Haptic interaction with a virtual 3D model:
A multimodal interactive system for 3D solar system

Zhenxing Li

University of Tampere
School of Information Science
Interactive Technology
M.Sc. thesis
Supervisor: Roope Raisamo
April 2013

University of Tampere
School of Information Science
Interactive Technology
Zhenxing Li: Haptic interaction with a virtual 3D model: A multimodal interactive system for 3D solar system
M.Sc. thesis, 81 pages, 7 index and appendix pages
April 2013

Haptic interaction has become more and more important in interactive technology. In current human-computer interaction, haptic interaction has been considered as an important additional interactive method. The benefits of haptic interaction mainly include high efficiency, accuracy and naturalness.

In this thesis, a multimodal interactive system was implemented based on a large volume 3D model of the solar system. This multimodal interactive system included two subsystems which separately used traditional computer interactive devices, a mouse and a keyboard, as well as a new haptic interaction device. These two interactive subsystems contained many relevant interactive functions for the user to interact with the model of the solar system and the models of celestial bodies inside it. In addition, the interactive methods for a large volume 3D model were studied in this research. Finally, a user study was employed to demonstrate the benefits of haptic interaction in a multimodal interactive system, and also the methods for improving current haptic technology had been discussed.

To sum up, the work of the thesis includes a theoretical discussion, the implementation of a multimodal interactive system and a user study, which focuses on the research for haptic interaction in the field of human-computer interaction.

Key words and terms: Haptic interaction, virtual 3D model, multimodal interactive system, human-computer interaction.

Acknowledgments

The work presented in this thesis was completed in the haptic laboratory of School of Information Science at the University of Tampere.

I would like to thank my professor, Roope Raisamo, for guiding me in the direction of my research, giving valuable advice for my thesis and providing laboratory and advanced hardware devices for supporting my work. And also I want to thank professor, Erkki Mäkinen, for reviewing my thesis. Furthermore, during the period of the user study, fifteen students participated in the testing of my multimodal interactive system, and I am very thankful to all of you for your time and comments.

Finally, I want to thank my wife and my parents. With your support, I have had enough time and energy to finish my research work and I owe my deepest gratitude to you. All in all, thank you all for your valuable support on the road of my life.

Li zhenxing

Tampere, 11.04.2013

Contents

1. Introduction 1
2. Multimodal human-computer interaction 4
    2.1 Principle of multimodal human-computer interaction 4
    2.2 Current haptic devices and software resources 6
    2.3 Historical researches related to haptic interaction 10
3. The visual 3D model of the solar system 12
    3.1 Software and hardware systems 12
    3.2 Implementation of 3D virtual solar system 13
        3.2.1 Implementing details of the revolution of planets 13
        3.2.2 Implementing details of the revolution of moons 16
        3.2.3 Implementing details of the rotation of all celestial bodies 18
        3.2.4 Implementing details of other improvements 20
4. Implementation of the multimodal interactive system 23
    4.1 Specification of the multimodal interactive system 23
    4.2 Implementation of 3D model of graphical user interface 26
    4.3 Implementation of functions of UI 30
        4.3.1 Implementing details of size-changing function in UI 30
        4.3.2 Implementing details of speed-changing function in UI 32
        4.3.3 Implementing details of other small functions in UI 35
        4.3.4 Implementing details of open and close function of UI 39
    4.4 Implementation of interactive system using a haptic device 43
        4.4.1 Implementing details of UI rotation 43
        4.4.2 Implementing details of Zoom in and out function 46
        4.4.3 Implementing details of information-showing function 50
        4.4.4 Implementing details of orbit-changing function of planets 53
    4.5 Implementation of interactive system using a mouse and a keyboard 60
        4.5.1 Implementing details of rotation of the UI and the solar system 60
        4.5.2 Implementing details of information-showing function 63
5. User study for the multimodal interactive system 65
    5.1 The setup and procedure of the experiment 66
    5.2 Results of the experiment 67

6. Discussion 72
7. Conclusion 76
References 77
Appendix 82

1. Introduction

In the long process of human evolution, humans have gradually developed a fast-functioning sensory system which includes five basic subsystems: visual, auditory, somatosensory, olfactory and gustatory. Each subsystem has its specific sensory organ, including the eyes, ears, skin, nose and tongue. Humans can perceive the outside physical world generally based on the signals detected by these organs, and the cognition of the physical world is eventually formed through our brain [Moller, 2002]. In other words, an object perceived by all or part of these human senses can be considered to be a “real” object. The olfactory and gustatory organs can only receive chemical signals emitted from objects and, thus, these two senses belong to the chemical senses [Kortum, 2008], and the visual, auditory and somatosensory senses are related to physical stimuli, such as colors to the eyes, sounds to the ears and vibrations to the skin. In current human-computer interaction (HCI), the visual, auditory and somatosensory senses are the most popular study fields for developing new interactive technologies. Due to the power of the human visual sense, most current researchers prefer to study and develop new visual interactive technologies, such as the technology of using the eyes to control virtual objects [Rudmann et al., 2003; Sundstedt, 2010]. In addition to visual technology, many researchers have started to study the interactive methods of auditory modality, for example, new speech-based user interfaces for inputting commands to computers [Arjunan et al., 2006]. However, simultaneously somatosensory modality is equally important. It can largely assist current visual and auditory interactive methods and make the interaction with virtual objects more natural. The human somatosensory system mainly includes tactile and kinesthetic sensations. Tactile sensation perceives information derived from cutaneous inputs and kinesthetic sensation perceives physic stimuli arising within the body regarding motion and position [Raisamo, 2011]. Both have great significance in human-computer interaction: the former sensation can be employed in the application of implementing new physical properties, such as different textures of virtual objects; and the latter can be used in the application of simulating torque for dynamic virtual objects. In interactive technology, the interactive technologies related to both of them are called “haptic technology” or “haptics”, and their devices and displays are used for exchanging mechanical energy with users.

Although haptic interaction is very important to users, haptic feedback is still an underused modality [Raisamo, 2011] and, thus, its benefit is still in the exploratory stage. According to Maybury and Wahlster [1998], there are six main benefits of haptic interaction used in multimodal human-computer interaction: efficiency, redundancy, perceptibility, naturalness, accuracy and synergy. However, these benefits are still theoretical and not yet verified by practices and experiments. In order to demonstrate the benefits of haptic interaction, the development of the relevant haptic technologies is necessary and pressing. However, haptic technology is not a single and independent research field but is related to mechanics, computer science, somatology and also electronics. Its basic research areas include human haptics, machine haptics and computer haptics [Saddik, 2007]:

- Human haptics focuses on the study of human sensing through tactile and kinesthetic sensations.
- Machine haptics includes designing and producing mechanical devices which augment or replace human touch.
- Computer haptics is considered to develop algorithms and software to generate and render touch feeling for virtual objects.

These three research fields, especially machine haptics and computer haptics, are still in the exploration and development stage, which largely limit the development of applications of haptic interaction and, thus, explain the fact that the benefits of haptic interaction have not been demonstrated.

For example, in the field of machine haptics, most current force-feedback haptic devices still belong to grounded haptic devices[1], which greatly limit their applications in mobile environments. Researchers such as Minamizawa et al. [2007] have started the study on this area. In the field computer haptics, computer haptic rendering refers to a group of algorithms used to compute and generate forces and torques for interaction [Lin and Otaduy, 2008], which currently can provide force feedbacks only for rigid virtual objects with regular shapes. The further studies of haptic renderings include the haptic renderings developed by Avila and Sobierajski [1996], who provided different haptic rendering methods for volume visualization, and by König et al. [2000], who developed a new non-realistic haptic rendering, and so on.

Although some applications of haptic interaction have been implemented, their study purpose is focused on the implementation of new haptic virtual objects. For instance, 3D virtual brushes used for haptic painting by Baxter et al. [2001]; the research of material texture by Huang et al. [2003]; and the simulation of the spine with haptic interface by Gibson and Zhan [2008]. They all employed the characteristics of haptic interaction to manipulate and interact with some special virtual objects. For example, Huang et al. used a fabric-typed material as the main virtual material, and they developed a force feedback system for simulating the touch feeling of this kind of materials. This study showed that haptic interaction as an additional interactive method can let virtual objects have more physical properties and make them more real. However, it is apparent that this virtual material cannot be interacted using traditional computer interactive devices such as a mouse and a keyboard. Due to this, it is difficult to

[1] A grounded device means the device is fixed or connected with the ground. For details, see Chapter 2.

compare the traditional interactive method with the haptic interactive method, and thus, these studies cannot be directly used to demonstrate the benefits of haptic interaction mentioned by Maybury and Wahlster [1998].

In order to systematically analyze the benefits of haptic interaction, a suitable application has to be developed, which can be interacted simultaneously using a traditional computer interactive method and a haptic interactive method. A user study based on these two interactive systems can be then employed to investigate the benefits of haptic interaction. Therefore, in this thesis, a multimodal interactive system based on a three-dimensional model of the solar system was designed and implemented for comparing the haptic interactive method with the traditional interactive method. The idea of creating the model of the solar system as the basic platform of application originated from Barnett et al. [2001] and Gazit et al. [2004]. They showed that the solar system is a very interesting object to most people and also a significant learning topic in astronomy. The traditional interactive system in this project employed a mouse and a keyboard as the interactive tools. Essentially, due to the design principle of the mouse and keyboard, the traditional interactive method allows user to control virtual objects only in a two-dimensional way, but in the new haptic interactive method, the interaction takes place in a three-dimensional work space. Through constructing and mixing these two independent interactive systems, it was possible to construct a complete multimodal interactive system. The users not only could use the whole multimodal system to interact with the model of the solar system, but also separate it into two basic subsystems to test the performance of each system. In the comparison of experimental results, efficiency, accuracy and naturalness were the main experimental parameters for evaluating haptic interaction. Finally, as the result, the user study of this thesis showed that current haptic interaction already has some expected benefits in a multimodal interactive system, such as naturalness, and also, as an additional interactive method, it is an integral part of the future 3D interactive system.

This thesis is divided into four parts. The general knowledge and background of relevant technologies and devices will be introduced firstly. The second part will describe the details of implementing the visual 3D model of the solar system. The specification of the two independent interactive systems and their implementation details will be described in the third part. Finally, a complete user study will be conducted in the last part of thesis, and its results will be discussed and compared with previous works.

2. Multimodal human-computer interaction

In order to construct a multimodal interactive system, it is necessary to understand the basic principles of human-computer interaction and the background of current existing haptic devices and technologies. Therefore, in this chapter, the main point is to study the basic principles of multimodal human-computer interaction, before describing the relevant products of hardware and software resources. Finally, some significant historical applications related to haptic interaction will be introduced.

2.1 Principle of multimodal human-computer interaction

In multimodal human-computer interaction, the whole interaction process can be divided into six steps which shape a completed interaction loop [Dix et al., 2003]:

1. Human cognitive process, based on input signals through human five senses.
2. Human output channels, based on the results of human cognitive process.
3. The input modalities (devices) of the computer, based on human output channels.
4. The “cognitive process” of computer, based on the data from the input devices.
5. The output modalities (devices) of computer, based on the result of “cognitive process” of the computer.
6. Human input channels, based on the output modalities of the computer.

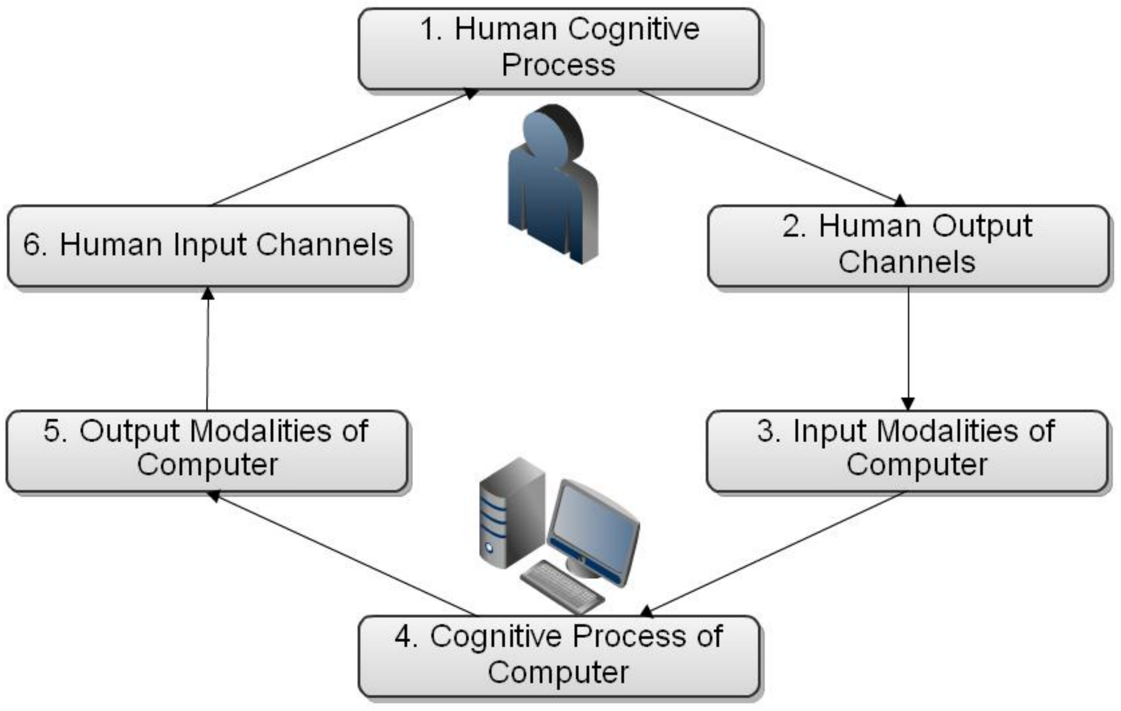

1. Human Cognitive Process
2. Human Output Channels
3. Input Modalities of Computer
4. Cognitive Process of Computer
5. Output Modalities of Computer
6. Human Input Channels

Figure 1: Multimodal human-computer interaction

The first step of Figure 1, the human cognitive process, is a complicated mental process. When input signals carried by neurons enter our brain, the complex nervous network of the brain will process these signals and form the cognition and understanding for the outside environment and objects [Coren et al., 1998]. This process relies on signals detected through our five main senses: sight, hearing, touch, smell and taste.

After the process of cognition, human beings will make physical responses through output channels including different body or hand movements, gaze, touch, speech and so forth. Nowadays, even body temperature and the signal of neurons in our brain have been used as output channels. For example, brain-computer interface [Esfahani and Sundararajan, 2012] is a novel interactive interface, which can collect brain activities from specific locations on the scalp of a user and make the computer to react to them. There are many human body reactions which can be used as output channels.

The computer then will sense these reactions employing different computer peripheral devices. The choice of devices depends on the type of human output channel. For example, the traditional devices for computer are keyboards and mouses which can detect finger pressing and hand movement. Depending on the position of user's finger pressing on the keyboard, text information can be identified and sent to the computer, and also the moving and clicking of a mouse can realize various computer commands, such as 'select', 'copy' and 'paste'. In addition to these devices, microphones can be used as the speech input device, and the camera-typed devices can be used as visual input devices to detect human hand movements, gaze or other body movements. For instance, Kinect [Francese et al., 2012] is one such successful application. In haptic interaction, force feedback devices can be both input and output media, and they are often used to receive and generate force and torque, and then exchange these mechanical signals between a user and a computer.

While the computer receives data from the peripheral devices, the data needs to be processed in order to understand the commands to make appropriate responses. This process is what is called "the cognitive process" of the computer. In fact, the process relies on various software and algorithms installed in the computer, and they can recognize and compute the input data, and then output appropriate results. For example, computer visual rendering is one such group of algorithms which can generate and render visual images, and haptic rendering is designed to generate forces and torques in response to interactions of haptic interface point inside virtual models [Lin and Otaduy, 2008].

After the cognitive process of the computer, the processed results are presented through interaction devices to the user. Currently, there are many different types of computer output devices. For instance, monitors and speakers are the common devices to present the visual and auditory signals to the user. In addition, haptic feedback devices and wearable tactile displays [Chouvardas et al., 2005] can be used as the output media for transmitting haptic signals. Olfactory and gustatory devices and interfaces are currently relatively rare compared with the above types.

Finally, the sensory system of our body, including the senses of sight, hearing, touch, smell and taste, will perceive the signals from computer output devices, and transmit them to our brain.

Based on the principle of a multimodal interactive system, the development of new haptic interactive system is actually a process to add haptic input and output modalities to both the computer and the user. However, although humans can easily adopt this new modality because of instinct, computers need new haptic devices and also new algorithms to support the use of haptic devices and process the input data. Therefore, the existing haptic hardware devices and their software resources should be introduced as the basis to implement the project of the thesis.

2.2 Current haptic devices and software resources

Based on human somatosensory system, researchers and engineers develop haptic devices in two different ways. One way focuses on creating the devices of tactile-feedback and another way is to develop force-feedback devices.

Tactile-feedback devices mainly provide physical stimulation on the skin, and the methods include pressure, vibration, electric stimulation, skin stretch, temperature and so forth [Raisamo, 2011]. The most common devices are vibration-based devices which can generate vibration signals to the human skin. Besides vibration devices, there are many other types, for example, pressure-based devices, surface acoustic wave devices, electro-rheological devices and electro-tactile stimulation devices [Chouvardas et al., 2005]. Nowadays, most of tactile-feedback devices are used to interact with some electronic and mechanical equipment instead of computers, and most of the tactile feedback does not need to be processed by complicated algorithms and software. For example, researchers often utilize motors to generate vibration signals, which are widely used in commercial products such as mobile phones, game consoles and toys to provide haptic feedback.

Force-feedback devices are currently the major human-computer haptic devices. Force-feedback devices can be used both as input and output modalities for computers to receive and generate force and torque. Since these devices are directly controlled by human hand, the mechanical structure of haptic devices should be designed in a way which can sense the movement of human hand and arm. Actually, the easiest way to design this device is to simulate the structure of human arm. Based on this principle, many current devices, such as Phantom and Falcon, have been designed. In order to clearly explain their structures, degree of freedom (DOF) needs to be introduced. It refers to an independent axis to specify the position and orientation of rigid objects in physics [Uicker et al., 2010]. For example, human wrist has six DOF. Three DOF is for displacement forces and another three DOF is for orientation forces [ElKoural and Singh, 2003]:

- Displacement: moving up and down, moving left and right, and moving forward and back.
- Orientation: tilting up and down (pitching), turning left and right (yawing), and tilting side to side (rolling).

In the design of haptic devices, DOF refers to the types of forces which can be received or generated through this device, instead of the DOF of self-structure of devices. For instance, three DOF haptic devices have six DOF positional sensing but they can receive and generate only three displacement forces (x, y, z in 3D spaces). And six DOF haptic devices could receive and generate both displacement and orientation forces (x, y, z, pitch, yaw and roll). In addition, in the structure of device, due to the need of the pivot of force, most of the current force-feedback devices are grounded devices. So, this situation limits their applications in mobile environments. Many researchers have realized this drawback and begun to design ungrounded devices for simulating force feedback [Minamizawa et al., 2007; Aoki et al., 2009]. However, because of the lack of force pivot, it is difficult to generate big forces with these ungrounded devices. This makes them look more like a type of tactile-feedback device. Grounded and ungrounded haptic devices are shown in Figure 2.

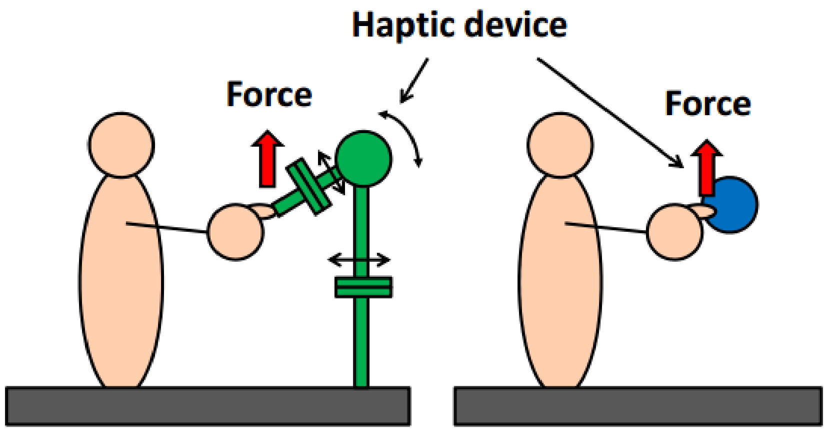


Figure 2: Grounded and ungrounded haptic devices [Aoki et al., 2009]

In addition to haptic hardware devices, the relevant software resources are necessary for supporting the use of haptic feedback devices, especially for force-feedback devices. Currently, there are several haptic software development kits (SDK) available [Kadlecek, 2011]: Chai3D, Reachin, OpenHaptic, H3D and GodObject. Among them, only Chai3D, OpenHaptic and GodObject have an independent haptic rendering, while the others are developed as a software package which integrates various haptic renderings from other SDK. For instance, H3D is a scene graph-based application programming interface (API) by SenseGraphics Company. It contains graphic rendering

(OpenGL [Shreiner et al., 2007]) and haptic renderings (Chai3D, OpenHaptic and GodObject), and users can randomly choice a haptic rendering which they need.

Although there are many different haptic renderings, their basic principles are almost the same. Current haptic rendering techniques basically have two types: point-based and ray-based [Raisamo, 2011]. In point-based haptic interaction, when haptic interface point (HIP) penetrates into the virtual object, algorithms will check the depth of penetration inside the object in order to calculate the reaction force. In ray-based haptic interactions, the haptic interface point is changed from one-dimensional point into a two-dimensional ray with orientation, and the reaction force is obtained by checking the places of ray and object. In most haptic renderings, the reaction force (F) is calculated using the linear spring law [Uicker et al., 2010]

$$F = k * x \ , \tag{2.1}$$

where k is spring constant, and x is the distance of the end of spring from its original position. So, in our case, k can be considered as the stiffness of object, and x is the depth of penetration. For a very rigid virtual object, the value of k should be set as high as possible, and otherwise, the object will become soft.

However, there is a problem emerged from this situation: how to detect the collision of haptic interface point with virtual object in an infinite 3D space? Due to the various shapes of virtual objects, checking every detail in the whole 3D space seems too time-consuming. Gottschalk et al. [1996] provided a good solution to this problem.

Let's render a 3D virtual sphere with radius R using 3DOF point-based device, as shown in Figure 3.

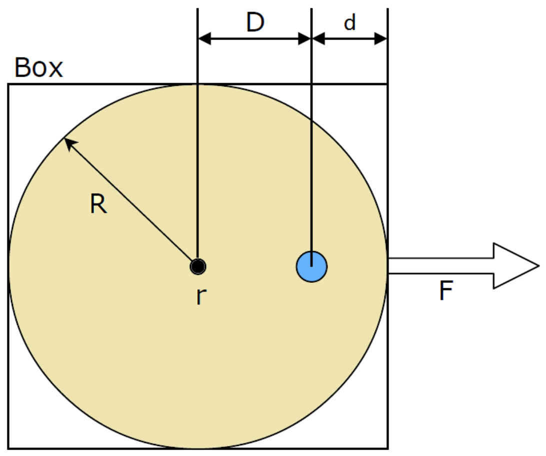


Figure 3: Cross section of virtual sphere (yellow) with HIP (blue)

First of all, the collision detection of interface point within sphere is divided in two steps [Gottschalk et al., 1996]. The first step is to check whether the point is inside the box, and if so, the second step will check if it is inside the sphere.

The first step is simple. We build up a rectangular coordinate system with center point r (0, 0, 0), and then assume that the coordinates of the eight vertexes of box are (X_n, Y_n, Z_n), where $1 \leq \mathrm{n} \leq 8$, and the coordinate of the interface point (HIP) is $(\mathrm{a}, \mathrm{b}, \mathrm{c})$, then judging if the following equations are true at the same time:

$$(X_n)_{min} \leq \mathrm{a} \leq (X_n)_{max} \ \&\& \ (Y_n)_{min} \leq \mathrm{b} \leq (Y_n)_{max} \ \&\& \ (Z_n)_{min} \leq \mathrm{c} \leq (Z_n)_{max} \quad ,(2.2)$$

where $(X_n)_{min}$ and $(X_n)_{max}$ are the minimum and maximum x-axis coordinate values among eight vertexes, and so on. It means that if the coordinate values of the interface point are in the range of maximum and minimum coordinate values of the eight vertexes, it can be concluded that the interface point is inside the box.

After the first step is passed, we can begin to consider if the interface point is inside the sphere. This can be done by measuring the distance (D) between the interface point and the center of sphere (r). If the distance is bigger than the radius of sphere (R), the point is outside the sphere. If the distance equals to or is smaller than the radius of sphere, the point is inside the sphere. Distance (D) and, thus, penetration depth (d) can be obtained by the following equations:

$$\mathrm{D} = \sqrt{a^2 + b^2 + c^2} \tag{2.3}$$

$$\mathrm{d} = \mathrm{R} - \mathrm{D} \ . \tag{2.4}$$

The next task is to calculate the values of force and torque based on the penetration depth (d). Because the haptic device we are employing is a three DOF device, we only need to consider the displacement forces. Orientation forces have to be ignored since they cannot be received and generated using this device. In calculation, all forces on the interface point are divided into three-directional force vectors using three-dimensional coordinate. Thus, the reaction force (F) is

$$\mathrm{F} = (F_x, F_y, F_z) \ . \tag{2.5}$$

Then, in order to calculate the above force vectors, the displacement values of penetration depth (d) on x-axis, y-axis and z-axis need to be calculated:

$$d_x = \frac{a}{D} * (R - D) \tag{2.6}$$

$$d_y = \frac{b}{D} * (R - D) \tag{2.7}$$

$$d_z = \frac{c}{D} * (R - D) \ . \tag{2.8}$$

According to the linear spring law (2.1), we get equations for sphere:

$$F_x = k * \frac{a}{D} * (R - D) \tag{2.9}$$

$$F_y = k * \frac{b}{D} * (R - D) \tag{2.10}$$

$$F_z = k * \frac{c}{D} * (R - D) \ . \tag{2.11}$$

Haptic rendering is developed based on creating such a group of equations (2.9 – 2.11) for virtual objects with different shape, such as cube, sphere, cone and other polygonal objects. Furthermore, the differences of current haptic renderings are mainly due to the differences of their algorithms and constant values inside them, such as the k in the equation (2.11).

