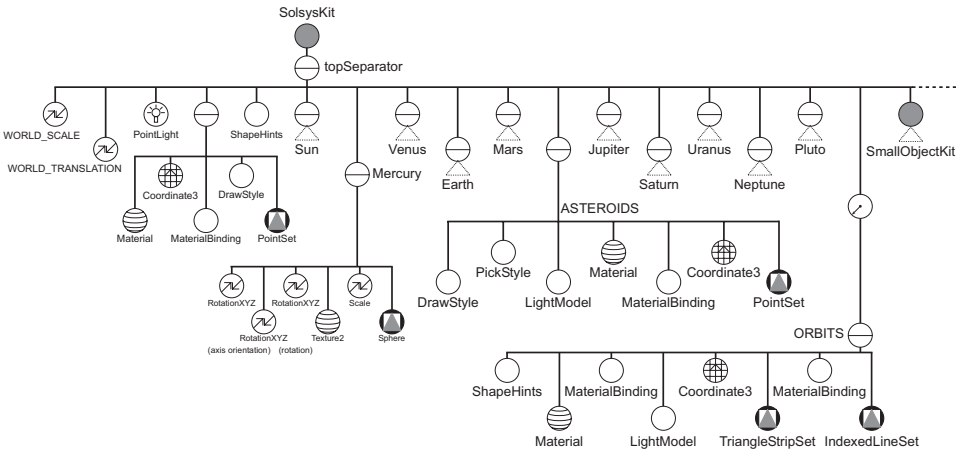


Figure 5.2: Structure of `SolsysKit`

5.3.5 The Solar System

Except for the starry background, all of the objects shown in VRMoSS belong to our Solar System. The implementation conceptually discerns the objects depending on the way their positions are determined.

The scene objects are contained in custom subclasses of `SoNodeKit` which makes instantiation, especially of comets and asteroids, rather easy. Each of these node kits makes use of a central engine for calculation of the object's position depending on the JDE value provided by the `MainController`.

`SolsysKit`

The `SolsysKit` is a node kit with rather open structure. It is the container for all of the objects of the Solar System. Its central `SolsysEngine` (see page 70) provides positions and rotational data for all planets, the Moon, Jupiter's satellites and the Sun. Another engine, `AsteroidsEngine` (see page 70), creates the positions of the objects in the Asteroid Belt.

It has just one separator (see Figure 5.2) which contains:

World Scale `SoScale` to scale the Solar System with all objects

World Translation `SoTranslation` to move object of interest into center

Sun Light `SoPointLight` at Sun's position, so that the planets are correctly illuminated

Planet Points In the overview presentation of the Solar System, the planets would be so small that they would completely vanish from the screen. To keep the planets visible at least as dot, an `SoPointSet` is

used, with according `SoMaterial` and `SoCoordinate3` nodes setting colors and positions.

Planet Models Each planet description is enclosed in a `SoSeparator` containing all elements for the planet and possible moon or ring system descriptions.

Asteroid Belt An `SoSeparator` containing `SoCoordinate3`, `SoMaterial` and `SoPointSet` nodes is used to represent up to thousands of asteroids. The positions come from the `AsteroidsEngine` described below.

Planet Orbits Nine `SoTriangleStripSet`s represent the orbits of the planets. The vertices were found by letting each planet orbit the Sun once with step width of (orbitalPeriod/360°) and writing positions with slightly larger and slightly smaller distances to file.

These flat structures tend to vanish from view if seen from the side or in too small scale (overview presentations). To keep them visible, the inner edge has been enforced by an `SoIndexedLineSet`, reusing the same vertex data.

Small Objects Single asteroids and comets with orbits and bodies of irregular shape can be added as well as spacecrafts.

All these objects (described below) are added into the `SolsysKit` to affect them with scaling, transformation and sunlight.

The Planets Each planet is modeled as textured rotating `SoSphere`, with its axis tilted according to [Sei92]. Jupiter and Saturn are also shown flattened (ellipsoidal) by their fast rotation. Earth's moon is presented in the right orientation at the correct position as described in [Mee98], Jupiter's moons are shown to rotate with bound rotation. (They rotate once around their axis while orbiting Jupiter.)

Textures NASA's Jet Propulsion Laboratory hosts a website (`http://maps.jpl.nasa.gov/`) where authors of computer models of the Solar System can find texture maps for all planets and many moons. Most of these textures are based on results from NASA space missions, like the MARINER missions of the early 1970s to Venus and Mercury, the VIKING missions to Mars in the mid-1970s, or the PIONEER 10/11 and VOYAGER missions to the outer planets. Only Pluto has not yet been visited by a spacecraft. The texture provided for Pluto is an artist's concept, based on images of Neptune's satellites. Mercury has been mapped only on one hemisphere. The texture repeats the surface of the known hemisphere onto the unmapped one.

The texture for Saturn's ring has been created by the author with ADOBE PHOTOSHOP, based on a ring map in [PC88].

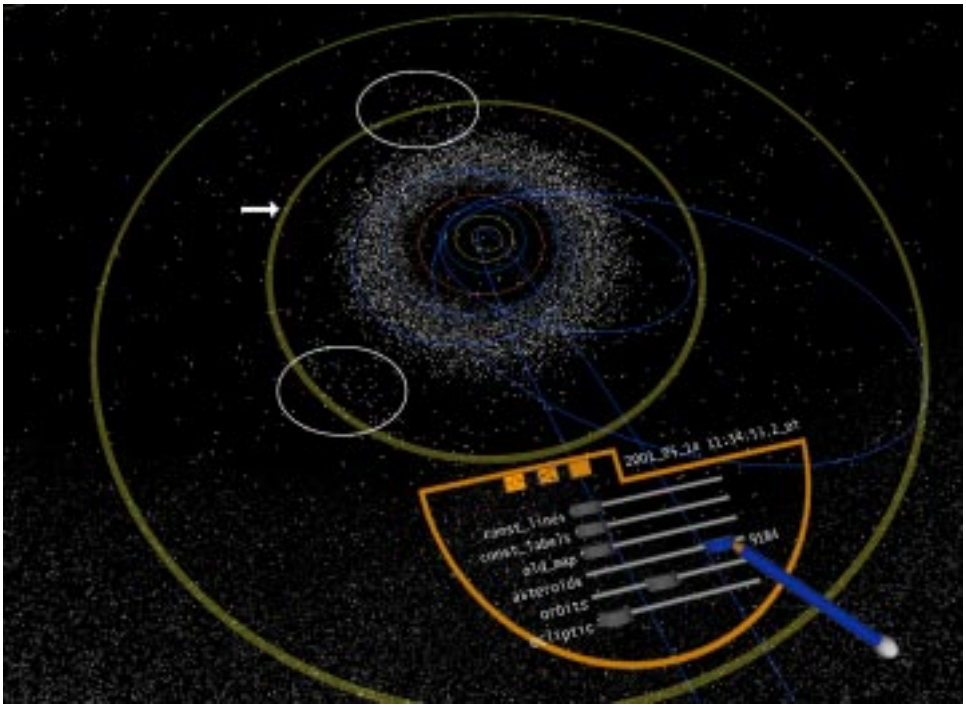


Figure 5.3: Presenting the Asteroid Belt with VRMoSS. Also clearly visible are the Troians, 2 groups of asteroids (circled) which are gravitationally bound to Jupiter and precede or follow the giant planet (arrowed) near special points of its orbit (Lagrange points).

`SolsysEngine` The planets' positions are generated using this custom engine, which has a time input, `jde` (`SFDouble`, Julian Day number, connected from the `MainController`), and several position and rotation outputs that are connected to planet transformation and rotation nodes in the `SolsysKit`.

`SolsysEngine` is essentially a wrapper class to the respective calls of the appropriate functions from the C source disk to Astronomical Algorithms [Mee91]. The spherical positions, accurate to within a few arcseconds, are then converted to rectangular heliocentric ecliptical coordinates.

Of course, the algorithm for calculation of the planets' positions has its temporal restrictions. The model should not be switched to times outside a few thousands of years around $J2000.0$, or the planets will be shown on wrong positions.

