Module 5

Interactive Input Methods and Graphical User Interfaces

- It is often useful to be able to specify graphical input interactively.
- During the execution of a program, for example, we might want to change the position of the camera or the location of an object in a scene by pointing to a screen position, or we might want to change animation parameters using menu selections.
- There are several kinds of data that are used by a graphics program, and a variety of interactive input methods have been devised for processing these data values.
- In addition, interfaces for systems now involve extensive interactive graphics, including display windows, icons, menus, and a mouse or other cursor-control devices.

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.
- But input procedures require interaction with display-window managers and specific hardware devices.
- 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, although most input devices can handle some data types better than others

We can think about input devices in two distinct ways.

1. **Physical devices**

   -keyboard or a mouse

2. **Logical devices**

   - properties are specified in terms of what they do from the perspective of the application program

```
int x;
scanf(&x);
printf("%d",x);
```

Physical Input devices

Two primary types of physical devices:

- **pointing devices**
- **keyboard devices**

keyboard

- The most commonly used input device is a keyboard. The data is entered by pressing the set of keys. All keys are labeled. A keyboard with 101 keys is called a QWERTY keyboard.

Pointing devices

- Devices such as the mouse, trackball, joy stick return incremental inputs (or velocities) to the operating system.
- The computer can then integrate these values to obtain a two- dimensional position.
- Thus as a mouse cursor moves across a surface, the integrals of the velocities x,y values can be converted to indicate the position for a cursor on the screen.
- Devices such as the data tablet return a position directly to the operating system.
- A typical data tablet has rows and columns of wires embedded under its surface.
- The position of the stylus is determined through the electromagnetic interactions between the signals travelling through the wires and sensors in the stylus.

Logical Input Devices

- Graphics standards maintains device independence by using logical input devices.
- A logical input device act as an intermediate between physical input device and the application.
- That is user inputs data from the physical input device to application program via the logical input device.

- The two major characteristics describe the logical behavior of an input device:

  (1) the measurements that the device returns to the user program

  (2) the time when the device returns those measurements.

Six Classes of Logical Input Devices

- **String:** A string device is a logical device that provides ASCII strings to the user program, implemented by means of a physical keyboard.
- **Locator:** A locator device provides a position in the world coordinates to the user program, implemented by means of a pointing device, such as a mouse or a trackball.
- **Pick:** A pick device returns the identifier of an object on the display to the user program, implemented with the same physical device as a locator.
- **Choice:** Choice devices allow user to select one of the discrete number of options. In OpenGL, we can use various widgets provided by the window system. A widget is a graphical interactive device, provided by either the window system or a toolkit.
- **Valuator:** Valuators provide analog input to the user program. On some graphics systems, there are boxes or dials to provide valuator input.
- **Stroke:** A stroke device returns an array of locations. Although we can think of a stroke device as similar to multiple uses of a locator , it is often implemented such that an action say, pushing down a mouse button , starts the transfer of data into the specified array, and a second action , such as releasing the button , ends this transfer.

Measure and Trigger

- The manner by which physical and logical input devices provide input to an application program can be described in terms of two entities:

  1) A measure process, and 2) A device trigger.

- The **measure** of a device is what the device returns to the user program.
- The **trigger** of a device is a physical input on the device with which the user can signal the computer.

  Example: The measure of a keyboard is a string,

  The trigger could be the “return” or “enter” key.

  Example: For a locator the measure includes the location and

  The trigger can be the button on the pointing device.

