Module 3

Q1. Define computer animation. Explain the stages to design animation sequences.

Ans : 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.

Q2. Discuss Interactive picture construction techniques.

Ans: 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.

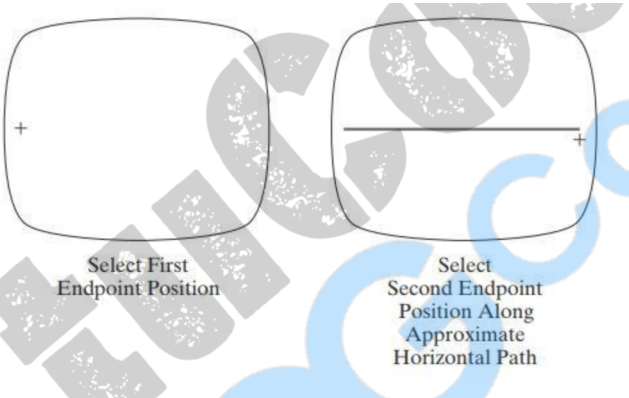


Figure 3.3: Horizontal line constraint.

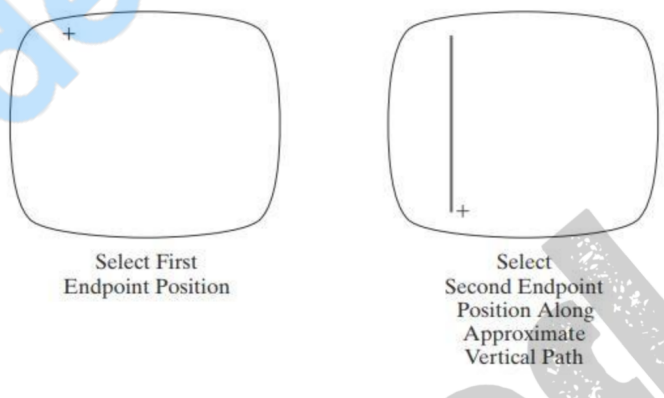
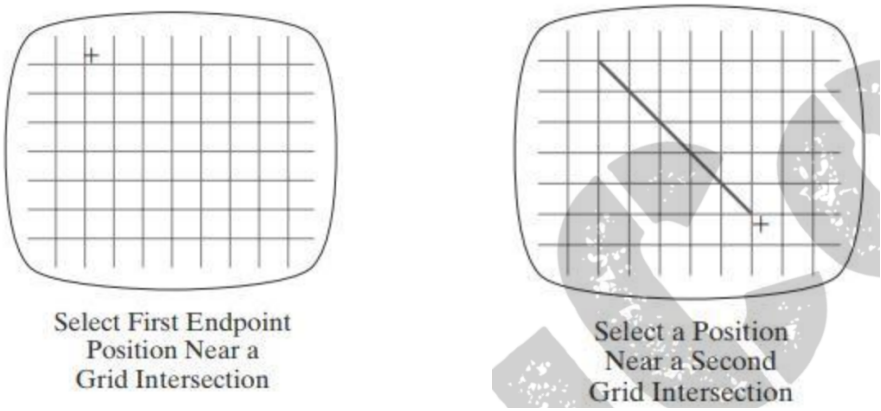


Figure 3.4: Vertical line constraint.

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

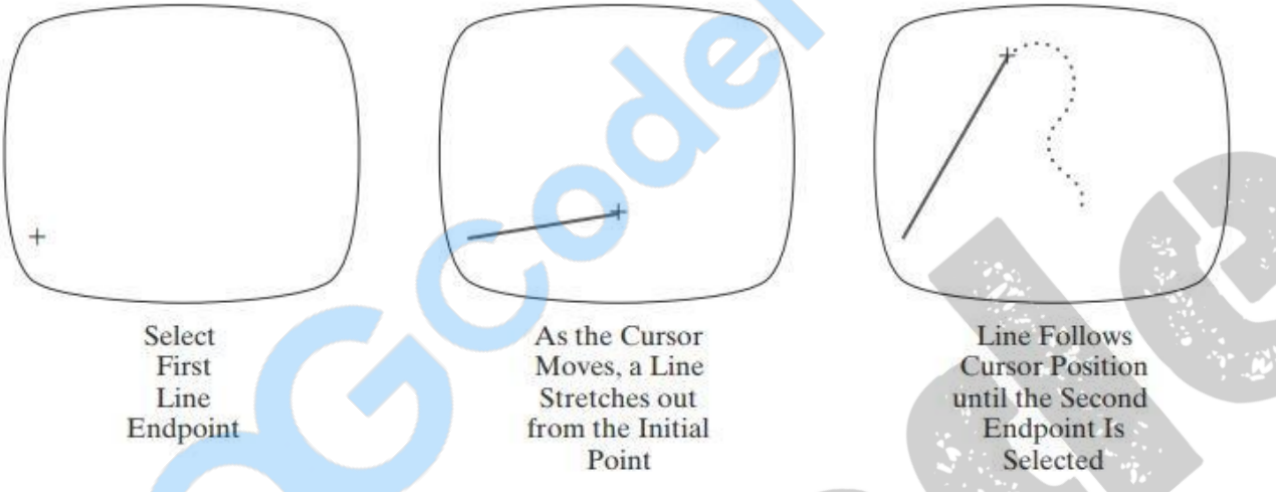
Another kind of constraint is a rectangular grid displayed in some part of the screen area. With an activated grid constraint, input coordinates are rounded to the nearest grid intersection. Figure 3.5 illustrates line drawing using a grid. Each of the cursor positions in this example is shifted to the 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.



