MODULE-3

Graphical Input Data

Graphics programs use several kinds of input data, such as coordinate positions, attribute values, character-string specifications, geometric-transformation values, viewing conditions, and illumination parameters. Many graphics packages, including the International Standards Organization (ISO) and American National Standards Institute (ANSI) standards, provide an extensive set of input functions for processing such data. But input procedures require interaction with display-window managers and specific hardware devices. Therefore, some graphics systems, particularly those that provide mainly device-independent functions, often include relatively few interactive procedures for dealing with input data.

A standard organization for input procedures in a graphics package is to classify the functions according to the type of data that is to be processed by each function. This scheme allows any physical device, such as a keyboard or a mouse, to input any data class.

Logical Classification of Input Devices

When input functions are classified according to data type, any device that is used to provide the specified data is referred to as a **logical input device** for that data type. The standard logical input-data classifications are

1. **LOCATOR:** A device for specifying one coordinate position.
2. **STROKE:** A device for specifying a set of coordinate positions.
3. **STRING:** A device for specifying text input.
4. **VALUATOR:** A device for specifying a scalar value.
5. **CHOICE:** A device for selecting a menu option.
6. **PICK:** A device for selecting a component of a picture.

1. Locator Devices

Interactive selection of a coordinate point is usually accomplished by positioning the screen cursor at some location in a displayed scene. Mouse, touchpad, joystick, trackball, spaceball, thumbwheel, dial, hand cursor, or digitizer stylus can be used for screen-cursor positioning.

Keyboards are used for locator input in several ways. A general-purpose keyboard usually has four cursor-control keys that move the screen cursor up, down, left, and right. With an additional four keys, cursor can be diagonally moved. Rapid cursor movement is accomplished by holding down the selected cursor key. Sometimes a keyboard includes a touchpad, joystick, trackball, or other device for positioning the screen cursor. For some applications, it may also be convenient to use a keyboard to type in numerical values or other codes to indicate coordinate positions. Other devices, such as a light pen, have also been used for interactive input of coordinate positions. But light pens record screen positions by detecting light from the screen phosphors, and this requires special implementation procedures.

2. Stroke Devices

This class of logical devices is used to input a sequence of coordinate positions, and the physical devices used for generating locator input are also used as stroke devices. Continuous movement of a mouse, trackball, joystick, or hand cursor is translated into a series of input coordinate values. The graphics tablet is one of the more common stroke devices. Button activation can be used to place the tablet into "continuous" mode. As the cursor is moved across the tablet surface, a stream of coordinate values is generated. This procedure is used in paintbrush systems to generate drawings using various brush strokes.

3. String Devices

The primary physical device used for string input is the keyboard. Character strings in computer-graphics applications are typically used for picture or graph labeling. Other physical devices can be used for generating character patterns for special applications. Individual characters can be sketched on the screen using a stroke or locator-type device. A pattern recognition program then interprets the characters using a stored dictionary of predefined patterns.

4. Valuator Devices

Valuator input can be employed in a graphics program to set scalar values for geometric transformations, viewing parameters, and illumination parameters. In some applications, scalar input is also used for setting physical parameters such as temperature, voltage, or stress-strain factors. A typical physical device used to provide valuator input is a **panel of control dials**. Dial settings are calibrated to produce numerical values within some predefined range. Rotary

potentiometers convert dial rotation into a corresponding voltage, which is then translated into a number within a defined scalar range, such as -10.5 to 25.5. Instead of dials, slide potentiometers are sometimes used to convert linear movements into scalar values.

Any keyboard with a set of numeric keys can be used as a valuator device. Joysticks, trackballs, tablets, and other interactive devices can be adapted for valuator input by interpreting pressure or movement of the device relative to a scalar range. For one direction of movement, say left to right, increasing scalar values can be input. Movement in the opposite direction decreases the scalar input value. Selected values are usually echoed on the screen for verification. Another technique for providing valuator input is to display graphical representations of sliders, buttons, rotating scales, and menus on the video monitor. Cursor positioning, using a mouse, joystick, spaceball, or other device, can be used to select a value on one of these valuators.

5. Choice Devices

Menus are typically used in graphics programs to select processing options, parameter values, and object shapes that are to be used in constructing a picture. Commonly used choice devices for selecting a menu option are cursor-positioning devices such as a **mouse, trackball, keyboard, touch panel,** or **button box**.

Keyboard function keys or separate button boxes are often used to enter menu selections. Each button or function key is programmed to select a particular operation or value, although preset buttons or keys are sometimes included on an input device.

For screen selection of listed menu options, cursor-positioning device is used. When a screen-cursor position (x, y) is selected, it is compared to the coordinate extents of each listed menu item. A menu item with vertical and horizontal boundaries at the coordinate values x_{min}, x_{max}, y_{min}, and y_{max} is selected if the input coordinates satisfy the inequalities

$$x_{min} \leq x \leq x_{max}, \quad y_{min} \leq y \leq y_{max} \qquad (1)$$

For larger menus with relatively few options displayed, a touch panel is commonly used. A selected screen position is compared to the coordinate extents of the individual menu options to determine what process is to be performed.

Alternate methods for choice input include keyboard and voice entry. A standard keyboard can be used to type in commands or menu options. For this method of choice input, some abbreviated format is useful. Menu listings can be numbered or given short identifying names. A similar encoding scheme can be used with voice input systems. Voice input is particularly useful when the number of options is small (20 or fewer).

6. Pick Devices

Pick device is used to select a part of a scene that is to be transformed or edited in some way. Several different methods can be used to select a component of a displayed scene, and any input mechanism used for this purpose is classified as a pick device.

Most often, pick operations are performed by positioning the screen cursor. Using a mouse, joystick, or keyboard, for example, we can perform picking by positioning the screen cursor and pressing a button or key to record the pixel coordinates. This screen position can then be used to select an entire object, a facet of a tessellated surface, a polygon edge, or a vertex. Other pick methods include **highlighting schemes**, **selecting objects by name**, or a combination of methods.

Using the cursor-positioning approach, a pick procedure could map a selected screen position to a world-coordinate location using the inverse viewing and geometric transformations that were specified for the scene. Then, the world coordinate position can be compared to the coordinate extents of objects. If the pick position is within the coordinate extents of a single object, the pick object has been identified. The object name, coordinates, or other information about the object can then be used to apply the desired transformation or editing operations. But if the pick position is within the coordinate extents of two or more objects, further testing is necessary. Depending on the type of object to be selected and the complexity of a scene, several levels of search may be required to identify the pick object.

When coordinate-extent tests do not uniquely identify a pick object, the distances from the pick position to individual line segments could be computed. Figure 3.1 illustrates a pick position that is within the coordinate extents of two line segments. For a two-dimensional line segment with pixel endpoint coordinates (x_1, y_1) and (x_2, y_2), the perpendicular distance squared from a pick position (x, y) to the line is calculated as

$$d^2 = \frac{[\Delta x(y - y_1) - \Delta y(x - x_1)]^2}{\Delta x^2 + \Delta y^2}$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

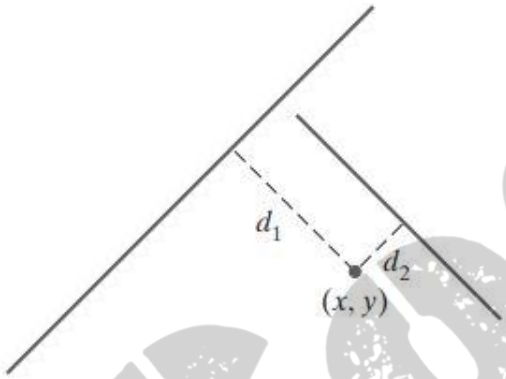

Figure 3.1: Distances to line segments from a pick position.

Another picking technique is to associate a **pick window** with a selected cursor position. The pick window is centered on the cursor position, as shown in Figure 3.2, and clipping procedures are used to determine which objects intersect the pick window. For line picking, we can set the pick-window dimensions *w* and *h* to very small values, so that only one line segment intersects the pick window.

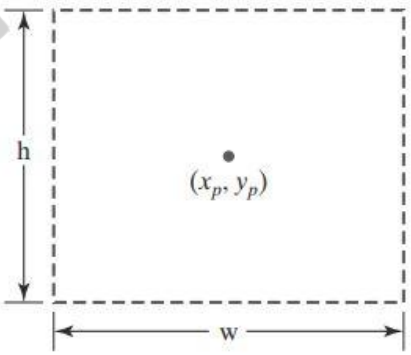

Figure 3.2: A pick window with center coordinates (x_p, y_p), width *w*, and height h.

Highlighting can also be used to facilitate picking. Successively highlight those objects whose coordinate extents overlap a pick position (or pick window). As each object is highlighted, a user could issue a "reject" or "accept" action using keyboard keys. The sequence stops when the user accepts a highlighted object as the pick object. Picking could also be accomplished simply by successively highlighting all objects in the scene without selecting a cursor position.

If picture components can be selected by **name**, keyboard input can be used to pick an object. This is a straightforward, but less interactive, pick-selection method. Some graphics packages allow picture components to be named at various levels down to the individual primitives. Descriptive names can be used to help a user in the pick process, but this approach has drawbacks. It is generally slower than interactive picking on the screen, and a user will probably need prompts to remember the various structure names.

