Module 2

Syllabus: Input and Interaction: Interaction, Input devices, Clients and Servers, Display Lists, Display Lists and Modeling, Programming Event Driven Input, Menus.

3.1 INTERACTION

One of the most important advances in computer technology was enabling users to interact with computer displays. More than any other event, Ivan Sutherland's Sketchpad project launched the present era of interactive computer graphics. The basic paradigm that he introduced is deceptively simple. The user sees an image on the display. She reacts to this image by means of an interactive device, such as a mouse. The image changes in response to her input. She reacts to this change, and so on. Whether we are writing programs using the tools available in a modern window system or using the human computer interface in an interactive museum exhibit, we are making use of this paradigm.

Although rendering is the prime concern of most modern APIs, including OpenGL, interactivity is an important component of most applications. OpenGL, however, does not support interaction directly. The major reason for this omission is that the system architects who designed OpenGL wanted to increase its portability by allowing the system to work in a variety of environments. Consequently, window and input functions were left out of the API. Although this decision makes renderers portable, it makes discussions of interaction that do not include specifics of the window system more difficult. In addition, because any application program must have at least a minimal interface to the window environment, we cannot avoid such issues completely if we want to write complete, nontrivial programs.

We can avoid such potential difficulties by using the GLUT toolkit. This toolkit provides the minimal functionality that is expected on virtually all systems, such as opening of windows, use of the keyboard and mouse, and creation of pop-up menus. We adopt this approach, even though it may not provide all the features of any particular windowing system and produces code that neither makes use of the full capabilities of any particular window system nor proves as efficient as code written for a particular environment. However, writing code for the standard window systems is based on the principles that we can illustrate most simply using GLUT.

We use the term window system, to include the total environment provided by systems such as the X Window System, Microsoft Windows, and the Macintosh Operating System. Graphics programs that we develop will render into a window within one of these environments. The terminology used in the window system literature may obscure the distinction between, for example, an X window and the OpenGL window into which our graphics are rendered. However, you will usually be safe if you regard the OpenGL window as a particular type of window on your system that can display output from OpenGL programs. Our use of the GLUT toolkit will enable us to

avoid the complexities inherent in the interactions among the window system, the window manager, and the graphics system.

3.2 INPUT DEVICES

3.2.1 Physical Input Devices

The pointing device allows the user to indicate a position on a display and almost always incorporates one or more buttons to allow the user to send signals or interrupts to the computer. The keyboard device is almost always a physical keyboard but can be generalized to include any device that returns character codes. For example, a tablet PC uses recognition software to decode the user's writing with a stylus but in the end produces character codes identical to those of the standard keyboard.

We will use the American Standard Code for Information Interchange (ASCII) in our examples. ASCII assigns a single unsigned byte to each character. Nothing we do restricts us to this particular choice, other than that ASCII is the prevailing code used. Note, however, that other codes, especially those used for Internet applications, use multiple bytes for each character, thus allowing for a much richer set of supported characters.

The mouse (Figure 3.1) and trackball (Figure 3.2) are similar in use and often in construction as well. When turned over, a typical mechanical mouse looks like a trackball. In both devices, the motion of the ball is converted to signals sent back to the computer by pairs of encoders inside the device that are turned by the motion of the ball. The encoders measure motion in two orthogonal directions.

There are many variants of these devices. Some use optical detectors rather than mechanical detectors to measure motion. Small trackballs are popular with portable computers because they can be incorporated directly into the keyboard. There are also various pressure-sensitive devices used in keyboards that perform similar functions to the mouse and trackball but that do not move; their encoders measure the pressure exerted on a small knob that often is located between two keys in the middle of the keyboard.

FIGURE 3.1 Mouse.

FIGURE 3.2 Trackball.

A typical data tablet (Figure 3.4) has rows and columns of wires embedded under its surface. The position of the stylus is determined through electromagnetic interactions between signals traveling through the wires and sensors in the stylus. Touch-sensitive transparent screens that can be placed over the face of a CRT have many of the same properties as the data tablet. Small, rectangular, pressure-sensitive touchpads are embedded in the keyboards of most portable computers. These touchpads can be configured as either relative- or absolute positioning devices. Some are capable of detecting simultaneous input from two fingers touching different spots on the pad and can use this information to enable more complex behaviors.