2.3 Historical researches related to haptic interaction

To begin with, as early as 25 years ago, in 1987, NASA Research Center provided a virtual environment display system with multimodal interaction [Fisher et al., 1987]. They designed a set of interactive devices to interact within a virtual environment, including a head-mounted visual display, tactile input glove, speech recognition device and gesture tracking device. These devices allowed user to interact with virtual object by user's position, voice and gesture. And specifically, the head-mounted display device provided user a wide-angle stereoscopic vision image, with a connection to a host computer. Speech recognition allowed the user to input commands to computer using speech, for example, using pronunciation of letters to give information to host computer instead of using a keyboard. Tactile input glove was employed to manipulate and control virtual objects. This glove device had many special sensors to detect the movement of different parts of human hand and arm, such as fingers, palm, wrist and elbow, and recorded their positions and orientations. Using these data, user could control the virtual objects in natural ways, such as grasping, rotating and so forth. Besides these devices, an independent gesture tracking device was designed for checking user's head movement, so that the head-mounted display could give different images to user depending on the position and orientation of user's head. The system was mainly intended for studying robotics, information management and human factors. For instance, in their experiment, they tried to create a large and efficient interactive environment for astronauts in space station, which could allow astronauts to handle amounts of works in a limited space. Overall, their interaction methods with virtual objects were multiple, containing human input channels vision and audition and human output channels speech and gesture. Despite of that many details of these devices have been kept in secret till now, their system is a good example of multimodal interactive system.

For designing a multimodal system, the combination of human interactive modalities needs to be seriously considered in order to attain an efficient and accurate multimodal interactive system. Jeong and Gluck [2002] provided an experiment to test human performance using different combinations of interactive modalities as human input channel. In the test, bivariate thematic maps were used as test objects, which were represented by two variables simultaneously. The variables could be visual, auditory or haptic signals. Through making the different group of variables, they studied and tested which group of modalities could make the process of understanding map easier to human. The groups of modalities included visual-visual, visual-auditory, visual-haptic and auditory-haptic. Visual signals in the experiment were different colors, auditory signal was short musical sound and haptic signal was vibration. Jeong and Gluck found out that, based on the mean completion time of task, visual-visual group was the fastest one but with the lowest accuracy due to the difficulty to understand various combinations of colors, and auditory-haptic group was the slowest one but with the highest accuracy. They concluded that auditory and haptic display maybe is the best solution for representing bivariate thematic maps. However, there is an interesting finding in the research they did not notice. According to the data in their paper, the mean completion time of visual-visual (86.56 seconds), visual-auditory (88.51 seconds) and visual-haptic (97.30 seconds) groups were almost the same, but auditory-haptic group needed 121.20 seconds, and their correct rates were 1.75, 3.41, 3.75 and 5.50, respectively. Although auditory-haptic group had the highest correct rates (5.50), it was the slowest (121.20 seconds). The visual-haptic group actually was the best group of modalities with the higher accuracy (correct rate 3.75) and higher processing speed (97.30 seconds). Therefore, this study indicates that visual-haptic combination could be the best input channel group for human, which could be used as a good basis for the research of this thesis.

In the interaction within large volume virtual environment, interacting with virtual objects is very difficult, such as navigation in a virtual world. Many research groups have realized this problem and tried to find a good solution for haptic interaction. Smith et al. [2007] solved this problem in a special way. They designed a large-scale haptic hardware device which allowed users to interact with virtual objects when they were walking inside the virtual environment. The virtual environment they built up was a large supermarket using CAVE visualization room [Cruz-Neira et al., 1992]. The structure of the haptic device was modeled as a real shopping trolley which could make user to easily adapt this haptic device. The structure of the whole device was complicated including a position control system and a velocity control system to detect the position and velocity of shopping trolley and transmit data to host computer. By adding different motors in the device, force effect could be generated to simulate interaction with virtual objects in this virtual market. Smith et al. provided a good solution to deal with the navigation problem in a large volume of virtual environment, but the hardware device was too expensive. Their virtual environment was presented using a 3D visualization technology and also interactive method is three-dimensional. In the situation of only having a normal 2D computer screen and a desktop haptic device, the problem of haptic interaction within large volume virtual environment still needs to be considered and solved.

3. The visual 3D model of the solar system

The implementation of a suitable application platform for the multimodal interactive system is very important in this project. This application has two basic requirements:

1. The interactive tasks of application have to be handled simultaneously by the traditional interactive tools, a mouse and a keyboard, and a haptic device.
2. The application should be interesting to users and have high learning and entertainment values.

According to Barnett [2001] as well as Gazit [2004], the solar system is highly complicated scientific concept in astronomy, but most people are interested in it. A virtual solar system can help people to easily understand lots of astronomical phenomena and study the relevant knowledge, such as the day-night cycle, the solar and lunar eclipses, the generation of seasons, and so on. Furthermore, a virtual 3D model with navigation tools can increase young learners' perceptual and cognitive abilities. Therefore, a virtual solar system is a suitable application for the multimodal interactive system, and both the traditional interactive tools and haptic devices can be employed for the interactive tasks of this virtual model.

The project software platform and hardware devices are introduced in this chapter. The implementation details of a visual three-dimensional model of the solar system are described as well. This implementation has been done in a previous advanced course.

3.1 Software and hardware systems

The hardware system contains many suitable peripheral devices such as speakers, mouses, keyboards and haptic devices. Speakers are normal devices used as auditory output devices, and mouse and keyboard as the common input devices of computer are used for interaction in the traditional two-dimensional way. Haptic devices are employed as the three-dimensional interactive tools, which can be simultaneously as both input and output devices. This kind of haptic devices has been designed and produced by many companies such as Omega series haptic devices of Force Dimension, Falcon of Novint and Phantom series of Geomagic.

In addition to hardware system, a suitable software system is also important. Since haptic, visual and auditory elements have to be implemented simultaneously, the software platform must handle both graphics and haptics and also integrate audio with them. Currently, there are some software development kits available as mentioned in the previous chapter, and the best choice for this project is H3D which is open source software. It integrates various haptic renderers including Openhaptics, CHAI3D, GodObject and Ruspini, and it uses open standards OpenGL to process the graphics [H3D Manual, 2009]. Basically, H3D is created mainly by C++, and X3D and Python are integrated in this software development platform. X3D is the ISO standard XML-based file format to design 3D computer graphics mainly for web [Brutzman and

Daly, 2007] and Python is a general-purpose, interpreted high-level programming language whose design philosophy emphasizes code readability [Kuhlman, 2011]. Therefore, H3D supports X3D language to design three-dimensional graphics and Python to implement behaviors and create logical events. Besides these two languages, C++ as the original language can be used to create new haptic and visual elements such as a new shape node with a special force feedback.

The computer peripheral devices could be changed depending on the available devices in lab. For example, the haptic device will be chosen between Omega device of Force Dimension and Phantom device of Geomagic. Basically, there is no essential difference between them, and the only difference is that they use different haptic renderers which will not affect the implementation of the project. The basic structure of project system is shown in Figure 4.

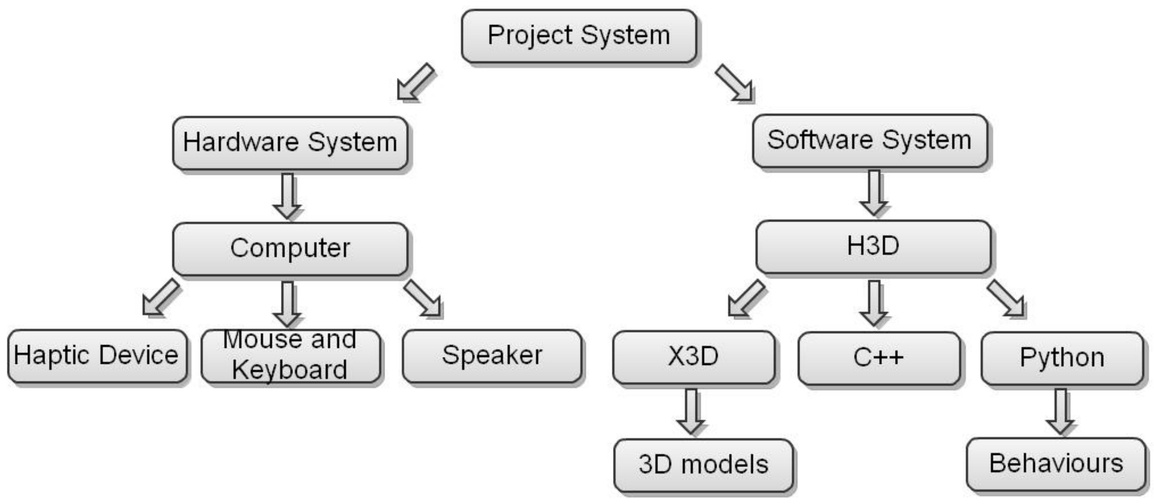


Figure 4: System architecture

3.2 Implementation of 3D virtual solar system

The implementation of the visual 3D model of the solar system is done using the programming language X3D. As we know, the solar system generally is made up with the Sun, the eight planets (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus and Neptune) a dwarf planet (Pluto), and the moons. Furthermore, each planet has its own self-rotation and orbit around the Sun, and simultaneously their moons have the same situation. In order to simulate the whole operation process of the solar system, the implementation of a dynamic 3D model has to handle events including time, position, angle of rotation and so on.

3.2.1 Implementing details of the revolution of planets

Let’s first discuss the revolution of each planet. In this section, two important events to be handled are time and position. Here, Earth is used as an example. First of all, when time is running, Earth keeps moving around the Sun following a fixed orbit. In this process, if the running time is divided into many small periods. Earth has a different position value within each period, and then following the running time, the position value of Earth is changed into different values which lead to a dynamic moving effect. It is clear that these position values should form the orbit data of Earth. Therefore, time and position need to be processed carefully for implementing the revolution of planets. The situation of revolution is shown in Figure 5.

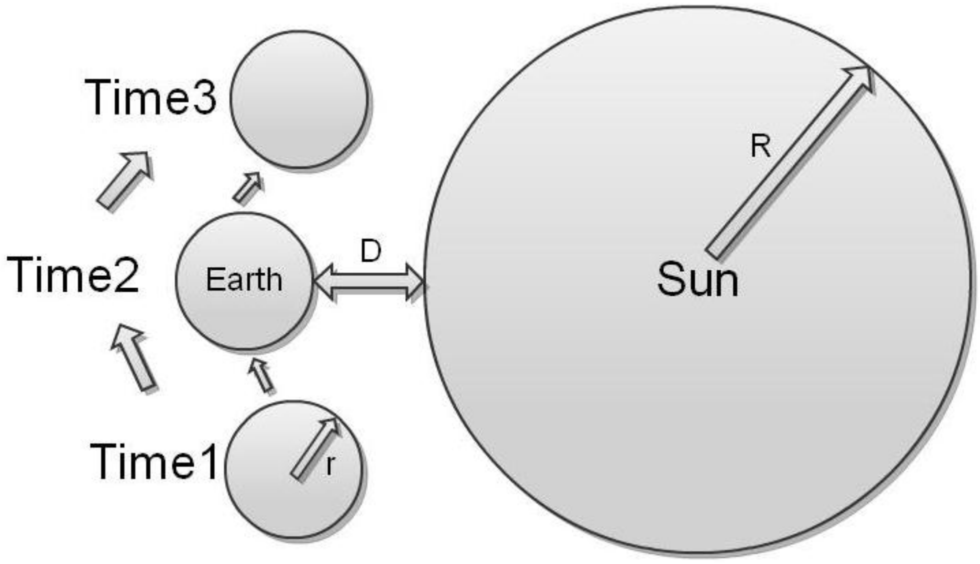


Figure 5: Principle of the revolution of a planet

Assuming the center point of the Sun is (0, 0, 0), the orbit data can be calculated using the equations (3.1) (3.2) and (3.3) which are derived from the formulas of circle [Brannan et al., 1999]:

$$x = (D + R + r) * \cos(\theta) \tag{3.1}$$

$$y = (D + R + r) * \sin(\theta) \tag{3.2}$$

$$z = 0 \ , \tag{3.3}$$

where x, y and z are coordinate values of points on the circular orbit, D is the distance between the Earth and the Sun, R is the radius of the Sun and r is the radius of Earth. Angle θ is a random value selected between 0 to 360 degrees, and choosing more angle values would get more orbit data and, thus, make the revolution of planet smoother.

After the orbit data have been calculated, a loop of time is needed to be created and divided into many small periods depending on the amount number of the orbit data, and each period must be corresponded to a corresponding orbit value. In this way, when Earth completes one revolution, the loop of time will be ended, and repeating this process makes Earth to keep moving following the orbit.

According to the above principle, the whole revolution process can be implemented in X3D. The code fragment below is an example for the revolution of Earth around the Sun.

In X3D:

```
<Transform DEF="EARTH">
<Shape>
<Appearance>
<ImageTexture url="C:\H3D\earth.jpg"/>//Adding texture picture to Earth
</Appearance>
<Sphere DEF="EARTHSHAPE" radius="0.015" solid="true"/> // Creating Earth
</Shape>
</Transform>
<TimeSensor DEF="TIME" loop="true" cycleInterval="60"/> // Creating time flow
<PositionInterpolator DEF="EARTHREVOLUTION"
  Key = "…" // Time periods from 0 to 1
  KeyValue = "…" /> // Orbit data
//Time flow is sent to position interpolator
<ROUTE fromNode="TIME" fromField="fraction_changed”
  toNode="EARTHREVOLUTION" toField="set_fraction">
//Orbit data from position interpolator are sent to position of Earth
<ROUTE fromNode="EARTHREVOLUTION" fromField="value_changed"
  toNode="EARTH" toField="translation"/>
```

Firstly, according to the specification of X3D [Brutzman and Daly, 2007], <Transform> is the main node which defines a coordinate system for its children and any visual 3D model is created through this node. Its parameters include the position of model (“translation”), the rotation of model (“rotation”), the scale of model (“scale”), and the rotation center of model (“center”). Moreover, “DEF” defines the name of each component used for positioning. Inside this node, <shape> is a child node which can be used to define the geometric shape of model. For example, <Sphere> is one type of geometric shapes, and its “radius” is 0.015, and setting ‘solid’ true means drawing one side (outside) of sphere. Besides this geometric shape, there are some other types of shapes such as <box> and <cylinder>. Then, adding <Image Texture> in <Appearance> which is a child of <shape> changes the texture image of model, and “url” is the address of this image.

To create a running time flow, <TimeSensor> must be used, and inside this node, setting “loop” true means that the time flow will keep repeating, and “cycleInterval” defines how much time for one loop of time flow. Then, <PositionInterpolator> is the node which can divide time flow into many periods and set their corresponding orbit values [X3D tutorial, 1999]. The field of “Key” inside <PositionInterpolator> node is used to set the time-divided values and its minimum and maximum values are 0 and 1, respectively. For example, if the values are “0, 0.5, 1”, it means that the incoming time flow (from <TimeSensor>) will be divided into two periods which can have three corresponding orbit values for them. “KeyValue” is the field to put the orbit data for planet, and the number of orbit data should be equal to the number of values in “Key”.

Now, all components should be connected with each other. Basically, <ROUTE> is the specific node for connecting and transmitting events between different components [X3D tutorial, 1999]. During the whole revolution process, there are two necessary connections:

1. Time flow must be sent to Position Interpolator: the field “fraction_changed” of time sensor can get the time flow, and then the field “set_fraction” of position interpolator is used to receive the time flow.
2. Orbit data obtaining from Position Interpolator needs to be sent to the position field of Earth: “value_changed” is used to get orbit value from Position Interpolator and then send to the “translation” of Earth.

Now, the revolution of Earth has been done, and other planets can use the similar X3D code to implement their revolution.

3.2.2 Implementing details of the revolution of moons

In addition to the nine planets, there are many moons running around their own planets in the solar system and they also make revolution to the Sun. This situation makes the implementation of the revolution of moons more complicated. A feasible method is to divide the implementation of revolution of moons into two steps which are shown in Figure 6 as well:

1. Implementing the revolution of moon around its planet.
2. Implementing the revolution of this planet system around the Sun.

In the first step, the orbit data of moon can be calculated using equations (3.4) (3.5) and (3.6), assuming the center of the planet is (0, 0, 0):

$$\mathrm{x} = (d + r + c) * \cos(\theta) \tag{3.4}$$

$$\mathrm{y} = (D + R + r) * \sin(\theta) \tag{3.5}$$

$$\mathrm{z} = 0 \ , \tag{3.6}$$

where c is the radius of moon and d is the distance between moon and the planet. Now we get the orbit data of moon, and then using the same method of planet revolution can implement that the moons move around their own planet.

The next step is to implement the movement of the whole planet system around the Sun. This can be done by providing double coordinate systems. Let’s use Earth system as an example: the transform nodes of the moon and Earth are independent, having independent positions, rotations, scales and other parameters. Now, a new transform node will be added and it includes both transform nodes of the moon and Earth. The position value of new transform node now will be the position of Earth and also the center point of revolution of the moon around Earth. Therefore, changing position value of the new transform node will lead to the movement of the whole Earth system around the Sun.

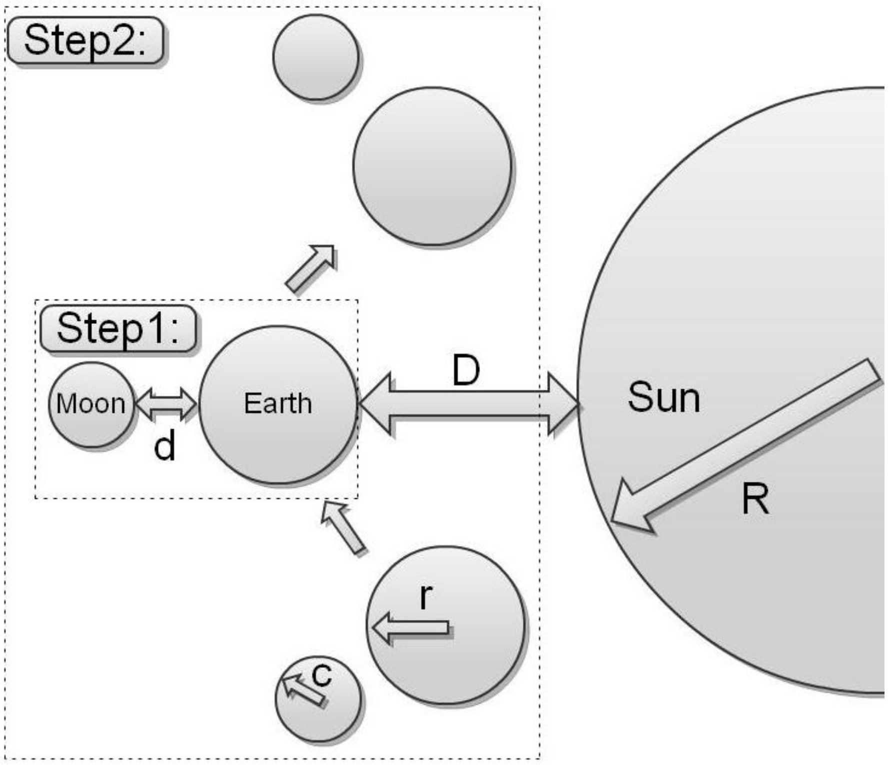


Figure 6: Principle of the revolution of the moon

Let’s see an example below:

In X3D:

```
<Transform DEF="EARTHSYSTEM" > // Double transform structure
<Transform DEF="EARTH">
  … // Setting for Earth shape, size and so on
</Transform>
<Transform DEF="MOON">
  … // Setting for moon shape, size and so on
</Transform>
</Transform>
// First step to implement the revolution of moon around Earth
<TimeSensor DEF="MOONEARTHTIME" loop="true" cycleInterval="10"/>
<PositionInterpolator DEF="MOONREVOLUTION"
  Key="…" // Time periods
  KeyValue="…"/> // Moon orbit data from equations (3.3) and (3.4)
<ROUTE fromNode="MOONEARTHTIME" fromField="fraction_changed"
  toNode="MOONREVOLUTION" toField="set_fraction"/>
<ROUTE fromNode="MOONREVOLUTION" fromField="value_changed"
  toNode="MOON" toField="translation"/>
// Second step to implement the revolution of Earth system around the Sun
<TimeSensor DEF="TIME" loop="true" cycleInterval="60"/>
<PositionInterpolator DEF="EARTHREVOLUTION"
```

```
  Key="…" // Time periods
  KeyValue="…"/>// Earth orbit data from equations (3.1) and (3.2)
//make time flow into different periods
<ROUTE fromNode="TIME" fromField="fraction_changed"
  toNode="EARTHREVOLUTION" toField="set_fraction"/>
//Send orbit data to Earth system
<ROUTE fromNode="EARTHREVOLUTION" fromField="value_changed"
  toNode="EARTHSYSTEM" toField="translation"/>
```

According to the above code fragment, the basic principle of revolution of moons is similar to the revolution of planets, and the only difference is that a new transform node replaces the old planet transform node which includes transform nodes of both planet and its moon. In this way, the moon will be running around the planet, and at the same time, the whole planet system will make revolution around the Sun. The revolution of other moons and planets can be implemented using the same method.

3.2.3 Implementing details of the rotation of all celestial bodies

While all celestial bodies are making revolution, they are also making self-rotation. Unlike the implementation of revolution which is done using time and position, the implementation of self-rotation relies on the processing of time and rotation angle. For example, based on the running time, a planet should change its angle to show the different faces. The situation is shown in Figure 7.

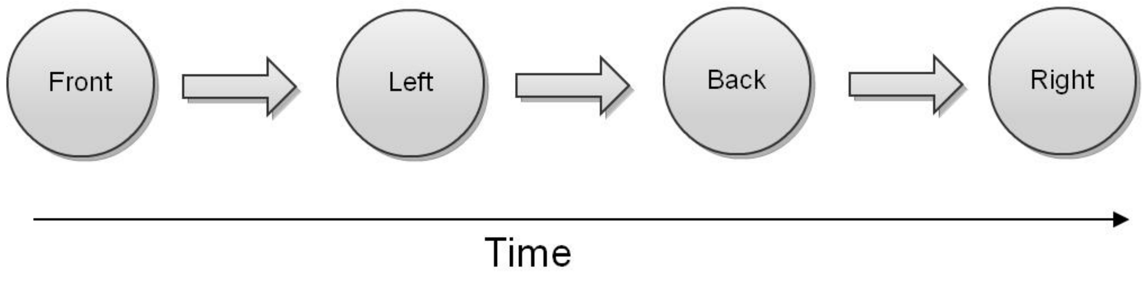


Figure 7: Principle of the rotation of a planet

In X3D, there is a special filed named <OrientationInterpolator> which can be employed to generate the different rotation values with the running time [X3D tutorial, 1999].

Here, the meaning of rotation value in 3D space [Arfken et al., 2012] should be introduced. For example, in rotation value (0, 0, 1, 3.1415), the first three numbers are the coordinate values of a point in 3D space, and the last value is the radian value of angle. So, this rotation value means the object will rotate about 3.1415 radians (about 180 degrees) along the direction of vector from (0, 0, 0) to (0, 0, 1). The details are also shown in Figure 8.

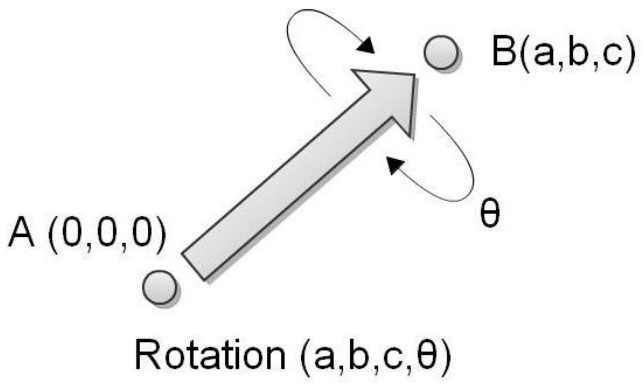


Figure 8: Rotation value in 3D space

Now, we can calculate the rotation values for a sphere. The first value for the front position should be (0, 0, 0, 0), the second value for the left position should be (0, 1, 0, 1.57) which means the sphere rotates about 90 degrees along Y-axis, the third value for the back position is (0, 1, 0, 3.14), the fourth value for the right position should be (0, 1, 0, 4.71) and the last one which is changed back to the front position is (0, 1, 0, 6.28). Therefore, during this process, the sphere would be rotated from 0 to 360 degrees along with Y-axis. Now, these rotation values and a running time can be used to implement the self-rotation function. Let’s see an example in X3D.

In X3D:

```
<Transform DEF="EARTH">
… // Setting for Earth shape, size and so on
</Transform>
<TimeSensor DEF="EARTHTIME" loop="true" cycleInterval="3"/>// New time flow
<OrientationInterpolator DEF="ORIENTATION"
  Key="0 0.25 0.5 0.75 1" // Time periods
  KeyValue="0 0 0 0, 0 1 0 1.57, 0 1 0 3.14, 0 1 0 4.71, 0 1 0 6.28"/> //Rotation values
<ROUTE fromNode="EARTHTIME" fromField="fraction_changed"
  toNode="ORIENTATION" toField="set_fraction"/>
<ROUTE fromNode="ORIENTATION" fromField="value_changed"
  toNode="EARTH" toField="rotation"/> // Send values to the rotation of planet
```

In <OrientationInterpolator>, “KeyValue” is the place to put the rotation values and “Key” is the place of time periods, which are very similar to <PositionInterpolator>. After all necessary nodes are created, a time flow will be sent to <OrientationInterpolator>, and then the rotation values will be sent from <OrientationInterpolator> to the “rotation” of Earth, so that the rotation angle of the Earth will be changed following the running time. Now, the function of self-rotation has been completed, and all planets can use the same method to implement this function.