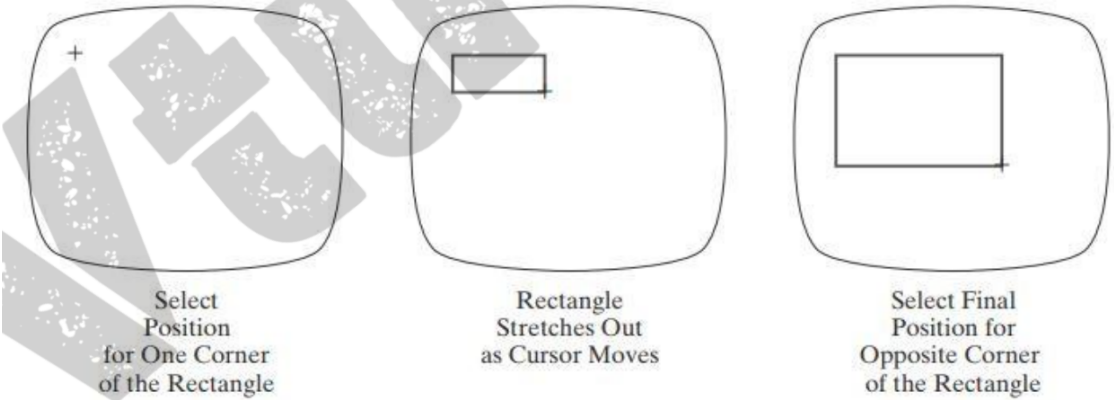
Construction of a line segment with endpoints constrained to grid intersection positions.

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.



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.



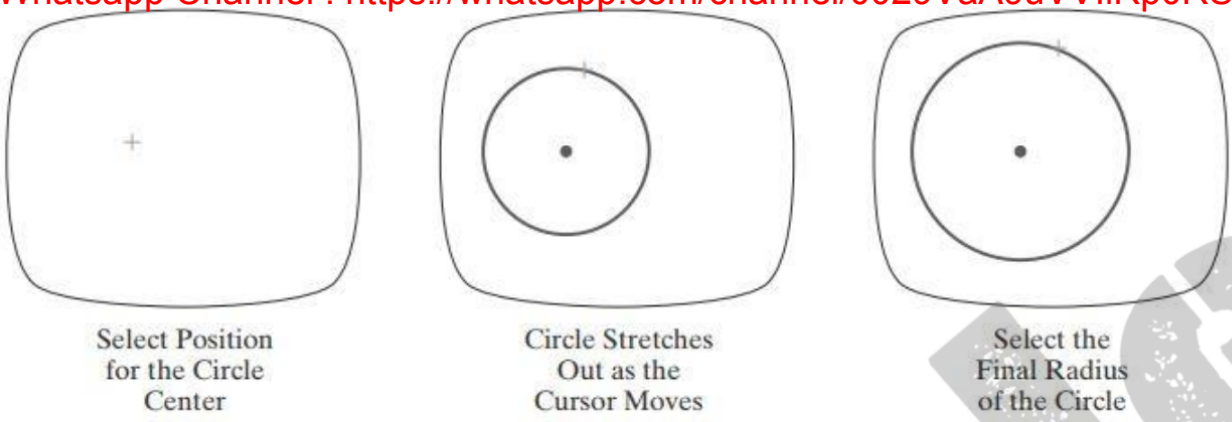


Figure 3.8: Constructing a circle using a rubber-band method.

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.