Input Functions for Graphical Data

Graphics packages that use the logical classification for input devices provide several functions for selecting devices and data classes. These functions allow a user to specify the following options:

1. The input interaction mode for the graphics program and the input devices. Either the program or the devices can initiate data entry, or both can operate simultaneously.
2. Selection of a physical device that is to provide input within a particular logical classification (for example, a tablet used as a stroke device).
3. Selection of the input time and device for a particular set of data values.

Input Modes

Some input functions in an interactive graphics system are used to specify how the program and input devices should interact. A program could request input at a particular time in the processing (**request mode**), or an input device could independently provide updated input (**sample mode**), or the device could independently store all collected data (**event mode**).

1) Request Mode

In **request mode,** the application program initiates data entry. When input values are requested, processing is suspended until the required values are received. This input mode corresponds to the typical input operation in a general programming language. The program and the input devices operate alternately. Devices are put into a wait state until an input request is made; then the program waits until the data are delivered.

2) Sample Mode

In **sample mode,** the application program and input devices operate independently. Input devices may be operating at the same time that the program is processing other data. New values obtained from the input devices replace previously input data values. When the program requires new data, it samples the current values that have been stored from the device input.

3) Event Mode

In **event mode,** the input devices initiate data input to the application program. The program and the input devices again operate concurrently, but now the input devices deliver data to an input queue, also called an ***event queue***. All input data is saved. When the program requires new data, it goes to the data queue.

Typically, any number of devices can be operating at the same time in sample and event modes. Some can be operating in sample mode, while others are operating in event mode. But only one device at a time can deliver input in request mode.

Echo Feedback

Requests can usually be made in an interactive input program for an echo of input data and associated parameters. When an echo of the input data is requested, it is displayed within a specified screen area.

Echo feedback can include:

- The size of the pick window
- The minimum pick distance

- The type and size of a cursor
- The type of highlighting to be employed during pick operations
- The range (mininum and maximum) for valuator input
- The resolution (scale) for valuator input.

Callback Functions

For device-independent graphics packages, a limited set of input functions can be provided in an auxiliary library. Input procedures can then be handled as callback functions that interact with the system software. These functions specify what actions are to be taken by a program when an input event occurs. Typical input events are moving a mouse, pressing a mouse button, or pressing a key on the keyboard.

Interactive Picture-Construction Techniques

A variety of interactive methods are often incorporated into a graphics package as aids in the construction of pictures. Routines can be provided for positioning objects, applying constraints, adjusting the sizes of objects, and designing shapes and patterns.

1) Basic Positioning Methods

We can interactively choose a coordinate position with a pointing device that records a screen location. How the position is used depends on the selected processing option. The coordinate location could be an endpoint position for a new line segment, or it could be used to position some object—for instance, the selected screen location could reference a new position for the center of a sphere; or the location could be used to specify the position for a text string, which could begin at that location or it could be centered on that location. As an additional positioning aid, numeric values for selected positions can be echoed on the screen. With the echoed coordinate values as a guide, a user could make small interactive adjustments in the coordinate values using dials, arrow keys, or other devices.

2) Dragging

Another interactive positioning technique is to select an object and drag it to a new location. Using a mouse, for instance, we position the cursor at the object position, press a mouse button,

move the cursor to a new position, and release the button. The object is then displayed at the new cursor location. Usually, the object is displayed at intermediate positions as the screen cursor moves.

3) Constraints

Any procedure for altering input coordinate values to obtain a particular orientation or alignment of an object is called a **constraint**. For example, an input line segment can be constrained to be horizontal or vertical, as illustrated in Figures 3.3 and 3.4. To implement this type of constraint, we compare the input coordinate values at the two endpoints. If the difference in the y values of the two endpoints is smaller than the difference in the x values, a horizontal line is displayed. Otherwise, a vertical line is drawn.

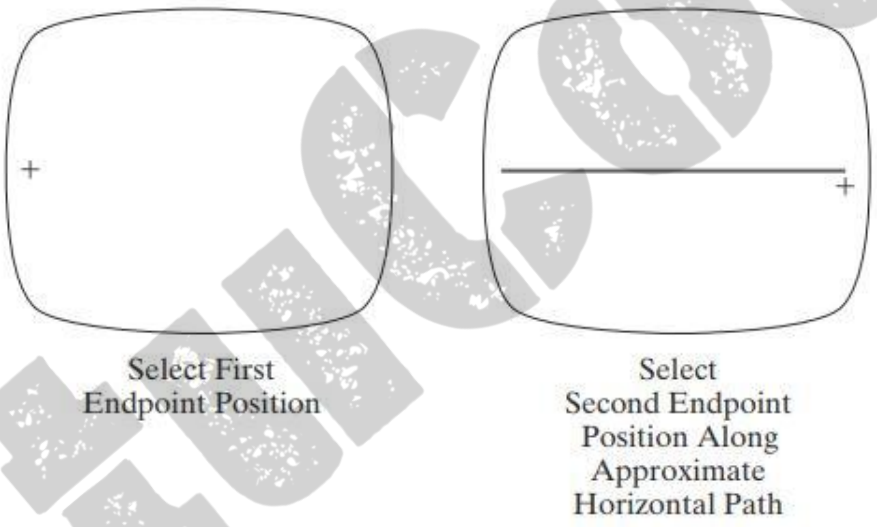

Figure 3.3: Horizontal line constraint.

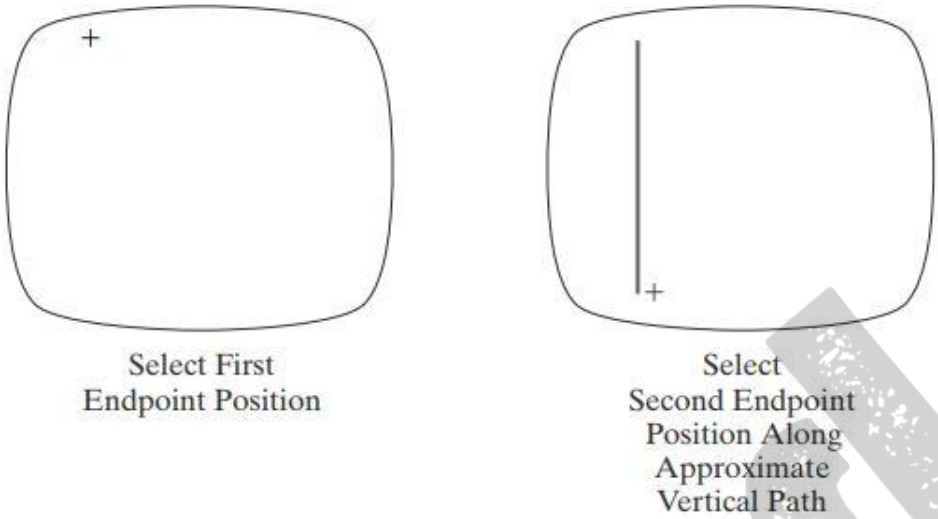

Figure 3.4: Vertical line constraint.

Other kinds of constraints can be applied to input coordinates to produce a variety of alignments. Lines could be constrained to have a particular slant, such as 45◦, and input coordinates could be constrained to lie along predefined paths, such as circular arcs.

4) Grids

Another kind of constraint is a rectangular grid displayed in some part of the screen area. With an activated grid constraint, input coordinates are rounded to the nearest grid intersection. Figure 3.5 illustrates line drawing using a grid. Each of the cursor positions in this example is shifted to the nearest grid intersection point, and a line is drawn between these two grid positions. Grids facilitate object constructions, because a new line can be joined easily to a previously drawn line by selecting any position near the endpoint grid intersection of one end of the displayed line. Spacing between grid lines is often an option, and partial grids or grids with different spacing could be used in different screen areas.

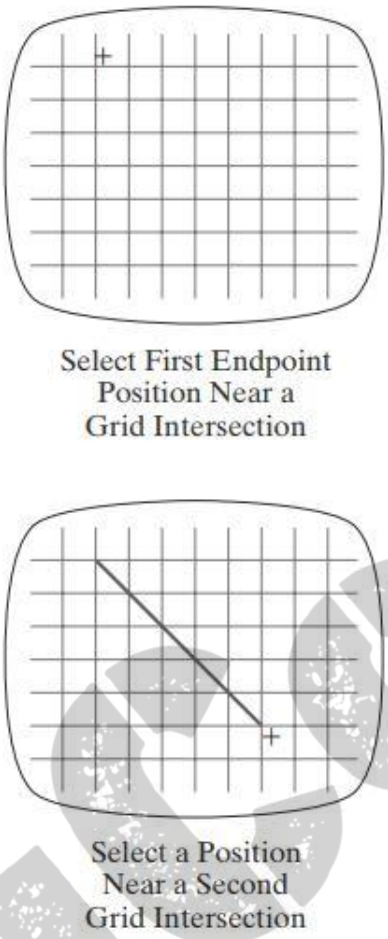

Figure 3.5: Construction of a line segment with endpoints constrained to grid intersection positions.