3.2.4 Implementing details of other improvements

Until now, the basic structure of a dynamic 3D solar system has been implemented. However, the current view of the 3D model looks very dreary and unreal, and some graphical improvements are necessary for it. There are three major improvements which should be implemented are the following:

- Adding orbit for each planet and moon, in order to let user easily understand the revolution situations of every celestial body.
- Adding background image and asteroid belt, which will make the whole solar system more real.
- Adding light sources for each celestial body, which will make the solar system more vivid.

The first improvement can be completed by using the orbit point data which are calculated from equations (3.1), (3.2), (3.3), (3.4), (3.5), and (3.6), and connecting each point. This will form a circular orbit. In X3D, <IndexedLineSet> and its child node <Coordinate Point> can be used to connect different space points to generate a line [X3D tutorial, 1999]. Here is an example:

In X3D:

```
<Transform DEF="JUPITERORBIT">
<Shape>
<Appearance>
<Material emissiveColor="0.6 0.6 0.6"/>
</Appearance>
<IndexedLineSet coordIndex="…">
<Coordinate point="…"/>
</IndexedLineSet>
</Shape>
</Transform>
```

In <IndexedLineSet>, “coordIndex” is a list of numbers beginning from 0, and represents index numbers for the position values in <Coordinate point>. In detail, when the number of index numbers in “coordIndex” matches the number of position values in <Coordinate point>, each index number will attach one position value. Then, <IndexedLineSet> node will connect the points with different position values one by one according to their index numbers. In this way, the orbit of every object can be created, and moreover, the color of line can be modified by using “emisiveColor” in <Material>. All orbits can be created using this method.

In order to do the second improvement, we need <Background> and <Disk2D> [X3D tutorial, 1999], which are two basic fields in X3D. Firstly, the main 3D space created by H3D is a cube 3D space, and there are totally six faces which can be attached with a background image. Therefore, in <Background>, there are “fronturl”, “backurl”, “topurl”, “bottomurl”, “lefturl” and “righturl” which can be employed to set the background image. Using all will cost much computer resources, and thus, in this project, only “fronturl” is used.

In X3D:

```
<Background frontUrl="C:\H3D \image\starry.jpg"/>
```

Then, to create the asteroid belt, the simplest way is to create a 2D disk which is hollow in the center and then attach a good texture image to it. The model of 2D disk is shown in Figure 9.

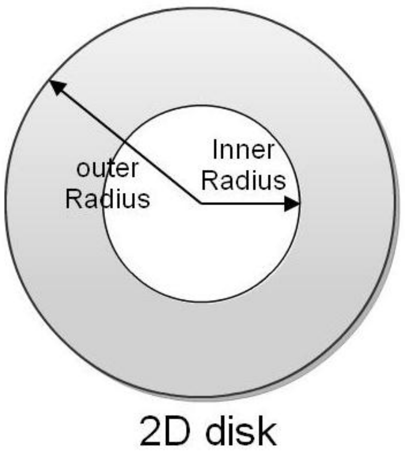


Figure 9: Two-dimensional disk

In X3D, <Disk2D> is the way to create this object [X3D tutorial, 1999]. Here is an example.

In X3D:

```
<Transform DEF="ASTEROIDROTATION">
<Shape>
<Appearance>
<ImageTexture url="C: \asteroid belt.jpg"/>
</Appearance>
<Disk2D innerRadius="0.6875" outerRadius="0.7625"/>
</Shape>
</Transform>
```

In <Disk2D>, “innerRadius” is the radius of hollow space, and “outerRadius” is the radius of disk. Furthermore, the texture image can be changed in <ImageTexture> and also since the real asteroid belt has self-rotation, the rotation function should be applied to it by using the same method as for planets.

The third improvement is to add light sources for each celestial body. Basically, the Sun should have red light which will affect all other planets and moons, and moreover, other planets and moons should have light and dark faces. In X3D, these effects can be done by placing <PointLight> in a suitable position in 3D place [X3D tutorial, 1999]. For example, the red light source should be placed in the center of the Sun:

In X3D:

```
<PointLight ambientIntensity= “0.1” color= “0.6 0 0” intensity= “1” on= “true”
                Attenuation= “0.5 0.5 0.6” location= “0 0 0”/>,
```

where “ambientIntensity” specifies the intensity of the ambient emission from the light, “color” defines the spectral color properties for both the direct and ambient light emission as an RGB value [Boughen, 2003], “intensity” specifies the brightness of the direct emission from the light, “On” indicates whether the light is enabled or disabled, “Attenuation” defines that the illumination falls off with distance from the light, and “location” is the location of the light source. Adjusting these values should follow the real situation of 3D model, and this adjustment has to be done lots of times to get the best performance. To add dark and light faces to planets and moons, the value of “color” needs to be changed into (1, 1, 1) and (-1, -1, -1) which represent white light and black light, respectively.

At last, because this is a large volume 3D model, the view point must be set for user to watch this model. The setting of view point can be done by adding <Viewpoint> in X3D [X3D tutorial, 1999].

In X3D:

```
<Viewpoint DEF="VP" position="0 0 1.5"/>
```

By using the above methods and codes, a complete 3D model of the solar system can be constructed. It includes the Sun, all planets, some moons and the asteroid belt. Due to the additional light sources and orbit lines, the scene is vivid and organized. A view of the solar system is shown in Figure 10.

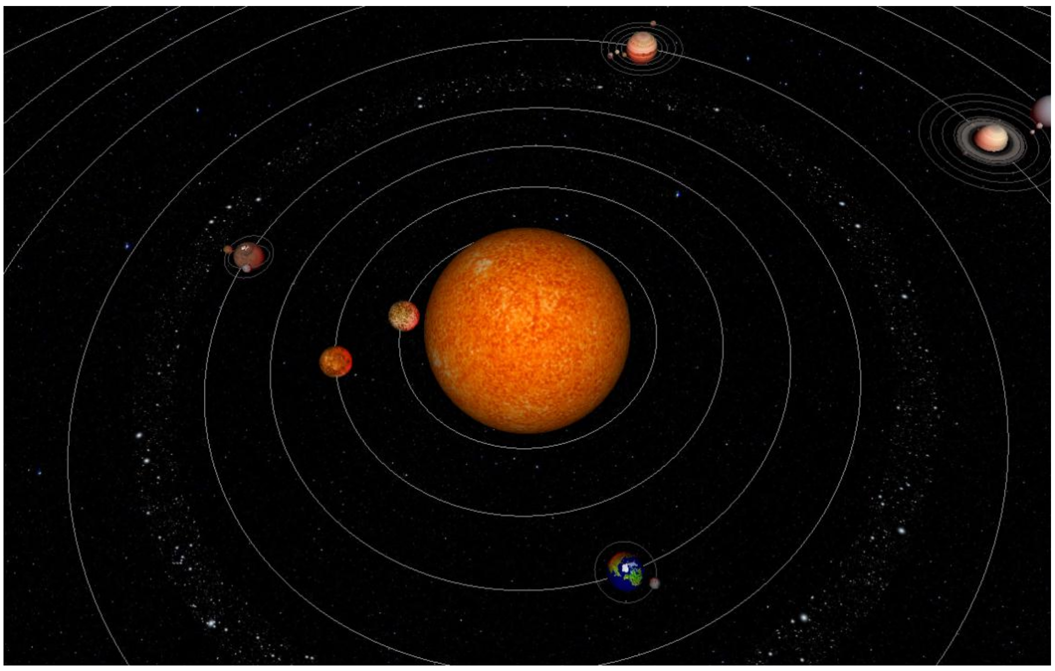


Figure 10: 3D model of the solar system

4. Implementation of the multimodal interactive system

The main purpose of this thesis is to create a multimodal interactive system based on a 3D model. Therefore, the specification of interactive system needs to be discussed and decided first in order to design a reasonable interactive system and demonstrate the benefits of haptic interaction. After that, the implementation details of the multimodal interactive system will be described in this chapter.

4.1 Specification of the multimodal interactive system

To begin with, based on the constructed 3D model of the solar system, the possible interactive elements for this model are the following:

- Size of objects: users will be interested in seeing a planet or a moon which can be changed in size.
- Speed of rotation and revolution of objects: the speed of operation of planets and moons can be modified by users.
- Orbit-changing function of objects: user can place each planet on a different orbit.
- Information-showing function of objects: the information about each planet and the Sun should be somehow shown to users as an introduction.
- Others: making the orbit line or moons invisible, closing the asteroid belt, opening background music and so on, mainly for users’ different requirements.

These five elements are the basic interactive elements of the model, and the implemented interactive system should contain these elements. Since this model is a large volume 3D model, there will be some troubles when a user uses a mouse or a haptic device to interact the virtual objects, and some special interactive methods should be implemented for dealing with this situation. In the project, the chosen methods to solve this problem are the following:

- Rotation function for 3D model: this function can change the spatial location of 3D model and virtual objects inside it without breaking their original structure and operation.
- Zoom in/out function: this is a common method to deal with the large volume of 3D model, which make the interaction with virtual objects inside model easier.

By adding these two interactive methods, the multimodal interactive system will be more flexible and can handle the interactive tasks within the large volume 3D model.

Now, since the most important issue is to implement these interactive elements using our hardware devices, the logical principle of devices should be known. A normal mouse has only two buttons (left and right) which have a basic logical event: when the button is pressed, the logical event is “True”, and when the button is released, the logical event is “False”. And a keyboard can be used only to send string-typed data to computer. The haptic device such as Omega has only one device button which has the similar logical event to the buttons of mouse. In this project, a mouse and a keyboard belong to one group of devices and a haptic device belongs to another group. The interactive methods and functions of these two groups should be independent but they can also work with each other in cases where some special functions cannot be implemented by one of the groups. In fact, in functionality, there is one big difference between these two groups of devices: mouses and keyboards can only be input medium of computer, but haptic devices can be both input and output medium since they not only transform force from user to computer but also generate force feedback to user. After knowing the basic principle of these devices, the next important task is to design the suitable functions for the discussed interactive elements using the above two groups of devices.

For this complicated multimodal interactive system, the related functions are too complex and overloaded if there is no any graphical user interface (GUI). Therefore, a visual user interface has been planned to add into the interactive system. And due to that the interaction will be taken place in 3D space, the traditional 2D graphical user interface is not suitable because it will take much space. Hence, a 3D user interface should be designed and implemented, and part of interactive functions could be integrated within this interface. The basic ideas about new GUI are the following:

- Cube UI with open and close buttons, which is a platform integrating with all buttons and slide bars.
- Touch buttons attached on the UI for functions of invisible orbit lines and moons, music opening and other functions, which handles “True” and “False” events.
- Slide bars attached on the UI for adjusting the size of planets and speed of rotation and revolution of planets, which deals with number-typed data.

The cube UI should be controlled by two groups of devices directly. For the mouse and keyboard group, a mouse can click the buttons and also draw the marker on the slider bar, and a keyboard can be used to change the orientation of the UI by sending some special string-typed data to computer. For another group, using a haptic device should have the same function for the UI. For example, if user employs a haptic device to touch a button, the button should be changed to a different state, and if it touches the line of slider bar, the marker will be moved on that place and its value will be changed as well. And for adjusting the orientation of the UI, there is a novel way using a haptic device to implement this function: using force as input. User can use a haptic device to push one side of cube, and the cube will be rotated following the direction of this force.

This method replaces the traditional method and directly uses hand movement of a user as input to adjust the rotation of a virtual cube UI.

Besides the functions integrated within the UI, there are two important functions which are planned to be implemented directly into the 3D model. They can be divided into information-showing functions and orbit-changing functions. Since there is only one physical button on a haptic device, there should be some buttons on the UI for opening these two functions. In fact, these two functions are mainly designed for haptic interaction. The reason is that a haptic device can touch virtual objects, which can be a novel interactive method to open the information images of the solar system objects or putting them into a different orbit. But in order to compare two interactive systems, a mouse and a keyboard somehow can be used to manage this function as well via clicking or sending string-typed data.

At last, implementing the rotation function and zoom in/out function for 3D model of the solar system has a special situation: since the haptic device selected in this project has only three degree of freedom which means that it cannot send or receive forces of orientation, the rotation function can only be performed somehow using a keyboard. The implementation of zoom in/out function is easier and more natural to be implemented using a 3D interactive tool. So, this function will be implemented by using a haptic device, instead of a mouse and a keyboard. Therefore, rotation function will be done by using the mouse and keyboard group and zoom in/out function will be implemented using a haptic device. The specification is in Table 1.

Table 1: Specification of the multimodal interactive system

Functions	Platforms	Devices
Size-changing function of planets	UI	Mouse/keyboard, Haptic device
Speed-changing of rotation and revolution of planets	UI	Mouse/keyboard, Haptic device
Orbit-changing function of planets	Solar system	Haptic device
Information-showing function of planets and the Sun	Solar system	Mouse/keyboard, Haptic device
Rotation functions separately for UI and the solar system	UI Solar system	Mouse/keyboard, Haptic device (UI)
Zoom in and zoom out function for UI and the solar system	UI Solar system	Haptic device
Other small functions	UI	Mouse/keyboard, Haptic device

It should be noticed that in the part of other small functions of Table 1, in addition to the invisible orbit line function, the invisible moons function, the invisible asteroid belt function and background music opening function, there will be some other function buttons which are used to solve the emerging technical problems. The details will be discussed in the following sections.

4.2 Implementation of 3D model of graphical user interface

In the implementation of interactive system, the 3D model of graphical user interface should be designed and implemented first. In the project, the shape of this user interface is planned as a three-dimensional cube and there are six surfaces on the cube totally. However, only four surfaces need to be used: front, back, left and right, and top and bottom surfaces will be removed in order to simplify the control of the UI. Since there are a lot of functions integrated within the UI, one of four surfaces should mainly contain buttons, and other three surfaces should be attached with slider bars which are used to change the size and speed of the planets. Therefore, in this UI, the front surface of the UI will be for placing press buttons, and other three surfaces are for slider bars.

Let’s first consider the method to create four surfaces which are used to make up the cube. Basically, the four surfaces should be the same in size but have different orientations, and front surface should have 90 degrees difference from the left and right surfaces and so on. The spatial structure of cube is shown in Figure 11.

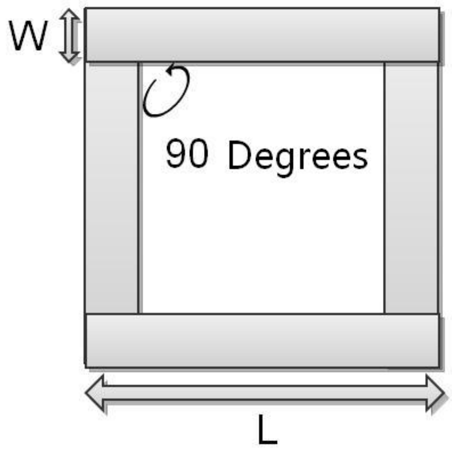


Figure 11: Top view of the UI structure

The four surfaces of cube can be created using four small cuboids, and because all UI components will be attached with the surfaces of cube, these small cuboids can be overlapping with each other without affecting the attached components.

In X3D, we can simply use <BOX> to create one cuboid [X3D tutorial, 1999], and there is an example below for the front surface:

In X3D:

```
<Transform translation="0 0 0.2" rotation= “0 0 0 0”>
<Shape>
```

```
<Appearance>
<ImageTexture url="C:\H3D \image\black.jpg"/>
<SmoothSurface stiffness="0.5"/> // Stiffness of model
</Appearance>
<Box DEF="Cube_front" size="0.45 0.4 0.05" solid="false"/>
</Shape>
</Transform>
```

In <Box>, the size “0.04 0.4 0.076” represents length, width and height of cuboid, and the spatial position of cuboid can be changed in “translation” of the parent node <Transform>. To change the orientation of cuboid, “rotation” can be used. For example, the left cuboid should have “rotation” value (0 1 0 1.57) which rotates the cuboid 90 degrees clockwise along Y-axis and the right cuboid should have “rotation” value (0 1 0 -1.57) which rotates the cuboid 90 degrees counterclockwise along Y-axis.

There is also an easier method to create this cube UI. We can just switch the width and length values (W, L) of front and back cuboids to get the left and right cuboids, and then place them into left and right positions to construct this cube. Furthermore, in order to let the haptic device touch the cube and also other 3D models including the objects in the solar system, the “stiffness” of <SmoothSurface> must be added in <Appearance> of <Shape> in all component <Transform> nodes [H3D manual, 2009]. Its value can be selected from 0 to 1. The bigger value the stiffer 3D models.

Now, when the cube has been constructed, the buttons and slider bars should be attached on their surfaces. In X3D, these two UI components can be created using <TouchButton> and <SliderBar> [H3D manual, 2009]. Consider an example button:

In X3D:

```
<Transform translation="0.19 0.224 0.22" >
<Frame desiredSize="0.05 0.05 0.01"> // Size of touch button area
<TouchButton DEF="INTERFACE_SWITCH" buttonMode="NORMAL" text="Button">
<FontStyle DEF="BUTTONFS" size="0.03" justify='"MIDDLE"'/> //Font style
<ImageTexture DEF="ButtonTexture" url="C:\H3D\image\button.png"/>
<GridInfo DEF="GI" sticky="W+E+N+S" padding="0.001 0.001"/>// Structure of button
</TouchButton>
</Frame>
</Transform>
```

Hence, to create a UI button, we need to first create a transform node which manages the position and orientation of button. The “desiredSize” of the parent node <Frame> is a necessary field for managing the size of button area [X3D tutorial, 1999], and its values represent the length, width and height of button area. Then, <TouchButton> is the main child node for touch button to be created, and the text on button can be changed via “text” and the type of button can be set by using “buttonMode”. There are three main types of buttons in H3D: normal, toggle-press and toggle-release. Normal button has the same function as the button of mouse, which sends “True” to computer when the button is pressed and “False” when the button is released, and also the state of button will be changed following this change. Toggle-press and toggle- release buttons are different from normal button: the state of button will be changed once when the button has been pressed or released, and if the state of button is “True”, it will send “True” value to computer all the time. When the button has been pressed or released again, the state of button will be changed into “False”, and then it will send “False” until the next button pressing or releasing.

After creating touch button, since the format of text on button and the structure of button area must be set, <FontStyle> and <GridInfo> must be included in the node of <TouchButton> [X3D tutorial, 1999]. In <FontStyle>, “size” specifies the size of text and “justify” defines the alignment of text, and also the color of text can be changed by “color” filed and the font of text can be modifies by “Family”. Then, <GridInfo> is used to manage the structure of button area. The basic spatial structure of button as well as other UI components has many layers including margins, padding, borders and content [Box model, 1999]. Its structure is shown in Figure 12.

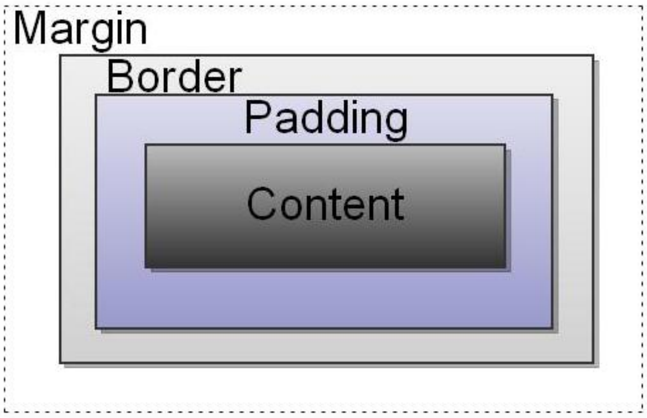


Figure 12: Box model of the UI component

In <GridInfo>, we can only set the size of padding for button and the sizes of other layers are default zero. The “padding” is the field to set the size of padding area of button. Furthermore, the “sticky” field is used to make the button stretch and fill all the space assigned to it, and its value “W+ E+ N+ S” means that its stretch directions are west, east, north and south. Now, the implementing details of touch button have been introduced.

Now, the implementation of slider bar is similar to touch button but more complex. Let’s see an example of slider bar.

In X3D:

```
<Transform rotation="0 1 0 3.1415926" translation="-0.004 0.16 -0.231" >
<Frame desiredSize="0.34 0.04 0.01">
<Label text="Mercury"> // First label
```

```
<ImageTexture DEF="ButtonPng" url="C:\H3D \image\button.png"/>
<GridInfo DEF="LABEL1" columnSpan="3" padding="0.001 0.01"
  Sticky="W+E+N+S"/>
</Label>
<SliderBar DEF="M_Size" valueRange="0.002 0.032" value="0.03"
  stepLength="0.001" markerColor="1 0 0"> // Slider bar
<GridInfo DEF="SBGI" columnSpan="9" column="3"
  padding="0.001 0.01" sticky="W+E+N+S"/>
</SliderBar>
<Label DEF="M_Size_label"> // Second label
<FontStyle USE="FS"/>
<ImageTexture USE="ButtonPng"/>
<GridInfo DEF="LABEL2" columnSpan="4" column="12"
  padding="0.009 0.01" sticky="W+E+N+S"/>
</Label>
</Frame>
</Transform>
```

In fact, the area of a slider bar in the UI includes three components which are two labels and one slider bar. The first label shows the name of slider bar and the second label shows the current value of slider bar, and this is a basic interaction between user and slider bar. In order to conveniently manage the area of slider bar, these three components should be placed in one <Frame>. Furthermore, their positions can be managed through different <GridInfo> nodes. In <GridInfo> node, beside the padding part, the management of content part as shown in Figure 12 can be set by using “column” and “columnSpan” [X3D tutorial, 1999]. For instance, for the first label, “column” is the beginning position of label and the width of area of label can be changed by “columnSpan”. Then the position of slider bar can be placed on the position after the pervious amount number of columns and so on. It is necessary to adjust the number of columns several times to get the best alignment of these components.

After the alignment of the three components, consider the setting of slider bar. In <SliderBar> [X3D tutorial, 1999], we can set its current value via “value” and the value range of slider bar can be set through “valueRange”, and “stepLength” is used to adjust the length of one step when user moves the marker on the slider bar. At last, the color of the marker can be modified by “markerColor”. Because there are many slider bars which are needed to be attached on the left, right and back surfaces of the UI, their transitions and orientations must be modified several times based on the spatial structure of the UI, and also all components could be covered by an image using <ImageTexture>.

Furthermore, the text of button and the text of the second label of slider bar should be changed depending on the state of button and slider bars, so that user can know the current state of button and the current value of slider bars. This is a small but important function in the UI, which will be introduced in detail in the next section.

Until now, the basic 3D model of user interface has been implemented, and Figure 13 shows a front view of its 3D model. In the next section, the basic functions related with the UI will begin to be introduced and their implementing methods will be described one by one.

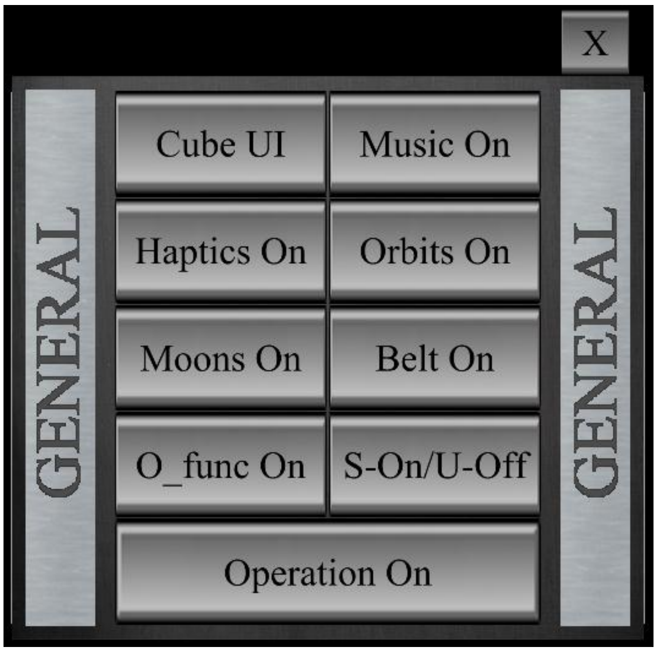


Figure 13: 3D model of graphical user interface

4.3 Implementation of functions of UI

Since the 3D cube UI has integrated with the size and speed changing functions for the solar system objects and also contains many touch buttons related to other small functions, it works as a central processing unit which is necessary to be implemented first. In this section, the implementing details of size-changing function are described, and the speed-changing function is introduced in the next section, and finally, other related functions of the UI are discussed.

4.3.1 Implementing details of size-changing function in UI

In Chapter 3, the implementing method of the model of each planet and moon has already been introduced. The planets and moons are created using <Sphere> in X3D and their sizes can be changed by modifying the values of “radius”. In order to change them through the values of slider bars of the UI, a connector must be built up to connect the value of radius and the value of slider bar. At the same time, the value of slider bar should be shown in the label to let user know the value of slider bar. The logical relationship is shown in Figure 14.