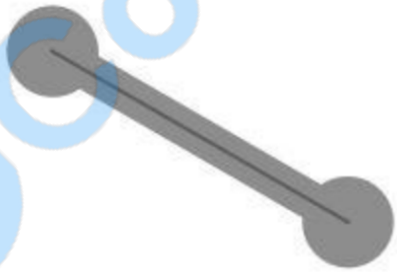


Figure 3.9:A gravity field around a line. Any selected point in the shaded area is shifted to a position on the line.

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.

Q3. Write a note on OpenGL Animation Procedures.

Ans: 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

Q4. Explain character animation in detail.

Ans : 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.

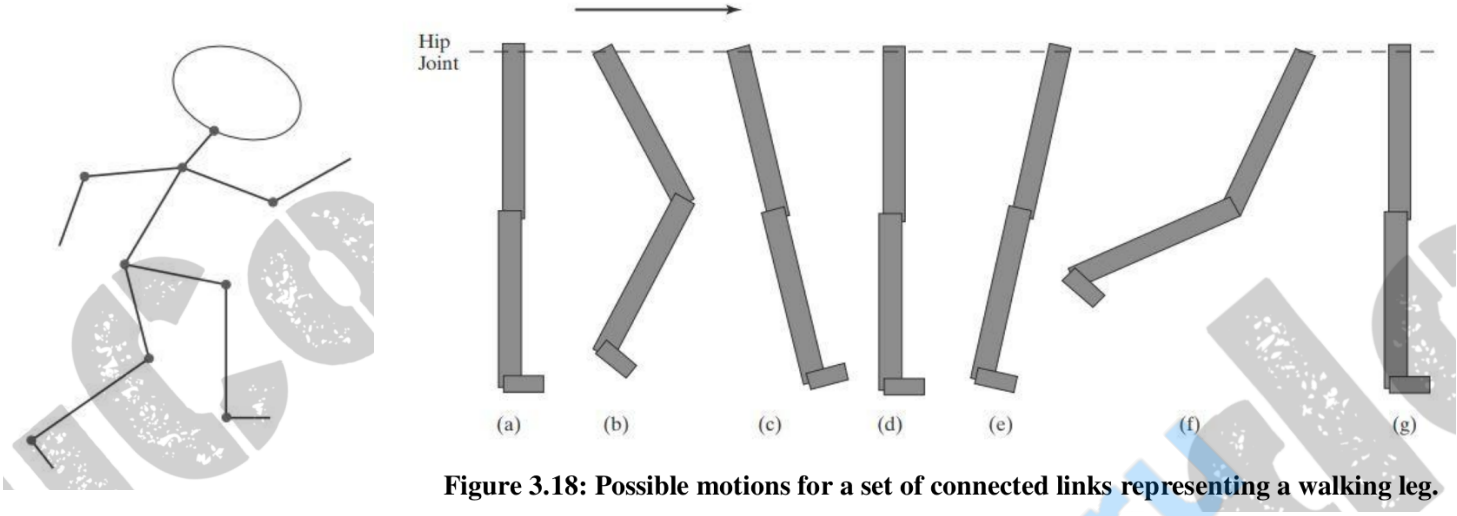


Figure 3.18: Possible motions for a set of connected links representing a walking leg.

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.

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.

Q5. Explain the logical classification of input devices.

Ans: 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 xmin, xmax, ymin, and ymax is selected if the input coordinates satisfy the inequalities

$x_{min} \leq x \leq x_{max}, y_{min} \leq y \leq y_{max}$ (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

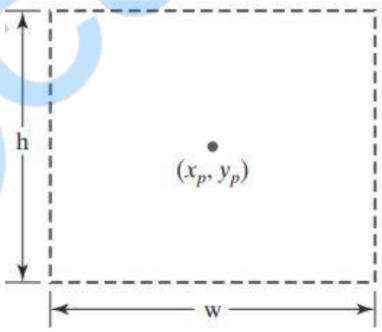
pressing a button or key to record the pixel coordinates. This screen position can then be used to select an entire object, a facet of a tessellated surface, a polygon edge, or a vertex. Other pick methods include highlighting schemes, selecting objects by name, or a combination of methods.

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 (x1, y1) and (x2, y2), the perpendicular distance squared from a pick position (x, y) to the line is calculated as

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

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.



A pick window with center coordinates (xp, yp), 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.

Q6. How are menus and submenus are created in open-gl? Illustrate with an example.