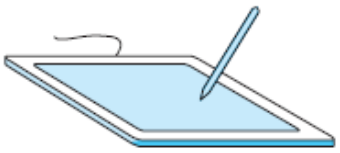

FIGURE 3.4 Data tablet.

The lightpen has a long history in computer graphics. It was the device used in Sutherland's original Sketchpad. The lightpen contains a light-sensing device, such as a photocell (Figure 3.5). If the lightpen is positioned on the face of the CRT at a location opposite where the electron beam strikes the phosphor, the light emitted exceeds a threshold in the photodetector and a signal is sent to the computer. The light pen was originally used on random scan devices so the time of the interrupt could easily be matched to a piece of code in the display list, thus making the light pen ideal for selecting application-defined objects. With raster scan devices, the position on the display can be determined by the time the scan begins and the time it takes to scan each line. Hence, we have a direct-positioning device. The lightpen is not as popular as the mouse, data tablet, and trackball. One of its major deficiencies is that it has difficulty obtaining a position that corresponds to a dark area of the screen. However, tablet PCs are used in a manner that mimics how the light pen was used originally; the user has a stylus with which she can move randomly about the tablet (display) surface.

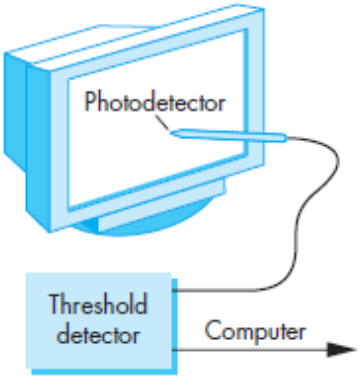

FIGURE 3.5 Lightpen.

One other device, the joystick (Figure 3.6), is particularly worthy of mention. The motion of the stick in two orthogonal directions is encoded, interpreted as two velocities, and integrated to identify a screen location. The integration implies that if the stick is left in its resting position, there is no change in the cursor position and that the farther the stick is moved from its resting position, the faster the screen location changes. Thus, the joystick is a variable-sensitivity device. The other advantage of the joystick is that the device can be constructed with mechanical elements, such as springs and dampers, that give resistance to a user who is pushing the stick. Such a mechanical feel, which is not possible with the other devices, makes the joystick well suited for applications such as flight simulators and game controllers.

FIGURE 3.6 Joystick.

For three-dimensional graphics, we might prefer to use three-dimensional input devices. Although various such devices are available, none have yet won the widespread acceptance of the popular two-dimensional input devices. A spaceball looks like a joystick with a ball on the end of the stick (Figure 3.7); however, the stick does not move. Rather, pressure sensors in the ball measure the forces applied by the user. The spaceball can measure not only the three direct forces (up–down, front–back, left–right) but also three independent twists. The device measures six independent values and thus has six degrees of freedom. Such an input device could be used, for example, both to position and to orient a camera. Other three-dimensional devices, such as laser scanners, measure three dimensional positions directly. Numerous tracking systems used in virtual reality applications sense the position of the user. Virtual reality and robotics applications often need more degrees of freedom than the two to six provided by the devices that we have described. Devices such as data gloves can sense motion of various parts of the human body, thus providing many additional input signals. Recently, in addition to being wireless, input devices such as Nintendo's Wii incorporate gyroscopic sensing of position and orientation.

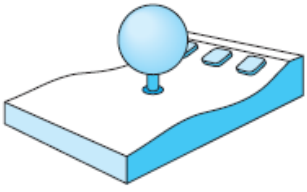

FIGURE 3.7 Spaceball.

3.2.2 Logical Devices

Some earlier APIs defined six classes of logical input devices. Because input in a modern window system cannot always be disassociated completely from the properties of the physical devices, OpenGL does not take this approach. Nevertheless, we describe the six classes briefly because they illustrate the variety of input forms available to a developer of graphical applications. We will see how OpenGL can provide the functionality of each of these classes.