The Asteroid Belt To visualize the asteroid belt (Figure 5.3), thousands of minor planets orbiting the sun mainly between Mars and Jupiter, the custom `AsteroidEngine` reads 20.000 sets of orbital elements from a file in its `initClass()` method and, in its `evaluate()` method, computes as many of

the asteroids' positions as needed. The engine has inputs `jde` (`SFDouble`, Julian Day number) and `number` (`SoSFInt32`, conrolling the number of asteroids for which positions will be calculated) and outputs `position` and `color` (both `SoMFVec3f`). To save execution time, the accuracy of the solution of Kepler's equation here is only about one degree, which is still perfectly adequate for the purpose of showing the motions of thousands of (unlabeled) objects.

The orbital elements for the asteroids have been taken from `http://www.naic.edu/~nolan/astorb.html`. These elements are actually osculating elements, thus strictly only usable for a few weeks around the epoch. Position is computed as described in section 3.1.6.

The asteroids are represented as an `SoPointSet`, with one asteroid per point. The color values are calculated as shades of gray depending on magnitude parameter H as found in the elements.

The Small Objects

As mentioned in section 3.1.6, we can model the motion of a small body in the Solar System around the sun as undisturbed Kepler orbit, as long as the respective object remains far away from large bodies that can influence its motion by gravitation. The motion can be calculated from six orbital parameters, which, strictly, are only valid at the specified epoch, but can be used for a short period (weeks) around this time. For comets, these orbital elements, as defined in section 3.1.6, are q, e, i, ω, Ω and T. Traditionally, for asteroids, having elliptical orbits, these elements are a, e, i, ω, Ω, and M, from which, together with the epoch, perihel date T can be deduced, while q can be calculated from a and e.

On the other hand, from a single set of position and speed data (the state vector), together with time of measurement, orbital elements q, e, i, ω, Ω and T can be deduced following procedures described in section 3.1.7. Such data are available for a number of spacecrafts from ESA and NASA.

We see that asteroids, comets and spacecrafts share the property of being describable with an undisturbed Kepler orbit, so a common abstract base class can be used which defines all common parts. Each derived node kit class has an accompanying engine class driving the respective content.

`BaseSmallObjectKit` A small object in VRMoSS is represented with at least two parts: its orbit and the body itself. An abstract node kit class, `BaseSmallObjectKit`, defines all parts needed for all small objects (See Figure 5.4).

An abstract `BaseSmallObjectEngine` defines the necessary inputs and outputs to use for the kit classes derived from `BaseSmallObjectKit`. The engine, heart of the node kit, calculates the object's position, orbit lineset coordinates, and orbit rotation values.

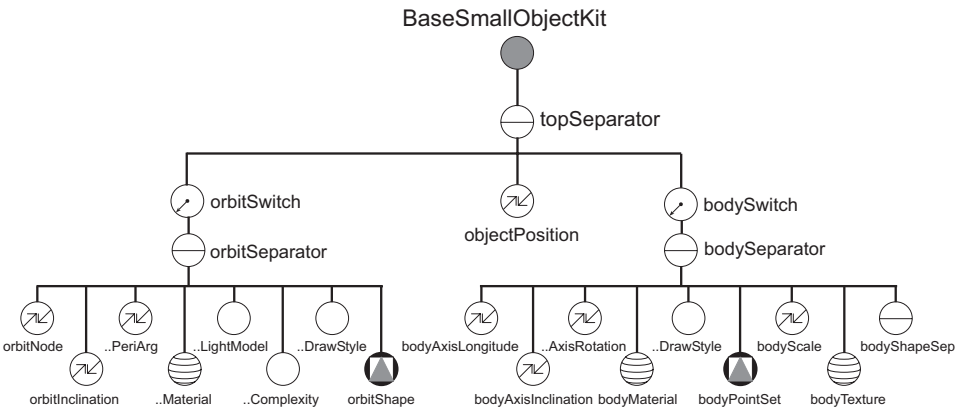


Figure 5.4: Structure of `BaseSmallObjectKit`

The object's orbit is rendered as `SoLineSet` The orbit's visibility is controlled with the kit's `orbitVisibility` field, with a `FloatSwitcherEngine` (see section 5.3.4) translating the field value into the orbit's transparency. If switched to fully transparent, the orbit part is switched off altogether.

The object's body is represented by a `SoPointSet` with one point plus the contents of the `bodyShapeSep` part of the kit.

`SmallObjectKit` This does not define new parts, but includes a real `SmallObjectEngine`, which provides object position, speed and orbit lineset vertices. This node kit can be used to represent an asteroid in detail.

In the last years, several asteroids have been visited by spacecrafts or observed from Earth with radar. From these data, Scott Hudson of Washington State University has created 3D models of some asteroids in POVRay format, which are freely available at `http://www.eecs.wsu.edu/~hudson/Research/Asteroids/models.html`. The data were converted with Maya (Alias Wavefront) to be usable with Open Inventor. Shape information for the asteroid can be inserted into the `bodyShapeSep` part of the `SmallObjectKit`.

`CometKit` In addition to the parts of a normal `SmallObjectKit`, orbit and body, a `CometKit` has the parts that make comets so special (see Fig. 5.5).

1. A tail part with a gas tail that points always strictly away from the sun and a dust part that also points away from the sun, but lags behind depending on the comet's speed.

2. The body has "hot spots" where gas evaporates when struck by sunlight. This emission is modeled with a particle system.

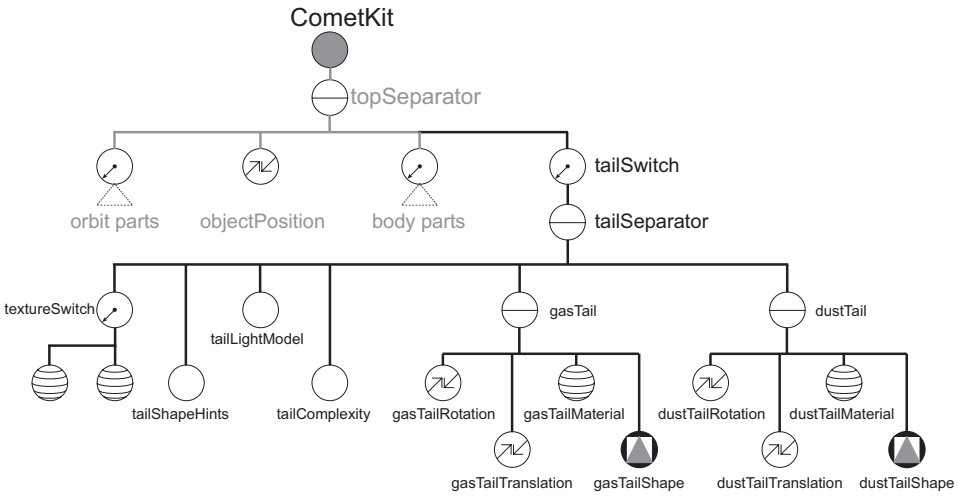


Figure 5.5: The `CometKit` class adds a tail part to the `BaseSmallObjectKit` and expands the body part with representations of outgassing activity (not shown).

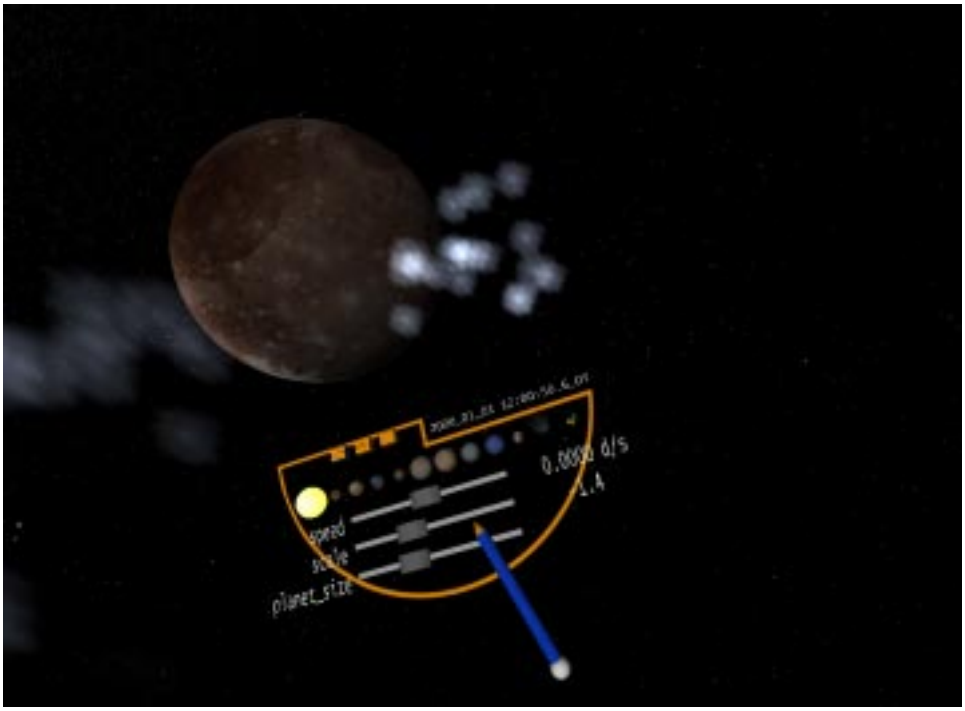


Figure 5.6: The core of a Comet. Note the outgassing activity, implemented as particle system of screen-aligned textured rectangles.