5) Rubber-Band Methods

Line segments and other basic shapes can be constructed and positioned using rubber-band methods that allow the sizes of objects to be interactively stretched or contracted. Figure 3.6 demonstrates a rubber-band method for interactively specifying a line segment. First, a fixed screen position is selected for one endpoint of the line. Then, as the cursor moves around, the line is displayed from the start position to the current position of the cursor. The second endpoint of the line is input when a button or key is pressed. Using a mouse, a rubber-band line is constructed while pressing a mouse key. When the mouse key is released, the line display is completed.

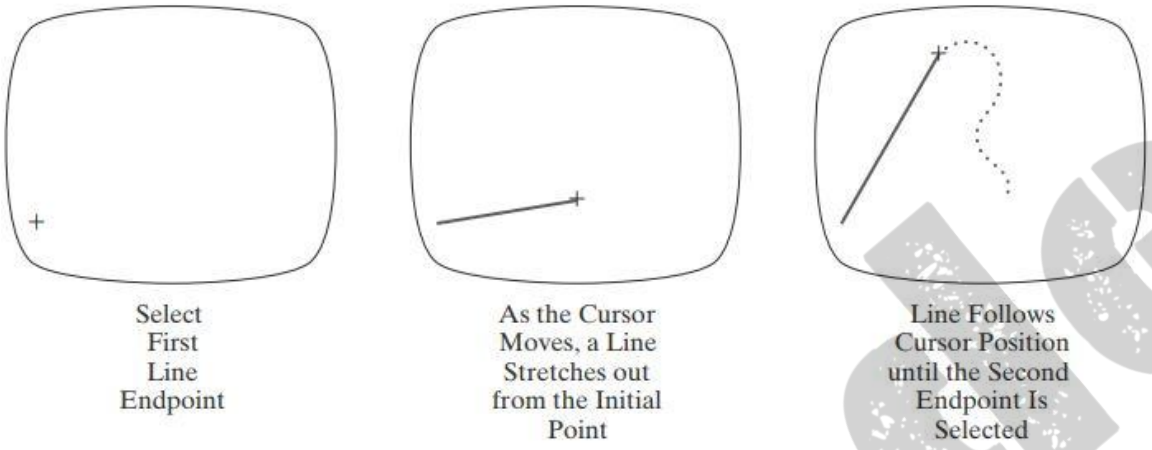

Figure 3.6: A rubber-band method for constructing and positioning a straight-line segment.

Similar rubber-band methods can be used to construct rectangles, circles, and other objects. Figure 3.7 demonstrates rubber-band construction of a rectangle, and Figure 3.8 shows a rubber-band circle construction.

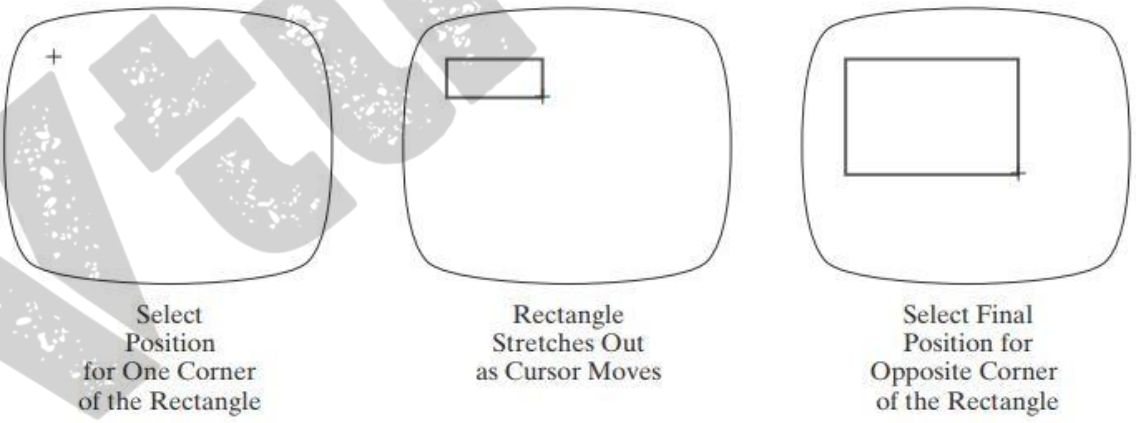

Figure 3.7: A rubber-band method for constructing a rectangle.

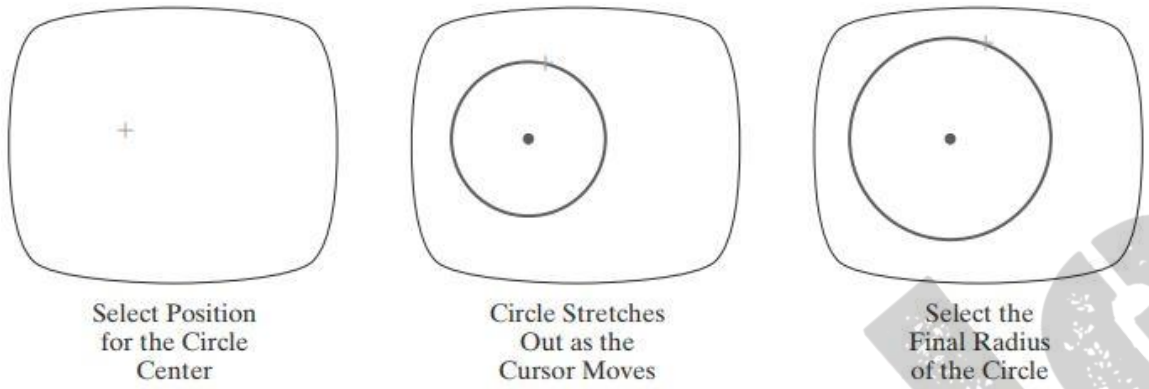

Figure 3.8: Constructing a circle using a rubber-band method.

6) Gravity Field

In the construction of figures, we sometimes need to connect lines at positions between endpoints that are not at grid intersections. Because exact positioning of the screen cursor at the connecting point can be difficult, a graphics package can include a procedure that converts any input position near a line segment into a position on the line using a *gravity field* area around the line. Any selected position within the gravity field of a line is moved ("gravitated") to the nearest position on the line. A gravity field area around a line is illustrated with the shaded region shown in Figure 3.9.

Gravity fields around the line endpoints are enlarged to make it easier for a designer to connect lines at their endpoints. Selected positions in one of the circular areas of the gravity field are attracted to the endpoint in that area. The size of gravity fields is chosen large enough to aid positioning, but small enough to reduce chances of overlap with other lines. If many lines are displayed, gravity areas can overlap, and it may be difficult to specify points correctly. Normally, the boundary for the gravity field is not displayed.

Figure 3.9:A gravity field around a line. Any selected point in the shaded area is shifted to a position on the line.

7) Interactive Painting and Drawing Methods

Options for sketching, drawing, and painting come in a variety of forms. Curve-drawing options can be provided using standard curve shapes, such as circular arcs and splines, or with freehand sketching procedures. Splines are interactively constructed by specifying a set of control points or a freehand sketch that gives the general shape of the curve. Then the system fits the set of points with a polynomial curve. In freehand drawing, curves are generated by following the path of a stylus on a graphics tablet or the path of the screen cursor on a video monitor. Once a curve is displayed, the designer can alter the curve shape by adjusting the positions of selected points along the curve path.

Line widths, line styles, and other attribute options are also commonly found in painting and drawing packages. Various brush styles, brush patterns, color combinations, object shapes, and surface texture patterns are also available on systems, particularly those designed as artists' workstations. Some paint systems vary the line width and brush strokes according to the pressure of the artist's hand on the stylus.

Virtual-Reality Environments

Interactive input is accomplished in virtual-reality environment with a data glove, which is capable of grasping and moving objects displayed in a virtual scene. The computer-generated scene is displayed through a head-mounted viewing system as a stereographic projection. Tracking devices compute the position and orientation of the headset and data glove relative to

the object positions in the scene. With this system, a user can move through the scene and rearrange object positions with the data glove

Another method for generating virtual scenes is to display stereographic projections on a raster monitor, with the two stereographic views displayed on alternate refresh cycles. The scene is then viewed through stereographic glasses. Interactive object manipulations can again be accomplished with a data glove and a tracking device to monitor the glove position and orientation relative to the position of objects in the scene.

