Chapter 4 System Design and Implementation

In this chapter, we introduce the input and output hardware used in DIVINE. And then some issues about implementation and design will be discussed in remainder sections.

4.1. Virtual Hand Model

As mentioned in section 2.3.3, the virtual hand model can be the feedback by which the user can be aware the position and orientation of the hand in 3D environment when the user moves his hand in real environment. The virtual hand model consists of simple 3D geometries and it can interact with visualized metaphors in the scene. Figure 4-1 shows the virtual hand model used in DIVINE.

Figure 4-1: The virtual hand model used in DIVINE.

4.2. VR Hardware Used in DIVINE

DIVINE is 3D visualization system. For more intuitive and efficient interaction as

described in section 2.3, the device-independent framework in DIVINE provides alternatives to use both traditional input devices, such as mice and keyboards, and VR input devices, such as data gloves and trackers. A head-mounted display can provide stereoscopic views that make users have sense of presence. We will have more discussion about these devices used in DIVINE in following sections.

4.2.1. Data Glove and Hand Tracker

In DIVINE, we use the data glove produced by Fifth Dimension Technology (5DT, http://www.5dt.com/) as shown in Figure 2-6. Referring to Figure 4-2, the data glove consists of a Lycra glove with embedded fiber optic sensors. These sensors are linked to the computer via an optoelectronics unit, a ribbon cable and an interface box. With connecting to the RS-232 serial port via the interface cable, the data glove is platform independent [47].

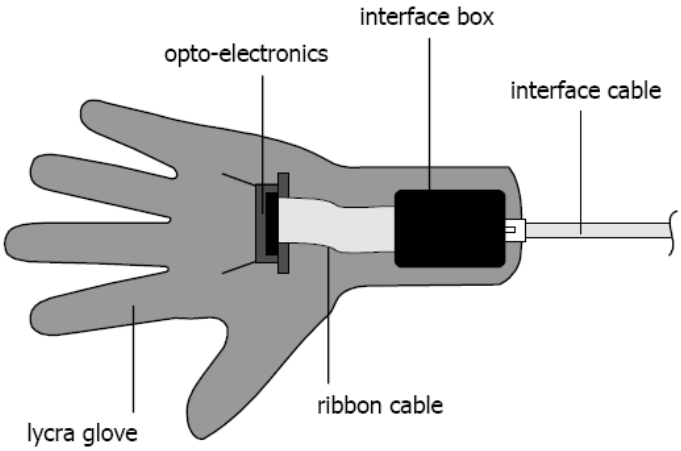

Figure 4-2: Components of the data glove.

The most important parts of the data glove are 14 sensors that measure finger

flexure (2 sensors per finger) as well as the abduction between fingers. The sensor mappings of the data glove are shown in Figure 4-3. Notice that sensors marked as 14 and 15 in the figure are not yet implemented.

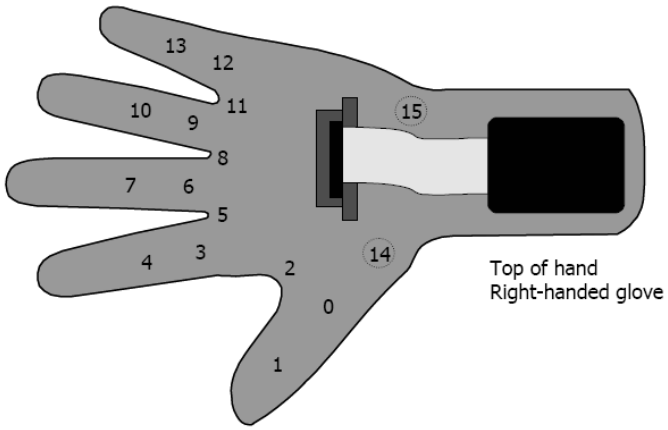

Figure 4-3: Sensor mappings of the data glove.

With the SDK provided by the data glove, we can get scaled records (between 0 and 1) of the sensors in DIVINE and make our virtual hand model flex fingers according to these records.

The data glove is responsible for finger flexure detection, while the hand tracker is used to get the position and orientation records of the hand. The 6-DOF tracker used in DIVINE is the Ascension Technology Corporation (http://www.ascension-tech.com/) "Flock of birds" motion tracking system. Figure 4-4 shows the components of the tracking system. In the figure, the left is the transmitter; the top right is the electronics unit, and the bottom right is the sensor. The transmitter can track the position and orientation of the sensor and send these records to our application. With attaching the

sensor to the data glove as demonstrated in Figure 4-5, we can track the motion and the finger flexure of the user's hand in DIVINE for further interaction.

Figure 4-4: A Flock of Birds system.