These special parts were included into VRMoSS from work done mainly by the author's colleagues, Zsolt Marx and Gottfried Eibner, at the Institute of Computer Graphics and Algorithms. Each tail is represented as textured paraboloid, where length and size are computed by the `CometEngine` (another subclass of `SmallObjectEngine`) depending on the comet's distance from Sun and absolute magnitude H_{10}, coming with the comet orbit elements. Tail rotation, an animated texture for the tail and the animation of the particles representing the gas and dust emitted from the comet core (see Figure 5.6) are driven by sensor callbacks. The animated tail texture is shared by all instances of `CometKit`.

Orbital elements for comets are available at `http://ssd.jpl.nasa.gov/data/ELEMENTS.COMET` (as of August 8th, 2001).

`SpacecraftKit` The abstract `SpacecraftKit` class adds a spacecraft's precomputed track to the parts defined in `BaseSmallObjectKit` (see Fig. 5.7). The vertex coordinates for the track line set have to be provided by the respective subclass of `SpacecraftEngine`. A `SpacecraftEngine` will read in its `initClass()` method a data file with date, position and speed values and can then interpolate the spacecraft's position. Additionally, at every instant, the instantaneous Kepler orbit is calculated to allow the visualization via the `SpacecraftKit`'s `orbit` parts.

Before launch date, the spacecraft's position should be connected to Earth's, and probably all parts of the kit should be switched to invisibility.

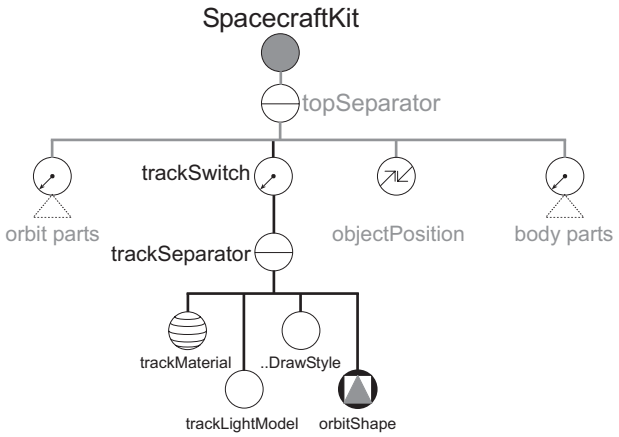


Figure 5.7: Structure of `SpacecraftKit`

`RosettaKit` The `RosettaKit` node kit, subclass of `SpacecraftKit`, uses the `RosettaEngine` to show the position and orientation of the ESA Rosetta

spacecraft. The spacecraft model, which is the `bodyShapeSeparator` part of the kit, shows the shape of the spacecraft modeled as cube with antenna and with solar panels. The body is turned so that the antenna which is mounted to the body always points towards Earth, and the solar panels point towards the sun as directly as possible, considering the mentioned main orientation.

Before launch date, the spacecraft's position is connected to Earth's position. After launch, the path and spacecraft body parts are switched visible, and the probe position is found by interpolation between 4 values with LAGRANGE's method: Having 4 times t_0, t_1, t_2, t_3 and corresponding object positions $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, the position $\mathbf{x}_t$, corresponding to a time t, where $t_1 < t < t_2$, can be found as

$$\begin{aligned} \mathbf{x}_t &= \mathbf{x}_0 \frac{(t-t_1)(t-t_2)(t-t_3)}{(t_0-t_1)(t_0-t_2)(t_0-t_3)} \\ &+ \mathbf{x}_1 \frac{(t-t_0)(t-t_2)(t-t_3)}{(t_1-t_0)(t_1-t_2)(t_1-t_3)} \\ &+ \mathbf{x}_2 \frac{(t-t_0)(t-t_1)(t-t_3)}{(t_2-t_0)(t_2-t_1)(t_2-t_3)} \\ &+ \mathbf{x}_3 \frac{(t-t_0)(t-t_1)(t-t_2)}{(t_3-t_0)(t_3-t_1)(t_3-t_2)} \end{aligned} \tag{5.1}$$

The `orbit` part of the `RosettaKit` show the instantaneous Kepler orbit derived from the current state vector (see section 3.1.7). This shows very clearly that most of the time the spacecraft moves along an undisturbed Kepler orbit, but when near Mars and Earth (twice), the track is influenced by the respective planet's gravity so that the spacecraft apparently "changes lane" and switches to a different Kepler orbit.

When the spacecraft comes near comet Wirtanen, it fires briefly a rocket engine to enter an orbit around the comet (Figure 5.8). This is modeled as semitransparent cone which is switched on during a short time, then the rocket, track and orbit lines are switched off, and the spacecraft position is transferred into a circular orbit around the comet where it will stay forever thereafter.

The ASH ROOTS mission storyboard required a dramaturgic addition: at one point, the solar panels should lose their orientation so that a power failure will occur. The additional input, `solarPanelsMisalignment` (an `SoSFFloat`), can be used to show this.

The path of ESA's ROSETTA spacecraft was found on `http://planetary.so.estec.esa.nl/RSOC/trajectory_data.html`. This file gives dates with position and speed values for a ROSETTA trajectory simulation performed by ESA.

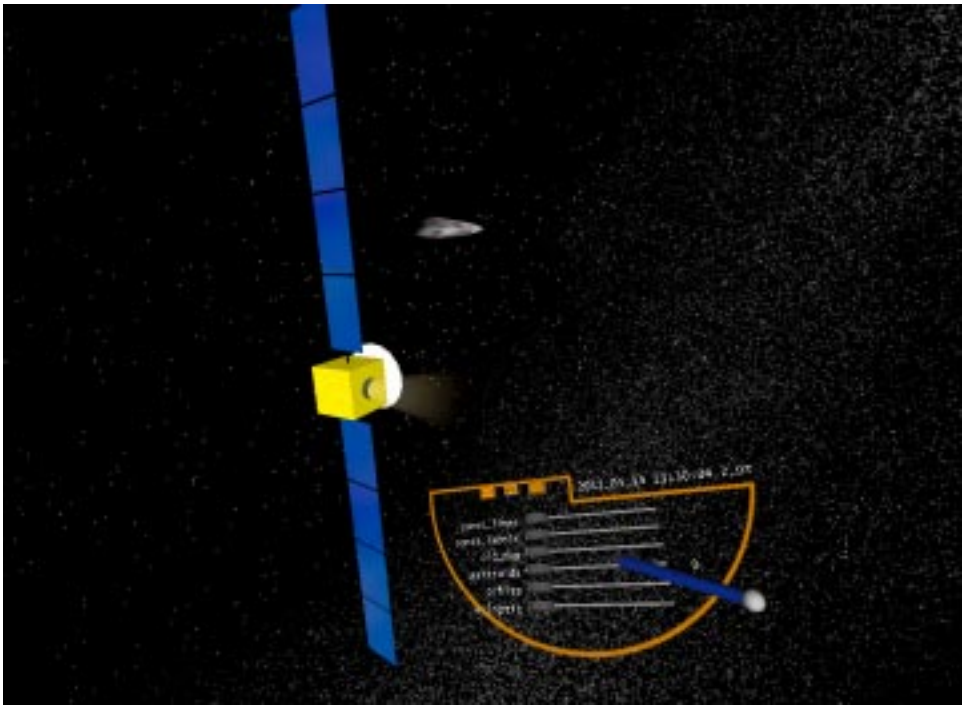


Figure 5.8: The Rosetta spacecraft approaching comet 46P/WIRTANEN. The main engine is just firing in this scene, bringing the spacecraft into orbit around the comet (upper center). Note the speckled band of the Milky Way.