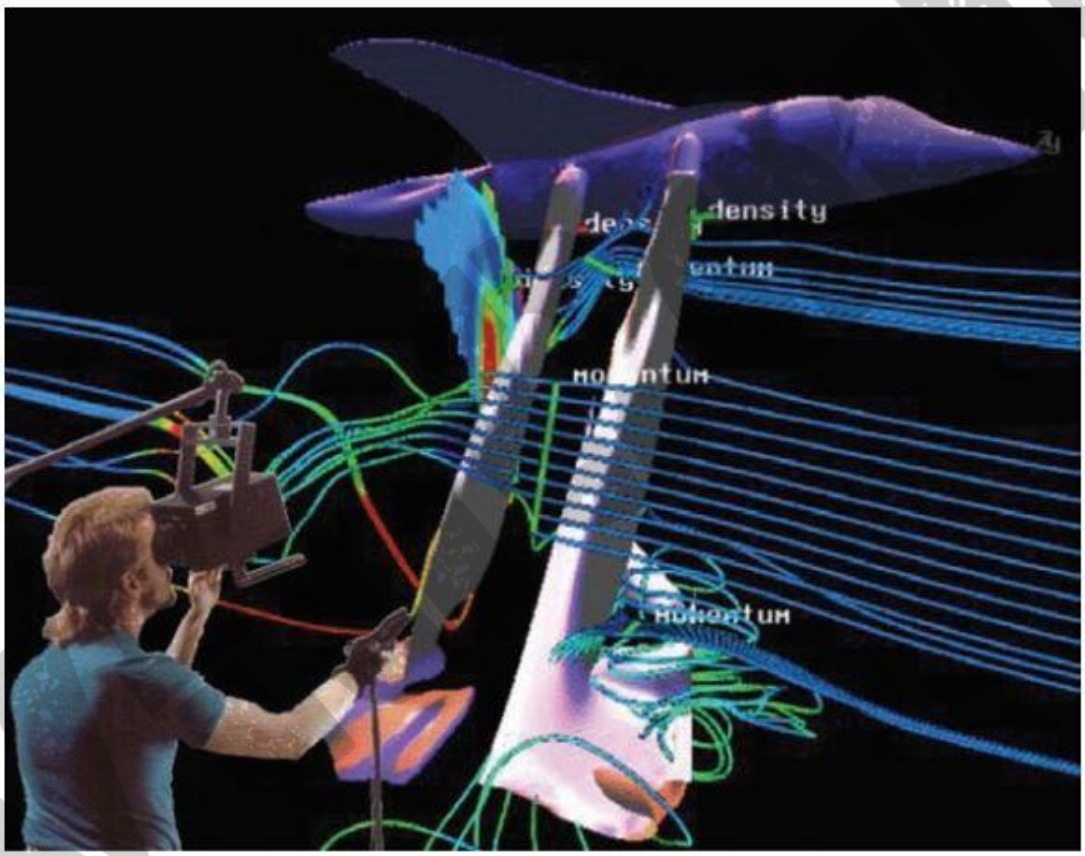

Figure 3.10: Using a head-tracking stereo display, called the BOOM and a Dataglove a researcher interactively manipulates exploratory probes in the unsteady flow around a Harrier jet airplane.

OpenGL Interactive Input-Device Functions

Interactive device input in an OpenGL program is handled with routines in the OpenGL Utility Toolkit (GLUT), because these routines need to interface with a window system. In GLUT, there are functions to accept input from standard devices, such as a mouse or a keyboard, as well as from tablets, space balls, button boxes, and dials. For each device, a procedure (the call back function) is specified and it is invoked when an input event from that device occurs. These GLUT commands are placed in the **main** procedure along with the other GLUT statements.

GLUT Mouse Functions

Following function is used to specify ("register") a procedure that is to be called when the mouse pointer is in a display window and a mouse button is pressed or released:

glutMouseFunc (mouseFcn);

This mouse callback procedure, named **mouseFcn**, has four arguments.

void mouseFcn(GLint button, GLint action, GLint xMouse, GLint yMouse)

Parameter **button** is assigned a GLUT symbolic constant that denotes one of the three mouse buttons.

Parameter **action** is assigned a symbolic constant that specifies which button action we want to use to trigger the mouse activation event. Allowable values for **button** are **GLUT_ LEFT_BUTTON**, **GLUT_MIDDLE_BUTTON**, and **GLUT_RIGHT_BUTTON**.

Parameter **action** can be assigned either **GLUT_DOWN** or **GLUT_UP**, depending on whether we want to initiate an action when we press a mouse button or when we release it. When procedure **mouseFcn** is invoked, the display-window location of the mouse cursor is returned as the coordinate position (**xMouse**, **yMouse**). This location is relative to the top-left corner of the display window, so that **xMouse** is the pixel distance from the left edge of the display window and **yMouse** is the pixel distance down from the top of the display window.

By activating a mouse button while the screen cursor is within the display window, we can select a position for displaying a primitive such as a single point, a line segment, or a fill area.

Another GLUT mouse routine that we can use is

glutMotionFunc (fcnDoSomething);

This routine invokes **fcnDoSomething** when the mouse is moved within the display window with one or more buttons activated. The function that is invoked has two arguments:

void fcnDoSomething (GLint xMouse, GLint yMouse)

where (**xMouse**, **yMouse**) is the mouse location in the display window relative to the top-left corner, when the mouse is moved with a button pressed.

Some action can be performed when we move the mouse within the display window without pressing a button:

glutPassiveMotionFunc(fcnDoSomethingElse);

The mouse location is returned to **fcnDoSomethingElse** as coordinate position (**xMouse**, **yMouse**), relative to the top-left corner of the display window.

GLUT Keyboard Functions

With keyboard input, the following function is used to specify a procedure that is to be invoked when a key is pressed:

glutKeyboardFunc (keyFcn);

The specified procedure has three arguments:

void keyFcn (GLubyte key, GLint xMouse, GLint yMouse)

Parameter **key** is assigned a character value or the corresponding ASCII code. The display-window mouse location is returned as position (**xMouse**,**yMouse**) relative to the top-left corner of the display window. When a designated key is pressed, mouse location can be used to initiate some action, independently of whether any mouse buttons are pressed.

For function keys, arrow keys, and other special-purpose keys, following command can be used:

glutSpecialFunc (specialKeyFcn);

The specified procedure has three arguments:

void specialKeyFcn (GLint specialKey, GLint xMouse,GLint yMouse)

Parameter **specialKey** is assigned an integer-valued GLUT symbolic constant. To select a function key, one of the constants **GLUT_KEY_F1** through **GLUT_KEY_F12** is used. For the arrow keys, constants such as **GLUT_KEY_UP** and **GLUT_KEY_RIGHT** is used. Other keys can be designated using **GLUT_KEY_PAGE_DOWN**, **GLUT_KEY_HOME**.

GLUT Tablet Functions

Usually, tablet activation occurs only when the mouse cursor is in the display window. A button event for tablet input is recorded with

glutTabletButtonFunc (tabletFcn);

and the arguments for the invoked function are similar to those for a mouse:

void tabletFcn (GLint tabletButton, GLint action,GLint xTablet, GLint yTablet)

We designate a **tablet button** with an integer identifier such as 1, 2, 3, and so on, and the button **action** is specified with either **GLUT_UP** or **GLUT_DOWN**. The returned values **xTablet** and **yTablet** are the tablet coordinates. The number of available tablet buttons can be determined with the following command

glutDeviceGet (GLUT_NUM_TABLET_BUTTONS);

Motion of a tablet stylus or cursor is processed with the following function:

glutTabletMotionFunc (tabletMotionFcn);

where the invoked function has the form

void tabletMotionFcn (GLint xTablet, GLint yTablet)

The returned values **xTablet** and **yTablet** give the coordinates on the tablet surface.

GLUT Spaceball Functions

Following function is used to specify an operation when a spaceball button is activated for a selected display window:

glutSpaceballButtonFunc (spaceballFcn);

The callback function has two parameters:

void spaceballFcn (GLint spaceballButton, GLint action)

Spaceball buttons are identified with the same integer values as a tablet, and parameter **action** is assigned either the value **GLUT_UP** or the value **GLUT_DOWN**. The number of available spaceball buttons can be determined with a call to **glutDeviceGet** using the argument **GLUT_NUM_SPACEBALL_BUTTONS**

Translational motion of a spaceball, when the mouse is in the display window, is recorded with the function call

glutSpaceballMotionFunc (spaceballTranlFcn);

The three-dimensional translation distances are passed to the invoked function as, for example:

void spaceballTranslFcn (GLint tx, GLint ty, GLint tz)

A spaceball rotation is recorded with

glutSpaceballRotateFunc (spaceballRotFcn);

The three-dimensional rotation angles are then available to the callback function, as follows:

void spaceballRotFcn (GLint thetaX, GLint thetaY, GLint thetaZ)

GLUT Button-Box Function

Input from a button box is obtained with the following statement:

glutButtonBoxFunc (buttonBoxFcn);

Button activation is then passed to the invoked function:

void buttonBoxFcn (GLint button, GLint action);

The buttons are identified with integer values, and the button action is specified as **GLUT_UP** or **GLUT_DOWN**.

GLUT Dials Function

A dial rotation can be recorded with the following routine:

glutDialsFunc(dialsFcn);

Following callback function is used to identify the dial and obtain the angular amount of rotation:

void dialsFcn (GLint dial, GLint degreeValue);

Dials are designated with integer values, and the dial rotation is returned as an integer degree value.

OpenGL Picking Operations

In an OpenGL program, interactively objects can be selected by pointing to screen positions. However, the picking operations in OpenGL are not straightforward. Basically, picking is performed using a designated pick window to form a revised view volume. Integer identifiers are assigned to objects in a scene, and the identifiers for those objects that intersect the revised view volume are stored in a pick-buffer array. Thus, to use the OpenGL pick features, following procedures has to be incorporated into a program:

- Create and display a scene.
- Pick a screen position and, within the mouse callback function, do the following:
    - Set up a pick buffer.
    - Activate the picking operations (selection mode).
    - Initialize an ID name stack for object identifiers.
    - Save the current viewing and geometric-transformation matrix.
    - Specify a pick window for the mouse input.