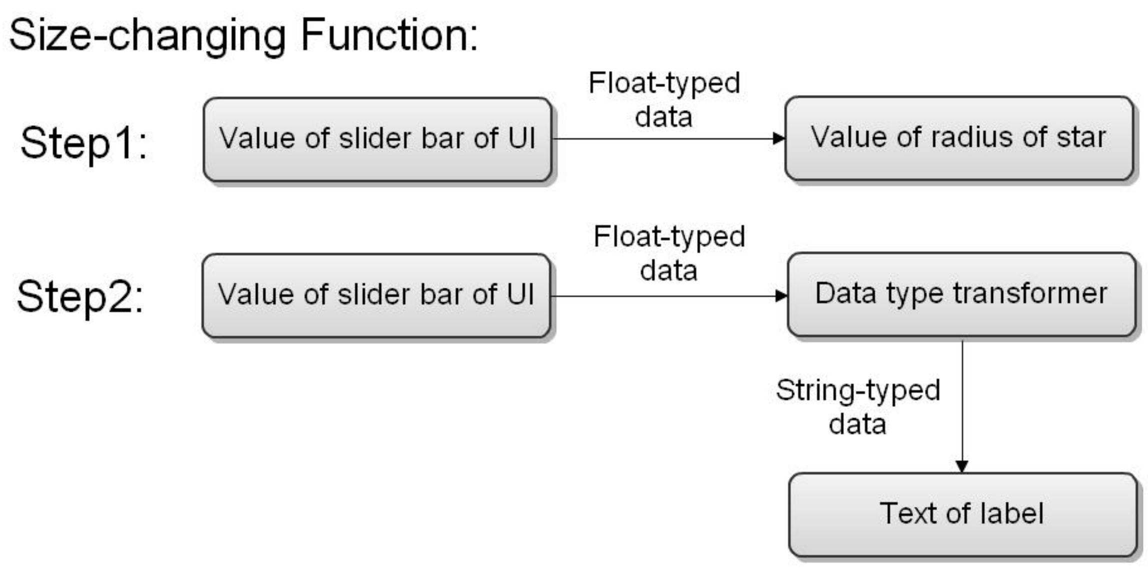


Figure 14: Logic diagram of size-changing function

Since the value of a slider bar of the UI has the same float-typed data as the value of “radius” in <Sphere> node, the data can be directly sent from value of a slider bar to value of radius of a planet. As the text of a label of the UI can receive string-typed data, the float-typed value must be transformed to string-typed data before sending to the text of label. The first step can easily done using <ROUTE> in X3D [X3D tutorial, 1999]. Here is an example:

In X3D:

```
<Sphere DEF="MERCURYSHAPE" solid="true" radius="" /> // Creating planet
…
<SliderBar DEF="M_Size" valueRange="0.002 0.032" value="0.03" stepLength="0.001"
  markerColor="1 0 0"/> //Creating slider bar
<ROUTE fromNode="M_Size" fromField="value" toNode="MERCURYSHAPE"
  toField="radius"/> // Connection between them
```

The <ROUTE> above works as a connector which send the “value” of <SliderBar> to the “radius” of <sphere> of Mercury. In this way, when user draws the marker of slider bar, the size of Mercury will be changed following the movement of marker.

In the second step, the data type transformer must be implemented in Python. The code fragment below is an example.

In Python:

```
class data( TypedField( MFString, SFFloat ) ):
    def update(self, event):
        Data = self.getValue ()
        Data= "%.02f" %Data
        return [Data]
m1=data()
```

The format of Python programming is very strict and the empty spaces decide the logical operation [Python tutorial, 1990]. In the above code fragment, “class” defines a logical event class, and “data” is the name of class. “TypedField” is a parameter of this class, which defines the types of input and output data [Jackson, 2011]. In this case, “SFFloat” is the type of input data, which means that it only contains single floating-point number [Prata, 2004], and “MFString” is the type of output data, which means that it contains multiple strings. “def update(self, event):” is used to update the incoming data, and “Data” is used to store the incoming data, which is an floating-point number with keeping the hundredth in the decimal. The program will return the “Data” as strings. At last, “m1” is an application of class “data”.

After creating the transformer, we can use the similar method of the first step to connect the value of slider bar, transformer and text of label.

In X3D:

```
// Input python file in X3D
<PythonScript DEF="UIPS" url="C:\H3D \Python\data.py"/>
…
//Adding slider bar
<SliderBar DEF="M_Size" valueRange="0.002 0.032" value="0.03" stepLength="0.001"
  markerColor="1 0 0"/>
<Label DEF="M_Size_label" text=""/> // Second label for showing value of slider bar
// Connect the value of size with the text of the label
<ROUTE fromNode="M_Size" fromField="value" toNode="UIPS" toField="m1"/>
<ROUTE fromNode="UIPS" fromField="m1" toNode="M_Size_label" toField="text"/>
```

The Python file must be input into X3D by using <PythonScript> and also give a name by using <DEF> [H3D manual, 2009]. And then, as the above codes shows, <ROUTE> is used to send float-typed data to data type transformer at first, and then send string-typed data from transformer to the text of label.

Now, we have completed the function of size-changing for Mercury, and other planets and moons can be handled by the same method. Python code can be reused through creating new applications, but one slider bar should be corresponding with one planet.

4.3.2 Implementing details of speed-changing function in UI

In the specification of the project, the speed-changing function should be divided into two functions: one is for the speed of self-rotation and another one is for the speed of revolution. But in the project, both of these two dynamic operations are implemented by employing a time flow, and thus, the speed for both situations can be changed only through adjusting the time flow. The implementing method of this function is similar to the one of size-changing function. The logical relationship is shown in Figure 15.

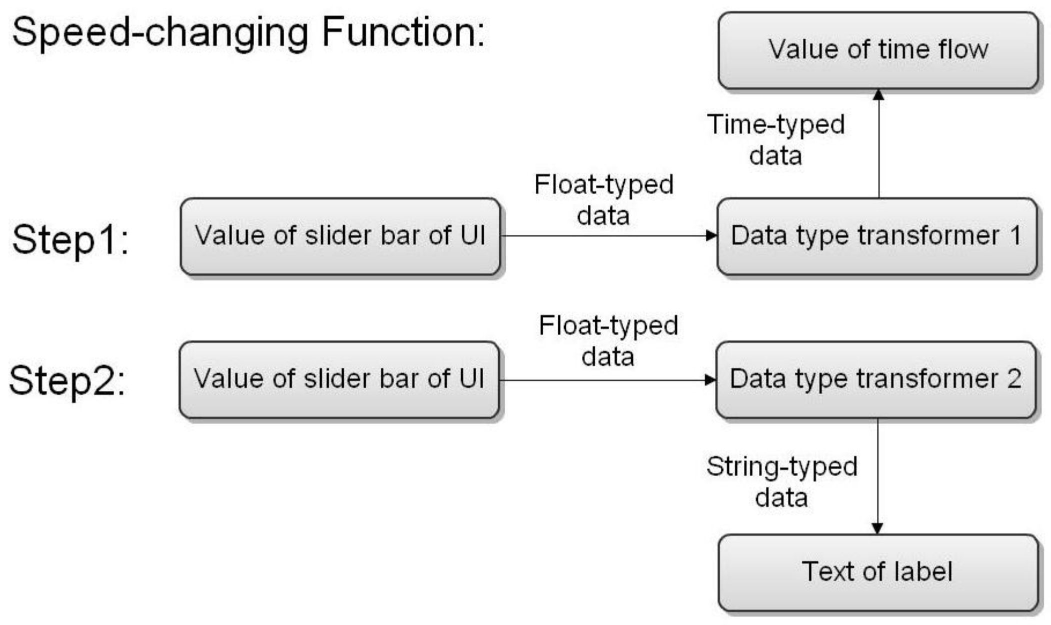


Figure 15: Logic diagram of speed-changing function

Since changing the text of label is the same as changing the size, we focus on the former. In the first step, since the value of time flow indicates the total length of time which is represented by “cycleInterval” in <TimeSensor>, this value is time-typed data. So, a new data type transformer is needed, and its implementing method is similar to the transformer for string-typed data. One example is below.

In Python:

```
class data2 (TypedField( SFTime, SFFloat ) ):
   def update(self, event):
      return event.getValue()
m2=data2()
```

In the above code fragment, the only difference is that the type of return value has been changed into “SFTime” which means that it contains a single time-typed data. After this, we can use <ROUTE> to connect the value of slider bar, transformer and value of time flow.

In X3D:

```
<PythonScript DEF="UIPS" url="C:\H3D \Python\data.py"/> // Transformer
<TimeSensor DEF="TIME" enabled="true" loop="true” cycleInterval=""/> // Time flow
…
<SliderBar DEF="M_Speed" valueRange="5 600" value="40" stepLength="40"
 markerColor="1 0 0"/> // Slider bar
<ROUTE fromNode="M_Speed" fromField="value" toNode="UIPS" toField="m2"/>
<ROUTE fromNode="UIPS" fromField="m2" toNode="TIME" toField="cycleInterval"/>
```

Now, this function has implemented. However, this function does not work if the time sensor is running. In other words, if the time sensor begins to work, the value of “cycleInterval” cannot be changed. Therefore, we have to stop the time sensor first, and then change the value of “cycleInterval”. This leads to the need of a logical button which can open and close the time sensor, and when time sensor stops to run, the relative planet will stop its self-rotation and revolution.

In the implementing process, the choice of type of button which is used to stop and restart time sensor is clear (“toggle-release” or “toggle-press”). The “normal” type of button cannot be used, since if user releases the button, the state of button will be immediately changed from “True” into “False”. Hence, user has to continue to hold the stop function button and adjust slider bar at the same time, which is impossible for user. Furthermore, for the convenience of user, the button should control all time sensors which are used to generate time flow for self-rotation and revolution of planets and moons, and also should have the changing text on button for explaining the state of button to user. Here is an example of stop and restart button.

In X3D:

```
<TimeSensor DEF="TIME" enabled="true" loop="true" cycleInterval="30"/>
<TimeSensor DEF="TIME2" enabled="true" loop="true" cycleInterval="30"/>
…// Different time sensors for rotation and revolution function
<TouchButton DEF="SWITCH" state="true" buttonMode="TOGGLE_PRESS">
// Stop and restart function
<ROUTE fromNode="SWITCH" fromField="state" toNode="TIME" toField="enabled"/>
<ROUTE fromNode="SWITCH" fromField="state" toNode="TIME2" toField=" enabled"/>
```

The data type for “state” of button and “enabled” of time sensor is Boolean. The value of “state” can be directly sent to the “enabled” of time sensor which controls weather the time sensor is on or off.

Now, the change of button state should imply change in button text. This can be done by implementing a new data type transformer. The basic logic is that when the button sends a Boolean-typed value to the transformer, the transformer will send a special string-typed data to the text of button. It should be noticed that this function is only suitable for a toggle-press or toggle-release type button. The Python and X3D codes for this function are here.

In Python:

```
class data (TypedField (MFString, SFBool)):
    def update(self, event):
        if event.getValue () == True:
            return ["Operation On"]
        elif event.getValue () == False:
            return ["Operation Off"]
newtext=data()
```

In X3D:

```
<PythonScript DEF="UIPS" url="C:\H3D \Python\data.py"/> // adding Python file in X3D
<ROUTE fromNode="SWITCH" fromField="state" toNode="UIPS" toField="newtext"/>
<ROUTE fromNode="UIPS" fromField="newtext" toNode="SWITCH" toField="text"/>
```

According to the above codes, if the state of button is true, the text of button will be “Operation On”, and if the state of button is false, the text of button will be “Operation Off”.

The whole speed-changing function has now been implemented, and there is one thing to be noticed: the value of “cycleInterval” refers to the length of time flow, and if user increases the value, the time to finish one loop of revolution or self-rotation becomes longer and, thus, the speed of rotation or revolution of planet will be decreased.

4.3.3 Implementing details of other small functions in UI

Besides the two above functions, there are some small functions implemented within the UI, (making the lines of orbits invisible, making the moons and asteroid belt invisible, and starting background music). These three functions are used to meet the requirements for different users. Since some users will prefer the traditional 2D model UI, there is additional function which can switch the UI between cube structure and flat structure. At last, there should be a function which can open and close this UI in order to meet the basic requirement of user. Let’s first consider the implementation of the first three functions.

Firstly by making the virtual objects invisible we can use “graphicsOn” in <ToggleGroup> [H3D manual, 2009]. <ToggleGroup> is a special parent node which can contain many other parent nodes such as <Transform>. By giving Boolean-typed value to its field “graphicsOn”, it can open and close the graphics of all nodes inside <ToggleGroup>. Besides “graphicsOn”, there is “hapticsOn” which enables or disables the function of the haptic device to touch the objects of nodes inside the <ToggleGroup>. Furthermore, to implement the invisible function, a toggle-press or toggle-release button is needed, and this button will control the visibility of the virtual objects, and then a connection should be built up between this button and the toggle group.

In X3D:

```
<ToggleGroup DEF="TG" graphicsOn="True" haptciOn="True"> // Toggle group
…// All nodes which want to be controlled
</ToggleGroup>
<TouchButton DEF="BUTTON" state="true" buttonMode="TOGGLE_PRESS">
<ROUTE fromNode="BUTTON" fromField="state" toNode="TG" toField="graphicsOn"/>
```

In the above code fragment, the toggle group contains all nodes which need to be invisible, and <ROUTE> is used to connect a new UI button with the toggle group.

Therefore, the function of making virtual objects invisible can be done using this method.

<Sound> can be employed to input music [X3D tutorial, 1999]. However, there is no field in <Sound> to enable or disable this node. The only feasible method to implement this function is to put the <Sound> node into another X3D file, and at each time that user wants to listen to music, a button of the UI will be used to input the new X3D file into the original file, and vice versa.

In new X3D:

```
<Sound maxBack="1000" maxFront="1000" DEF="BACKSOUND">
<AudioClip loop="true" url="C:\H3D \music\backmusic.wav"/>
</Sound>
```

In original X3D:

```
<Inline DEF="MUSIC" load="false" url="C:\H3D\ BackgroundMusic.x3d"/>
<TouchButton DEF="BUTTON2" state="true" buttonMode="TOGGLE_PRESS">
<ROUTE fromNode="BUTTON2" fromField="state" toNode="MUSIC" toField="load"/>,
```

where "maxBack" and "maxFront" in <Sound> node define the maximum back and front values of sound field which is shaped as an ellipsoid, <Inline> is used to input another outside file into the original X3D file, and "load" is a field of <Inline> which is used as a switch to control the loading of this new file. At last, <ROUTE> connects "state" of button with the "load" of <Inline>.

In order to let user to have an experience of using a normal 2D model of the UI, a new function of flat UI has been implemented. Basically, this function is implemented by adjusting rotation value of four cuboids which are made up the UI. For example, rotating the left cuboid 90 degrees counterclockwise along Y-axis will make it become another front cuboid, and it stands just on the left-side of the original front cuboid. Figure 16 shows the details method to implement this function.

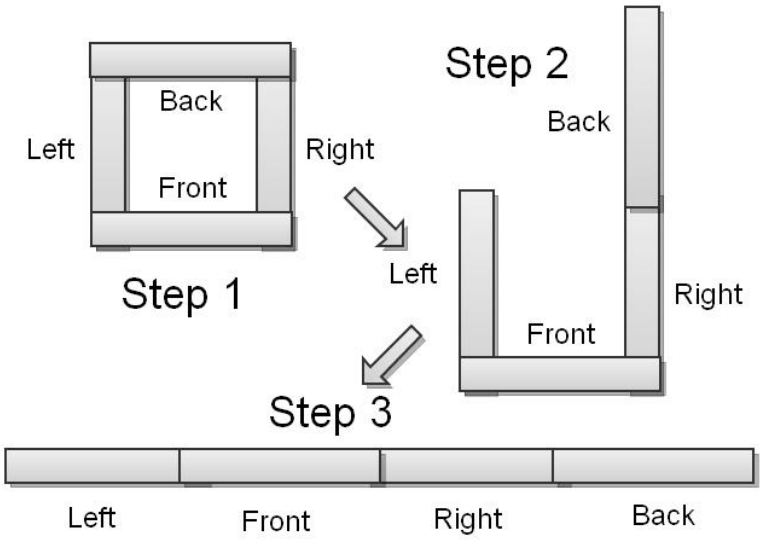


Figure 16: Implementing method of the flat UI

Based on Figure 16, the implementation method can be divided into three steps:

1. In the first step, the four cuboids with its components will be organized into four different groups: left group, right group, back group and front group by putting the components of each group into the different <Transform> nodes.
2. In step two, the components of back group will be rotated clockwise along the Y-axis and the rotation center is the center point of overlapping area of right cuboid and back cuboid.
3. In step three, right group and back group will be firstly put in the same group named right-back group, which means that two <Transform> nodes will be both put into another <Transform> node.

In this way, the components of left group will be rotated 90 degrees counterclockwise along the Y-axis and the rotation center is the center point of overlapping area of left cuboid and front cuboid, and the components of right-back group will be rotated 90 degrees clockwise along the Y-axis and the rotation center is the center point of overlapping area of front cuboid and right cuboid. Front group will not be rotated.

Now, for grouping these four cuboids and their components, the multiple transform structure must be used in X3D. The code fragment below is an example.

In X3D:

```
<Transform center="-0.225 0 0.225" DEF="LeftBox">
… // including left cuboid and all attached UI components
</Transform >
<Transform center="0.225 0 0.225" DEF="RightBackBox">
<Transform DEF="RightBOX">
… // including right cuboid and all attached UI components
</Transform >
<Transform center="0.225 0 -0.225" DEF="BackBox">
… // including back cuboid and all attached UI components
</Transform >
</Transform >
```

In the above code fragment, “center” in <Transform> is the field which is used to set the center of rotation, and its value should be the coordinate value of a point in 3D space and the default value is (0, 0, 0) [X3D tutorial, 1999]. After grouping the cuboids, a new toggle-press or toggle-release touch button must be added, which is used to control this function, and also three rotation value generators must be implemented in Python for generating different rotation values. Let’s first consider its logic diagram which is shown in Figure 17.

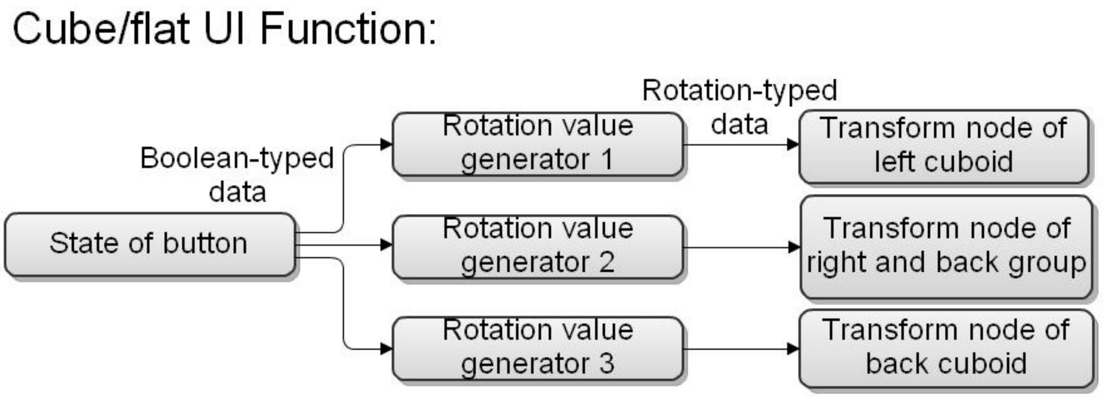

Cube/flat UI Function:
Boolean-typed data
Rotation-typed data
State of button
Rotation value generator 1
Rotation value generator 2
Rotation value generator 3
Transform node of left cuboid
Transform node of right and back group
Transform node of back cuboid

Figure 17: Logic diagram of the cube/flat UI function

Figure 17 shows that the button will send the Boolean-typed value to three different generators at the same time, and each generator will immediately generate rotation-typed value to the transform nodes based the input Boolean-typed value. In this way, the cube UI will become flat UI. The three generators have to be implemented first. The code fragment below is an example of generator for the rotation of left cuboid.

In Python:

```
class generator ( TypedField( SFRotation, SFBool ) ):
   def update ( self, event ):
      if event.getValue() == True:
            self.changed = Rotation (0, 0, 0, 0)
      else: self.changed = Rotation (0, 1, 0, -1.57)
      return self.changed
ortationvalue = generator ()
```

According to the above code fragment, when the program receives “True”, it will send the rotation (0, 0, 0, 0) which means it will not be rotated, and if the program receives “False”, it will send rotation (0, 1, 0, -1.57) which means that it will be rotated counterclockwise about 90 degrees along the Y-axis. This generator is used for left cuboid. The other two generators with different output rotation values can be implemented using the same method. At last, we should connect all components following the logic diagram.

In X3D:

```
<PythonScript DEF="UIPS" url="C:\H3D \Python\data.py"/>
<TouchButton DEF="ROTATION_B" state="true" buttonMode="TOGGLE_PRESS"/>
// connect button with generators
<ROUTE fromNode="ROTATION_B" fromField="state" toNode="UIPS"
  toField="ortationvalue">
…// connecting with other two generators
<ROUTE fromNode="UIPS" fromField="ortationvalue" toNode="RightBackBox"
  toField="rotation"> // sending rotation-typed values to different nodes
```

…// **other two transform nodes connections**

It should be noticed that the rotation value from generator 2 will be sent to the "rotation" of <Transform> node of right-back group, and the rotation value from generator 3 will be only sent to the "rotation" of <Transform> node of back group. In H3D, the tasks of children node will be processed firstly, and then H3D will process the parent node, so that the steps of implementation are just following the steps shown in Figure 16.

Now, the cube/flat UI function has been successfully implemented and user can switch between two different types of the UI and choose the suitable one. The next subsection will discuss another important function which is used for opening and closing the UI.

4.3.4 Implementing details of open and close function of UI

Since the UI cannot be always shown in the scene, an open function button and another close function button must be implemented for the UI. To implement this function, a feasible and easy method is to utilize <ToggleGroup> which can make virtual objects invisible and untouched. The method is shown in Figure 18.

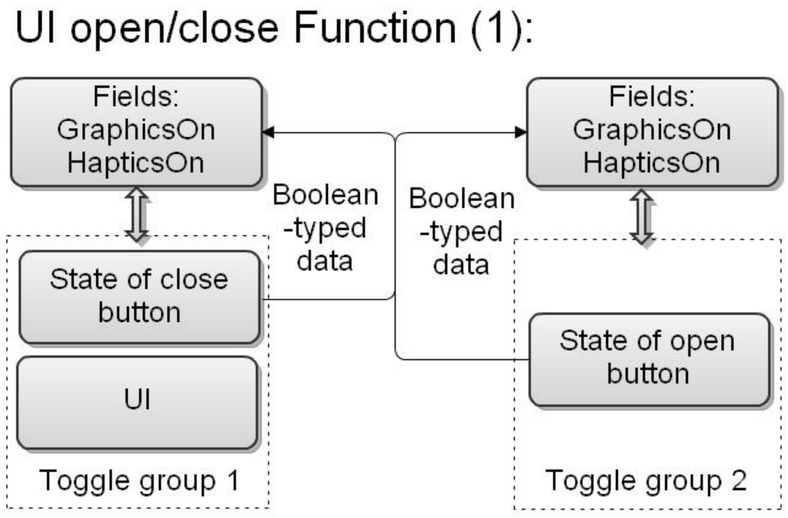


Figure 18: Logic diagram of the UI open/close function

Firstly, the UI with close button should be in the same toggle group, and open button should be put in another toggle group. Then, the close button can trigger the event of the first toggle group to make the UI and close button itself invisible and untouched. Simultaneously it will trigger the event of the second toggle group to make the open button visible and can be touched. And the open button has the opposite function, which will make itself invisible and untouched, and lead to the UI and close button visible and can be touched. However, there are technical problems emerging in this implementing process. First of all, it is clear that both close and open buttons must control the fields of both toggle groups at the same time, and the state of one button should be changed back if another button is used. This situation leads to a technical problem due to the limits of available types of button in X3D. Since there are only three types of buttons available (except radio type buttons which are useless in the project), and the logic of both toggle-release and toggle-press type buttons is that the state of buttons will be changed only if user press or release the button, it follows that one clicking action will only lead to one change of state of button. The state of normal type button will be "True" while the button is pressing, and will be changed into "False" immediately while the button is released. These three types of buttons clearly cannot be directly used in the above application because the needed button is that it can only steadily send a Boolean-typed data "True" after a completed clicking action including pressing and releasing, and the Boolean-typed data will not be changed to "False" until another button is clicked. This situation causes a complicated programming process. Although the UI can be invisible and untouched, it still exists in the scene, which will bring unexpected problems when user interacts with the solar system using a haptic device. In order to solve these problems, an improved method has been provided.

The new method combines the feature of view point with the function of <Togglegroup>. The improved method is shown in Figure 19.

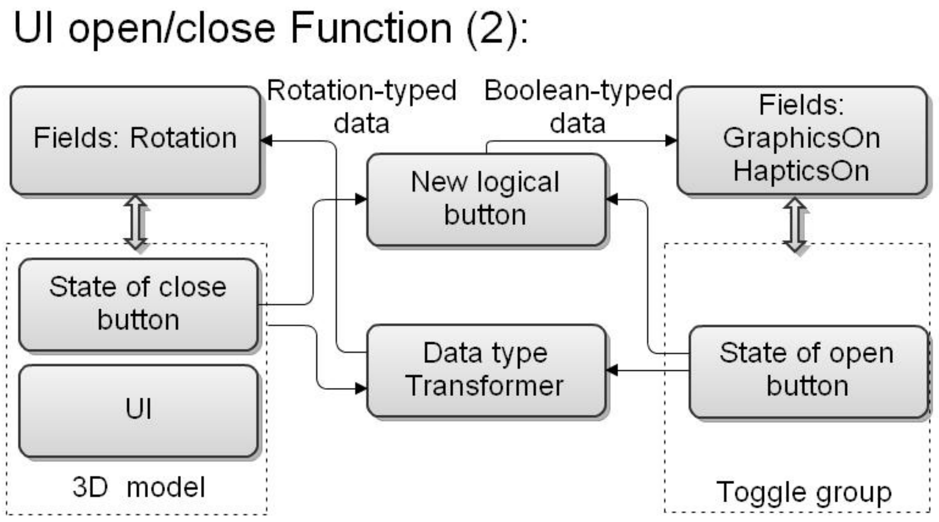


Figure 19: Logic diagram of the UI open/close function of the new method

The basic principle is that since the view point of user can be set by using <Viewpoint>, to make the UI invisible and untouched, the simplest way is to rotate the whole UI and close button away from the field of view point, and the open button will rotate the UI and close button back. Furthermore, programming a new logical button in Python for controlling the toggle group of open button is necessary. In short, the open button can rotate the UI and close button back to the field of view point and also it will become invisible and untouched. And close button will rotate the UI and itself away from the field of view point and make open button visible and can be touched.