Ans: 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.

```
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.

Q7. Explain the various Mouse and Keyboard functions available in GLUT.

Ans : 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.

1. 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.

2. 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 The display-window mouse location is returned as position ASCII code. (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.

3. 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.

4. 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)

5. 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.

6. 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.

Q8. Explain the various components and types of Computer Animation Languages.

Ans: Computer animation involves creating moving images by defining and controlling sequences of visual data. To facilitate the development of animations, several specialized animation languages and tools have been developed. These languages and tools provide functionalities that go beyond general-purpose programming languages, offering dedicated components for designing, controlling, and rendering animations. Here's a detailed explanation of the various components and types of computer animation languages:

Components of Computer Animation Languages

1. **Graphics Editor**
    - **Purpose:** Allows animators to design and modify object shapes and scenes.
    - **Features:**
        - **Spline Surfaces:** Enable smooth and flexible curve-based modeling of shapes.
        - **Constructive Solid Geometry (CSG):** Allows for creating complex objects by combining simpler shapes using boolean operations (union, intersection, difference).
        - **Other Representation Schemes:** Such as parametric surfaces or voxel-based representations.
    - **Use Case:** Design and edit objects and environments to be used in animations.
2. **Key-Frame Generator**
    - **Purpose:** Defines the major positions or states of objects at specific points in time (key frames).
    - **Functionality:**
        - **Key Frames:** Key frames capture essential poses or configurations of objects. The animator specifies these frames to establish important moments in the animation sequence.
        - **In-Between Generator:** Automatically generates intermediate frames (in-betweens) between key frames to create smooth transitions.
    - **Use Case:** Define critical poses or configurations and let the system interpolate between them.
3. **In-Between Generator**
    - **Purpose:** Creates intermediate frames between key frames to produce fluid motion.
    - **Functionality:**
        - **Interpolation Methods:** Can use linear interpolation, spline interpolation, or other methods to generate smooth transitions.
        - **Automation:** Reduces the need for manual frame-by-frame animation, making it easier to produce smooth motion.
    - **Use Case:** Automatically generate smooth transitions between key frames to create realistic animations.

4. **Standard Graphics Routines**
    - **Purpose:** Provide essential functions for rendering and manipulating graphics.
    - **Functionality:**
        - **Viewing and Perspective Transformations:** Adjusts how objects are viewed and projected onto the screen.
        - **Geometric Transformations:** Includes translation, rotation, and scaling to move and manipulate objects.
        - **Visible-Surface Identification:** Determines which surfaces are visible in a scene and which are obscured.
        - **Surface-Rendering Operations:** Includes techniques for shading, texture mapping, and other visual effects.
    - **Use Case:** Perform fundamental operations required for rendering scenes and objects.

Types of Computer Animation Languages

1. **Key-Frame Systems**
    - **Purpose:** Focus on defining key frames and generating in-betweens.
    - **Features:**
        - **Rigid Body Animation:** Defines objects as rigid bodies with joints and limited degrees of freedom.
        - **Degrees of Freedom:** Controls movement of objects, such as a robot arm with various rotational and translational movements.
    - **Example:** Animation software that allows animators to specify key frames and automatically interpolates intermediate frames.
2. **Parameterized Systems**
    - **Purpose:** Allows for detailed control over motion and shape characteristics.
    - **Features:**
        - **Adjustable Parameters:** Control degrees of freedom, motion limits, and shape changes.
        - **Dynamic Behavior:** Parameters can be adjusted to modify object behavior and appearance during animation.
    - **Example:** Systems where animators can tweak parameters to control the flexibility of a character's movement or deformation of objects.
3. **Scripting Systems**
    - **Purpose:** Use scripts to define object specifications and animation sequences.
    - **Features:**
        - **Script-Based Definitions:** Allows users to define objects and animations through a scripting language.
        - **Library Construction:** Build and manage a library of objects and motions using scripts.
    - **Example:** A scripting language that allows users to write scripts for creating and animating complex scenes or simulations.

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:
o Set up a pick buffer.
o Activate the picking operations (selection mode).
o Initialize an ID name stack for object identifiers.
o Save the current viewing and geometric-transformation matrix.
o Specify a pick window for the mouse input.
o Assign identifiers to objects and reprocess the scene using the revised view volume. (Pick information is then stored in the pick buffer.)
o Restore the original viewing and geometric-transformation matrix.
o Determine the number of objects that have been picked, and return to the normal rendering mode.
o 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 ofthe viewport.When these coordinates are coordinates are relative to the given with mouse upper-left corner, input,the mouse andthusthe input yMouse value has to be inverted. The double-precision values for the width and height ofthepickwindowarespecifiedwithparameters **widthPick** and **heightPick**.Parameter vpArray designates an integer array containing the coordinate position and sizeparameters for the current viewport.