5.3.6 `StarsKit`

To enhance the impression of being in the Solar System, it was necessary to add a background with stars and the Milky Way. VRMoSS does not simulate the space between the stars, thus it was sufficient to model the rest of the universe as large sphere. The radius of this sphere had to be smaller than the far clipping plane distance of the stereo viewer, thus it was necessary to exclude the starry sphere from any scaling operations.

The Stars

From the data given in the Yale BSC5 (see section 3.2), vertex and color values for an `SoPointSet` were generated with an auxiliary program using the ART toolkit (see section 4.4). This point set consists of a spherical arrangement of 9096 points of different colors, closely resembling the starry sky. Invalid entries, i.e., catalog errors, were sorted out in the process.

However, the stars' brightnesses could not be implemented exactly as commanded by the astronomical magnitude scale (see section 3.2.1), and also the colors needed special attention.

Brightness The BSC contains stars down to mag 6.5, which can be observed with keen eyesight under very good conditions. If we draw all stars of magnitude zero and brighter as "white" (8-bit RGB 255/255/255) and try to implement the brightness drop-off of $1 : 2.5$ (a loss of 60% of brightness) per magnitude step, stars of 6th magnitude would be drawn with RGB 1/1/1, which is practically black. Experiments have shown, however, that stars, if modeled in the described form, appear too dim: most monitors and projectors have nonlinear response to signal strength and do not show dim color well (gamma problem).

To remedy this situation, a smaller brightness drop-off had to be found by a series of experiments with different color values, using the model:

$$b = \texttt{mag\_step}^{m} \tag{5.2}$$

Here, b describes the linear brightness of the star, normalized to $0 \ldots 1$. Stars of magnitude 0 and brighter have $b = 1$. `mag_step`, 0.4 in nature, varied between 0.6 and 0.9. The best value was found to be around 0.75. Higher values made differences between stars of different magnitude unclear, smaller values made dim stars too dark to be seen clearly.

Still, the starry sky looked too dark, so an additional brightness `brighter` was added to the stars before applying formula 5.2:

$$b = \texttt{mag\_step}^{m-\texttt{brighter}} \tag{5.3}$$

The best combination, which is now used, is `mag_step` $= 0.75$ and `brighter` $= 0.5$.

Colors The color indices included in the BSC would be easily usable to find RGB colors for the stars. These indices describe differences in the magnitudes comparing measurements through narrow band pass filters which are directly correlated with color impression. However, the data are incomplete, therefore their spectral classes (see section 3.2.2) have been used in the preparation of the stellar colors. The stars without valid entries are handled as being white.

To represent these spectral classes, table 5.1 was devised. The respective RGB triplet was multiplied with the brightness value from equation 5.3 and renormalized to $(0 \ldots 1, 0 \ldots 1, 0 \ldots 1)$ to get the final RGB color value for each star. This color is contained in the `rgb` field of the `StarsKit`'s `SoBaseColor` node.

The Constellations

As mentioned in section 3.3, it is important to use an expressive selection of stars to represent the constellations with stick figures. The author has decided to create his own set of lines, based on the books of THOMAS [TT45]

Spectral class	Red	Green	Blue
O	0.8	1.0	1.2
B	0.9	1.0	1.1
A	1.0	1.0	1.0
F	1.05	1.05	0.9
G	1.1	1.1	0.8
K	1.2	1.0	0.8
M	1.45	0.9	0.65

Table 5.1: The figures represent multipliers for RGB values.

and Rey [Rey76] and own imagination, which should really help to identify the depicted constellations.

The lines are implemented as `SoIndexedLineSet` and reuse the stars' coordinates as vertices.

As additional aid for identification, labels have been added, implemented as `SoText2` nodes translated into their respective locations in the starry sphere. A `SoSwitch` is used to select one `SoSeparator` which itself contains the translation and text nodes. Figure 5.9 demonstrates the constellation lines and labels.

The Milky Way

The BSC includes only the brightest stars which can be seen individually with the naked eye. The diffuse glow of the billions of stars of the Milky Way had to be reproduced with different means.

In [EH88], Elsässer and Haug presented maps of stellar densities in the Milky Way. From the map showing the visual distribution, an image was prepared using Adobe PhotoShop. The stars were created as "noise" which is properly brightened in areas of higher stellar density. This map is used as texture for an `SoQuadMesh`, which represents the inside of parts of a globe.

The map in [EH88] is given in the old Galactic Coordinate system Type I (see section 3.1.5). The transformation of the coordinates into Type II and further into ecliptical coordinates was performed with the Equation Solver application of an HP-49G pocket calculator using formulæ from Landolf-Börnstein [Hel65] and commands from Urania [Zot00].

The Old Sky Map

The project title, "Access to Scientific Space *Heritage*", together with his special interest in that field, prompted the author to implement a bit of classical astronomy: it should be possible to show the stars in front of a

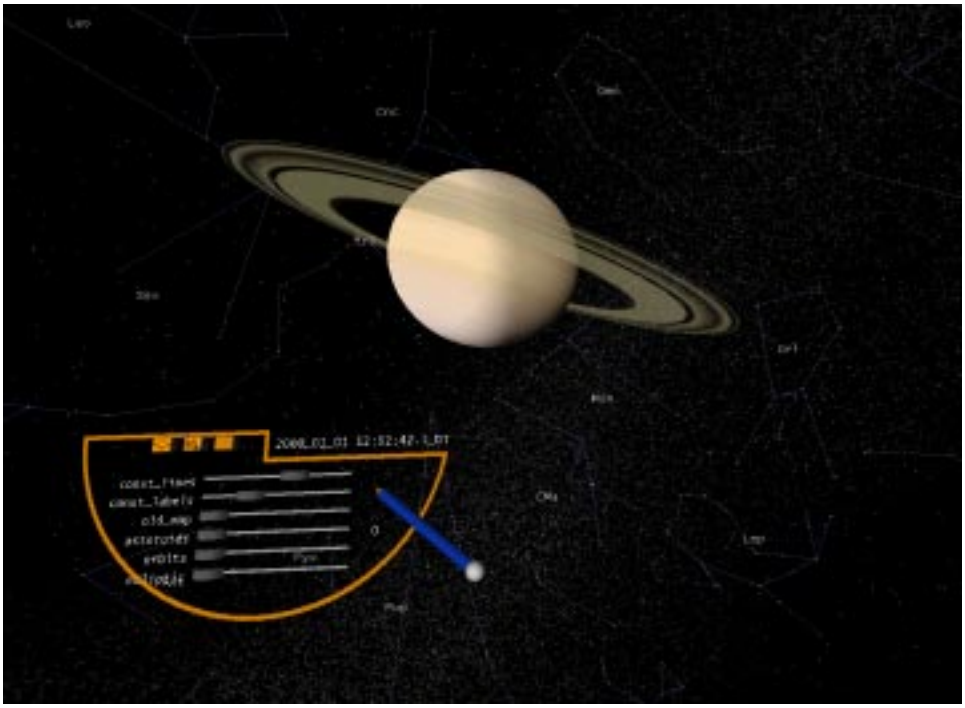


Figure 5.9: Saturn in front of the constellations, depicted with "stick figures" and labeled for identification.

figural constellation painting found mostly on celestial globes and maps of the 17th and 18th centuries.

To produce the globe, a hemispheric sky map from the 17th century, reproduced in Sesti's book, The Glorious Constellations [Ses91], was scanned. The Panorama Tools, free software tools by Helmut Dersch [Der01], was used to create a rectangular image that is used as texture map for a large `SoSphere` (Figure 5.10).

A small problem is the fact that the labels on the old map are printed as seen "from the outside" of the sky, and thus are shown reversed in the VRMoSS, which always shows the "inside" of the sky globe.