Therefore, the improved method needs two new components which are the data type transformer for receiving Boolean-typed data and sending rotation-typed data, and a new logical virtual button for sending a stable Boolean-typed data once a logical event “True” is received.

Let’s first discuss the implementation of new logical button. To create this virtual button, the type of open and close buttons of the UI must be selected first. The best choice is normal type which will send “True” if button is pressed. Actually, this button must be divided into two independent logical buttons, and one button is only for close button, which will continue to send “True” when logical event “True” is received for making open button visible, and another button is used for open button, which will continue to send “False” when logical event “True” is received for making open button invisible and cannot be touched. Here is a simple example of logical button for close button.

In Python:

```
class closebutton( AutoUpdate( SFBool ) ):
  def update ( self, event ):
    if event.getValue():
       return True
add_closeswitch= closebutton ()
```

In the above code fragment, “AutoUpdate(SFBool)” defines that this class can receive Boolean-typed data and also send Boolean-typed data [Jackson, 2011]. If the program receives a Boolean-typed value “True”, then it will continue to return “True”. This logical button is for the close button. For the open button, the code is similar to the one of the close button, which will continue to return a value “False”.

Now, to create the data type transformer, the rotation value must be calculated first. In our case, the best solution is that the UI rotates 180 degrees around the view point, and in this way, the UI definitely will be moved out from the view of user. To do that, the center of rotation needs to be set first by using the “center” field of <Transform> and its value should be the value of the view point.

In X3D:

```
<Transform DEF="UISYSTEM_POSITION" center="0 0 1.5">
… //UI components
</Transform>
```

Then, we need to implement the transformer in Python. The transformer should include two independent classes for close and open buttons. Below is an example for the close button.

In Python:

```
class position( TypedField( SFRotation, SFBool ) ):
  def update( self, event):
        if event.getValue():
            self.changed = Rotation( 0,1,0, 3.1415926 )
        return self.changed
closebutton_rotation= position()
```

The rotation (0, 1, 0, 3.1415926) means that it will be rotated 180 degrees along Y-axis. This class is used for close button, and for open button, the rotation value of class should be changed to (0, 0, 0, 0). At last, following the logic diagram, we need to connect all components together using <ROUTE>.

In X3D:

```
 <PythonScript DEF="UIPS" url="C:\H3D \Python\data.py"/>
<TouchButton DEF="C_BUTTON" buttonMode="NORMAL"/> // close button
<TouchButton DEF="O_BUTTON" buttonMode="NORMAL"/> // open button
<ToggleGroup DEF=" TG">
… // Open button
</ToggleGroup>
// making open button visible and can be touched
<ROUTE fromNode="C_BUTTON" fromField="state" toNode="UIPS"
  toField=" add_closeswitch"/>
<ROUTE fromNode="UIPS" fromField="add_closeswitch" toNode="TG"
  toField=" GraphicsOn"/>
<ROUTE fromNode="UIPS" fromField="add_closeswitch" toNode="TG"
  toField=" HapticsOn"/>
// making open button invisible and untouched
<ROUTE fromNode="O_BUTTON" fromField="state" toNode="UIPS"
  toField=" add_openwitch"/>
<ROUTE fromNode="UIPS" fromField="add_openwitch" toNode="TG"
  toField=" GraphicsOn"/>
<ROUTE fromNode="UIPS" fromField="add_openwitch" toNode="TG"
  toField=" HapticsOn"/>
//rotating UI and close button away from view of user
<ROUTE fromNode="C_BUTTON" fromField="state" toNode="UIPS"
  toField="closebutton_rotation"/>
<ROUTE fromNode="UIPS" fromField="closebutton_rotation"
  toNode=" UISYSTEM_POSITION" toField="rotation"/>
//rotating UI and close button back
<ROUTE fromNode="O_BUTTON" fromField="state" toNode="UIPS"
  toField="openbutton_rotation"/>
<ROUTE fromNode="UIPS" fromField="openbutton_rotation"
  toNode=" UISYSTEM_POSITION" toField="rotation"/>
```

Until now, all functions of the UI have been introduced and their implementation details have been described. The next section will begin to describe the first interactive system using a haptic device.

4.4 Implementation of interactive system using a haptic device

The implementation of interactive system which employs a haptic device as the interactive tool is the most important part in this project. In this part, there are four important functions to be implemented:

- A special function for rotating the 3D model of the UI using a haptic device.
- Zoom in/out function for both the solar system and the UI using a haptic device.
- Information-showing function for planets and the Sun using a haptic device
- Orbit-changing function for all planets using a haptic device.

The first two functions are mainly used to control the UI and the solar system, in order to overcome the problems of large volume 3D model. The last two functions are to directly interact with the solar system. These four functions could be the most important applications in this project for haptic interaction. Let’s first consider the rotation function for the 3D model of the UI.

4.4.1 Implementing details of UI rotation

In the previous sections, the structure of the UI has been introduced. It has four different surfaces, and each surface has different UI components for different functions. Basically, all of these components can be touched by a haptic device. However, due to the spatial structure of the UI, user has to rotate this UI to different surface and then choose the component which he needs. To implement this rotating function in the situation of using a haptic device, a special implementing method is used in this project, which employs push force of user’s hand as the input signal to rotate the UI. In fact, this implementing method is originated from the real situation of a rotating cube: when human uses his hand to push the edge of a real cube, and if this cube is fixed on the ground but can be rotated randomly, it will be rotated depending on the direction and power of the incoming force. This situation is shown in Figure 20.

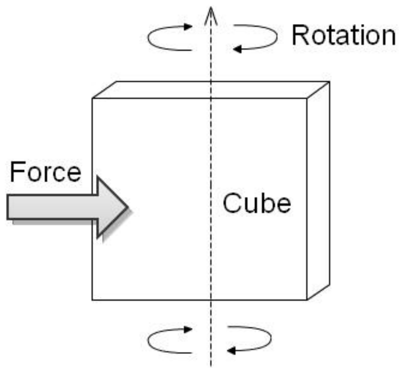


Figure 20: Rotation of a real cube

Based on this principle, there are two things needed to be considered: the first one is the direction of force and the second one is the magnitude of force. The direction of force will lead to the different direction of rotation of cube, and the magnitude of force affects the speed of rotation. In order to simulate the rotation of real cube for the UI, both need to be handled.

Firstly, the direction of rotation can be implemented directly by creating the force sensor areas on the edges of the UI cube. For example, two force sensor areas are created separately on the left and right side of one surface of the UI as white areas in Figure 13. When the left sensor area detects force, the cube will be rotated clockwise, and if the right sensor area detects force, the cube will be rotated counterclockwise. Secondly, the changing speed of rotation can be implemented by logically connecting the magnitude of force with the angle of rotation, and when user uses more power to push the UI, the UI will be rotated to a bigger angle, which will make user feel that the speed of rotation has been increased. The logic diagram of this function is shown in Figure 21.

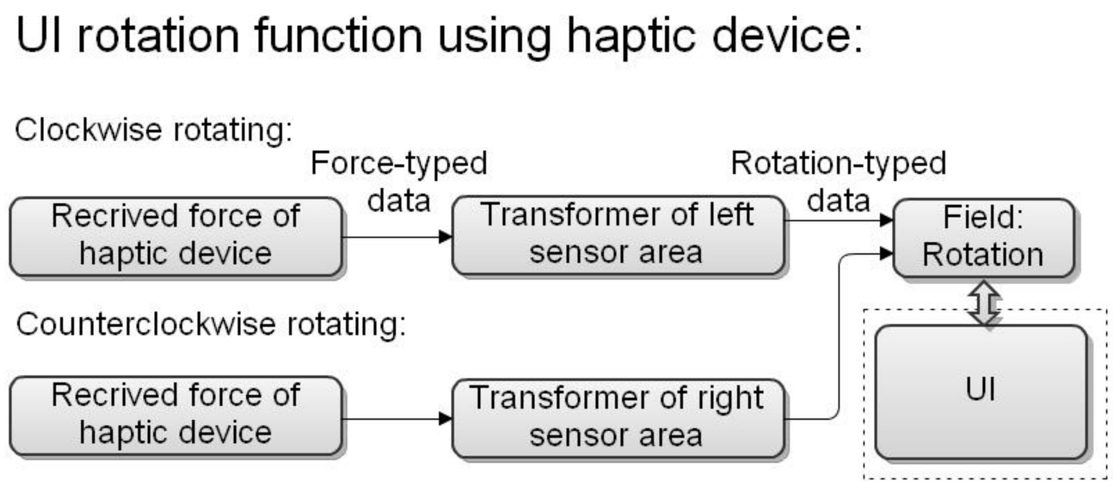

UI rotation function using haptic device:
Clockwise rotating:
Force-typed data
Rotation-typed data
Recrived force of haptic device
Transformer of left sensor area
Field: Rotation
Counterclockwise rotating:
Recrived force of haptic device
Transformer of right sensor area
UI

Figure 21: Logic diagram of the UI rotation function using a haptic device

In this function, the most important task is to implement the transformers for both sensor areas, and the transformers can receive force-typed data and calculate the biggest magnitude of force along the direction which is perpendicular to the surface of the UI, and a suitable mathematical logical equation needs to be found out for generating the rotation-typed value. Since the force-typed data from the haptic device is three-dimensional value which includes three victors of force respectively along the X-axis, Y-axis and Z-axis, the easiest way to calculate the magnitude of force is to choose the largest value among these three values. Let’s assume that the force applied to the left sensor area is (force_x, force_y, force_z), then the biggest magnitude of force should be

$$\text{Force} = \max(\text{abs}(\text{force\_x}), \text{abs}(\text{force\_y}), \text{abs}(\text{force\_z})) , \qquad (4.1)$$

where max () in Python [Jackson, 2011] means the choice of the biggest value among these three values, and abs () is used to calculate the absolution of three vectors.

Once the magnitude of force is obtained, the logical equation which connects force with rotation value has to be built up. In our case, the best method is to use the magnitude of force directly as the radian of rotation, and therefore, the equation is

$$\text{Rotation\_new} = \text{Rotation}(0, 1, 0, \text{Force}/n) * \text{Rotation\_old}\,, \qquad (4.2)$$

where Rotation_new is the new rotation value of the cube UI, and Rotation_old is the current rotation value of the cube UI. Rotation (0, 1, 0, Force/n) means the UI will be rotated (Force/n) radian along the Y-axis, and the value of n will be selected depending on the experimental situation of the UI rotation: the bigger value of n the bigger the force to rotate the UI. Now, we need to program this in Python.

In Python:

```
class Force(TypedField(SFRotation,(SFRotation,MFVec3f))):
    def update(self,event):
        try:
            routes_in = self.getRoutesIn()
            Rotation_changed = routes_in[0].getValue()
            force= routes_in[1].getValue()
        except:
            return Rotation_changed
        if(len(force)>0):
            force = max(abs(force[0].z), abs(force[0].x), abs(force[0].y))
        else:
            force = 0
        if(force > 0):
            Rotation_changed = Rotation (0, 1, 0, force/20)* Rotation_changed
        return Rotation_changed
left_sensorarea = Force ()
```

In the above code fragment, the class “Force” has two different values as input: rotation-typed value which is the current rotation value of the cube UI, and three-dimensional vectors which are the force vectors. And it can output a rotation-typed value, the new rotation value for the cube UI. Then “self.getRoutesIn()” is used to identify the two input values and input them into the correct data storages of program [Jackson, 2011]. The following code fragment is to implement the equations (4.1) and (4.2). This program is used for left sensor area because the value of “force” is always a positive value which means that the rotation of the UI is always clockwise. The program for right sensor area is almost the same as this one, and the only difference is that rotation value should be (0, 1, 0, -force/20) which will lead the UI to be rotated counterclockwise.

After the creating of transformers, all of the components should be connected with each other following the logic diagram of Figure 21.

In X3D:

```
<Transform DEF= "UI">
… // All UI components
<Box DEF="front_left_sensor " size="0.05 0.38 0.005" solid="true"/> // left force area
…//other force areas
</Transform>
<PythonScript DEF="FORCE" url="C:\H3D\Python\force.py"/>
<ROUTE fromNode="UI" fromField="rotation" toNode="FORCE"
  toField="left_sensorarea"/> // input current rotation value of UI
<ROUTE fromNode="front_left_sensor" fromField="force" toNode="FORCE"
  toField="left_sensorarea"/> //input force of left sensor area
<ROUTE fromNode="FORCE" fromField="left_sensorarea" toNode="UI"
  toField="rotation"/> //output new rotation value to UI
```

When the sensor area detects force, the transformer will calculate a new rotation value to the UI, so that user will feel like he is pushing the UI to move. The other left force areas can employ the same methods. It should be noticed that these areas must be place the left-side of surfaces of the UI.

4.4.2 Implementing details of Zoom in and out function

Zoom in and out function is the important solution for the interaction within large volume 3D model. The typical method in current human-computer interaction is to use keys of a keyboard to zoom in and out. However, this method is suitable only to the two-dimensional objects, and for a lager 3D model, this method seems unnatural and inconvenient. Since the haptic device is a 3D interactive tool, we can use the natural movement of human hand to implement this function. For example, pulling the model will zoom in the model and pushing will zoom out the model.

In order to implement this function, we begin by investigating the logical structure of 3D model in H3D. As we know, the different components of the solar system and the UI models all have their own transform node which includes values of translation, rotation, scale and so on, and all components are placed in the same large 3D space which is created by H3D software. If we can create a new 3D space which contains all our created 3D models and also is inside the main 3D space, the position change of the new 3D space will lead to the moving of all 3D models, but other parameters, such as view point, in the main 3D space will be unchanged. This situation provides a good solution for this function, and the detailed steps are following and also shown in Figure 22:

1. Creating a new logical 3D space which contains all 3D models.
2. Moving the logical 3D space with all models to implement the zoom in and zoom out function.

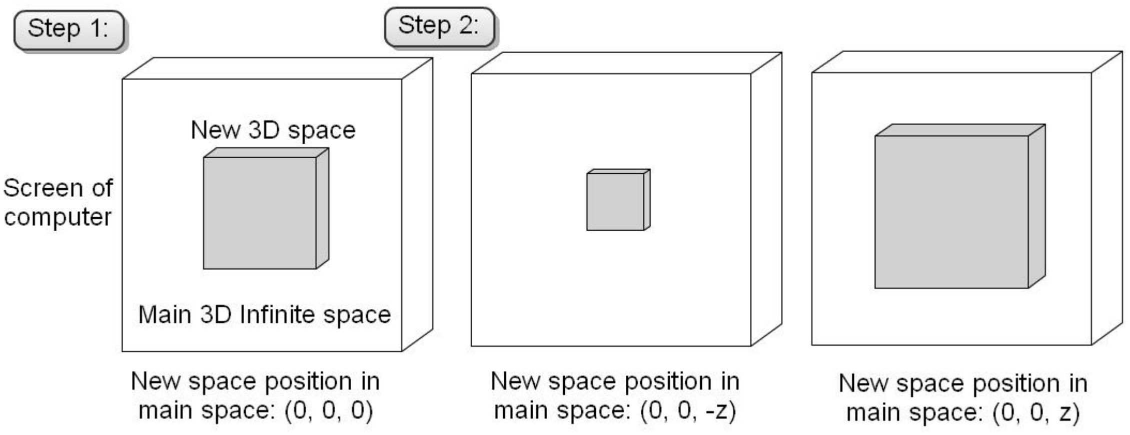

Step 1:
Step 2:
Screen of computer
New 3D space
Main 3D Infinite space
New space position in main space: (0, 0, 0)
New space position in main space: (0, 0, -z)
New space position in main space: (0, 0, z)

Figure 22: Principle of Zoom in/out function using a haptic device

For the first step, in H3D, there is a parent node <MatrixTransform> which can be used to create this logical new 3D space [H3D Manual, 2009], and the only work is to put all nodes of components into one <MatrixTransform >. In our case, since there are two models (the solar system and the UI), and these two models should be put into two different <MatrixTransform> nodes.

To implement the second step, based on the mathematical equation of this <matrixTransform>, we can modify the parameters of this equation to adjust the position of this logical space. The equation [H3D Manual, 2009] is the following:

$$P' = T * C * R * SR * S * -SR * -C * P \ , \qquad (4.3)$$

where P' is new matrix value, P is the old matrix value, C is the value of center, SR is the value of scale orientation, T is the value of translation, R is the value of rotation, and S is the value of scale.

Therefore, based on the equation (4.3), the only parameter which needs to be modified is translation T, and this translation value should be related with the position of the haptic device in the main 3D space. The best solution is to directly obtain the changed value of positions of the haptic device, and then using this value as the translation parameter T in equation (4.3). In this way, the equations are the following:

$$P' = T * P \qquad (4.4)$$

$$T = ((x - x'), (y - y'), (z - z')), \qquad (4.5)$$

where x is the new position of the haptic device in main 3D space, and x' is the old position of the haptic device in main 3D space, and so on. Because P and P' are both four-dimensional matrices, in order to calculate the equation (4.4), the value of T in equation (4.5) has to be changed into four-dimensional matrixes as well. This gives

$$T = [(1, 0, 0, (x - x')), (0, 1, 0, (y - y')), (0, 0, 1, (z - z')), (0, 0, 0, 1)] \quad . (4.6)$$

Now, the relationship between the position of the haptic device and the position of new 3D space has been built up. In order to implement them in X3D and Python, let's first consider the logic diagram of this zoom in and out function for the UI which is shown in Figure 23.

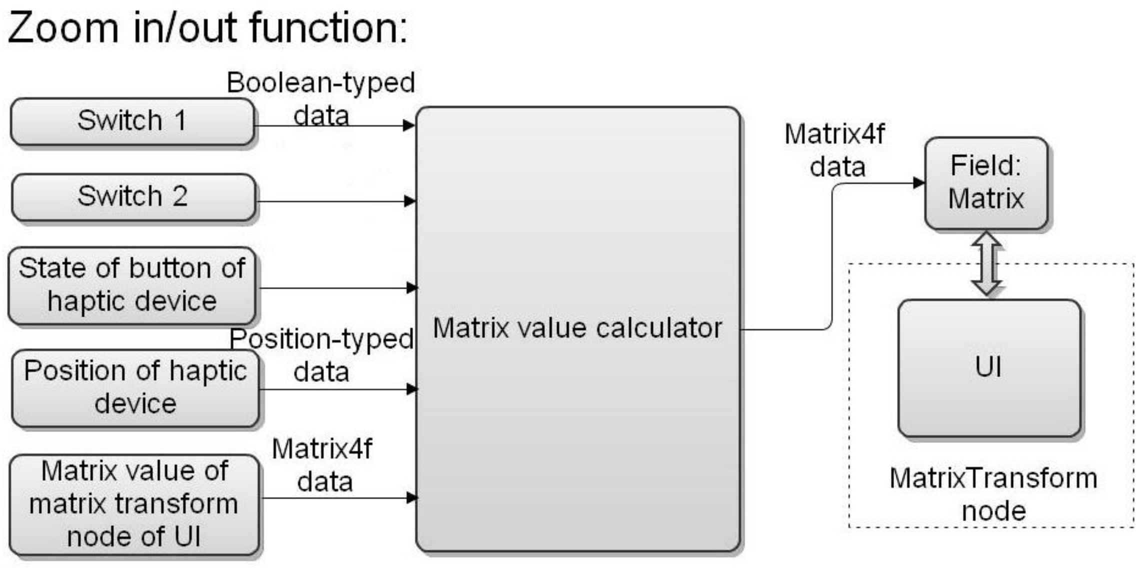


Figure 23: Logic diagram of the UI zoom in and out function using a haptic device

There are two switches added to control that if this function works or not. The purpose of one switch is for switching this function between the UI and the solar system: because zoom in and out function is needed to be applied to the UI model and the solar system model separately, this switch will decide which model this function is applied to. Another switch is for separating this function from other functions: since there is only one physical button on the used haptic device and this button will be used for other functions, such as orbit-changing function, it is necessary to add a switch to identify which function the button of the haptic device is used to. These two switched will be implemented using toggle-press or toggle-release button attached on the UI. Besides these two switches, the core input parameters for this function are the state of main button of the haptic device, position value of haptic and current matrix value of <Transform> node of the UI. Through using the matrix calculator, the output matrix value will be sent back to <Transform> node of the UI. Now, the most important task is to implement the matrix value calculator in Python. According to the equations (4.4), (4.6) and logic diagram, the calculator can be implemented in the code fragment shown below.

In Python:

```
    class zoom (TypedField(SFMatrix4f, (SFBool, SFVec3f,SFBool,SFBool, SFMatrix4f ))):
```

```
  def update(self, event):
      inputs = self.getRoutesIn()
      tracker_button = inputs[0].getValue()
      tracker_position = inputs[1].getValue()
      switch = inputs[2].getValue()
      switch1 = inputs[3].getValue()
      Matrixdata = inputs[4].getValue()
    if tracker_button and switch == False and switch1 == False:
      difference_position = tracker_position - self.last_position
      Object_state = Matrix4f (1, 0, 0, difference_position.x,
                               0, 1, 0, difference_position.y,
                               0, 0, 1, difference_position.z,
                               0, 0, 0, 1) * Matrixdata
      return Object_state
    self.last_position = tracker_position
Zoom = zoom ()
```

The class “zoom” above can receive five different values which are three Boolean-typed data for two switches and button of the haptic device, and one “vec3f” value for storing the position value of the haptic device, and one “matrix4f” value for the matrix value of <MatrixTransform> of the UI. The following parts of the code in class are to implement equations (4.4) and (4.6), and they will be worked only if the states of button and switches are all passing. At last, this class will output a new “matrix4f” value for the UI.

After the creation of the calculator, the next task is to connect all components in X3D, based on the logic diagram of Figure 23.

In X3D:

```
<MatrixTransform DEF="M_T">
… // all UI component
</MatrixTransform>
//Input haptic device into X3D
<IMPORT inlineDEF="H3D_EXPORTS" exportedDEF="HDEV" AS="HDEV"/>
// Two switches
<TouchButton DEF="SWITCH1" buttonMode="TOGGLE_PRESS"/>
<TouchButton DEF="SWITCH2" buttonMode=" TOGGLE_PRESS "/>
<PythonScript DEF="ZOOMINOUT" url="C:\H3D\Python\zoom.py"/>
// Route all parameters into calculator one by one following the order of Python codes
<ROUTE fromNode="HDEV" fromField="mainButton" toNode="ZOOMINOUT"
  toField="Zoom"/> //Button of haptic device
<ROUTE fromNode="HDEV" fromField="trackerPosition" toNode="ZOOMINOUT"
```

```
  toField="Zoom"/> //Position of haptic device
<ROUTE fromNode="SWITCH1" fromField="state" toNode="ZOOMINOUT"
  toField="Zoom"/> //Switch 1
<ROUTE fromNode="SWITCH2" fromField="state" toNode="ZOOMINOUT"
  toField="Zoom"/> //Switch 2
// Input Matrix value of UI
<ROUTE fromNode=" M_T" fromField="matrix" toNode="ZOOMINOUT"
  toField="Zoom"/>
// Output the new matrix value to UI
<ROUTE fromNode=" ZOOMINOUT" fromField="Zoom" toNode="M_T"
  toField="matrix"/> // output Matrix value of UI
```

<IMPORT> is used to add the functions of a haptic device into X3D and "Mainbutton" is the state of button of device and "trackerPosition" is the position value of the haptic device in 3D space [X3D tutorial, 1999]. In sum, this code fragment is only for the UI, and the program for the solar system is similar to this one. The condition of the switch should be different for the two systems: for example, setting "False" is the condition for the UI, and setting "True" is the condition for the solar system, so that while user clicks the button, if the button state is "True", the function will only work for the solar system, and if button state is "False", the function will only work for the UI. Now, the zoom in and out function has been successfully implemented.

4.4.3 Implementing details of information-showing function

In order to let user study the basic knowledge of the solar system, the information needs to be shown to user. Since a haptic device can directly touch all of the objects, the best method to implement this function is to use touch as the trigger event to open the view of the information. This information view should contain the models of planets and their moons, and also the introduction text. Now, the problem is that how to "open" this view in practice. There are two methods to implement:

1. Creating a new X3D file which contains all components of view, and then using touch as trigger event to input this file.
2. Creating all components of information view in the original X3D file, but in the different position of the main 3D place, and then using touch as trigger event to change the view point of user for them.

The above two methods are both feasible. However, the first method requires much computing resources and sometimes the computer will be stuck when the program is loading the new X3D file. Therefore, in this project, the second method is used to implement this function. Then, the logic diagram of function is shown in Figure 24.