Input Modes

- In addition to multiple types of logical input devices, we can obtain the measure of a device in three distinct modes:

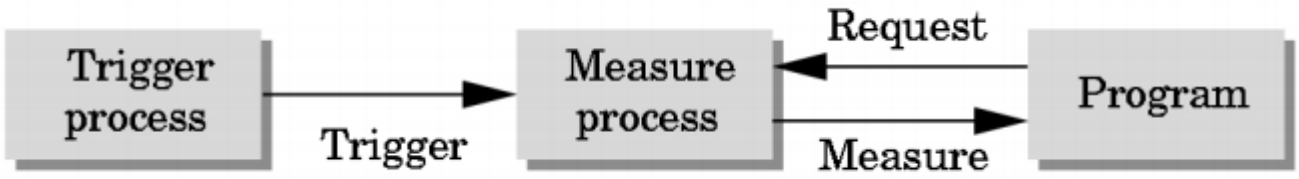
  1) Request mode, 2) Sample mode, and 3) Event mode.

- It defined by the relationship between the measure process and the trigger.
- Normally, the initialization of an input device starts a measure process.

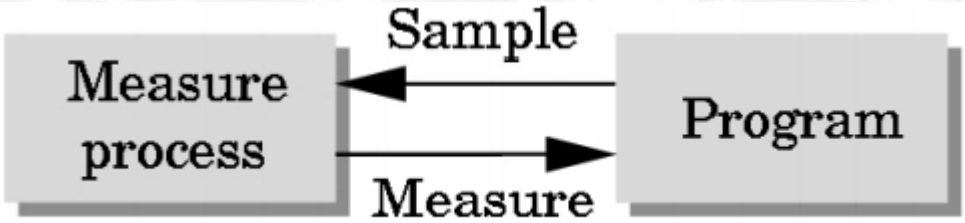
Request mode:

In this mode the measure of the device is not returned to the program until the device is triggered.

Ex: A locator can be moved to different point of the screen. The Windows system continuously follows the location of the pointer, but until the button is depressed, the location will not be returned.

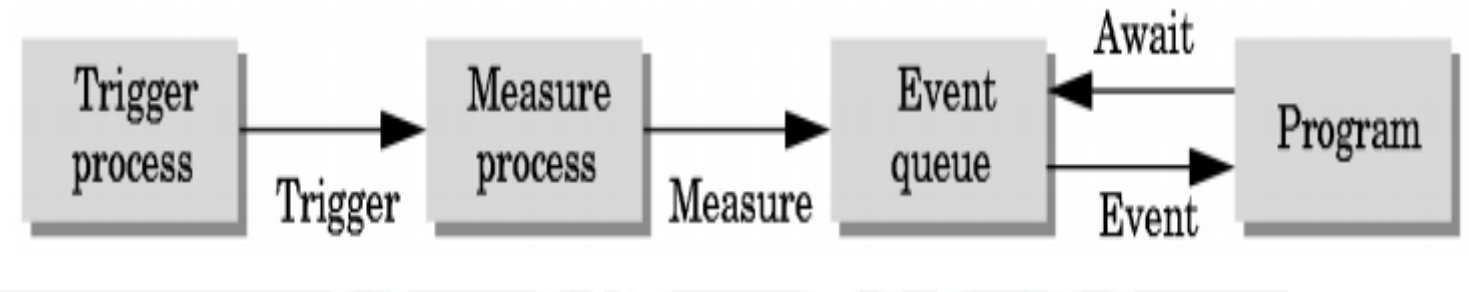


Sample mode: Input is immediate. As soon as the function call in the user program is encountered, the measure is returned, hence no trigger is needed.