The BSC stars do not perfectly match the stars as painted on the sky map. First, precession (see section 3.1.5), which shifted the stars since the time the original map was created, had to be compensated by adding a rotation along the ecliptic. Still, some distortion introduced in the process makes it practically impossible to find a perfect match between the old map and the modern star positions. On some areas, the BSC stars lie almost exactly on the map's stars, on other areas there is a noticeable offset. Using a map without stars, but with only constellation outline drawings, e.g. as found in Thomas's atlas [TT45], could help here.

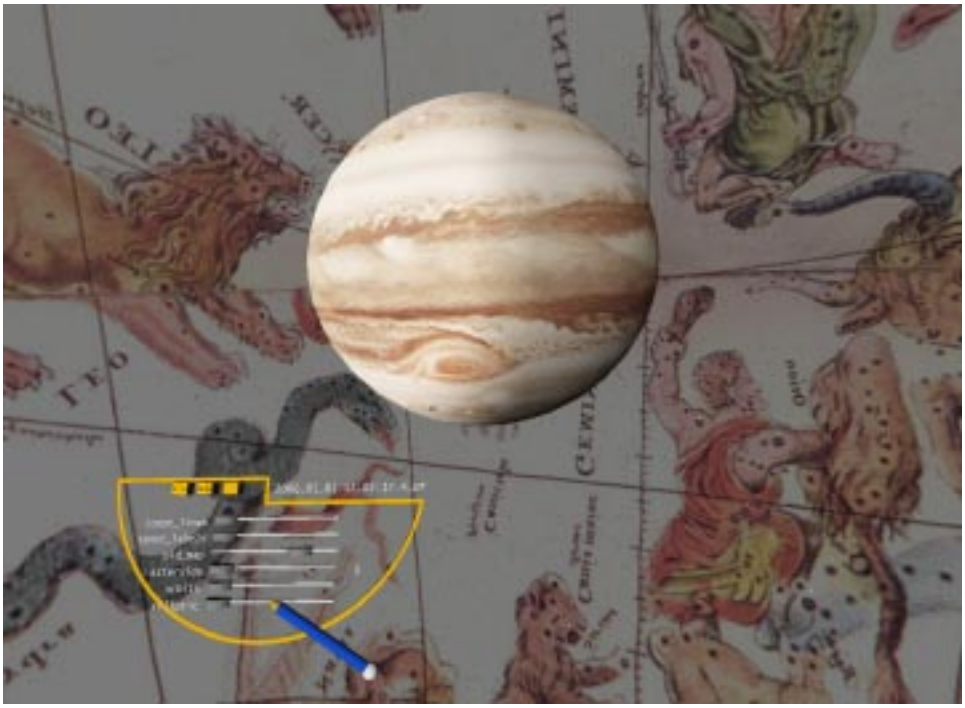


Figure 5.10: The classical constellation paintings of an old sky map used on a globe.

5.3.7 Scaling

In the VRMoSS, only the Solar System is scaled, not the stellar sphere surrounding the visitor.

The problem with scaling is the enormous range of sizes that has to be covered: small objects like comet cores, measuring only a few km, shall be visible in closeup view, or a planet like Earth, about 1000 times larger, or Jupiter, 140.000km in diameter, or the Sun, still ten times larger, or even the whole Solar System, with almost $12 \times 10^9 km$ diameter. The implemented scaling function now allows finer control in the large magnifications (closeup views) and also in the overall views, with a fast transfer in between by combining power and logarithmic functions.

The `scale` field in the `MainController` is connected to a custom `Scale-Engine`, which translates the linear input into a scaling factor which is used in the `SolsysKit`'s `WORLD_SCALE` node.

5.3.8 Bringing an object into Focus

It is not possible to only shift the Solar System to one side to bring a planet into closeup view and keep it centered: its motion will soon bring it out of

view again. Therefore it was necessary to implement a mechanism that locks the object of interest in focus.

This is implemented by connecting the interesting object's `position` field to a custom `NegateSFVec3f` engine which transmits the negated value of its `input` to its `output` field, and which is itself connected to the `translation` field of the `WORLD_TRANSLATION` node inside the `SolsysKit`.

Switching between objects is not done immediately, which would appear too abrupt, but by using a combination of `SoInterpolateVec3f` and `SoOneShot` engines and field sensors, to smoothly shift the scene to the next selected object within a short amount of time. If switching to another object is initiated while a transfer is already under way, the scene is immediately shifted from the current intermediate location to the new target.

5.3.9 Direct Scene Interaction

The STUDIERSTUBE API provides a class, `SoDragKit`, which allows to move a virtual object by touching it with the pen, pressing the pen button and dragging it to a new location. The `SoDragKit` performs a bounding box test on its `content` part to find out it it is interested in the 3D event sent by the pen.

It would probably make no sense for VRMOSS to allow dragging the planets from their natural places. However, it should be possible to grab the complete model and turn it around, including the star sphere enclosing our model.

If implemented by using a standard `SoDragKit`, the pen would be inside the Solar System's bounding box all the time, preventing 3D events from reaching the user interface elements on the PIP. Therefore, a `SoWorldDragKit` was derived which excludes the bounding box area of the PIP, thus allowing the PIP elements to receive pen/button events in its area. Most of the scene graph, namely the visible scene objects with `SolsysKit` and `StarsKit`, are the `content` part of the `SoWorldDragKit`.

5.3.10 Communication with the Outside World

The `XMLProcessor` class enables VRMOSS to communicate with another application, which may be the ASH Mission Server or, in a setup outside of an ASH VCR, any other application that provides XML messaging, e.g., a GUI or data viewer application made in FLASH or Java.

The `XMLProcessor` owned by the application's `MainController` is called by the sensor callback function of its private `SoOneShotSensor`. Objects of this sensor class call their registered callback whenever the application is idle, or after a certain time has passed. Thus, the callback is executed as low-priority task.

The `XMLProcessor` object provides the application with a server socket which listens for a client on a port defined on the command line (default: Port 2001) in non-blocking mode. Thus, even if no client is connected, the application will run without problems. Only one connection is allowed at a time.

Unfortunately, the XERCES-C++ DOM parser cannot read directly from a port. Thus, incoming messages are first copied to a memory buffer, then the DOM parser reads from that buffer:

When connection is established, the `XMLProcessor` attempts to read characters from the port. On success, all characters until a starting <, indicating the begin of an XML element, are discarded. Then, all characters until a closing > are copied into a memory buffer. If no > can be read, the message is discarded. Care must be taken that no > character is included in any character string, or this mechanism will produce a bad XML string.

As soon as a complete XML element is available, a DOM parser owned by the `XMLProcessor` parses the contents of the buffer. The resulting document object is then analyzed, and appropriate methods in the `MainController` are called.

To write out XML messages, the message contents are just passed into the `putSharedData()` method, which writes out the XML message to the port, if a client is connected.

Should a client connection break during reading from or writing to the port, the message is lost. The connection will be retried during the next execution of the sensor callback.

SolSys.dtd

The file `SolSys.dtd` shows the commands available in VRMOSS. Currently, this is only a formal specification for the ASH developers, the file is not used directly.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Document Type Declaration (DTD) for the XML commands sent between
Solar System Application "SolarStube" and its client, be it the Mission Server
or, in a different setup, a single Flash client or any other application.

Author: GZ Georg Zotti, TU Vienna, Institute for Computer Graphics

Version 0.0 August  2, 2001 GZ Initial version.
Version 0.1 August  3, 2001 GZ adapted to ASH XML data delivery protocol
Version 0.2 August  7, 2001 GZ changed Gregorian date string format to ASH style. Changed speed to direct values
Version 0.3 August 20, 2001 GZ added sol_window

This is a rather informal specification to be read by humans. (Well, programmers ;-)
Commands have to be understood by both partner applications.
Any incomprehensible message should be silently ignored.