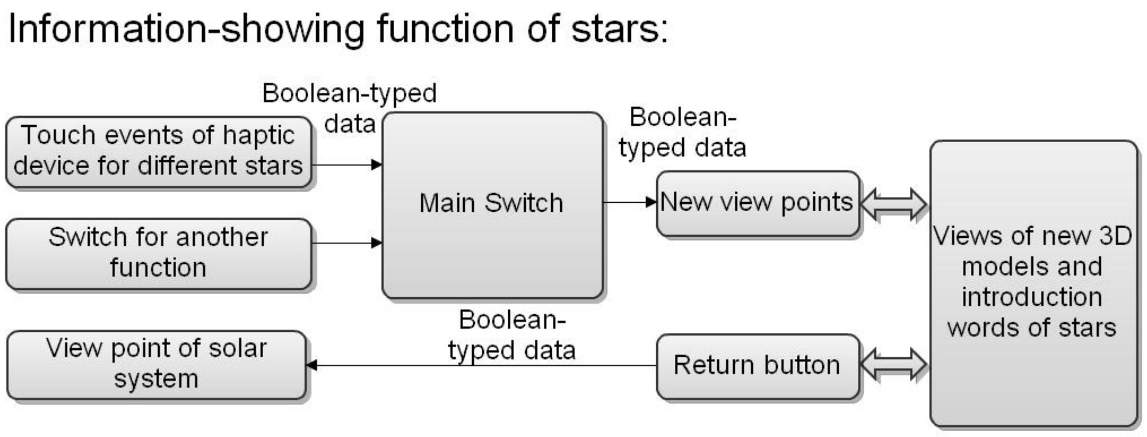

Information-showing function of stars:
Touch events of haptic device for different stars
Boolean-typed data
Main Switch
Boolean-typed data
New view points
Switch for another function
Views of new 3D models and introduction words of stars
View point of solar system
Boolean-typed data
Return button

Figure 24: Logic diagram of information-showing function using a haptic device

Since touch events of the haptic device has to be used for the orbit-changing function of planets, there is an additional switch which is used to separate these two functions. Both the touch event and the new switch output a Boolean-typed data to main switch which will be implemented in Python. The main switch controls if the view point changes. And also, since user will return to the main view point after checking the information, there should be a button below the text in the information view, which can be used to change back to the original view point.

Therefore, the tasks needed to be completed in this function are these: making 3D models palpable, creating new view points, creating new 3D models and text, and implementing main switch in Python. First, because the creating of view point and 3D model has been introduced in Chapter 3, we focus on the implementation of palpable 3D model and creating text in X3D.

To make a 3D model palpable, the method has been discussed in pervious parts, which is using “stiffness” of <SmoothSurface> in <Appearance> of <Shape>. And the value of “stiffness” can be changed from 0 to 1, and the bigger value leads to the stiffer model. Besides <SmoothSurface>, there are other types of surfaces, such as <FrictionalSurface> which will make surface with friction, and so forth [X3D tutorial, 1999]. In our case, here is an example.

In X3D:

```
<Transform DEF="SUN" translation ="0 0 0">
<Shape DEF="SUNSHAPE">
<Appearance>
… // Sun model
<SmoothSurface stiffness="0.1"/> // making model palpable
</Appearance>
</Shape>
</Transform>
```

Creating the text is simple as well in X3D, which can be implemented by using “string” of <Text> in <Shape> [X3D tutorial, 1999]. The blank is a string and can be put into “string”, and thus, changing a newline can be done by using blanks in separate double quotation marks. The below is an example.

In X3D:

```
<Transform DEF="SUNWORD" translation ="100.3 100.5 100">
<Shape>
<Text string=' "SUN"
  "    " "is a star…" '/> // Blank line and other texts
</Shape>
</Transform>
```

The introduction text can be created using the above method. Now, the main switch is needed to be implemented in Python, and the function of this switch is this: when touch event and new switch send “True” to main switch, main switch will output “True” to open the new viewpoint. So the code fragment is as following:

In Python:

```
class mainswitch(TypedField( SFBool, (MFBool, SFBool))):
    def update ( self, event ):
        routes_in = self.getRoutesIn()
        touch= routes_in[0].getValue()
        switch=routes_in[1].getValue()
        if touch and switch== True:
            return True
        return False
Mainswitch=mainswitch()
```

Now, we should connect two input parameters with main switch one by one, and then send the output value for opening the new view point. Lastly, the return button has to be implemented as well. Let’s see the Sun as an example.

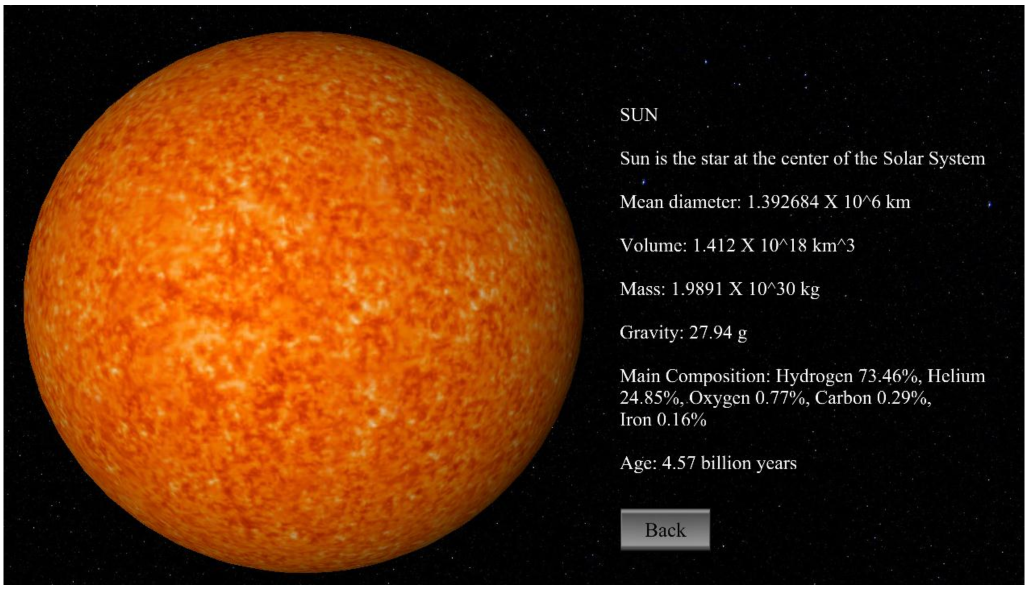
In X3D:

```
<Viewpoint DEF="VP" position="0 0 1.5"/> //VP for the solar system
<Viewpoint DEF="VPSUN" position="100 100 102" set_bind="false"/> // VP for sun
<PythonScript DEF="switch" url="C:\H3D\Python\switch.py"/>
<TouchButton DEF="SWITCH" buttonMode="TOGGLE_PRESS"/> //New button
<ROUTE fromNode="SUNSHAPE" fromField="isTouched" toNode="switch"
  toField="Mainswitch"/> // Input parameter of touch event
<ROUTE fromNode="SWITCH" fromField="state" toNode="switch"
  toField="Mainswitch"/>// Input parameter of new switch
<ROUTE fromNode="switch" fromField="Mainswitch" toNode="VPSUN"
  toField="set_bind"/> // output value to open the information view of the Sun
<TouchButton DEF="SWITCH2" buttonMode="NORMAL"/>// return original viewpoint
<ROUTE fromNode="SWITCH2" fromField="state" toNode=" VP"
```

toField="set_bind"/> **//return the viewpoint back to the original one**

The above code fragment shows that there are two viewpoints: one is for the user to watch the information of the Sun (“VPSUN”) and another one is for watching the solar system (“VP”). If the touch event is detected from the 3D model of the Sun and also function switch is on, the current view point will be changed into “VPSUN” for watching the information of the Sun. And if the return button is touched, the view point will be changed back to “VP” for the solar system.

To sum up, the information views can be done by using the same method. There are some points needed to be mentioned: First, the input parameters of touch events are different for every planet and the Sun, which is generated from the 3D model of each celestial body. And function switch button should be attached on the UI for users, and there should be ten return buttons which are separately placed under the introduction text of each planet and the Sun. A view of information image of the Sun is shown in Figure 25.


SUN

Sun is the star at the center of the Solar System

Mean diameter: 1.392684 X 10^6 km

Volume: 1.412 X 10^18 km^3

Mass: 1.9891 X 10^30 kg

Gravity: 27.94 g

Main Composition: Hydrogen 73.46%, Helium 24.85%, Oxygen 0.77%, Carbon 0.29%, Iron 0.16%

Age: 4.57 billion years

Figure 25: The view of information of the Sun

4.4.4 Implementing details of orbit-changing function of planets

The orbit-changing function is the most complicated function in this project. The basic idea of this function is this: user can employ a haptic device randomly to touch every planet and then press the main button of device to pick up the planet. The planet will be moved following the haptic device, and then when user release the main button of device, the planet will be running on the new orbit depending on its current location. In order to implement this function, there are several technical problems which have to be solved first:

1. How to separately select and pick up each planet using only one physical button of device?

2. How to make the planet following with the haptic device?
3. How to put the planet into a different orbit?

One button clearly is not enough to make computer know which planet user wants to pick up. Thus, there must be an independent checking program to identify which planet user wants to select, and then user can press the physical button of device to pick it up. There are two basic ideas about implementing this in Python:

- Distance: considering the distances between the haptic device and all planets, if one distance is smaller than a special number, it can be confirmed that this planet is the one that user wants to pick up.
- Touch: considering the touch events of all planets, if touch event of a planet is triggered, it can be confirmed that this planet is the one that user wants to pick up.

In the first method, the needed input data is the position of planet and the position of the haptic device, and then the distance formula of two points in 3D space could be utilized [Abbena et al., 2006]. Assuming values of positions of two points in 3D space are (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2), the formula is

$$\text{Distance} = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2}\ . \tag{4.7}$$

According to (4.7), we can calculate the distance between the planet and haptic device and if the distance is small enough, it means that this planet is the selected one. The details of the first method are shown in Figure 26. Basically, this method is feasible but the assuming distance value which decides if the planet has been selected or not is difficult to be determined: if this value is too big, user will pick up multiple planets; if this value is too small, user will be very difficult to pick up a planet in a large 3D environment.

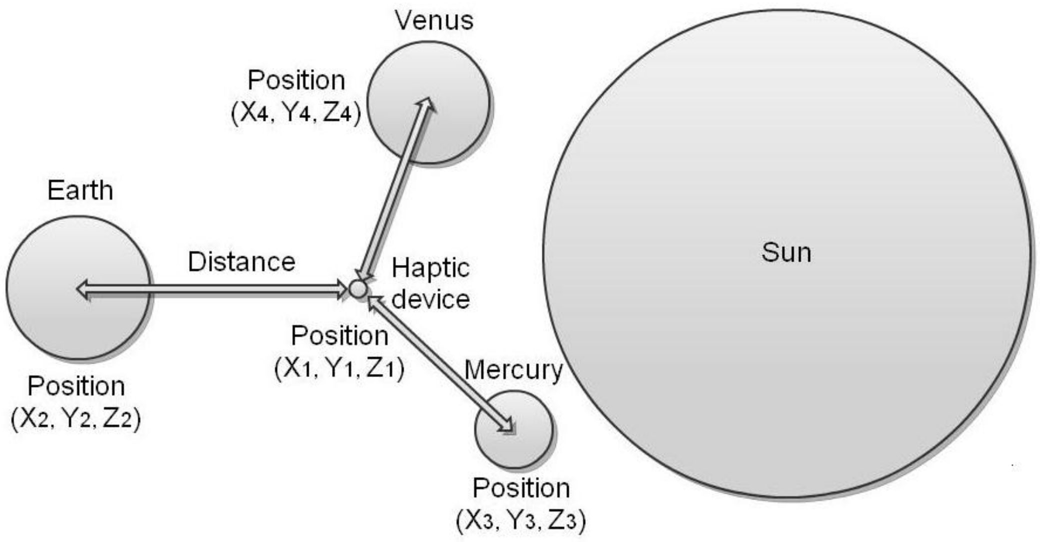


Figure 26: Principle of the selection of planets

The second method does not have the problem of the first method, because user has to touch the planet to select it. However, this method has its own drawback: since there is always a small mathematical deviation in the position system of H3D, when the planet has been selected and moved following the haptic device, there is a big chance that the haptic device will not touch the surface of 3D model of planet, and thus, the planet will drop.

In order to overcome the drawbacks of both methods, the best way is to implement them at the same time, and the final principle of checking program is this: there are two conditions to decide if the planet has been selected or not, which are distance and touch, and the planet is confirmed to be selected if one of them is true.

Secondly, for the second problem, the principle of revolution of planets needs to be reviewed. Due to that all planets are moving on their owned fixed orbit with time, and the orbit, in fact, is a database containing all coordinates of points of orbit, the best method to make the planet following with the haptic device is to replace the original coordinates of points of orbit with new coordinates which all are the same value as the position coordinate value of the haptic device. In this way, the selected plant will be following the movement of the haptic device. Here, due to that the coordinate value of a haptic device is a three-dimensional value and the data of orbit will a matrix of three-dimensional value, a simple mathematical transform is needed. For example:

$$\mathrm{Position} = \mathrm{Vec3f}\,(\mathrm{a}, \mathrm{b}, \mathrm{c}) \tag{4.8}$$

$$\mathrm{Orbit} = [\mathrm{Positon}, \mathrm{Postion}, \ldots, \mathrm{Position}]\ . \tag{4.9}$$

Thirdly, in order to solve the third problem, a new checking algorithm is necessary to be implemented: this program can check the last position of planet when user releases the button of device, and then measure the distance between the planet and the center of the solar system. Then, based on the distance, a different orbit database is used to replace the current coordinate values in the orbit database of planet. In our case, the center of the solar system is always (0, 0, 0), so the distance can be easily calculated by equation (4.7). The details of solution of the third problem are shown in Figure 27.

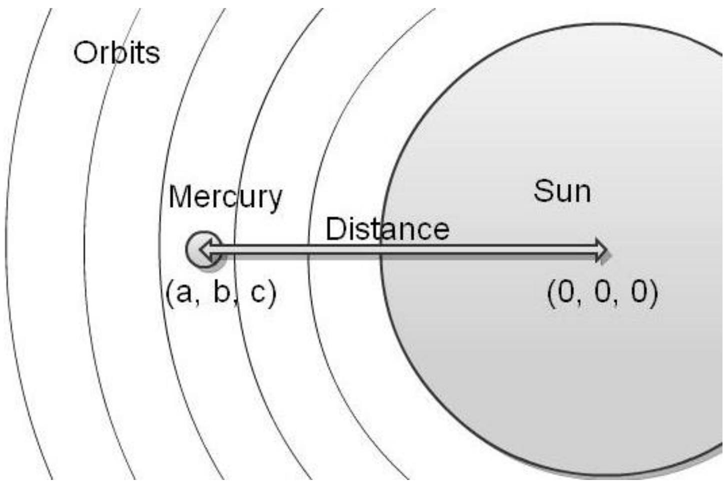


Figure 27: Principle of the orbit-changing for planets

Based on the analysis above, the complete principle of orbit-changing function is this: there should be nine orbit databases which contain the coordinate values of nine orbits, and when the user press the button of the haptic device, the first check algorithm will work to check if one of planets has been selected. If one planet is confirmed to be selected, a data calculator will work and it can replace the original orbit data of selected planet with the coordinate values of the haptic device. At then, when user releases the button of the haptic device, the second check algorithm will work to calculate the distance between the current position of the planet and the center of the solar system and then choose a suitable orbit data from nine orbit databases to replace the current coordinate values in the selected planet orbit database. The logic diagram of orbit-changing function is shown in Figure 28.

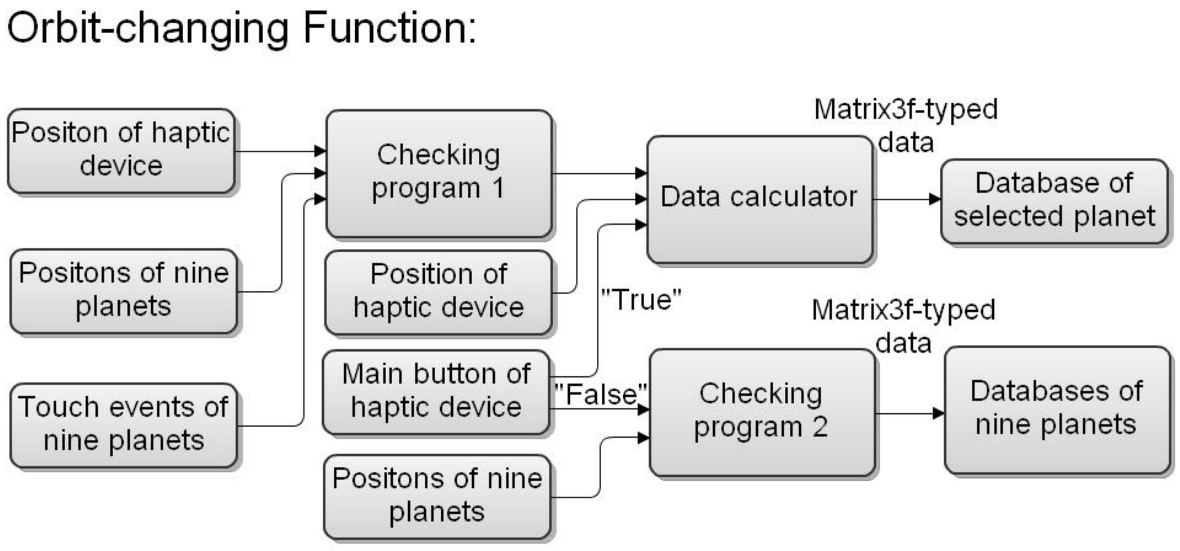


Figure 28: Logic diagram of the orbit-changing function of planets

According to the logic diagram in Figure 28, two checking programs and data calculator are the core components of this function. The first checking algorithm needs three input parameters which are the position of the haptic device, the position of planet and touch event, and it will output a Boolean-typed data as a switch for data calculator. It can be implemented in Python and the code fragment below gives the details.

In Python:

```
Class check1 (TypedField( SFBool,(MFBool, SFVec3f, SFVec3f ) ) ):
    def update ( self, event ):
        routes_in = self.getRoutesIn()
        if len(routes_in[0].getValue())> 0:
            touch = routes_in[0].back() // Get the last value of touch event
        else: touch = False
        Device_position = routes_in[1].getValue()
        planet_position = routes_in[2].getValue()
        L= math.sqrt( math.pow ((Device_position.x- planet_position.x ), 2) +
                           math.pow ((Device_position.y- planet_position.y), 2) +
                           math.pow ((Device_position.z- planet_position.z), 2))
        if touch == True or L < 0.005: // **Touch or distance**
            return True
        else: return False
Check1=check1()
```

The class “check1” above can receive the positions of a planet and a haptic device, and also the touch events from 3D model of planets. Since the touch events are multiple Boolean-typed data and the only valid data is the last one, “back ()” is used to select the last Boolean-typed data [Jackson, 2011]. Based the equation (4.7), the distance can be calculated, and then if the condition of touch or distance is passing, the algorithm will output a “True” value. At last, “Check1” is an application of this class. The same method can be applied to all the nine planets.

After the implementation of the first checking algorithm, the second checking algorithm can be implemented in Python as well. The example code fragment is the following:

In Python:

```
class calculator (TypedField(MFVec3f,(SFVec3f, SFBool, SFBool, SFVec3f,
                 MFVec3f, …, SFBool))):
  def update(self, event):
    routes_in = self.getRoutesIn()
    Device_position=routes_in[0].getValue()
    Device_button= routes_in[1].getValue()
    Switch= routes_in[2].getValue() //The first switch
    planet_position= routes_in[3].getValue()
    … // Positions of other planets
    Switch2 = routes_in[13].getValue()// The second switch
    if Device_button == True and Switch == True and Switch2== True: //data calculator
      a = Vec3f (Device_position.x, Device_position.y, Device_position.z)
      Result = [a, a…, a]
    else: // The second checking program
      Distance = math.sqrt(math.pow(planet_position.x, 2) +
                           math.pow(planet_position.y, 2) +
                           math.pow(planet_position.z, 2))
      if   Distance <= 0.3125:
          Result= mercury_position
      elif Distance > 0.3125 and Distance <= 0.4375:
          Result = venus_position
      elif Distance > 0.4375 and Distance <= 0.5625:
```

```
                Result = earth_position
                … // output other databases according to the different distance
        return Result
mercuryfollowing=calculator()
```

According to the above code fragment, there are two switches which are used to decide if planet is following the haptic device or not. The first switch is the one to separate this function from zoom in and out function which is also using the physical button of the haptic device. And the second switch is the Boolean-typed value generated from the first checking program. Generally, if the value of first checking program is "True" and also main button of device is pressing, the data calculator ("if" part in codes) will begin to work and the return value will be the database built up based on the equation (4.8) and (4.9); if one of them is "False", then the second program ("else" part in codes) will begin to check the distance and return a suitable orbit data to planet. At last, each planet should be using one application of the above class. Now, all components of logic diagram should be connected with each other in X3D, and let's consider an example of Mercury.

In X3D:

```
<Transform DEF="MERCURY" translation="0 0 0.3125">
<PositionInterpolator DEF="MERCURYREVOLUTION" key= "…" keyValue="…">
… //Other parameters
< /Transform >
<Transform DEF="ORBIT_DATABASE">
… //Nine orbit databases including all orbit data from ORBIT_ONE to ORBIT_NINE
< /Transform >
<IMPORT inlineDEF="H3D_EXPORTS" exportedDEF="HDEV" AS="HDEV"/>
//Inputting python file including the codes of checking programs and data calculator
<PythonScript DEF="orbit" url="C:\H3D\Python\orbit.py"/>
//The first checking program
<ROUTE fromNode="MERCURYSHAPE" fromField="isTouched" toNode="orbit"
  toField="Check1"/> //Button of device
<ROUTE fromNode="HDEV" fromField="trackerPosition" toNode="orbit"
  toField="Check1"/> // Position of device
<ROUTE fromNode="MERCURY" fromField="translation" toNode="orbit"
  toField="Check1"/> //Touch events
//Data calculator and the second checking program
<ROUTE fromNode="HDEV" fromField="trackerPosition" toNode="orbit"
  toField="mercuryfollowing"/>// Position of device
<ROUTE fromNode="HDEV" fromField="mainButton" toNode="orbit"
  toField="mercuryfollowing"/>//Button of device
```

```
<ROUTE fromNode="ORBIT_SWITCH" fromField="state" toNode="orbit"
  toField="mercuryfollowing"/>// First switch
<ROUTE fromNode="MERCURY" fromField="translation" toNode="orbit"
  toField="mercuryfollowing"/>// Position of mercury
<ROUTE fromNode="ORBIT_ONE" fromField="keyValue" toNode="orbit"
  toField="mercuryfollowing"/>// input orbit data for Python
… //Other routes for orbit two to nine
<ROUTE fromNode="switch" fromField="add_switch60" toNode="orbit"
  toField="mercuryfollowing"/>//Second switch
<ROUTE fromNode="orbit" fromField="mercuryfollowing"
  toNode="MERCURYREVOLUTION" toField="keyValue"/> //changing orbit data
```

Since the default orbit for Mercury is the first orbit in the orbit database, the default value of translation of “MERCURY” should equal to or be smaller than 0.3125 which calculated from equation (4.7). In this way, Mercury will be running on the first orbit before the use of orbit-changing function. This method can be applied to other planets, and also user can pick up multiple planets if he touches multiple planets using a haptic device.

Until now, all functions which are using a haptic device as the interactive tool have been implemented. In fact, there is only one button on the haptic device and also the interactive method mainly is touch, and this situation lead to the mess of functionality of this interactive system. For example, touching a planet will open the information image, but at the same time, this action is also the basic method for selecting planet in orbit-changing function. In order to separate these functions, some buttons have been added to the UI and their logical events have been implemented into all functions above. To make them clear, Figure 29 is a summary for these buttons and their logical function.

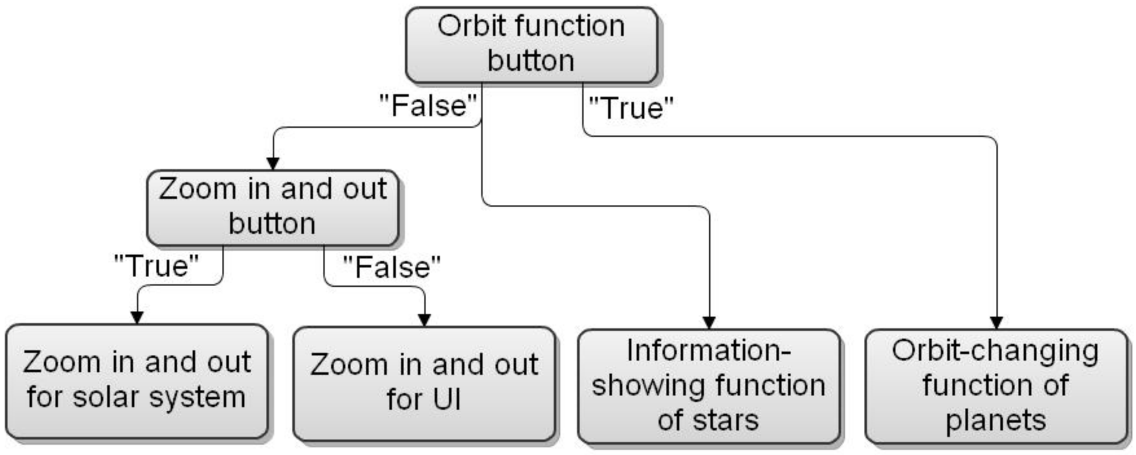

Orbit function button
"False"
"True"
Zoom in and out button
"True"
"False"
Zoom in and out for solar system
Zoom in and out for UI
Information-showing function of stars
Orbit-changing function of planets

Figure 29: Logic diagram of buttons used for separating functions

There are two toggle-press buttons which are used to separate the implemented functions in this interactive system. Generally, Orbit function button is the main button:

when the state of this button is “True”, the physical button of the haptic device is only used for orbit-changing function and touching planet will be only used to select planet to move; when the state of this button is “False”, touching planets can lead to show the information of planets. In this situation, if the state of zoom in and out button is “True”, the button of the haptic device can be used to zoom in or zoom out the 3D model of the solar system, and if the state of zoom in and out button is “False”, the button of the haptic device can be used to zoom in or zoom out the 3D model of the UI. At last, orbit function button and zoom in and out button should be attached to the UI with other UI components for user.