- Assign identifiers to objects and reprocess the scene using the revised view volume. (Pick information is then stored in the pick buffer.)
- Restore the original viewing and geometric-transformation matrix.
- Determine the number of objects that have been picked, and return to the normal rendering mode.
- Process the pick information.

A pick-buffer array is set up with the following command

glSelectBuffer (pickBuffSize, pickBuffer);

Parameter **pickBuffer** designates an integer array with **pickBuffSize** elements. The **glSelectBuffer** function must be invoked before the OpenGL picking operations (selection mode) are activated. An integer information record is stored in pick-buffer array for each object that is selected with a single pick input. Several records of information can be stored in the pick buffer, depending on the size and location of the pick window. Each record in the pick buffer contains the following information:

1. The stack position of the object, which is the number of identifiers in the name stack, up to and including the position of the picked object.
2. The minimum depth of the picked object.
3. The maximum depth of the picked object.
4. The list of the identifiers in the name stack from the first (bottom) identifier to the identifier for the picked object.

The integer depth values stored in the pick buffer are the original values in the range from 0 to 1.0, multiplied by 2^{32} - 1.

The OpenGL picking operations are activated with
glRenderMode (GL_SELECT);

Above routine call switches to selection mode. A scene is processed through the viewing pipeline but not stored in the frame buffer. A record of information for each object that would have been displayed in the normal rendering mode is placed in the pick buffer. In addition, this

command returns the number of picked objects, which is equal to the number of information records in the pick buffer. To return to the normal rendering mode (the default), **glRenderMode** routine is invoked using the argument **GL_RENDER**. A third option is the argument **GL_ FEEDBACK**, which stores object coordinates and other information in a feedback buffer without displaying the objects.

Following statement is used to activate the integer-ID name stack for the picking operations:

glInitNames ( );

The ID stack is initially empty, and this stack can be used only in selection mode. To place an unsigned integer value on the stack, following function can be invoked:

glPushName (ID);

This places the value for parameter **ID** on the top of the stack and pushes the previous top name down to the next position in the stack. The top of the stack can be replaced using

glLoadName (ID);

To eliminate the top of the ID stack, following command is used:

glPopName ( );

A pick window within a selected viewport is defined using the following GLU function:

gluPickMatrix (xPick, yPick, widthPick, heightPick, vpArray);

Parameters **xPick** and **yPick** give the double-precision, screen-coordinate location for the center of the pick window relative to the lower-left corner of the viewport. When these coordinates are given with mouse input, the mouse coordinates are relative to the upper-left corner, and thus the input **yMouse** value has to be inverted. The double-precision values for the width and height of the pick window are specified with parameters **widthPick** and **heightPick**. Parameter **vpArray** designates an integer array containing the coordinate position and size parameters for the current viewport.

OpenGL Menu Functions

GLUT contains various functions for adding simple pop-up menus to programs. With these functions, we can set up and access a variety of menus and associated submenus. The GLUT menu commands are placed in procedure **main** along with the other GLUT functions.

Creating a GLUT Menu

A pop-up menu is created with the statement

glutCreateMenu (menuFcn);

where parameter **menuFcn** is the name of a procedure that is to be invoked when a menu entry is selected. This procedure has one argument, which is the integer value corresponding to the position of a selected option.

void menuFcn (GLint menuItemNumber)

The integer value passed to parameter **menuItemNumber** is then used by **menuFcn** to perform an operation. When a menu is created, it is associated with the current display window.

To specify the options that are to be listed in the menu, a series of statements that list the **name** and **position** for each option is used. These statements have the general form

glutAddMenuEntry (charString, menuItemNumber);

Parameter **charString** specifies text that is to be displayed in the menu.

The parameter **menuItemNumber** gives the location for that entry in the menu.

For example, the following statements create a menu with two options:

glutCreateMenu (menuFcn);

glutAddMenuEntry ("First Menu Item", 1);

glutAddMenuEntry ("Second Menu Item", 2);

Next, we must specify a mouse button that is to be used to select a menu option.

This is accomplished with

glutAttachMenu (button);

where parameter **button** is assigned one of the three GLUT symbolic constants referencing the left, middle, or right mouse button.

Creating and Managing Multiple GLUT Menus

When a menu is created, it is associated with the current display window. We can create multiple menus for a single display window, and we can create different menus for different windows. As each menu is created, it is assigned an integer identifier, starting with the value 1 for the first menu created. The integer identifier for a menu is returned by the **glutCreateMenu** routine, and we can record this value with a statement such as

menuID = glutCreateMenu (menuFcn);

A newly created menu becomes the **current menu** for the current display window. To activate a menu for the current display window, following statement is used.

glutSetMenu (menuID);

This menu then becomes the current menu, which will pop up in the display window when the mouse button that has been attached to that menu is pressed.

To eliminate a menu following command is used:

glutDestroyMenu (menuID);

If the designated menu is the current menu for a display window, then that window has no menu assigned as the current menu, even though other menus may exist.

The following function is used to obtain the identifier for the current menu in the current display window:

currentMenuID = glutGetMenu( );

A value of 0 is returned if no menus exist for this display window or if the previous current menu was eliminated with the **glutDestroyMenu** function.

Creating GLUT Submenus

A submenu can be associated with a menu by first creating the submenu using **glutCreateMenu**, along with a list of suboptions, and then listing the submenu as an additional option in the main menu. Submenu can be added to the option list in a main menu (or other submenu) using a sequence of statements such as

```
submenuID = glutCreateMenu (submenuFcn);
    glutAddMenuEntry ("First Submenu Item", 1);
    ...
    …
    …
```

```
glutCreateMenu (menuFcn);
   glutAddMenuEntry ("First Menu Item", 1);
   ...
   ...
   ...
glutAddSubMenu ("Submenu Option", submenuID);
```

The **glutAddSubMenu** function can also be used to add the submenu to the current menu.

Modifying GLUT Menus

To change the mouse button that is used to select a menu option, first cancel the current button attachment and then attach the new button. A button attachment is cancelled for the current menu with

glutDetachMenu (mouseButton);

where parameter **mouseButton** is assigned the GLUT constant identifying the button (left, middle, or right) that was previously attached to the menu.

After detaching the menu from the button, **glutAttachMenu** is used to attach it to a different button.

Options within an existing menu can also be changed.

For example, an option in the current menu can be deleted with the function

glutRemoveMenuItem (itemNumber);

where parameter **itemNumber** is assigned the integer value of the menu option that is to be deleted.

Designing a Graphical User Interface

A common feature of modern applications software is a graphical user interface (GUI) composed of display windows, icons, menus, and other features to aid a user in applying the software to a particular problem. Specialized interactive dialogues are designed so that programming options are selected using familiar terms within a particular field, such as architectural and engineering design, drafting, business graphics, geology, economics, chemistry, or physics. Other considerations for a user interface (whether graphical or not) are the accommodation of various skill levels, consistency, error handling, and feedback.

The User Dialogue

For any application, the *user's model* serves as the basis for the design of the dialogue by describing what the system is designed to accomplish and what operations are available. It states the type of objects that can be displayed and how the objects can be manipulated. For example, if the system is to be used as a tool for architectural design, the model describes how the package can be used to construct and display views of buildings by positioning walls, doors, windows, and other building components. A circuit-design program provides electrical or logic symbols and the positioning operations for adding or deleting elements within a layout. All information in the user dialogue is presented in the language of the application.

Windows and Icons

Typical GUIs provide visual representations both for the objects that are to be manipulated in an application and for the actions to be performed on the application objects. In addition to the standard display-window operations, such as opening, closing, positioning, and resizing, other operations are needed for working with the sliders, buttons, icons, and menus. Some systems are capable of supporting multiple window managers so that different window styles can be accommodated, each with its own window manager, which could be structured for a particular application. Icons representing objects such walls, doors, windows, and circuit elements are often referred to as **application icons.** The icons representing actions, such as rotate, magnify, scale, clip, or paste, are called **control icons,** or **command icons.**

Accommodating Multiple Skill Levels

Usually, interactive GUIs provide several methods for selecting actions. For example, an option could be specified by pointing to an icon, accessing a pulldown or pop-up menu, or by typing a keyboard command. This allows a package to accommodate users that have different skill levels.

A less experienced user may find an interface with a large, comprehensive set of operations to be difficult to use, so a smaller interface with fewer but more easily understood operations and detailed prompting may be preferable. A simplified set of menus and options is easy to learn and remember, and the user can concentrate on the application instead of on the details of the interface. Simple point-and-click operations are often easiest for an inexperienced user of an applications package.

Experienced users, typically want speed. This means fewer prompts and more input from the keyboard or with multiple mouse-button clicks. Actions are selected with function keys or with

simultaneous combinations of keyboard keys, because experienced users will remember these shortcuts for commonly used actions.

Help facilities can be designed on several levels so that beginners can carry on a detailed dialogue, while more experienced users can reduce or eliminate prompts and messages. Help facilities can also include one or more tutorial applications, which provide users with an introduction to the capabilities and use of the system.

Consistency

An important design consideration in an interface is **consistency**. An icon shape should always have a single meaning, rather than serving to represent different actions or objects depending on the context.