Figure 4-5: A data glove with a tracker

4.2.2. Head-Mounted Display and Head Tracker

We have introduced HMD in section 2.3.2. Within various kinds of HMD, what we used in DIVINE is the I-glasses head-mounted display as shown in Figure 2-7. As the introduction of the product, the monitor glasses contain two LCD screens, one in front of each eye, which enable the viewing of flicker-free VGA content. The HMD also provides adjustable headphones for stereo audio. With two screens used in the HMD, we can have stereoscopic viewing.

The view point of an HMD will always look into the same direction if it lacks orientation control. However, passive control, such as using the keyboard or mouse, for a user to change the orientation is inadequate because a user wearing an HMD is difficult to look and even to use these input devices. For this reason, we attach a tracker to our HMD. When the user turns the head, the tracker gets the orientation

information actively and makes the viewing direction in virtual environment change as where the user looking at. InterSense's InterTrax^2 is a 3-DOF tracking system and can be easily attached to most known personal display devices, i.e., HMD. Figure 4-6 shows the InterTrax^2 head tracker and Figure 4-7 shows the HMD attached with an InterTrax^2 tracker.

Figure 4-6: A InterTrax^2 head tracker.

Figure 4-7: A HMD with a tracker.

4.3. Dynamic Gesture Recognition

While using a virtual hand to manipulate visualized metaphors in DIVINE, it is important to recognize what gesture the virtual hand is. For example, a user may want to drag some object and move it in the virtual environment by grabbing the object. As this scenario, the system has to recognize the "grabbing" gesture which triggers the drag action of the target object.

A gesture is a sequence of poses [48]. Referring to Figure 4-8 (a), a grabbing gesture is the sequence from a flat pose to a fist pose. If the system determines the gesture only in the time that the gesture occurs instead of tracking back previous poses,

the gesture may be misused. Considering the previous case, it may be a pointing pose before the fist pose rather than a flat one as shown in Figure 4-8 (b).

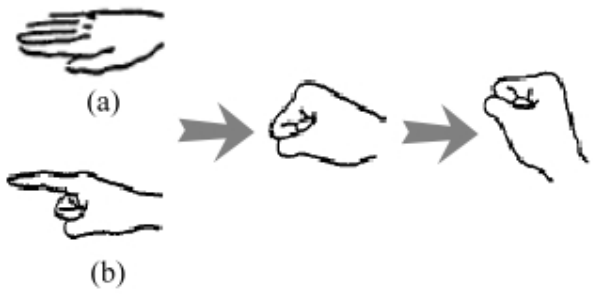

Figure 4-8: (a) A normal grabbing gesture; (b) An error grabbing gesture.[2]

The 5DT data glove has defined 16 hand poses listed in [47], including fist, flat and pointing pose. While recognizing gestures like grabbing in DIVINE, the system will check the hand pose as well as the flexure trend of fingers.

4.4. Collision Detection

Collision detection is one of the important topics in 3D games and VR, which is used in two main contexts – it detects any collision between objects which an application allows to collide and it is used in path planning where an animation agent may need to be sent along a path that keeps it maximally clear of other objects. *Bounding volumes* (BV) intersection detection, which rapidly tests for an intersection between two pairs of bounding volumes, is one of the fundamental approaches for collision detection. However, the detection result may be wrong if the BVs intersect

[2] These pictures of hand poses are captured from [48].

while the objects themselves are not intersecting. Figure 4-9 shows various bounding volume selections in 2D analogies [49].

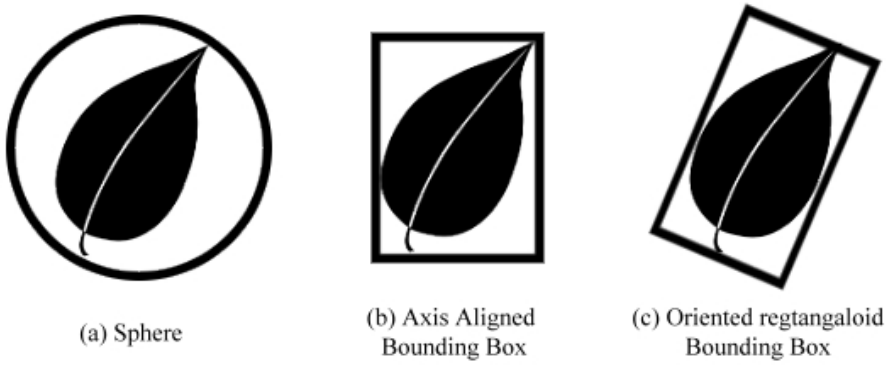

Figure 4-9: Bounding volumes in 2D analogue: (a) sphere; (b) AABB; (c) OBB.

In DIVINE, we detect the occurrence of collisions between our virtual hand model and visualized metaphors in the scene. By the collision detection working together with the dynamic gesture recognition, we can use our virtual hand model to manipulate visualized metaphors directly.