Use:
<!DOCTYPE SolSys SYSTEM "url-path-to/SolSys.dtd" >
... but will be of no practical use. Just read lower part of this file for valid, meaningful XML messages.
-->

<!ELEMENT stop_task EMPTY>
<!ELEMENT register_subscription EMPTY>
<!ELEMENT unregister_subscription EMPTY>
<!ELEMENT put_shared_data EMPTY>
```

```
<!ELEMENT data_delivery EMPTY>

<!ATTLIST register_subscription   data_id CDATA #FIXED "sol_date" >
<!ATTLIST unregister_subscription data_id CDATA #FIXED "sol_date" >
<!ATTLIST put_shared_data data_id ( sol_center    | sol_set_gdate    | sol_set_jd      |
                                    sol_set_speed | sol_set_ecliptic |
    sol_set_scale | sol_set_orbits   | sol_set_old_map |
    sol_set_const_lines | sol_set_const_labels | sol_set_asteroids )
                        data_type ( integer | float | double | enum | string ) #REQUIRED
                value     CDATA #REQUIRED
>
<!ATTLIST data_delivery data_id   ( sol_center    | sol_scale        | sol_set_gdate   | sol_set_jd     |
                                    sol_set_speed | sol_set_ecliptic |
    sol_set_scale | sol_set_orbits   | sol_set_old_map |
    sol_set_const_lines | sol_set_const_labels | sol_set_asteroids |
    sol_window )
                      data_type ( integer | float | double | enum | string ) #REQUIRED
value     CDATA #REQUIRED
>


<!-- Based on the formal looking specification, SolarStube may receive the following messages from the mission server - ->

<stop_task /> <!-- stop the Solar System Program. Whole application will terminate! - ->

<register_subscription data_id="sol_date" /> <!-- will push date on every image! - ->
<unregister_subscription data_id="sol_date" /> <!-- will stop pushing date on every image! - ->

<data_delivery data_id="sol_window" data_type="enum" value="front" /> <!-- Make SolarStube the foreground application - ->
<data_delivery data_id="sol_window" data_type="enum" value="back"  /> <!-- hide SolarStube to allow bigscreen Flash etc. - ->

<data_delivery data_id="sol_center" data_type="enum" value="Sun"      />
<data_delivery data_id="sol_center" data_type="enum" value="Mercury"  />
<data_delivery data_id="sol_center" data_type="enum" value="Venus"    />
<data_delivery data_id="sol_center" data_type="enum" value="Earth"    />
<data_delivery data_id="sol_center" data_type="enum" value="Mars"     />
<data_delivery data_id="sol_center" data_type="enum" value="Jupiter"  />
<data_delivery data_id="sol_center" data_type="enum" value="Saturn"   />
<data_delivery data_id="sol_center" data_type="enum" value="Uranus"   />
<data_delivery data_id="sol_center" data_type="enum" value="Neptune"  />
<data_delivery data_id="sol_center" data_type="enum" value="Pluto"    />
<data_delivery data_id="sol_center" data_type="enum" value="Comet"    />
<data_delivery data_id="sol_center" data_type="enum" value="Probe"    />

<data_delivery data_id="sol_scale" data_type="enum" value="up"   />
<data_delivery data_id="sol_scale" data_type="enum" value="down" />

<data_delivery data_id="sol_set_gdate"        data_type="string"  value="YYYY_MM_DD HH:MM:SS" />
<data_delivery data_id="sol_set_jd"           data_type="double"  value="0" /> <!-- any: e.g. 2451545.5=2000.0 - ->
<data_delivery data_id="sol_set_speed"        data_type="float"   value="0" /> <!-- days/second, e.g. -1000..1000 - ->
<data_delivery data_id="sol_set_ecliptic"     data_type="float"   value="0" /> <!-- 0..1 - ->
<data_delivery data_id="sol_set_scale"        data_type="float"   value="0" /> <!-- -10..10 - ->
<data_delivery data_id="sol_set_orbits"       data_type="float"   value="0" /> <!-- 0..1 - ->
<data_delivery data_id="sol_set_old_map"      data_type="float"   value="0" /> <!-- 0..1 - ->
<data_delivery data_id="sol_set_const_lines"  data_type="float"   value="0" /> <!-- 0..1 - ->
<data_delivery data_id="sol_set_const_labels" data_type="enum"  value="off" /> <!-- off/short/latin/english - ->
<data_delivery data_id="sol_set_asteroids"    data_type="integer" value="0" /> <!-- 0..10000 - ->


<!-- SolarStube may send the following messages to the mission server (or any other current client!) - ->
<task_stopped  task_id="sol" /> <!-- on program termination. - ->

<put_shared_data data_id="sol_date" data_type="string" value="YYYY.MM.DD HH:MM:SS.d" />
<!-- only if <register_subscription data_id="sol_date"/> received before! - ->

<put_shared_data data_id="sol_center" data_type="enum" value="Sun" />
<put_shared_data data_id="sol_center" data_type="enum" value="Mercury" />
<put_shared_data data_id="sol_center" data_type="enum" value="Venus" />
<put_shared_data data_id="sol_center" data_type="enum" value="Earth" />
<put_shared_data data_id="sol_center" data_type="enum" value="Mars" />
<put_shared_data data_id="sol_center" data_type="enum" value="Jupiter" />
<put_shared_data data_id="sol_center" data_type="enum" value="Saturn" />
<put_shared_data data_id="sol_center" data_type="enum" value="Uranus" />
<put_shared_data data_id="sol_center" data_type="enum" value="Neptune" />
<put_shared_data data_id="sol_center" data_type="enum" value="Pluto" />
<put_shared_data data_id="sol_center" data_type="enum" value="Comet" />
<put_shared_data data_id="sol_center" data_type="enum" value="Probe" />

<put_shared_data data_id="sol_set_jd"           data_type="double"  value="0" /> <!-- any: e.g. 2451545.5=2000.0 - ->
<put_shared_data data_id="sol_set_speed"        data_type="float"   value="0" /> <!-- -6..6 maps from -1000d/s..1000d/s - ->
<put_shared_data data_id="sol_set_ecliptic"     data_type="float"   value="0" /> <!-- 0..1 - ->
<put_shared_data data_id="sol_set_scale"        data_type="float"   value="0" /> <!-- -10..10 - ->
<put_shared_data data_id="sol_set_orbits"       data_type="float"   value="0" /> <!-- 0..1 - ->
<put_shared_data data_id="sol_set_old_map"      data_type="float"   value="0" /> <!-- 0..1 - ->
```

```
<put_shared_data data_id="sol_set_const_lines"  data_type="float"   value="0" /> <!-- 0..1 - ->
<put_shared_data data_id="sol_set_const_labels" data_type="enum"  value="off" /> <!-- off/short/latin/english - ->
<put_shared_data data_id="sol_set_asteroids"    data_type="integer" value="0" /> <!-- 0..10000 - ->