Examples of consistency:

- Always placing menus in the same relative positions so that a user does not have to hunt for a particular option.
- Always using the same combination of keyboard keys for an action.
- Always using the same color encoding so that a color does not have different meanings in different situations.

Minimizing Memorization

Operations in an interface should also be structured so that they are easy to understand and to remember. Obscure, complicated, inconsistent, and abbreviated command formats lead to confusion and reduction in the effective application of the software. One key or button used for all delete operations, for example, is easier to remember than a number of different keys for different kinds of delete procedures.

Icons and window systems can also be organized to minimize memorization. Different kinds of information can be separated into different windows so that a user can identify and select items easily. Icons should be designed as easily recognizable shapes that are related to application objects and actions. To select a particular action, a user should be able to select an icon that resembles that action.

Backup and Error Handling

A mechanism for undoing a sequence of operations is another common feature of an interface, which allows a user to explore the capabilities of a system, knowing that the effects of a mistake

can be corrected. Typically, systems can undo several operations, thus allowing a user to reset the system to some specified action. For those actions that cannot be reversed, such as closing an application without saving changes, the system asks for a verification of the requested operation.

Good diagnostics and error messages help a user to determine the cause of an error. Interfaces can attempt to minimize errors by anticipating certain actions that could lead to an error; and users can be warned if they are requesting ambiguous or incorrect actions, such as attempting to apply a procedure to multiple application objects.

Feedback

Responding to user actions is another important feature of an interface, particularly for an inexperienced user. As each action is entered, some response should be given. Otherwise, a user might begin to wonder what the system is doing and whether the input should be reentered.

Feedback can be given in many forms, such as **highlighting an object**, **displaying an icon or message,** and **displaying a selected menu option in a different color**. When the processing of a requested action is lengthy, the display of a flashing message, clock, hourglass, or other progress indicator is important. It may also be possible for the system to display partial results as they are completed, so that the final display is built up a piece at a time.

Standard symbol designs are used for typical kinds of feedback. A cross, a frowning face, or a thumbs-down symbol is often used to indicate an error, and some kind of time symbol or a blinking “at work” sign is used to indicate that an action is being processed.
This type of feedback can be very effective with a more experienced user, but the beginner may need more detailed feedback that not only clearly indicates what the system is doing but also what the user should input next.

Clarity is another important feature of feedback. A response should be easily understood, but not so overpowering that the user’s concentration is interrupted. With function keys, feedback can be given as an audible click or by lighting up the key that has been pressed. Audio feedback has the advantage that it does not use up screen space, and it does not divert the user’s attention from the work area. A fixed message area can be used so that a user always know where to look for

messages, but it may be advantageous in some cases to place feedback messages in the work area near the cursor.

Echo feedback is often useful, particularly for keyboard input, so that errors can be detected quickly. Selection of coordinate points can be echoed with a cursor or other symbol that appears at the selected position

Design of Animation Sequences

Note:

Computer animation generally refers to any time sequence of visual changes in a picture.

Constructing an animation sequence can be a complicated task, particularly when it involves a story line and multiple objects, each of which can move in a different way. A basic approach is to design animation sequences using the following development stages:

1. Storyboard layout
2. Object definitions
3. Key-frame specifications
4. Generation of in-between frames

1. Storyboard Layout

The **storyboard** is an outline of the action. It defines the motion sequence as a set of basic events that are to take place. Depending on the type of animation to be produced, the storyboard could consist of a set of rough sketches, along with a brief description of the movements, or it could just be a list of the basic ideas for the action. Originally, the set of motion sketches was attached to a large board that was used to present an overall view of the animation project. Hence, the name “storyboard.”

2. Object Definitions

An **object definition** is given for each participant in the action. Objects can be defined in terms of basic shapes, such as polygons or spline surfaces. In addition, a description is often given of the movements that are to be performed by each character or object in the story.

3. Key-Frame Specifications

A **key frame** is a detailed drawing of the scene at a certain time in the animation sequence. Within each key frame, each object (or character) is positioned according to the time for that frame. Some key frames are chosen at extreme positions in the action; others are spaced so that the time interval between key frames is not too great. More key frames are specified for intricate motions than for simple, slowly varying motions. Development of the key frames is generally the responsibility of the senior animators, and often a separate animator is assigned to each character in the animation.

4. Generation of in-between frames

In-betweens are the intermediate frames between the key frames. The total number of frames, and hence the total number of in-betweens, needed for an animation is determined by the display media that is to be used. Film requires 24 frames per second, and graphics terminals are refreshed at the rate of 60 or more frames per second. Typically, time intervals for the motion are set up so that there are from three to five in-betweens for each pair of key frames. Depending on the speed specified for the motion, some key frames could be duplicated.

There are several other tasks that may be required, depending on the application. These additional tasks include **motion verification, editing, and the production and synchronization of a soundtrack**. Many of the functions needed to produce general animations are now computer-generated.

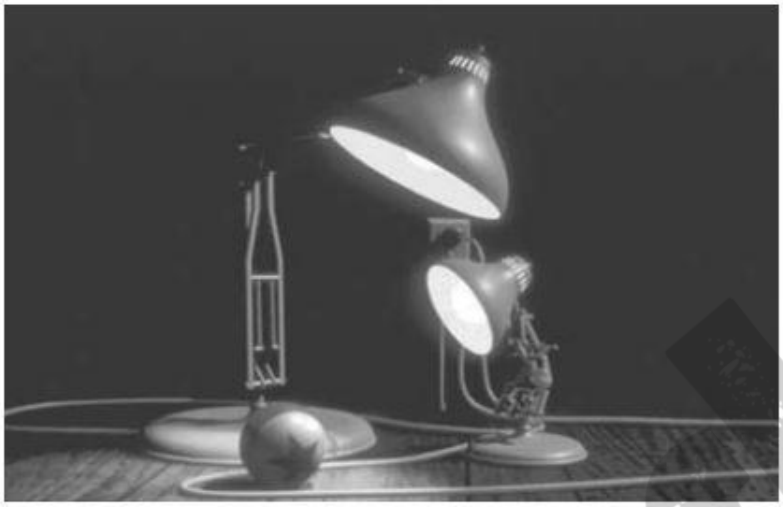

Figure 3.11: One frame from the award-winning computer-animated short film Luxo Jr. The film was designed using a key-frame animation system and cartoon animation techniques to provide lifelike actions of the lamps. Final images were rendered with multiple light sources and procedural texturing techniques.

Figure 3.12: One frame from the short film Tin Toy, the first computer-animated film to win an Oscar. Designed using a key-frame animation system, the film also required extensive facial-expression modeling. Final images were rendered using procedural shading, self-shadowing techniques, motion blur, and texture mapping.

Traditional Animation Techniques

Film animators use a variety of methods for depicting and emphasizing motion sequences. These include object deformations, spacing between animation frames, motion anticipation and follow-through, and action focusing.

One of the most important techniques for simulating acceleration effects, particularly for nonrigid objects, is **squash and stretch.**

Figure 3.13 shows how **squash and stretch** technique is used to emphasize the acceleration and deceleration of a bouncing ball. As the ball accelerates, it begins to stretch. When the ball hits the floor and stops, it is first compressed (squashed) and then stretched again as it accelerates and bounces upwards.

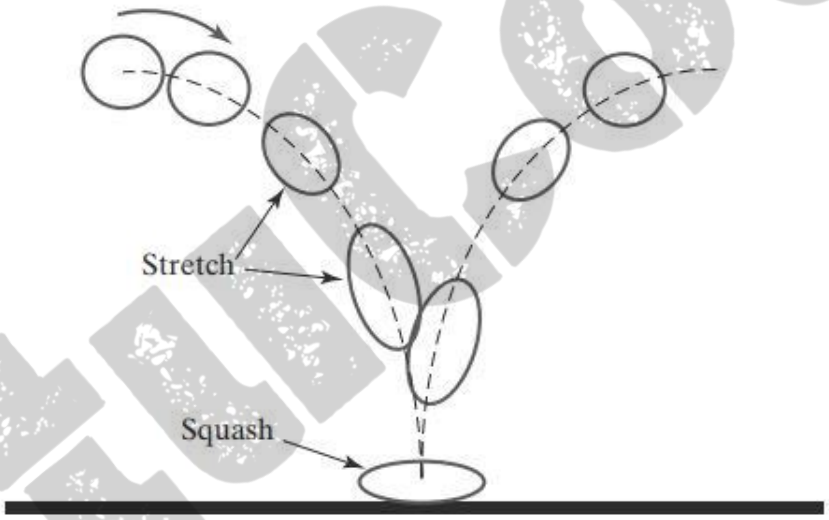

Figure 3.13 : A bouncing-ball illustration of the "squash and stretch" technique for emphasizing object acceleration.

Another technique used by film animators is **timing.** **Timing** refers to the spacing between motion frames. A slower moving object is represented with more closely spaced frames, and a faster moving object is displayed with fewer frames over the path of the motion. This effect is illustrated in Figure 3.14, where the position changes between frames increase as a bouncing ball moves faster.