With the APIs provided by OGRE, we can do simple collision detection in DIVINE. The abstract class - *SceneQuery* in OGRE performs a query on a scene, i.e., to retrieve a list of objects and/or geometry sections which are potentially intersecting a given region. Because the query results are generated based on *bounding volumes* of the geometries, we have to filter the results further for more accuracy. The derived classes – *IntersectionSceneQuery*, *RaySceneQuery*, and *RegionSceneQuery* classes, can provide different scene query results for different purposes. In our virtual hand model, there is an invisible region scene query object in the end of every finger, which

returns a list of objects that may have collision with our hand model. Then, there will be one object (usually the first one) selected from the list as the action target for further interaction.

4.5. Abstract Action Code and Input Interpreter

The “device-independent” framework is the main contribution in the thesis. To implement the framework, abstract action codes plays an important part in the system. Based on the framework, different input devices can be used to interact with the same visualized metaphor. Abstract action codes are used to isolate the actions affected on visualized metaphors and input devices. For example, the visualized metaphor will receive the same action code of “picking” while we pick it by either doing a pointing with a data glove or clicking the left button of a mouse. Thus, the metaphor does not need to distinguish the input sources. Table 4-1 summarizes the existing abstract action codes. In the table, we consider that it is different between ‘picking’ and ‘selecting’ a VM object. When we are ‘picking’ a VM object, there are actually collisions occurred between the VM object and our virtual hand. However, VM objects are ‘selected’ while the objects are in the area that we group them by drawing a circle or a rectangle.

Comparatively, abstract action codes are more constant than various inputs of input devices and will almost not be changed in the future. The change of the interface means the recompilation and change for the VM programming.

The same action that affects on the metaphor may be triggered by doing different actions produced by various input devices. The abstract action codes and by which input device they are triggered are listed in Table 4-2. According to the defined abstract action codes, visualization metaphor programmers can design various reaction of VM independently without considering various input devices.

Abstract action code	Description about the action (using mouse for example)
AC_ACTION_PICKED	To pick some metaphor up at the cursor's location.[3]
AC_ACTION_SELECTED	To select some metaphor at the cursor's location.
AC_ACTION_RELEASED	To release some metaphor from the other mode at the cursor's location.
AC_ACTION_MOVED	The cursor is moving with some metaphor.
AC_ACTION_ENTERED	The cursor entered into some metaphor.
AC_ACTION_EXITED	The cursor exited some metaphor.
AC_ACTION_DRAGGED	The cursor is dragging with some metaphor or only in dragging mode without any metaphor.
AC_ACTION_DRAGENTERD	The cursor is in dragging mode and entering another metaphor.
AC_ACTION_DRAGEXITED	The cursor is in dragging mode and exiting some metaphor.
AC_ACTION_DRAGMOVED	The cursor is in dragging mode and moving around.

Table 4-1: The existing abstract action code in DIVINE.

For various input devices satisfying the same set of abstract action codes, there is an input interpreter for every input device, such as mouse interpreter, keyboard

[3] A cursor can refer to a virtual hand model or a mouse cursor.

interpreter, and data glove interpreter. The input interpreter is responsible for translating the input command of to corresponding abstract action code which is received by the metaphor. For example, we define "Ctrl + P" on the keyboard as a "picking" command and it will be interpreted as the "AC_ACTION_PICKED" abstract action code for the metaphor. The metaphor then is informed that it has been picked. If we use another input device in the future, we only have to develop a corresponding input interpreter rather than modifying the whole input handling structure.

Abstract action code	**Triggered by**		
	Keyboard	**Mouse**	**Data glove**
AC_ACTION_PICKED	Press "Ctrl + P"	Click left button at some metaphor.	Make a gesture of pointing.
AC_ACTION_SELECTED	Press "Ctrl + S"	Click right button at some metaphor.	Make a gesture of pointing.
AC_ACTION_RELEASED	N/A[4]	Click the button without metaphor.	Make a gesture of flat.
AC_ACTION_MOVED	N/A	Move the mouse cursor.	Move the hand model.
AC_ACTION_ENTERED	N/A	The mouse cursor entered.	The hand model entered.
AC_ACTION_EXITED	N/A	The mouse cursor exited.	The hand model exited.
AC_ACTION_DRAGGED	N/A	The mouse cursor dragged.	The hand model dragged.
AC_ACTION_DRAGENTERD	N/A	The mouse	The hand model

[4] N/A denotes that this action code is not implemented in this device.

		cursor dragged and entered.	dragged and entered.
AC_ACTION_DRAGEXITED	N/A	The mouse cursor dragged and exited.	The hand model dragged and exited.
AC_ACTION_DRAGMOVED	N/A	The mouse cursor dragged and moved.	The hand model dragged and moved.

Table 4-2: Abstract action codes that currently triggered by input devices.