4.5 Implementation of interactive system using a mouse and a keyboard

In order to compare the 3D interactive method using a haptic device with the normal tradition method, the implementation of another interactive system which is employing traditional peripheral devices of computer as interactive tool is necessary. Basically, this interactive system should be designed in the way that users are familiar with, therefore a mouse and a keyboard would be the best interactive tools for this new interactive system. There are mainly two functions in this interactive system:

- A function for rotating 3D models of the UI and the solar system using mouse/keyboard.
- Information-showing function for the solar system objects using mouse/keyboard.

These two functions are similar to the ones using a haptic device, and since the unnatural interaction method of mouse/keyboard, zoom in and out function is not included in the implementation plan. But a function which can rotate the 3D model of the solar system is implemented, and this function, as we discussed before, cannot be implemented using a haptic device because we use a 3DOF device which cannot receive and generate the forces of orientation.

4.5.1 Implementing details of rotation of the UI and the solar system

In the traditional interactive method, there are basically two methods to rotate a 3D model: the first method is to use a mouse to drag the 3D model for rotating, and another method is to use keys on a keyboard to rotate the 3D model. Generally, these two methods are both suitable and choosing one of them is enough. In this project, the method I selected is to use keys on a keyboard to rotate the 3D model, and in order to separate this function for the UI and the solar system, the simplest way is to use different keys to trigger the function.

Since the 3D model of the solar system must be rotated in 360 degrees, there should be four keys on a keyboard to control the solar system. However, the cube UI only has four surfaces, which means that only left-right rotation is enough. So, only two keys needs to be used for rotating the UI. Let’s first consider the rotation function for the UI depicted in Figure 30.

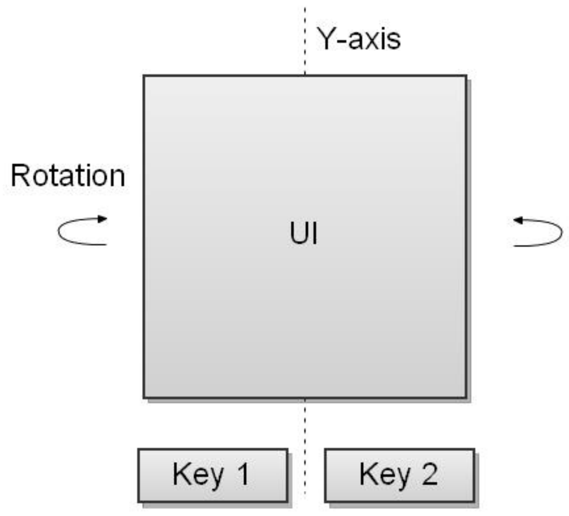


Figure 30: Situation of the UI rotation using keys

Each one of two keys is only in charge of one direction of rotation, and the rotation angle should be decided as well while the keys are used. Firstly, since users are familiar with the left key which controls left rotation and right key which controls right rotation, we can design it in this way: ‘Z’ is key 1 and ‘X’ is key 2 (‘Z’ is just on the left side of ‘X’ on the keyboard). Secondly, the angle of rotation should be 45 degrees so that user not only can perceive the rotation of the cube UI when user presses the keys but also it will not seriously affect the efficiency of this function (90 degrees is the most efficient angle but user cannot perceive the rotation of the UI).

Now, we need to check the logic diagram of this function, which is shown in Figure 31.

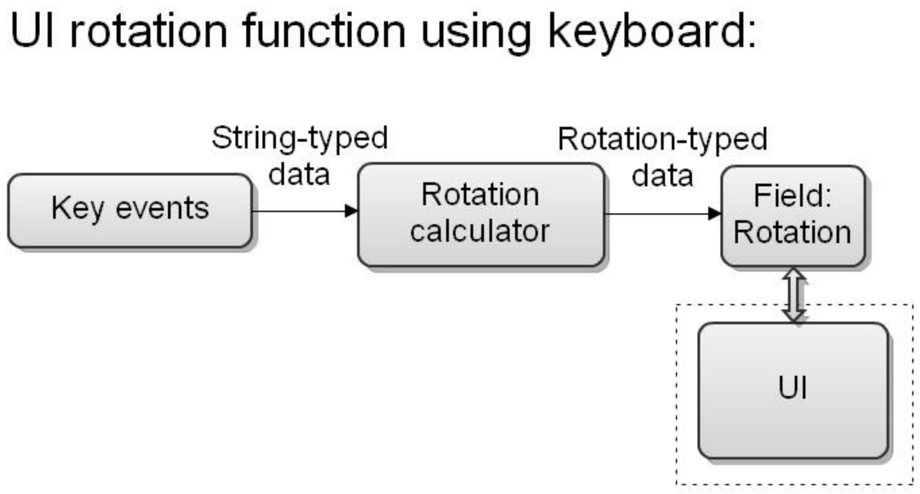


Figure 31: Logic diagram of the UI rotation function using a keyboard

The core component in the logic diagram is rotation calculator which can generate rotation-typed value when the key events have been triggered. The below codes of Python are an example for this calculator.

In Python:

```
class calculator2 ( TypedField( SFRotation, SFString ) ):
    def __init__( self ):
        TypedField( SFRotation, SFString ).__init__( self )
        self.changed = Rotation()
    def update( self, event):
        if event.getValue() == 'z' or event.getValue() =='Z':
            self.changed = Rotation( 0,1,0,-0.785398 ) * self.changed
        elif event.getValue() == 'x' or event.getValue() =='X':
            self.changed = Rotation( 0,1,0,0.785398 ) * self.changed
        return self.changed
UI_rotation=calculator2 ()
```

The class “calculator2” can get as input a string-typed data and can output a rotation-typed data. “def __init__(self)” is used to store the initial value of rotation each time before the calculation [Jackson, 2011], and the initial value is stored in parameter “self.changed”. Then, if ‘Z’ key have been triggered, this returned parameter will be 45 degrees (=0.785398 radian) along Y-axis and direction is clockwise, and if ‘X’ key has been triggered, the returned value will be 45 degrees counterclockwise along Y-axis.

Now, all components need to be connected with each other following the logic diagram of Figure 31.

In X3D:

```
<Transform DEF="UI">
  … // All UI components
<Transform>
<KeySensor DEF="KEYSENSOR" /> // Input key events into X3D
<PythonScript DEF="switch" url="C:\H3D\Python\switch.py"/>// “Calculator2” inside it
<ROUTE fromNode="KEYSENSOR" fromField="keyPress" toNode="switch"
  toField="UI_rotation"/> // Input key events
<ROUTE fromNode="switch" fromField=" UI_rotation" toNode="UI"
  toField="rotation" /> // output new rotation value to UI
```

In above code fragment, <KeySensor> is used to input the key events into X3D and “keyPress” is the field which sends out the string of key [X3D tutorial, 1999]. When the class of Python receives the special strings from a keyboard, it will send a fixed rotation-typed data to the UI, and the UI will be rotated depending on this rotation value.

Rotation function for the solar system is almost the same as the above one, and the difference is that there are two more keys used. These two keys will rotate the whole solar system along the X-axis, and the angle of rotation should be much smaller than 45 degrees, for example, 5 degrees in one time. In this way, the rotation function for 3D models of the UI and the solar system has been successfully implemented.

4.5.2 Implementing details of information-showing function

The information-showing function has been implemented in the interactive system of using haptic devices, and during the process of implementation, the trigger event to show the information of celestial body is touch event using the haptic device. To implement this function in the interactive system of using mouse/keyboard, the best trigger event is the action of clicking of a mouse which can be done by adding <TouchSensor> in every <Transform> node of celestial bodies. The basic logic diagram is shown in Figure 32.

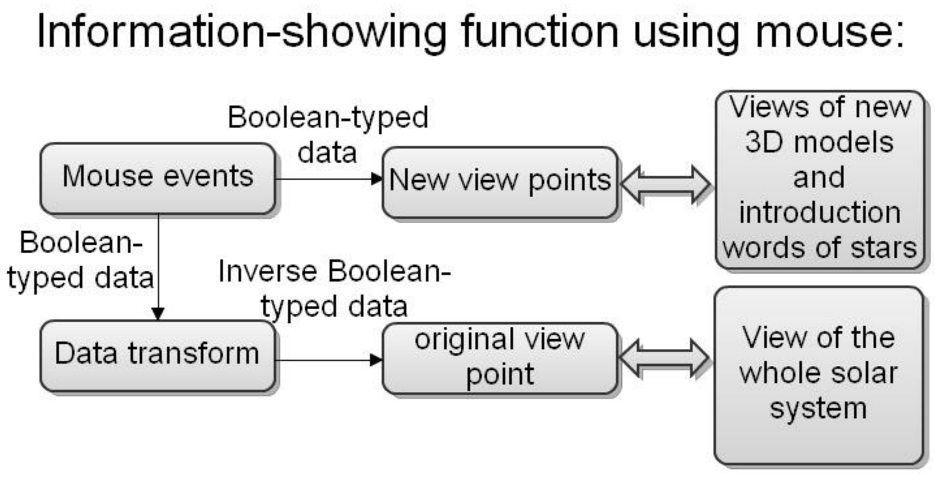

Information-showing function using mouse:
Mouse events
Boolean-typed data
New view points
Views of new 3D models and introduction words of stars
Boolean-typed data
Inverse Boolean-typed data
Data transform
original view point
View of the whole solar system

Figure 32: Logic diagram of the information-showing function using a mouse

When user presses the button of mouse on one of the objects, the mouse event will be triggered and sent a “True” value to open the new view point for this object. Simultaneously this “True” value will be sent to data transform to generate a “False” value which can close the original view point, so that user can watch the information about the object. And when user releases this button, the mouse will be sent a “False” value to close the new view point and this “False” value will be sent to the data transform as well which will output a “True” value, and thus, it will open the original view point for the main solar system. In this way, user can easily use this information-showing function via a mouse. Therefore, the data transform should be implemented using Python, and then all components should be connected in X3D.

In Python:

```
class InverseSwitch ( AutoUpdate( SFBool ) ):
    def update ( self, event ):
        if event.getValue()== True:
            return False
        else:
            return True
inverse= InverseSwitch()
```

In X3D:

```
<Viewpoint DEF="VP" position="0 0 1.5"/> // view point of the solar system
// view point for information of the Sun
<Viewpoint DEF="VPSUN" position="100 100 102" set_bind="false"/>
<Transform DEF="SUN">
  … // Model of the Sun
<MouseSensor DEF="T_SUN"/> // Mouse sensor
</Transform>
<PythonScript DEF="switch" url="C:\H3D\Python\switch.py"/> // "inverseswitch" inside
<ROUTE fromNode="T_SUN" fromField="isActive" toNode="VPSUN"
  toField="set_bind"/> // Connection for new view point
// Connections for data transform and original view point
<ROUTE fromNode="T_SUN" fromField="isActive" toNode="switch" toField="inverse"/>
<ROUTE fromNode="switch" fromField="inverse" toNode="VP" toField="set_bind"/>
```

The above code fragment is for the Sun, and this function for nine planets can be done using the same method. However, since there are ten information views for all celestial bodies in the solar system, there should have ten new viewpoints. To open these views, the logical event must follow the local diagram above in order to let the viewpoint of the solar system to be the default viewpoint.

Until now, all functions which are related with the mouse and keyboard have been implemented and also the implementation of the whole multimodal interactive system has been completed. Because all the UI components are originally designed in the way which can be controlled by a mouse and a haptic device, the cube UI now can be fully used in both two interactive systems. In this way, the UI could be also a test object for user study when users are using different interactive tools to interact with it, and also users can check the performance of these two interactive systems. The details of the user study will be introduced in Chapter 5.

5. User study for the multimodal interactive system

A user study was conducted in order to test the performance of the multimodal interactive system. This user study employed a questionnaire to collect the opinions of users. The questionnaire was divided into two parts: the first part of the questionnaire was used to record the experience and background of the participant, related with haptic devices and knowledge. It included three questions:

1. Do you often use personal computer and its basic peripheral devices such as mouse and keyboard? Answer: Often/ Not often/ Never.
2. Do you have any experience of using 3D interactive tool such as haptic device before? Answer: Yes/No
3. Do you have any knowledge about haptics and haptic interaction? Answer: Yes/No

The second part of the questionnaire was used after the participant finished all experimental tasks, and recorded the comments and suggestions of participant. This part of the questionnaire included six questions:

1. For a virtual 3D object, which method you prefer to control it and why? Answer: 3D interactive method (like using haptic device)/2D interactive method (like using mouse and keyboard)
2. How do you think the 3D interactive way with haptic interaction compared with only using gesture and why? Answer: Necessary/Not necessary
3. Which one of interactive tools you prefer to use when you need to control UI and why? Answer: Haptic device/ mouse and keyboard.
4. Which interactive system is more efficient and which interactive system is more accurate? Answer: Interactive system with haptic device/ interactive system with mouse and keyboard.
5. What is the main problem of the less-efficient/accurate interactive system?
6. In your opinion, what is the current problem for interactive system with haptic interaction and how to improve it?

The second part of the questionnaire was the main point of this user study, and each question in the second part had different purpose:

1. The first question was to understand the opinion of user for the novel 3D interactive methods, such as using gesture and force feedback to control 3D virtual object instead of traditional 2D interactive method.
2. The second question was to investigate the opinion of user for haptic interaction, and to understand if force feedback is necessary, or only using gesture is enough for the user.

3. The third question was to investigate which one is the best tool to control the current popular graphical user interface and the main reason for it.
4. The fourth question was to focus on the critical evaluation elements of interactive systems: efficiency and accuracy.
5. The fifth question was to understand the main reason to cause the less-efficient and less-accurate in two interactive systems.
6. A general question for investigating the weakness of haptic interaction in current human-computer interaction and the answer of this question could be the improved way for the future development of haptic interaction.

The detailed questionnaire has been attached in the appendix. Generally, the following content of this chapter is divided into two parts: the first part introduces the basic setup and process of experiment; the second part describes the collected information from questionnaire and summarizes the results of experiment.

5.1 The setup and procedure of the experiment

Generally, the experiment was an individual experiment and a participant took approximately thirty minutes to finish the whole experiment. There were totally 15 participants joining the experiment. The process of experiment was divided into three phases:

1. The first phase: Inquiring the related background information of the participant and filling in the first part of questionnaire.
2. The second phase: Introducing the experiment setup, preparing the basic knowledge to the participant and making him familiar with haptic devices.
3. The third phase: Completing the given tasks and filling in the second part of the questionnaire.

During the first and second phases, the participant used one minute to answer the first part of the questionnaire, and then the basic devices used in the experiment was introduced to the participant, mainly including force feedback haptic devices: Omega and Phantom, mouse and keyboard, and speaker. (The images of Omega and Phantom have been attached in the appendix.) If the participant lacked of the knowledge of haptic interaction, some basic knowledge was taught to him or her. After that, it took about ten minutes to let the participant to be familiar with haptic devices. When everything had been done, the experiment entered into the third phase.

In the third phase, some simple tasks were given to the participant, and these tasks were random, for example, checking the information of Earth and Jupiter using a mouse and a haptic device, moving Mars into the orbit of Saturn using a haptic device, increasing the size and speed of Venus through the UI using a mouse and a haptic device, and so on. This process took about 15 minutes. After participant completed the given tasks, the questions of the second part of the questionnaire were asked to be answered.

5.2 Results of the experiment

According to the answers of the first part of the questionnaire, the backgrounds of participants were similar to each other. This is shown in Figure 33.

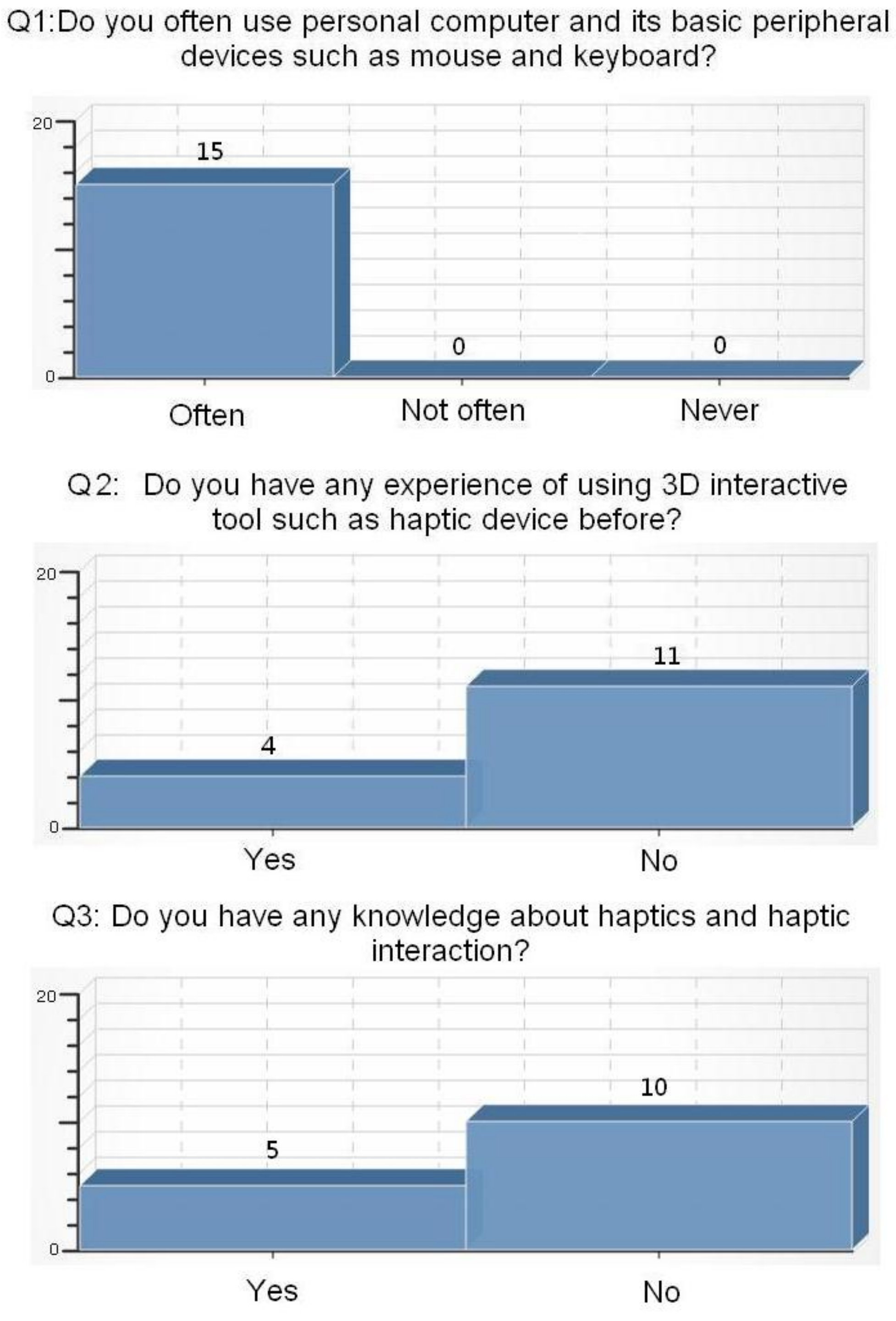


Figure 33: Answers of the first part of the questionnaire

All the fifteen participants were familiar with personal computers and basic devices, but only four participants had used haptic devices before this experiment. Furthermore, these four participants had studied basic knowledge of haptic interaction, and there was one participant who knew some knowledge of haptic interaction but never had a chance to use a haptic device before. Other ten participants had no knowledge about haptic interaction.

After the experiment, all participants gave their answers to the second part of the questionnaire. The answers of the first question are summarized in Figure 34.

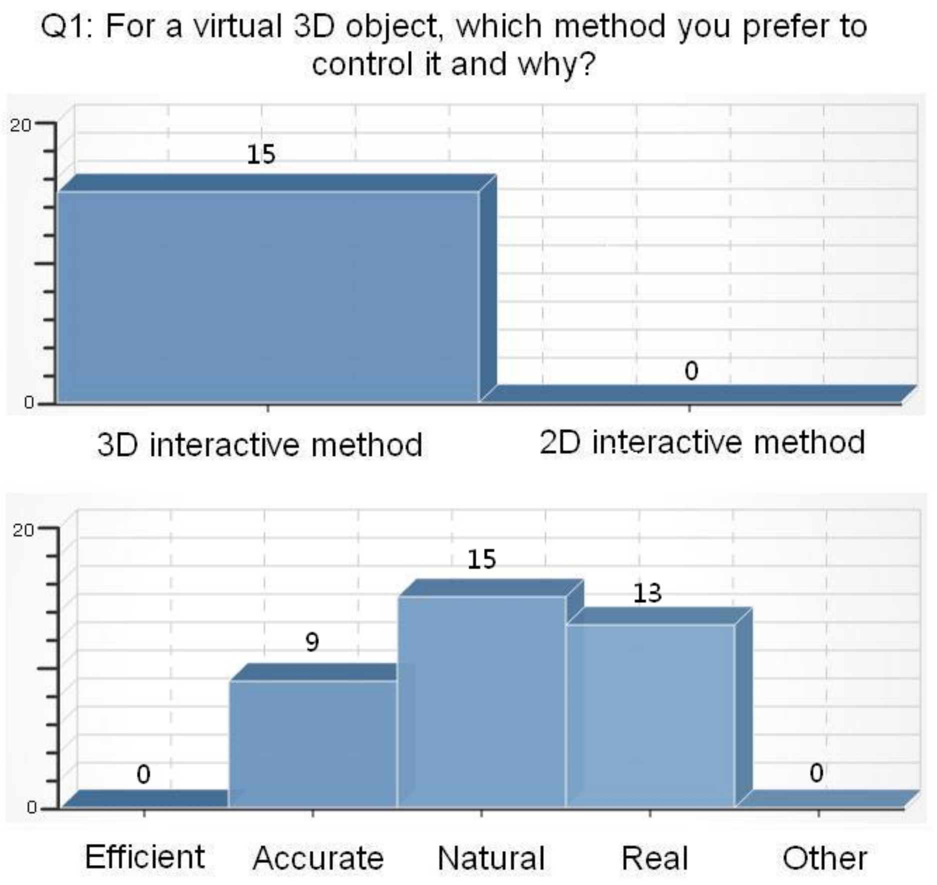


Figure 34: Answers of the first question in part two

Figure 34 shows that all participants preferred to use 3D interactive method to control a 3D virtual object, instead of 2D interactive method. And about the reason for their selection, all fifteen participants believed that 3D interactive method is more natural than 2D interactive method, and thirteen participants (about 87%) simultaneously believed that 3D interactive method is more real, and nine participants (60%) thought that this method is more accurate than 2D method but there was no participant who considered that 3D method is faster.

The second question in part two was about haptic interaction, and the answers to this question are shown in Figure 35.

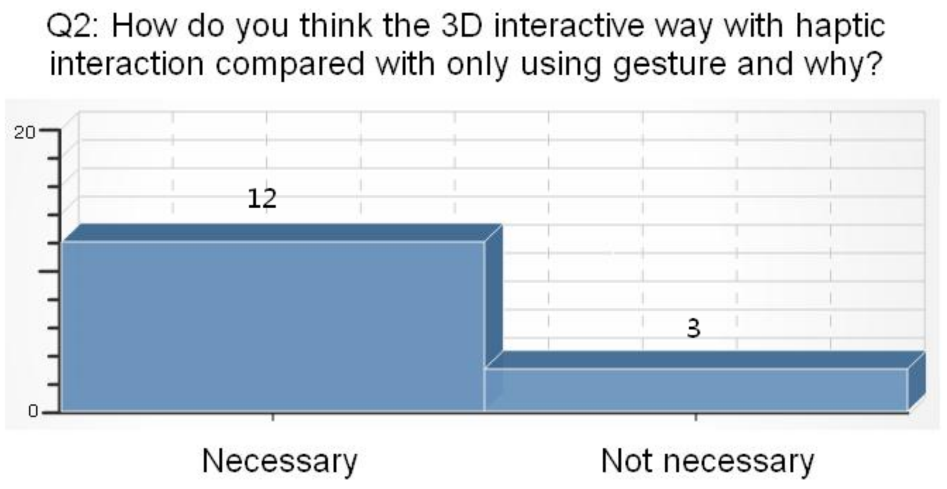


Figure 35: Answers of the second question in part two

The answers of the second question was clear as well: twelve participants (80%) believed that haptic interaction is necessary in a 3D interactive system, and only three participants (20%) thought that haptic interaction is not necessary. The main reason for participants who believed that haptic interaction is necessary is that haptic interaction can make virtual objects perceptible and real. Other three participants believed that using gestures is more convenient when they wanted to control the virtual object, and among them, one participant had also mentioned that the force feedback of virtual object may possibly hurt user's hand due to the big force feedback.

The next question of part two was for graphical user interface, and the answers to this question are shown in the Figure 36.

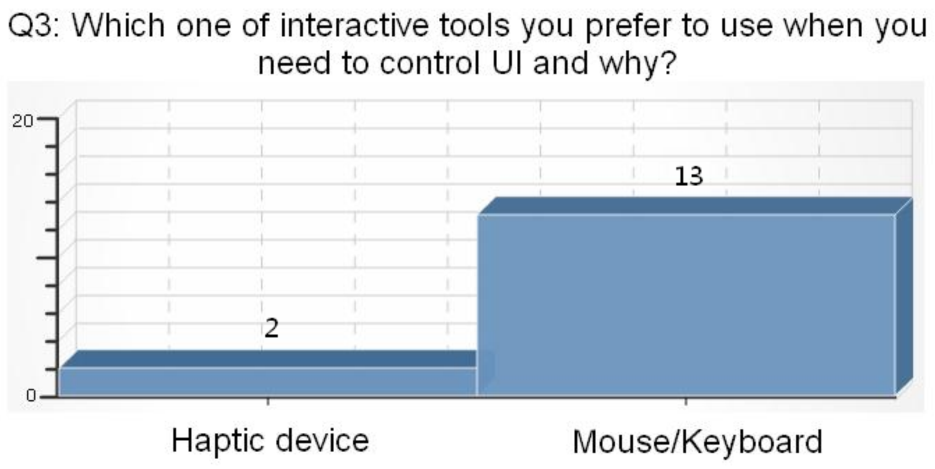


Figure 36: Answers of the third question in part two

The answers of participants to this question were surprising. Although all participants preferred to use 3D interactive method, only two participants (about 13%) liked to use a haptic device to interact with the 3D cube UI and others (about 87%) preferred to use a mouse and a keyboard. According to the answers of participants, the main reason is that most of participants could not bear the low efficiency and inconvenience of the haptic device when they wanted to interact with the UI, especially the slider bar. And other two participants who had different option liked the way of using force to rotate the UI.