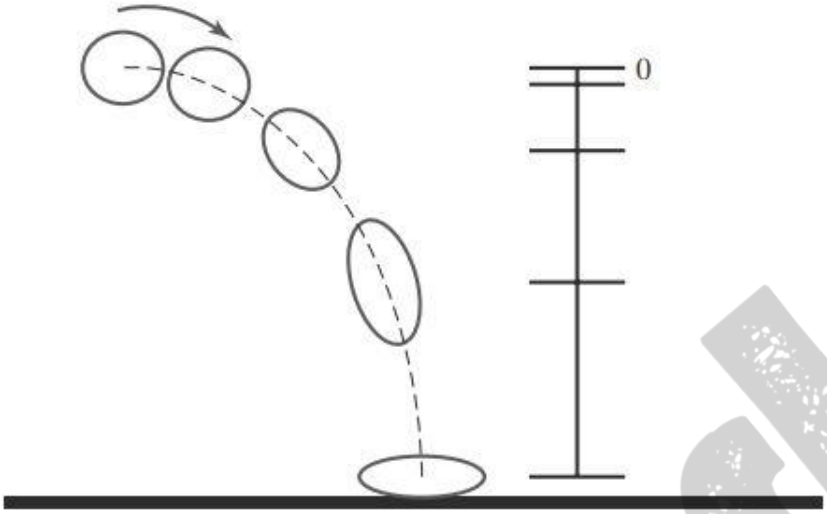

Figure 3.14: The position changes between motion frames for a bouncing ball increase as the speed of the ball increases.

Object movements can also be emphasized by creating preliminary actions that indicate an **anticipation** of a coming motion. For example, a cartoon character might lean forward and rotate its body before starting to run; or a character might perform a "windup" before throwing a ball. **Follow-through actions** can be used to emphasize a previous motion. After throwing a ball, a character can continue the arm swing back to its body; or a hat can fly off a character that is stopped abruptly. An action also can be emphasized with **staging. Staging** refers to any method for focusing on an important part of a scene, such as a character hiding something.

General Computer-Animation Functions

Many software packages have been developed either for general animation design or for performing specialized animation tasks.

Typical animation functions include

- Managing object motions
- Generating views of objects
- Producing camera motions
- The generation of in-between frames

Some animation packages, such as **Wavefront** for example, provide special functions for both the overall animation design and the processing of individual objects. Others are **special-purpose packages** for particular features of an animation, such as a system for generating in-between frames or a system for figure animation.

A set of routines is often provided in a general animation package for storing and managing the object database. Object shapes and associated parameters are stored and updated in the database. Other object functions include those for generating the object motion and those for rendering the object surfaces. Movements can be generated according to specified constraints using two dimensional or three-dimensional transformations. Standard functions can then be applied to identify visible surfaces and apply the rendering algorithms.

Another typical function set simulates camera movements. Standard camera motions are **zooming, panning, and tilting**. Finally, given the specification for the key frames, the in-betweens can be generated automatically.

Computer-Animation Languages

Routines can be developed to design and control animation sequences within a general-purpose programming language, such as C, C++, Lisp, or Fortran. But several specialized animation languages have been developed.

Computer animation languages typically include :

- A graphics editor
- A key-frame generator
- An in-between generator
- Standard graphics routines

The **graphics editor** allows an animator to design and modify object shapes, using spline surfaces, constructive solid geometry methods, or other representation schemes.

An important task in an animation specification is ***scene description***. Scene description includes the positioning of objects and light sources, defining the photometric parameters (light-source

intensities and surface illumination properties), and setting the camera parameters (position, orientation, and lens characteristics).

Another standard function is ***action specification.*** **Action specification** involves the layout of motion paths for the objects and camera. Usual graphics routines are needed for viewing and perspective transformations, geometric transformations to generate object movements as a function of accelerations or kinematic path specifications, visible-surface identification, and the surface-rendering operations.

Key-frame systems were originally designed as a separate set of animation routines for generating the in-betweens from the user-specified key frames. Now, these routines are often a component in a more general animation package. In the simplest case, each object in a scene is defined as a set of rigid bodies connected at the joints and with a limited number of degrees of freedom.

Example:

The single-armed robot in Figure 3.15 has 6 degrees of freedom, which are referred to as **arm sweep, shoulder swivel, elbow extension, pitch, yaw,** and **roll**. The number of degrees of freedom for this robot arm can be extended to 9 by allowing three-dimensional translations for the base (Figure 3.16). If base rotations are allowed, the robot arm can have a total of 12 degrees of freedom. The human body, in comparison, has more than 200 degrees of freedom.

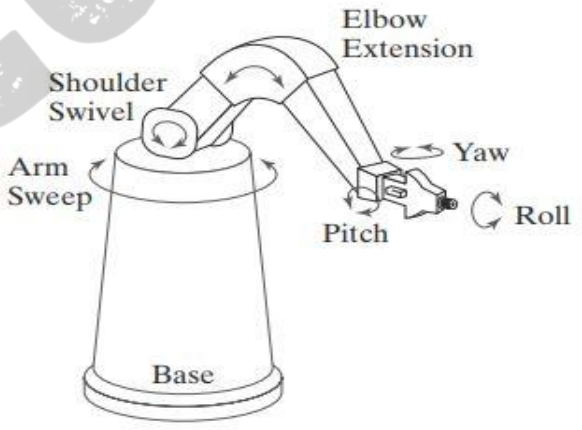

Figure 3.15: Degrees of freedom for a stationary, single-armed robot

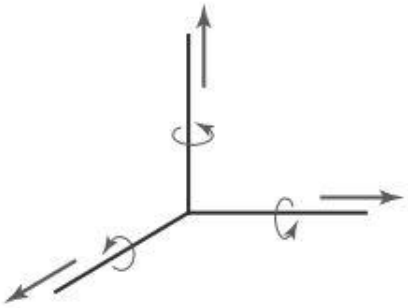

Figure 3.16: Translational and rotational degrees of freedom for the base of the robot arm

Parameterized systems allow object motion characteristics to be specified as part of the object definitions. The adjustable parameters control such object characteristics as degrees of freedom, motion limitations, and allowable shape changes.

Scripting systems allow object specifications and animation sequences to be defined with a user-input *script*. From the script, a library of various objects and motions can be constructed.

Character Animation

Animation of simple objects is relatively straightforward. It becomes much more difficult to create realistic animation of more complex figures such as humans or animals. Consider the animation of walking or running human (or humanoid) characters. Based upon observations in their own lives of walking or running people, viewers will expect to see animated characters move in particular ways. If an animated character's movement doesn't match this expectation, the believability of the character may suffer. Thus, much of the work involved in character animation is focused on creating believable movements.

Articulated Figure Animation

A basic technique for animating people, animals, insects, and other critters is to model them as **articulated figures. Articulated figures** are hierarchical structures composed of a set of rigid links that are connected at rotary joints (Figure 3.17). Animate objects are modeled as moving stick figures, or simplified skeletons, that can later be wrapped with surfaces representing skin, hair, fur, feathers, clothes, or other outer coverings.

The connecting points, or hinges, for an articulated figure are placed at the shoulders, hips, knees, and other skeletal joints, which travel along specified motion paths as the body moves. For example, when a motion is specified for an object, the shoulder automatically moves in a certain way and, as the shoulder moves, the arms move. Different types of movement, such as walking, running, or jumping, are defined and associated with particular motions for the joints and connecting links.

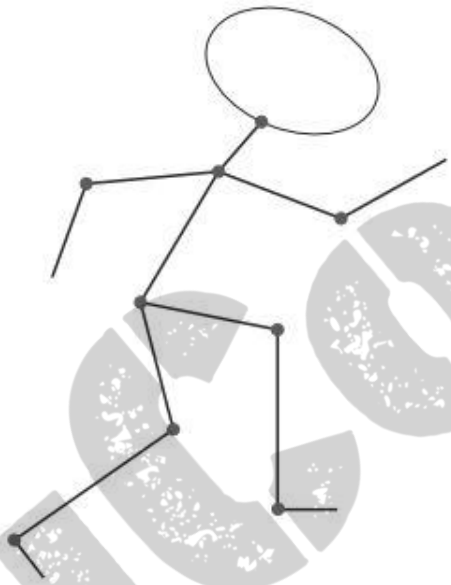

Figure 3.17: A simple articulated figure with nine joints and twelve connecting links, not counting the oval head

A series of walking leg motions, for instance, might be defined as in Figure 3.18. The hip joint is translated forward along a horizontal line, while the connecting links perform a series of movements about the hip, knee, and angle joints. Starting with a straight leg [Figure 3.18(a)], the first motion is a knee bend as the hip moves forward [Figure 3.18(b)]. Then the leg swings forward, returns to the vertical position, and swings back, as shown in Figures 3.18(c), (d), and (e). The final motions are a wide swing back and a return to the straight vertical position, as in Figures 3.18(f) and (g). This motion cycle is repeated for the duration of the animation as the figure moves over a specified distance or time interval.