1. **String**: A string device is a logical device that provides ASCII strings to the user program. This logical device is usually implemented by means of a physical keyboard. In this case, the terminology is consistent with that used in most window systems and OpenGL, which usually do not distinguish between the logical string device and a physical keyboard.
2. **Locator**: A locator device provides a position in world coordinates to the user program. It is usually implemented by means of a pointing device, such as a mouse or a trackball. In OpenGL, we usually use the pointing device in this manner, although we have to do the conversion from screen coordinates to world coordinates within our own programs.
3. **Pick:** A pick device returns the identifier of an object on the display to the user program. It is usually implemented with the same physical device as a locator, but has a separate software interface to the user program. In OpenGL, we can use a process called selection (Section 3.8) to accomplish picking.
4. **Choice:** Choice devices allow the user to select one of a discrete number of options. In OpenGL, we can use various widgets provided by the window system. A widget is a graphical interactive device, provided by either the window system or a toolkit. Typical widgets include menus, scrollbars, and graphical buttons. Most widgets are implemented as special types of windows. For example, a menu with n selections acts as a choice device, allowing us to select one of n alternatives. Widget sets are the key element defining a graphical user interface, or GUI.
5. **Valuators:** Valuators provide analog input to the user program. On some graphics systems, there are boxes or dials to provide valuator input. Here again, widgets within various toolkits usually provide this facility through graphical devices such as slide bars and radio boxes.
6. **Stroke:** A stroke device returns an array of locations. Although we can think of a stroke device as similar to multiple uses of a locator, it is often implemented such that an action, say, pushing down a mouse button, starts the transfer of data into the specified array, and a second action, such as releasing the button, ends this transfer.

3.2.3 Input Modes

The manner by which input devices provide input to an application program can be described in terms of two entities: a measure process and a device trigger. The measure of a device is what the device returns to the user program. The trigger of a device is a physical input on the device with which the user can signal the computer. For example, the measure of a keyboard should include a single character or a string of characters, and the trigger can be the Return or Enter key.

The application program can obtain the measure of a device in three distinct modes. Each mode is defined by the relationship between the measure process and the trigger. Once a measure process is started, the measure is taken and placed in a buffer, even though the contents of the buffer may not yet be available to the program. For example, the position of a mouse is tracked continuously by the underlying window system and a cursor is displayed regardless of whether the application program needs mouse input.

In request mode, the measure of the device is not returned to the program until the device is triggered. This input mode is standard in non graphical applications. For example, if a typical C program requires character input, we use a function such as scanf. When the program needs the input, it halts when it encounters the scanf statement and waits while we type characters at our terminal. We can backspace to correct our typing, and we can take as long as we like. The data are placed in a keyboard buffer whose contents are returned to our program only after a particular key, such as the Enter key (the trigger), is pressed. For a logical device, such as a locator, we can move our pointing device to the desired location and then trigger the device with its button; the trigger will cause the location to be returned to the application program. The relationship between measure and trigger for request mode is shown in Figure 3.8.

Sample-mode input is immediate. As soon as the function call in the user program is encountered, the measure is returned. Hence, no trigger is needed (Figure 3.9). In sample mode, the user must have positioned the pointing device or entered data using the keyboard before the function call, because the measure is extracted immediately from the buffer.

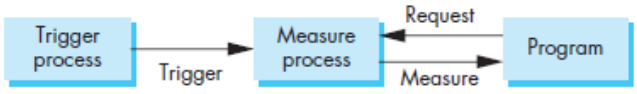

FIGURE 3.8 Request mode.

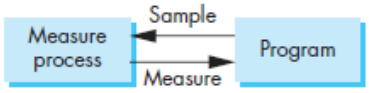

FIGURE 3.9 Sample mode.

One characteristic of both request- and sample-mode input in APIs that support them is that the user must identify which device is to provide the input. Consequently, we ignore any other information that becomes available from any input device other than the one specified. Both request and sample modes are useful for situations where the program guides the user but are not useful in applications where the user controls the flow of the program. For example, a flight simulator or computer game might have multiple input devices such as a joystick, dials, buttons, and switches most of which can be used at any time. Writing programs to control the simulator with only sample- and request-mode input is nearly impossible because we do not know what devices the pilot will use at any point in the simulation. More generally, sample- and request-mode input are not sufficient for handling the variety of possible human computer interactions that arise in a modern computing environment.