-->
```

5.3.11 Keyboard Control

The advanced VRMoSS setup with PIP is not easily transportable. Some development work has been done at home, therefore it was necessary to have some control over the scene via standard methods.

Table 5.2 shows all keyboard commands available in VRMoSS. All keys adjusting some brightness switch in small increments between 0 and 1, then down to 0 again.

Key	Action
F	orderFront (window)
B	orderBack (window)
X	eXit
Q	Quit (same as X)
Keypad $+/-$	Time forward/reverse
Keypad $\times/\div$	Time faster/slower
P	Pause
C	switch constellation labels
L	adjust constellation lines brightness
M	adjust star map brightness
O	adjust orbit brightness
F$< n >$	Bring planet n into focus, scale to default scale factor.
	F10: Sun; F11: Comet Wirtanen; F12: Rosetta
$< n >$ R W	Bring planet n into focus (without rescaling).
	0: Sun; W: Comet Wirtanen; R: Rosetta
Page up/down	scale up/down
Cursor $\leftarrow\rightarrow$	rotate ecliptic longitude
Cursor $\uparrow\downarrow$	rotate ecliptic latitude

Table 5.2: Keyboard Controls in VRMoSS

Chapter 6

Discussion and Possible Extensions

6.1 Other Usage Scenarios

Besides being the central part of an ASH VCR installation, VRMoSS can be presented in various other setups.

6.1.1 Standalone Installation

VRMoSS can run on its own, without any connection to a mission server. Thus, depending on an institution's needs (and also on budget), it can be set up as standalone installation in a museum of science or nature, science theater or planetarium. Further details have to be considered:

Stereo or Mono

A great difference in presentation style comes from this decision: While a monoscopic (single) projector setup can be viewed just like an ordinary monitor (without any special devices), viewing a stereo projection requires some sort of special eyeglasses, which the hosting institution will have to provide. These will usually just be cheap cardboard eyeglasses with polarizing filters for one-time use, but may also be more durable models which will have to be recollected from the visitors, cleaned, stored, *etc.* Usually, this stereo setup thus will raise much more overhead, but tests showed that the presentation, especially with many orbit curves shown, will be much clearer in stereo.

With PIP

The most advanced and attractive form of interaction certainly is control with the PIP. However, the magnet tracker hardware is expensive and rather

delicate, and it probably will be necessary to have a person in charge to supervise visitors and their actions.

With TouchScreen, Flash GUI

A different scenario could be a setup where the user can take over control via a Touch Screen running an application that can send XML messages, e.g., a FLASH GUI. This is still rather uncommon and would also allow experiments in user interface design. A monoscopic projector setup could work with a single PC (with graphics hardware capable of driving two monitors), stereo projection will require two PCs on a network.

With Keyboard Navigation

For the simplest setup in single PC installations, limited keyboard interaction is available. This has been implemented mainly for testing purposes during development and sometimes uses non-obvious keypresses, but during a presentation, trained personal could also use such a setup on a single PC.

Scripted with XML Message Server

A fully-automatic presentation could be created with the addition of a server application that can send XML commands on certain times. XML offers itself naturally as scripting language in this context.

6.1.2 Desktop Application

Of course, VRMOSS can also run on a normal desktop PC with standard screen. Stereo setup will require shutter glasses, or use of the crossed-eye technique described on page 53.

6.1.3 Multi-User Scenario in Studierstube

VRMOSS should be a very interesting application for the "real" STUDIERSTUBE environment, with multiple users wearing HMDs and being able to "walk around" in the Solar System and discuss details. It might be interesting to have a planet moving through the room, for example the Earth-Moon system could be inspected in detail. With a see-through AR setting, it might be necessary to omit the starry background, or else it would probably be overlaid on the discussion partner(s). However, integration should be rather straightforward.

6.1.4 HalfDome Setup

Instead of using a flat projection screen, there are projection setups which have a spherical half dome and projector with a fisheye lens. The image

fills most of the user's field of view, creating the impression of "really being inside", but only in monoscopic mode. Also, only few users near the center of the dome will see an undistorted image.

6.2 Possible Extensions

Many details found in our highly complex Solar System have not been implemented in VRMoSS. Some ideas are listed in this section. However, many of the proposed additions will take lots of programming effort for development, and later computing power for execution, and might therefore just be understood as seen on the wish list.

6.2.1 Free-fly mode

A really interesting extension could be the implementation of a free-flight mode that may be controlled with the PIP interface. In this scenario, gravitational attraction by the planets should affect the user's position in space. The PIP could be used as control instrument, e.g., its orientation could influence the user's spacecraft's attitude, and the pen could function as joystick, controlling thrusters.

6.2.2 More Spacecraft Paths

More spacecrafts could be included in VRMoSS. Path data for many of NASA's spacecrafts, most notably the Pioneer and Voyager missions, can be found and downloaded from `http://nssdc.gsfc.nasa.gov/space/helios/heli.html`.

6.2.3 Shadows

Neither Open Inventor nor the OpenGL API provide direct support for shadows cast by solid objects into space. Still, it would add more realism to implement some of these effects. Interesting phenomena involving shadows between objects of the Solar System would be:

- Saturn's main body and rings casting shadows on each other
- Jupiter's moons casting tiny shadows on Jupiter or hiding in the giant planet's shadow
- Earth and Moon, causing Solar and Lunar eclipses.

6.2.4 Planet Moons

Currently, only Earth's Moon and Jupiter's Galilean moons are included in the VRMoSS. More moons should be added. Procedures for computing the positions of some of Saturn's moons can be found in [Mee98], for the other planets' moons they may be found in [Sei92].

6.2.5 Sun

Currently, the Sun is just a sphere with emissive color and a mostly transparent texture adding sun spots. However, looking at the real Sun in detail reveals very complex structures and high activity. Much could be done to improve realism: an animated texture could simulate the convection cells of the solar granulation, particle systems could be added to show prominences and flares. Probably the PIP could be used as "filter" that can show interesting aspects of the Sun, like an X-ray view or its magnetic field.

6.2.6 Planet Textures

The current solution uses a single, static texture per planet. Observing a planet in closeup creates a "painted marble" impression. Dynamic processes in the planets' atmospheres are not represented, especially noticeable in the highly variable gas atmospheres of Jupiter, Saturn and Neptune.

Also, Earth's clouds are not shown at all, and there is no "night side" texture with brightly glowing city lights.

6.2.7 Dynamical Loading of New Asteroid Data

Currently, only one set of orbital elements for the asteroids is loaded on startup. These data are, strictly seen, only usable for a few weeks around the epoch. It might be useful to collect asteroid data files and change the data used depending on the current time within VRMoSS. Still better, of course, would be a connection to a real numerical simulator.

6.3 Concluding Remarks

During the development of the ASH VCR, the VRMoSS has been demonstrated and tested in several meetings. The ASH science partners, all from the planetarium world, highly appreciated the system and what could be demonstrated with it.

VRMoSS can be used to explain lots of things about the Solar System. During a presentation, a trained operator (the author) demonstrated within about half an hour most capabilities of the system and the flight of the Rosetta spacecraft.

Things that seemed to impress most were the up to 10.000 moving asteroids, showing different orbital speeds and also the Troian groups trapped by Jupiter, quite realistic impressions of comets C/1996B1 HYAKUTAKE and C/1995O1 HALE-BOPP, showing the tails growing near the Sun and always pointing away from the Sun, and the visualization of Rosetta's instantaneous Kepler orbit, impressively demonstrating the swing-by effects.

Longer "guided tours" through space are easily possible. VRMOSS might however face a problem, if only trained demonstrators are able to see and show details. There are many impressive setting options in VRMOSS, and each task might require considering those settings. Typical visitors in the ASH VCR will see and use the PIP for the first time, which might overcharge and distract them from grasping the content which the ASH VCR learning environment should provide. Also, getting to know and live with the still existing deficiencies of the magnetic tracking system (short range, field distortions) takes its time and might even lead to frustration with people ignorant of the background.