The fourth question was to compare the two interactive systems in this project, and the answers to this question are shown in Figure 37.

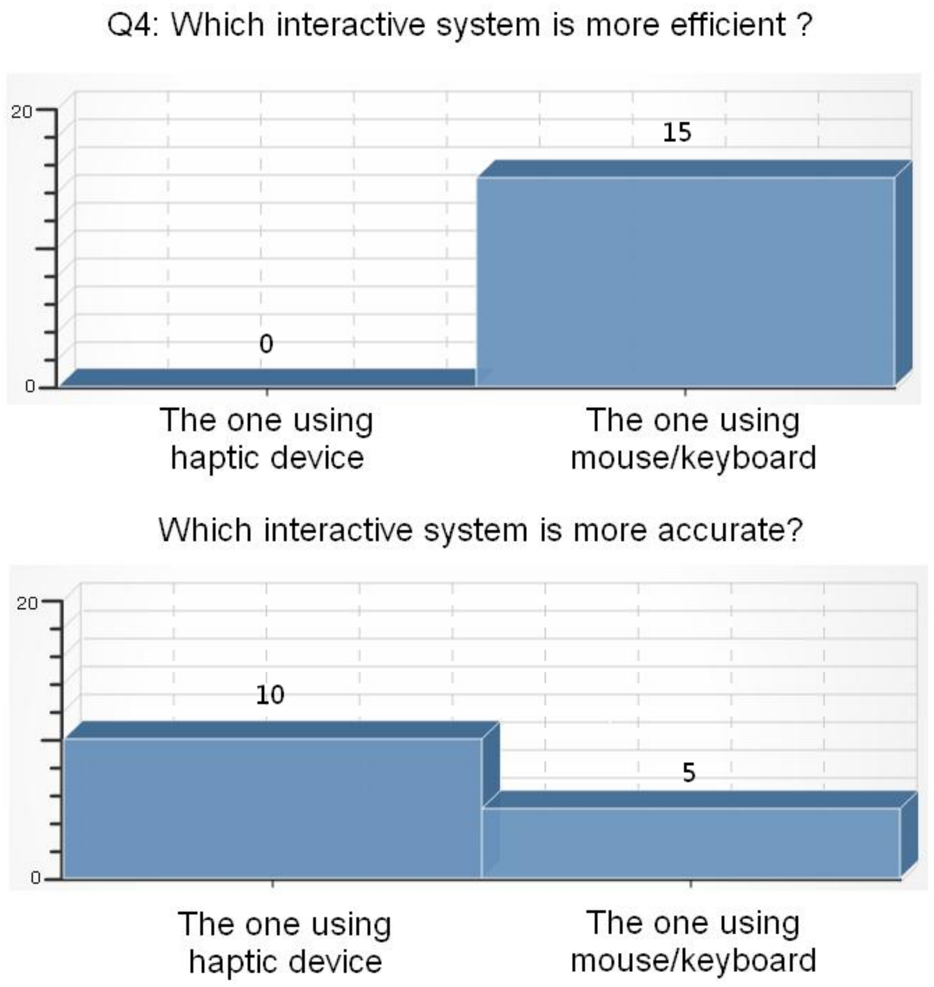


Figure 37: Answers of the fourth question in part two

All users believed that the 2D interactive method was faster than the 3D interactive method, but for accuracy, ten participants (about 67%) thought that using a haptic device was more accurate, and only five participants (about 33%) believed the 2D interactive method was still more accurate.

The fifth question was to investigate the main reason for the low efficiency and low accuracy of the fourth question. Based on the answers of participants, the main reason causing the low efficiency of interactive system with haptic devices was that the third dimension Z is difficult to be estimated for users through the normal two-dimensional screen of computer, and also the current graphical user interface is unsuitable to be controlled by the haptic device. The main reason causing the high accuracy of interactive system with haptic devices was that in addition to the visual feedback, haptic interaction could provide a force feedback which may improve the accuracy of an interactive system. The reason of those five participants who thought that using a mouse and a keyboard has a high accuracy was due to their high proficiency in using a mouse and a keyboard.

The last question was a comprehensive evaluation for the problem of the current haptic interaction. The most serious problems all participants mentioned were low efficiency and inconvenience. The low efficiency was mainly due to the control of the UI and the offset of 2D screen, and the inconvenience mainly came from the use of haptic devices. The improved methods from the answers of participants mainly included the design of new haptic devices, new user interfaces, new 3D imaging systems and so on.

In Chapter 6, we will discuss the results of this user study compared with some previous works, and find out the way for the future development of haptic technology in human-computer interaction.

6. Discussion

Maybury and Wahlster [1998] mentioned that haptic interaction has six main benefits, which are efficiency, redundancy, perceptibility, naturalness, accuracy and synergy. Among them, accuracy, efficiency and naturalness are three important evaluation parameters for an interactive system [Lamming and Newman, 1995]. Based on the results of the user study, we will discuss these three benefits compared with previous studies and also consider a method for improving current haptic technology.

First of all, by adding haptic interaction in a multimodal interactive system, it is widely accepted that the accuracy of interaction would be improved. For instance, MacGregor and Thomas [2001] studied the use of haptic mouse for some common computer tasks. According to their experimental results, the kinesthetic haptic feedback improved the detection rate of errors which means users made fewer errors with the help of haptic feedback. So, this study demonstrated that haptic interaction in a multimodal interactive system can improve the accuracy of interaction. In the user study of this thesis, the same conclusion could be made. Most participants of the user study believed that the 3D interactive system with haptic interaction has higher accuracy than normal 2D interactive system, especially in the tasks of testing information showing function and orbit-changing function. For example, in the task of orbit-changing function, when participants employed a haptic device to move one of the planets, the force feedback gave a clear signal to user for the selection, and all participants thought that this force feedback improved the accuracy of interaction between them and the virtual planet. Therefore, we may conclude that to interact with a virtual object in a multimodal interactive environment, haptic feedback could provide a great help in improving the accuracy of interaction.

Secondly, the efficiency of interactive system with haptic interaction has been studied as well, but the conclusions are different. For example, Oakley and his colleagues [2000] used touch to reduce visual overload in the environment of normal 2D desktop. They found out that the use of haptic interaction could not help to reduce the completion time for given tasks, but the errors made by participants were significantly reduced. This experimental result is similar to the result of this thesis. According to the results of the user study, most users preferred to use the novel 3D interactive method with haptic interaction to interact with 3D virtual objects even if they did not have any previous idea or knowledge of haptic interaction (based on their answers to the questions of the first part of the questionnaire), and the main reason is due to the high accuracy, naturalness and perceptibility of haptic interaction in a 3D interactive environment, but there was no participant who believed that haptic interaction in the project of this thesis could improve the efficiency of interaction, and on the contrary, most participants through that the 3D interactive system using a haptic device has low efficiency. Therefore, both studies have shown a similar result that haptic interaction as an additional interactive method may not improve the efficiency of interaction. However, this result cannot prove that ‘efficiency’ is not one of the benefits of haptic interaction.

Sallnäs and her colleagues [2000] also conducted an experimental study for haptic interaction used in a collaborative desktop virtual environment. They let two participants as a group simultaneously feel and manipulate dynamic objects in a shared desktop virtual environment. It was found out that the use of force feedback significantly improved the performance of participants to solve the given tasks. This result means that the accuracy of solving the given tasks was improved during the experiment and the task completion time was reduced as well. In order to figure out the reason to cause the difference of these two results, the experimental procedures and methods of two different studies have to be investigated.

On one hand, Oakley et al.’s [2000] experiment was to test the different haptic effects on a standard graphical button and also a scroll bar. These objects are the traditional components of a 2D graphical user interface. This illuminates that adding haptic feedback to these 2D GUI components may not improve the efficiency of interaction, but could improve the accuracy of interaction. On the other hand, Sallnäs et al. [2000] tested regular 3D cubes and the experiment was to let participants arrange these cubes in a 3D virtual space. The results of their experiment proved that haptic interaction with 3D objects has more advantages, which not only may improve the accuracy of interaction but also could improve its efficiency. This result has been verified again by the user study of this thesis.

In this thesis project, some functions of the project were integrated into a graphical user interface, such as the speed-changing and size-changing functions. This UI is modeled as a 3D cube attached with many 2D components, mainly including press buttons and slider bars. Other functions of the project had been directly implemented into the 3D models of the solar system. This situation leads to a composite conclusion including the results of both Oakley et al. and Sallnäs et al.. Firstly, in the task of interacting with 2D GUI components, the user study showed that not only haptic feedback could not improve the efficiency of interaction, but also 3D interactive environment causes the difficulty for interacting with these 2D components. This is the main reason that most participants in the user study thought that the 3D interactive system using a haptic device has low efficiency. Secondly, there were still two participants who liked to use a haptic device to interact with the UI. They liked to use a haptic device to directly interact with the 3D UI cube, instead of those attached 2D UI components. Therefore, we may conclude that haptic feedback may improve the efficiency of interaction when the interactive objects are 3D virtual objects in a 3D interactive environment, and for 2D virtual objects in a 3D interactive environment, haptic feedback is not the best solution for improving the efficiency of interaction under the current human-computer interactive technology.

Thirdly, in addition to the accuracy and efficiency, the naturalness is the one of the most important benefits of haptic interaction. There are many previous studies showing the naturalness of haptic interaction. For instance, Lahav and his colleagues [2004] developed a haptic virtual environment for blind people to explore unknown spaces. In their study, the participants were all blind, which means that they lacked the crucial visual information, and the interactive tool was a joystick which gave force and audio feedbacks to participants. Their study showed that haptic feedback is very useful to humans to quickly master the navigation for an unknown space while they permanently or temporarily lose visual feedback. This study demonstrated that the naturalness of haptic interaction is the main reason to let participants quickly realize the situation of unknown spaces. In the project of this thesis, the situation is the almost same as Lahav et al.'s study that most participants were very easy to understand and learn the novel 3D interactive methods. For example, almost all participants used less than fifteen minutes to learn how to use a haptic device to interact with virtual objects, such as pulling and pushing 3D models to zoom in and out, touching virtual objects to select and so on. This result is mainly due to the naturalness of haptic interaction. Therefore, we may conclude that haptic interaction could largely improve the naturalness of interaction and provide more natural interactive methods such as grabbing, pulling, pushing and other hand movements.

Although there are many benefits of haptic interaction, current haptic technology still suffers a number of limitations. Saddik [2007] believed that the most serious problem to the current haptic technology is the limitations of haptic interfaces and devices, including price, weight, size, type, usability and other technical limitations. The user study of this thesis also has shown some similar results. For instance, most participants mentioned that the inconvenience of haptic devices is one of the main reasons causing the efficiency problem of 3D interactive system. They thought that the haptic device which has only one interactive point with virtual objects is the main limitation, and they preferred to use a haptic device with multiple interactive points. This suggestion, in my opinion, could be a good direction of the development of the future haptic devices. Simultaneously, most participants believed that developing more suitable haptic interfaces is necessary, for example, utilizing force to rotate 3D model is a good application in this project and also they liked to zoom in and out the 3D model by drawing the model. In addition to the problems of haptic interfaces and devices, the third suggestion of the participants is to develop a new imaging system. The current display for presenting the information of computer often is a traditional monitor which only has 2D screen, and in the 3D interactive environment, 2D screen will make users have visual error for the data along z-axis. Therefore, this 2D imaging display would cause users to waste lots of time on adjusting the position of the haptic device for touching virtual objects. Moreover, it is obvious that this visual error also affects the accuracy of a 3D interactive system. Based on the results of the user study, participants are all looking forward to the development of a new 3D imaging system without any monitor, but this is still a challenge to the current visual imaging technology. In sum, in order to improve haptic interaction, new haptic interfaces and devices and also new 3D imaging systems could be a good development direction for the future haptic interactive technology.

The user study of the thesis generally has shown two important results:

1. 3D interactive system with haptic interaction has more advantages than traditional 2D interactive system.
2. Haptic interaction is important and necessary in this 3D interactive system, and it has been demonstrated that haptic interaction may largely improve the accuracy and naturalness of interaction in a multimodal interactive system, and the efficiency is still a potential benefit for haptic interaction.

To sum up, as we know, the current method of human-computer interaction is generally a 2D way, and users can only use a device such as a mouse to interact with virtual objects on a 2D surface. This interactive method is just a temporary solution for the current interactive system. Following with the development of new technologies such as new 3D imaging systems and new haptic devices and interfaces, traditional monitors and the 2D interactive system will be gradually discarded, and a new 3D interactive system with haptic interaction will become the popular interactive system in the future.

7. Conclusion

The purpose of this thesis was to explore haptic interaction in current human-computer interaction. In order to demonstrate the benefits of haptic interaction, a multimodal interactive system has been implemented based on a visual 3D model of the solar system, and user study has been conducted for testing the performance of this interactive system.

Firstly, a complete background of haptic interaction has been introduced, and the relevant hardware and software systems are described in detail. Then, three historical studies about haptic interaction have been given for systematically understanding the research points and problems which should be noticed for the project of this thesis. The following sections began to introduce the implementation details of the visual 3D model of the solar system and its multimodal interactive system. The specification of the project was given first and then each function of interactive system has been described. At last, a small but organized user study has been conducted. This user study uses a questionnaire to collect the feedbacks and comments of users about the performance of the multimodal interactive system.

The user study showed that the design of 3D interactive methods with haptic interaction in this project was basically successful, and haptic feedback from virtual object not only made the interaction more natural but also increased the accuracy of interaction. Therefore, users preferred to use this natural 3D interactive system to interact with virtual objects. It has been proven that haptic interaction can largely improve the accuracy, perceptibility and naturalness of interaction in human-computer interaction. However, its drawback is obvious as well: the current main problem of this interactive system is the low efficiency which is mainly due to the limits of the existing haptic devices and the traditional imaging system and display. Besides these two factors, haptic user interfaces and tools are relatively rare compared with visual user interfaces, and more new user interfaces should be developed in this research field, in order to implement a fast and convenient 3D multimodal interactive system.

After all, following with the development of interactive technologies and devices, the traditional 2D interactive method using a mouse and a keyboard will be gradually quit the main stage of interaction, and as a new generation of interactive system, 3D interactive system with haptic interaction will become the popular interactive system for human-computer interaction. At that time, the interaction between humans and computers will enter into a new stage.

References

[Abbena et al., 2006] Elsa Abbena, Alfred Gray and Simon Salamon, *Modern Differential Geometry of Curves and Surfaces with Mathematics*, Chapman and Hall/CRC, 2006.

[Aoki et al., 2009] Takafumi Aoki, Hironori Mitake, Danial Keoki, Shoichi Hasegawa and Makoto Sato, Wearable haptic device to present contact sensation based on cutaneous sensation using thin wire. *ACE '09 Proceedings of the International Conference on Advances in Computer Entertainment Technology,* 2009, 115-122.

[Arfken et al., 2012] George B. Arfken, Hans J. Weber and Frank E. Harris, *Mathematical Methods for Physicists, Seventh Edition: A Comprehensive Guide*, Academic Press, 2012.

[Arjunan et al., 2006] Sridhar P. Arjunan , Hans Weghorn, Dinesh K Kumar and Wai C. Yau, Vowel recognition of English and German language using facial movement (SEMG) for speech control based HCI, *VisHCI '06 Proceedings of the HCSNet Workshop on Use of Vision in Human-Computer Interaction Australian Computer Society*, 2006.

[Avila and Sobierajski, 1996] Ricardo S. Avila and Lisa M. Sobierajski, A haptic interaction method for volume visualization, *Proceedings of the 7th conference on Visualization, IEEE*, 1996, 197-ff.

[Barnett et al., 2001] Michael Barnett, Sasha A. Barab and Kenneth E. Hay, The virtual solar system project: Student modeling of the solar system, *The Journal of College Science Teaching*, **30,** 5, 2001, 300-305.

[Baxter et al., 2001] Bill Baxter, Vincent Scheib, Ming C. Lin and Dinesh Manocha, DAB: interactive haptic painting with 3D virtual brushes, *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques,* 2001, 461-468.

[Boughen, 2003] Nicholas Boughen, Light wave 3d 7.5 Lighting, Wordware Publishing, Inc, 2003.

[Box model, 1999] CSS box model: http://www.w3schools.com/css/css_boxmodel.asp

[Brannan et al., 1999] David A. Brannan, Matthew F. Esplen and Jeremy J. Gray, *Geometry*, The Press Syndicate of the University of Cambridge, 1999.

[Brutzman and Daly, 2007] Don Brutzman and Leonard Daly, *X3D: Extensible 3D Graphics for Web Authors*, Morgan Kaufmann, 2007.

[Chouvardas et al., 2005] Vasilios G. Chouvardas , Amalia N. Miliou and Miltiadis K. Hatalis, Tactile displays: a short overview and recent developments, *5th International Conference on Technology and Automation*, 2005.

[Coren et al., 1998] Stanley Coren, Lawrence M. Ward and James T Enns, *Sensation and Perception*, Harcourt College Publishing, 1998.

[Cruz-Neira et al., 1992] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon and John C. Hart, The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM*, **35**, 6, 1992, 64 – 72.

[Dix et al., 2003] Alan Dix, Janet E. Finlay, Gregory D. Abowd and Russell Beale, *Human-Computer Interaction*, Prentice Hall, 2003.

[ElKoural and Singh, 2003] George ElKoural and Karan Singh, Handrix: Animating the human hand, *Eurographics/SIGGRAPH Symposium on Computer Animation*, 2003.

[Esfahani and Sundararajan, 2012] Ehsan Tarkesh Esfahani and V. Sundararajan, Classification of primitive shapes using brain-computer interfaces, *Computer-Aided Design*, **44**, 10, 2012, 1011-1019.

[Fisher et. al., 1987] S. S. Fisher, M. McGreevy, J. Humphries and W. Robinett, Virtual environment display system, *I3D '86 Proceedings of the 1986 Workshop on Interactive 3D Graphics*, 1987, 77-87.

[Francese et al., 2012] Rita Francese, Ignazio Passero and Genoveffa Tortora, Wiimote and Kinect: gestural user interfaces add a natural third dimension to HCI. *Proceedings of the International Working Conference on Advanced Visual Interfaces*, 2012, 116-123.

[Gazit et al., 2004] Elhanan Gazit, David Chen and Yoav Yair, Developing understanding of basic astronomical concepts by using a virtual solar system, *ICLS '04 Proceedings of the 6th International Conference on Learning Sciences, International Society of the Learning Sciences*, 2004, 601-601.

[Gibson and Zhan, 2008] Lan Gibson and Gao Zhan, Finite element simulation of the spine with haptic interface. *Proceedings of the 2nd International Convention on Rehabilitation Engineering & Assistive Technology*, 2008, 20-25.

[Gottschalk et al., 1996]S. Gottschalk, M. C. Lin, and D. Manocha, OBB-Tree: A hierarchical Structure for Rapid Interference Detection, *SIGGRAPH '96 Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, 1996, 171-180.

[Huang et al., 2003] G. Huang, D. Metaxas and M. Govindaraj, Feel the "fabric": an audio-haptic interface, *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003, 52-61.

[H3D Manual, 2009] H3D official Manual, 2009, available as http://www.h3dapi.org/uploads/api/H3DAPI_2.1/docs/H3D%20API%20Manual.pdf

[Jackson, 2011] Cody Jackson, *Learning to Program Using Python*, Cody Jackson Publisher, 2011

[Jenong and Gluck, 2002] Wooseob Jeong and Myke Gluck, Multimodal bivariate thematic maps: Auditory and haptic display. *Proceedings of the American Society for Information Science and Technology*, **39**, 1, 2002, 279-283.

[Kadlecek, 2011] Petr Kadlecek, Overview of current developments in haptic APIs, *The 15th Central European Seminar on Computer Graphics,* 2011.

[Kortum, 2008] Philip T. Kortum, *HCI Beyond the GUI: Design for Haptic, Speech, Olfactory, and Other Nontraditional Interfaces.* Morgan Kaufmann, 2008.

[Kuhlman, 2011] Dave Kuhlman, *A Python Book: Beginning Python, Advanced Python, and Python Exercises,* Platypus Global Media, 2011.

[König et al., 2000] Henry König, Jochen Schneider and Thomas Strothotte, Haptic exploration of virtual buildings using non-realistic haptic rendering. *Proceedings of the 7th International Conference on Computers Helping People with Special Needs*, 2000, 377-384.

[Lahav and Mipduser, 2004] Orly Lahav and David Mioduser, Exploration of unknown spaces by people who are blind using a multi-sensory virtual environment, *Journal of Special Education Technology,* **19**, 3, 2004.

[Lamming and Newman, 1995] Michael G. Lamming, William M. Newman, *Interactive System Design*, Addison Wesley, 1995.

[Lin and Otaduy, 2008] Ming C. Lin and Miguel Otaduy, *Haptic Rendering: Foundations, Algorithms and Applications,* A K Peters, 2008.

[MacGregor and Thomas, 2001] Carolyn MacGregor and Alice Thomas, Does multi-modal feedback help in everyday computing tasks? *EHCI '01 Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction,* 2001, 251-262.

[Maybury and Wahlster, 1998] Mark T. Maybury and Wolfgang Wahlster, *Readings in Intelligent User Interfaces,* Morgan Kaufmann Publishers, 1998.

[Minamizawa et al., 2007] Kouta Minamizawa, Hiroyuki Kajimoto, Naoki Kawakami and Susumu Tachi, A wearable haptic display to present the gravity sensation - preliminary observations and device design. *WHC '07 Proceedings of the Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems,* 2007, 133-138.

[Moller, 2002] Aage R. Moller, *Sensory Systems: Anatomy Physiology and Pathophysiology*, Academic Press, 2002.

[Oakley et al., 2000] Ian Oakley, Marilyn Rose McGee, Stephen Brewster and Philip Gray, Putting the Feel in 'look and feel', *Proceedings of the SIGCHI conference on Human Factors in Computing Systems,* **2,** 1, 2000.

[Prata, 2004] Stephen Prata, *C++ Primer Plus*, Publisher of Sams, 2004.

[Python tutorial, 1990] Python tutorial online website, http://docs.python.org/2/tutorial/index.html

[Raisamo, 2011] Roope Raisamo, Haptic user interface, University of Tampere, School of Information Science, TAUCHI, Course Reports, 2011.

[Rudmann et al., 2003] Darrell S. Rudmann, George W. McConkie, and Xianjun Sam, Zheng, Eye tracking in cognitive state detection for HCI, *Proceedings of the 5th International Conference on Multimodal Interfaces, ACM, New York*, 2003

[Saddik, 2007] El Saddik, The potential of haptics technologies, *IEEE, Instrumentation & Measurement Magazine,* **10**, 1, 2007, 10-17.

[Sallnäs et al., 2000] Eva-Lotta Sallnäs, Kirsten Rassmus-Gröhn and Calle Sjöström, Supporting presence in collaborative environments by haptic force feedback, ACM Transactions on Computer-Human Interaction, **7,** 4, 2000.

[Shreiner et al., 2007] Dave Shreiner, Mason Woo, Jackie Neider and Tom *DavisOpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R).* Addison-Wesley Professional, 2007.

[Smith et al., 2007] T. A. Smith, A. L. Barrow, R. G. Barrow Reading and W. S. Harwin, A novel haptic interface for navigation in large volume environments, *WHC '07 Proceedings of the Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems IEEE,* 2007, 482-487.

[Sundstedt, 2010] Veronica Sundstedt, Gazing at games: using eye tracking to control virtual characters, *SIGGRAPH '10 ACM SIGGRAPH 2010 Courses*, *ACM New York*, 2010.

[Uicker et al., 2010] John Uicker, Gordon Pennock, and Joseph Shigley, *Theory of Machines and Mechanisms,* Oxford University Press, 2010.

[X3D tutorial, 1999] X3D official tutorial online website, http://www.web3d.org/x3d/learn/tutorial/#tutorials.

Appendix

Haptic devices:

Omega series of Force Dimension

Phantom series of Sensable

Questionnaire

Please answer the questions before test.

1. Do you often use personal computer and its basic peripheral devices such as mouse and keyboard?

□ Often □ Not often □ Never

2. Do you have any experience of using 3D interactive tool such as haptic device before?

□ Yes □ No

3. Do you have any knowledge about haptics and haptic interaction?

□ Yes □ No

Please answer the questions after test.

1. For a virtual 3D object, which method you prefer to control it and why?

□ 3D interactive method □ 2D interactive method

Reasons: □ Efficient □ Accurate □ Natural □ Real □ Other_____

2. How do you think the 3D interactive way with haptic interaction compared with only using gesture and why?

□ Necessary □ Not necessary

Reason: ____

3. Which one of interactive tools you prefer to use when you need to control UI and why?

□ Haptic device □ Mouse/Keyboard

Reason: ____

4. Which interactive system is more efficient and which interactive system is more accurate?

Efficiency: □ The one using haptic device □ The one using mouse/Keyboard

Accuracy: □ The one using haptic device □ The one using mouse/Keyboard

5. What is the main problem of the less-efficient interactive system and the less-accurate system?

Answer: ____

6. In your opinion, what is the current problem for interactive system with haptic interaction and how to improve it?

Answer: ____