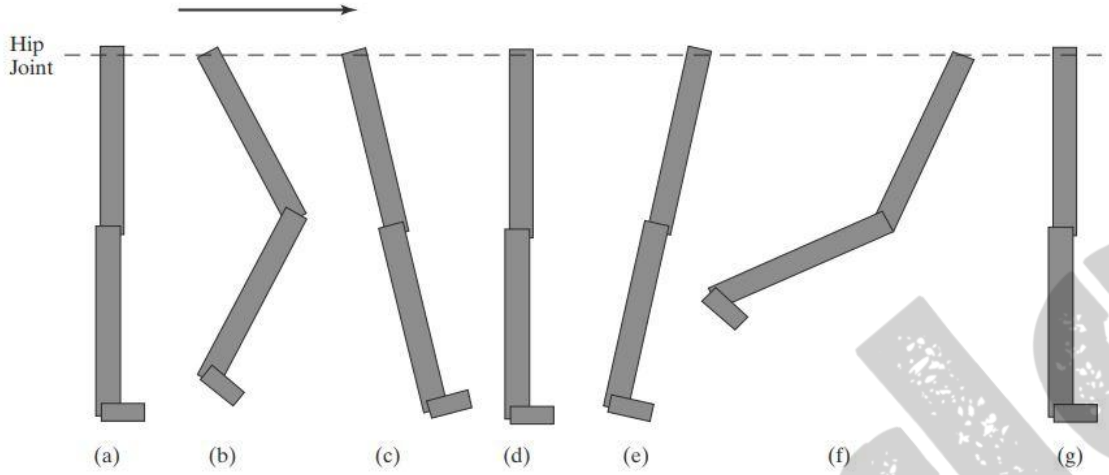

Figure 3.18: Possible motions for a set of connected links representing a walking leg.

As a figure moves, other movements are incorporated into the various joints. A sinusoidal motion, often with varying amplitude, can be applied to the hips so that they move about on the torso. Similarly, a rolling or rocking motion can be imparted to the shoulders, and the head can bob up and down.

Motion Capture

An alternative to determining the motion of a character computationally is to digitally record the movement of a live actor and to base the movement of an animated character on that information. This technique, known as ***motion capture*** or ***mo-cap***, can be used when the movement of the character is predetermined (as in a scripted scene). The animated character will perform the same series of movements as the live actor.

The classic motion capture technique involves placing a **set of markers** at strategic positions on the actor's body, such as the arms, legs, hands, feet, and joints. It is possible to place the markers directly on the actor, but more commonly they are affixed to a special skintight body suit worn by the actor. The actor is them filmed performing the scene. Image processing techniques are then used to identify the positions of the markers in each frame of the film, and their positions are translated to coordinates. These coordinates are used to determine the positioning of the body of the animated character. The movement of each marker from frame to frame in the film is tracked and used to control the corresponding movement of the animated character.

To accurately determine the positions of the markers, the scene must be filmed by multiple cameras placed at fixed positions. The digitized marker data from each recording can then be used to triangulate the position of each marker in three dimensions. Typical motion capture systems will use up to two dozen cameras.

Optical motion capture systems rely on the reflection of light from a marker into the camera. These can be relatively simple passive systems using photoreflective markers that reflect illumination from special lights placed near the cameras, or more advanced active systems in which the markers are powered and emit light.

Non-optical systems rely on the direct transmission of position information from the markers to a recording device. Some non-optical systems use inertial sensors that provide gyroscope-based position and orientation information.

Some motion capture systems record more than just the gross movements of the parts of the actor's body. It is possible to record even the actor's facial movements. Often called ***performance capture*** **systems**, these typically use a camera trained on the actor's face and small light-emitting diode (LED) lights that illuminate the face. Small photoreflective markers attached to the face reflect the light from the LEDs and allow the camera to capture the small movements of the muscles of the face, which can then be used to create realistic facial animation on a computer-generated character.

Periodic Motions

When animation is constructed with repeated motion patterns, such as a rotating object, the motion should be sampled frequently enough to represent the movements correctly. The motion must be synchronized with the frame-generation rate so that enough frames are displayed per cycle to show the true motion. Otherwise, the animation may be displayed incorrectly

A typical example of an undersampled periodic-motion display is the wagon wheel in a Western movie that appears to be turning in the wrong direction. Figure 3.19 illustrates one complete cycle in the rotation of a wagon wheel with one red spoke that makes 18 clockwise revolutions per second. If this motion is recorded on film at the standard motion-picture projection rate of 24 frames per second, then the first five frames depicting this motion would be as shown in Figure

3.20. Because the wheel completes 3/4 of a turn every 1/24 of a second, only one animation frame is generated per cycle, and the wheel thus appears to be rotating in the opposite (counterclockwise) direction.

In a computer-generated animation, the sampling rate in a periodic motion can be controlled by adjusting the motion parameters. Angular increment for the motion of a rotating object can be set so that multiple frames are generated in each revolution. Thus, a 3° increment for a rotation angle produces 120 motion steps during one revolution, and a 4° increment generates 90 steps. For faster motion, larger rotational steps could be used.

The motion of a complex object can be much slower than we want it to be if it takes too long to construct each frame of the animation.

Another factor that we need to consider in the display of a repeated motion is the effect of round-off in the calculations for the motion parameters. We can reset parameter values periodically to prevent the accumulated error from producing erratic motions. For a continuous rotation, we could reset parameter values once every cycle (360º).

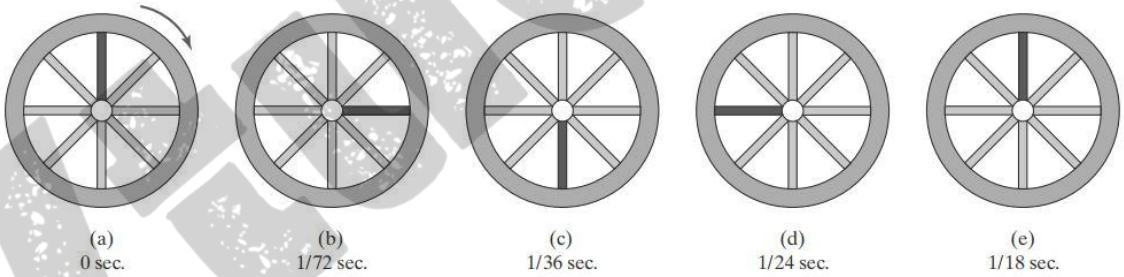

Figure 3.19: Five positions for a red spoke during one cycle of a wheel motion that is turning at the rate of 18 revolutions per second.

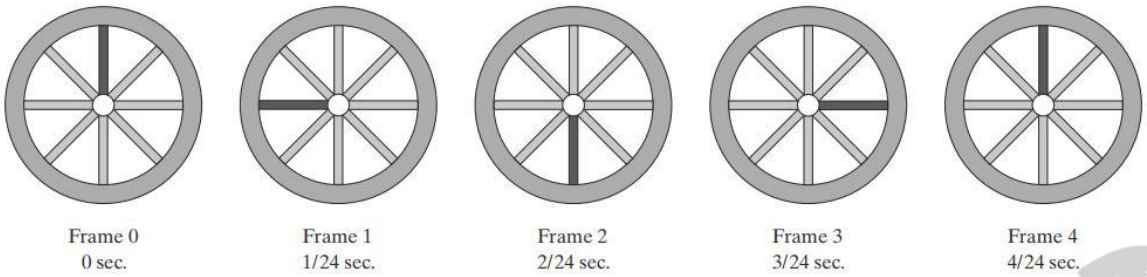

Figure 3.20: The first five film frames of the rotating wheel in Figure 19 produced at the rate of 24 frames per second

OpenGL Animation Procedures

Raster operations and color-index assignment functions are available in the core library, and routines for changing color-table values are provided in GLUT. Other raster-animation operations are available only as GLUT routines because they depend on the window system in use. In addition, computer-animation features such as double buffering may not be included in some hardware systems.

Double-buffering operations, if available, are activated using the following GLUT command:

glutInitDisplayMode (GLUT_DOUBLE);

This provides two buffers, called the ***front buffer*** and the ***back buffer,*** that we can use alternately to refresh the screen display. While one buffer is acting as the refresh buffer for the current display window, the next frame of an animation can be constructed in the other buffer. We specify when the roles of the two buffers are to be interchanged using

glutSwapBuffers ( );

To determine whether double-buffer operations are available on a system, we can issue the following query:

glGetBooleanv (GL_DOUBLEBUFFER, status);

A value of **GL_TRUE** is returned to array parameter **status** if both front and back buffers are available on a system. Otherwise, the returned value is **GL_FALSE**.

For a continuous animation, we can also use

glutIdleFunc (animationFcn);

where parameter **animationFcn** can be assigned the name of a procedure that is to perform the operations for incrementing the animation parameters. This procedure is continuously executed whenever there are no display-window events that must be processed. To disable the **glutIdleFunc**, we set its argument to the value **NULL** or the value 0

Question Bank

1. Explain in detail about logical classification of input devices.
2. Explain request mode, sample mode and event mode.
3. Explain in detail about interactive picture construction techniques.
4. Write a note on virtual reality environment.
5. Explain different OpenGL interactive Input-Device functions.
6. Explain OpenGL menu functions in detail.
7. Explain about designing a graphical user interface.
8. Write a note on OpenGL Animation Procedures.
9. Explain character animation in detail.
10. Write a note on computer animation languages.
11. Explain briefly about general computer animation functions.
12. Explain in detail about traditional animation techniques.
13. Explain in detail about different stages involved in design of animation sequences.
14. Write a note on periodic motion.