To enhance the learning aspect for ASH, the passive mode with the presentation controlled by prerecorded messages instead of by the user, has been added. However, if used too extensively, it would possibly question the inclusion of a PIP altogether.

Further tests and integration with a presentable mission in the prototype VCR during the completion phase of the ASH project should allow to finetune the mixture of operation in active and passive modes.

Abbreviations

AML ASH Mission Language

API Application Programming Interface

ART Advanced Rendering Toolkit (Inst. of Computer Graphics, TU Wien)

ASH Access to Scientific Space Heritage (IST/EU)

BSC Yale Bright Star Catalog

CMG Client Manager (ASH)

CRT Cathode Ray Tube (display device)

CSCW Computer Supported Collaborative Working

C-VCR Commercial VCR (ASH)

DOM Document Object Model (XML)

DTD Document Type Declaration (XML)

EMG Episode Manager (ASH)

ESA European Space Agency

EU European Union

GSFC Goddard Space Flight Center (NASA)

GUI Graphical User Interface

HMD Head Mounted Display

HTML Hypertext Markup Language

IAU International Astronomical Union

IST Information Society Technologies (EU)

JPL Jet Propulsion Laboratory (NASA)

MDS Mission Data Space (ASH)

MMG Mission Manager (ASH)

MSI Mission State Information (ASH)

MSV Mission Server (ASH)

NASA National Aeronautics and Space Administration (USA)

OIV Open Inventor (SGI)

PIP Personal Interaction Panel

P-VCR Prototype VCR (ASH)

RAID Redundant Array of Inexpensive Disks

RGB triplets of red, green and blue color values, either in device values (usually $0 \ldots 255$) or in normalized values ($0 \ldots 1$)

ROOTS ROsetta Observing The Solar system (ASH mission)

SAX Simple API for XML

SDS Shared Data Space (ASH)

SSD System Specification Document (ASH)

SSV Simulation Server (ASH)

SGI Silicon Graphics Industries

SGML Standard Generalized Markup Language

STB Studierstube (Inst. of Computer Graphics, TU Wien)

UCL User Client (ASH)

VCR Virtual Control Room (ASH)

VRML Virtual Reality Modeling Language

VRMoSS Virtual Reality Model Of the Solar System

W3C World Wide Web Consortium

WWW World Wide Web

XML Extensible Markup Language

Index

Bibliography

[Bar91] Hans-Jochen Bartsch. *Taschenbuch Mathematischer Formeln.* Fachbuchverlag Leipzig, fourteenth edition, 1991.

[BG91] Lee E. Brotzman and Susan E. Gessner, editors. *Selected Astronomical Catalogs*, volume 1. Astronomical Data Center (ADC, GSFC, NASA), Goddard Space Flight Center, Greenbelt, Maryland, 1991.

[Bry18] Otto J. Bryk, editor. *Johann Kepler – Die Zusammenklänge der Welten.* Klassiker der Naturwissenschaft&Technik. Eugen Diederichs, Jena, 1918.

[Bur78] Robert Burnham, Jr. *Burnham's Celestial Handbook.* Dover Publications, Inc., New York, 1978.

[COR01] CORDIS. IST Overview. `http://www.cordis.lu/ist/overv-1.htm`, September 2001.

[Der01] Helmut Dersch. Panorama Tools. Website: `http://www.fh-furtwangen.de/~dersch/`, 1998–2001.

[Dre53] J. L. E. Dreyer. *A History of Astronomy from Thales to Kepler.* Dover Publications, Inc., New York, second edition, 1953.

[EH88] H. Elsässer and U. Haug. Über eine lichtelektrische Flächenphotometrie der südlichen und nördlichen Milchstraße in zwei Farben und die Struktur des galaktischen Systems. In Hermann Mucke, editor, *Seminarpapiere*, volume 16 of *Sternfreunde-Seminar*, pages 26–38, Vienna, 1988. Österreichischer Astronomischer Verein. (Reprint from: Mitteilungen des Astronomischen Instituts der Universität Tübingen, Nr. 48, in: Zeitschrift für Astrophysik 50, pages 122–144, 1960).

[ESA] ESA Rosetta Website. `http://sci.esa.int/rosetta/`.

[Hea99] Paul J. Heafner. *Fundamental Ephemeris Computations.* Willmann-Bell, Inc., Richmond, Virginia, first edition, 1999. For use with JPL data.

[Hel65] K. H. Hellwege, editor. *Landolt-Börnstein — Numerical Data and Functional Relationships in Science and Technology*, volume 1: Astronomy and Astrophysics of *Group IV: Astronomy, Astrophysics and Space Research*. Springer Verlag, Berlin, Heidelberg, New York, 1965.

[HM01] Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell*. O'Reilly &Associates, Inc., Sebastopol, California, 2001.

[Mar00] Stephen P. Maran. Selling Astrophysics in a Crystal Palace. *Sky & Telescope*, 99(5):46–47, May 2000.

[Mee91] Jean Meeus. *Astronomical Algorithms*. Willmann-Bell, Richmond, Virginia, 1991.

[Mee98] Jean Meeus. *Astronomical Algorithms*. Willmann-Bell, Richmond, Virginia, second edition, 1998.

[Mei92] Ludwig Meier. *Der Himmel auf Erden – Die Welt der Planetarien*. Johann Ambrosius Barth, Leipzig, Heidelberg, 1992.

[PC88] James B. Pollack and Jeffrey N. Cuzzi. Planetenringe. In *Planeten und ihre Monde*, Verständliche Forschung, pages 158–171. Spektrum der Wissenschaft Verlagsgesellschaft mbH&Co. KG, Heidelberg, 1988.

[Rey76] H.A. Rey. *The Stars — A New Way To See Them*. Houghton Mifflin Company, Boston, 1976. (1997 reprint).

[Sei92] P. Kenneth Seidelmann. *Explanatory Supplement to the Astronomical Almanac*. University Science Books, Sausalito, California, 1992.

[Ses91] Giuseppe Maria Sesti. *The Glorious Constellations — History and Mythology*. Harry N. Abrams, Inc., New York, 1991.

[SFH00a] Dieter Schmalstieg, Anton Fuhrmann, and Gerd Hesina. Bridging Multiple User Interface Dimensions with Augmented Reality. In *Proceedings of the 3rd International Symposium on Augmented Reality (ISAR 2000)*, pages 20–30, Munich, Germany, October 2000.

[SFH+00b] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, L. Miguel Encarnação, Michael Gervautz, and Werner Purgathofer. The Studierstube Augmented Reality Project. Technical Report TR-186-2-00-22, Institute of Computer Graphics and Algorithms, Vienna University of Technology, December 2000.

[SG97a] Zsolt Szalavári and Michael Gervautz. The Personal Interaction Panel — A Two-Handed Interface for Augmented Reality. In *Proceedings of EUROGRAPHICS'97*, volume 16, 3 of *Computer Graphics Forum*, pages 335–346. Budapest, Hungary, September 1997. Available at `http://www.studierstube.org`.

[SG97b] Zsolt Szalavári and Michael Gervautz. Using the Personal Interaction Panel for 3D Interaction. In *Proceedings of the Conference on Latest Results in Information Technology*, pages 3–6, Budapest, Hungary, May 1997. Available at `http://www.studierstube.org`.

[Tho29] Oswald Thomas. *Himmel und Welt.* Arbeitsgemeinschaft für Kultur und Aufbau, München, second edition, 1929.

[Tho56] Oswald Thomas. *Astronomie.* Das Bergland-Buch, Salzburg/Stuttgart, seventh edition, 1956.

[TT45] Oswald Thomas and Richard Teschner. *Atlas der Sternbilder.* Das Bergland-Buch, Salzburg, 1945.

[TW+01] Robert F. Tobler, Alexander Wilkie, et al. ART — Advanced Rendering Toolkit: A modular, portable rendering package implemented as a set of libraries in Objective-C. ART website: `http://www.artoolkit.org`, 1997–2001.

[TZW+01] Christoph Traxler, Georg Zotti, Morten Wagner, Bernard Fontaine, and Georges Focant. System Specification. Deliverable 4, work package 4, ASH consortium, 2001. version final 4.7.0.

[Wer94a] Josie Wernecke. *The Inventor Mentor: Programming Object-oriented 3D Graphics with Open Inventor, Release 2.* Addison-Wesley, 1994.

[Wer94b] Josie Wernecke. *The Inventor Toolmaker: Extending Open Inventor, Release 2.* Addison-Wesley, 1994.

[WZ71] Alfred Weigert and Helmut Zimmermann. *ABC der Astronomie.* Werner Dausien, Hanau/Main, third edition, 1971.

[Zot00] Georg Zotti. *Urania — The Astronomical Companion for the HP-48 and HP-49 Pocket Calculator Series, Version 2.1.* Vienna, 2000.

Acknowledgements

This work would not have been possible without other people's work.

First of all, I would like to express my thanks to Prof. Hermann Mucke, director of the Zeiss Planetarium Vienna 1963–2000, the Urania public observatory 1971–2000, and secretary of the Astronomical Society of Austria (Österreichischer Astronomischer Verein). With his personal ambition and countless inspiring lectures on astronomy, especially for children and laypersons, he excited strong astronomical interest in many visitors, me included. Now retired from the planetarium, he still teaches astronomy in a new environment, the Sterngarten, an "open air planetarium" built by the Society on the western outskirts of Vienna. I feel honored to have worked as assistant, later as guide at the Urania observatory, and wish him and his wife many more years to go!

I would like to thank members of the Institute of Computer Graphics and Algorithms of the Vienna University of Technology, most notably Michael Gervautz and Christoph Traxler for suggestions and helpful hints, and the Studierstube teams at the Institute and at VRVis, the Vienna Competence Center for Virtual Reality and Visualization, most notably Rainer Splechtna, Anton Fuhrmann, and Zsolt Szalavári for their support.

My colleagues Zsolt Marx and Gottfried Eibner built the basis of a comet model with animated tail and gas particles evaporating from the comet core. They deserve my thanks for letting me adapt and use it in VRMoSS.

Last, but not least I would like to thank other members of the ASH team for providing a great system to teach and learn astronomy. I hope that the ASH project will continue with interesting missions that make use of VRMoSS and show many young people the fascination of astronomy and space exploration.