Our third mode, eventmode, can handle these other interactions. We introduce it in three steps. First, we show how event mode can be described as another mode within our measure–trigger paradigm. Second, we discuss the basics of client and servers where event mode is the preferred interaction mode. Third, we show an eventmode interface to OpenGL using GLUT, and we write demonstration programs using this interface.

Suppose that we are in an environment with multiple input devices, each with its own trigger and each running a measure process. Each time that a device is triggered, an event is generated. The device measure, including the identifier for the device, is placed in an event queue. This process of placing events in the event queue is completely independent of what the application program does with these events. One way that the application program can work with events is shown in Figure 3.10. The application program can examine the front event in the queue or, if the queue is empty, can wait for an event to occur. If there is an event in the queue, the program can look at the first event's type and then decide what to do. If, for example, the first event is from the keyboard but the application program is not interested in keyboard input, the event can be discarded and the next event in the queue can be examined.

3.3 CLIENTS AND SERVERS

We have looked at our graphics system as a monolithic box with limited connections to the outside world, rather than through our carefully controlled input devices and a display. Networks and multiuser computing have changed this picture dramatically, and to such an extent that, even if we had a single-user isolated system, its software probably would be configured as a simple client–server network.

If computer graphics is to be useful for a variety of real applications, it must function well in a world of distributed computing and networks. In this world, our building blocks are entities called servers that can perform tasks for clients. Clients and servers can be distributed over a network (Figure 3.11) or contained entirely within a single computational unit. Familiar examples of servers include print servers, which can allow sharing of a high-speed printer among users; compute servers, such as remotely located high-performance computers, accessible from

user programs; file servers that allow users to share files and programs, regardless of the machine they are logged into; and terminal servers that handle dial-in access. Users and user programs that make use of these services are clients or client programs. Servers can also exist at a lower level of granularity within a single operating system. For example, the operating system might provide a clock service that multiple client programs can use. It is less obvious what we should call a workstation connected to the network: It can be both a client and a server, or perhaps more to the point, a workstation may run client programs and server programs concurrently.

The model that we use here was popularized by the X Window System. We use much of that system's terminology, which is now common to most window systems and fits well with graphical applications.

A workstation with a raster display, a keyboard, and a pointing device, such as a mouse, is a graphics server. The server can provide output services on its display and input services through the keyboard and pointing device. These services are potentially available to clients anywhere on the network.

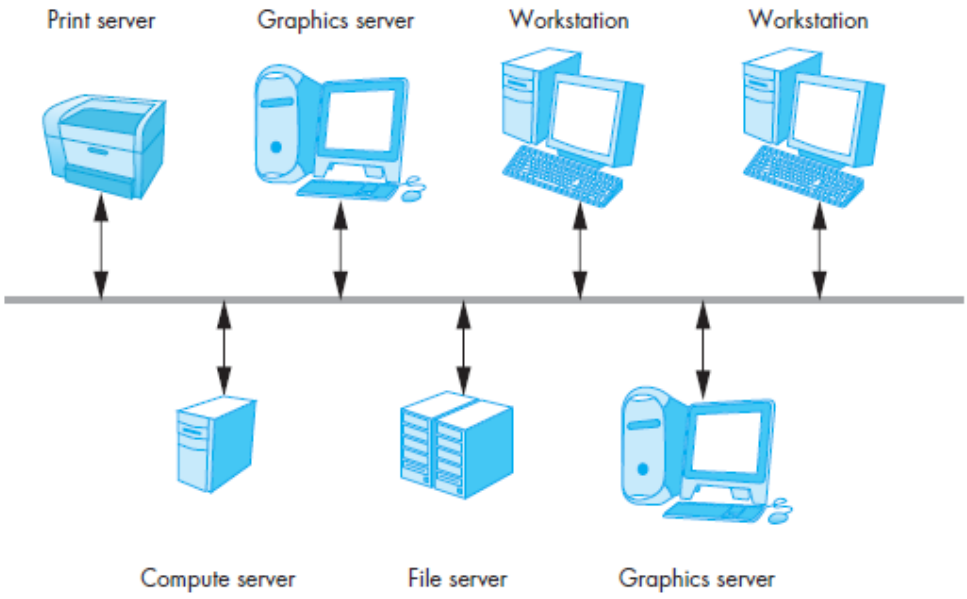

FIGURE 3.11 Network.

Continued…….