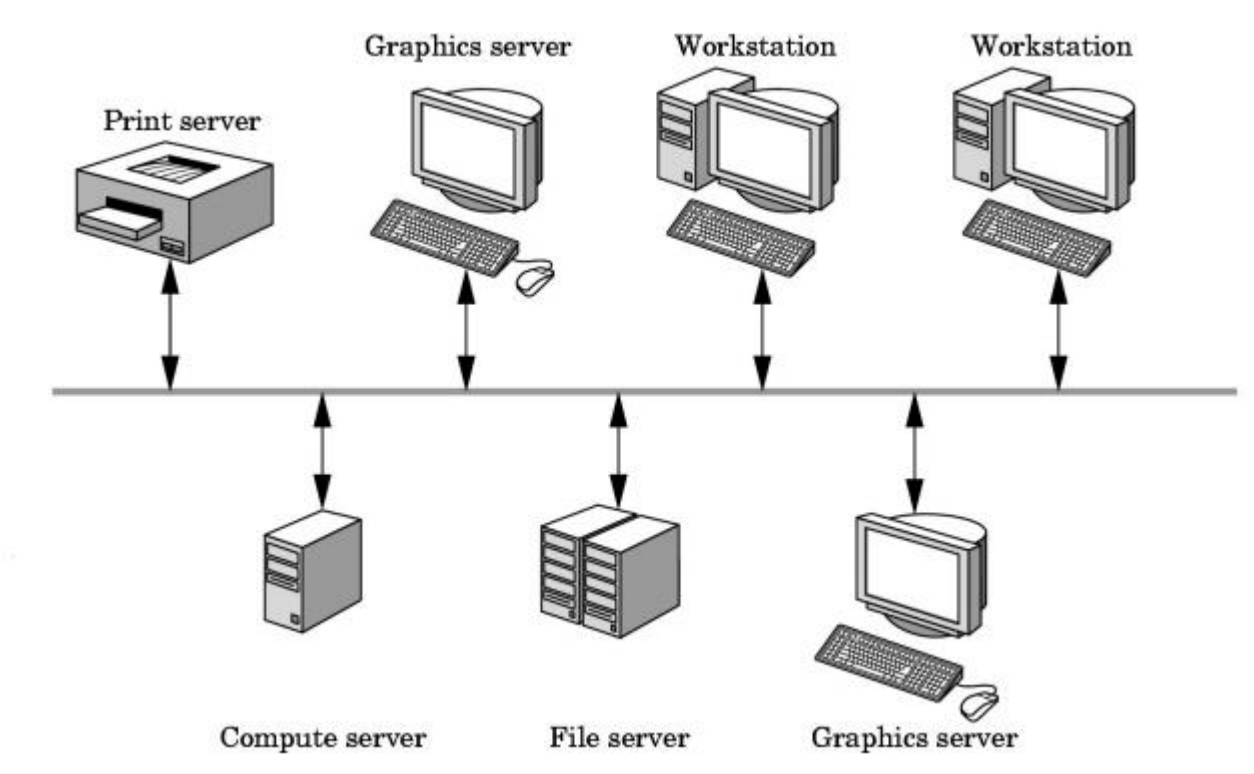
Event mode: The previous two modes are not sufficient for handling the variety of possible human-computer interactions that arise in a modern computing environment. In an environment with multiple input devices, each with its own trigger and each running a measure process. Each time that a device is triggered, an event is generated.

- The device measure, with the identifier for the device, is placed in an event queue. The user program executes the events from the queue.
- When the queue is empty, it will wait until an event appears there to execute it.
- Another approach is to associate a function called a callback with a specific type of event. This is the approach we are taking.



Clients and Servers

- In this world our building blocks are entities called servers that can perform tasks for clients.
- Examples of servers include print servers, which can allow sharing of high-speed printer, compute servers, file servers.
- Users and the user programs that make use of these services are clients or client programs.



Display Lists

- Often it can be convenient or more efficient to store an object description as a named sequence of statements.
- We can do this in OpenGL using a structure called a **display list.**
- Once a display list has been created, we can reference the list multiple times with different display operations.
- On a network, a display list describing a scene is stored on the server machine, which eliminates the need to transmit the commands in the list each time the scene is to be displayed.
- We can also set up a display list so that it is saved for later execution, or we can specify that the commands in the list be executed immediately.

Display Lists

- Display lists illustrate how we can use clients and servers on a network to improve interactive graphics performance.
- And display lists are particularly useful for hierarchical modeling, where a complex object can be described with a set of simpler subparts.
- It provides the advantage of reduced network traffic.
- Executing the commands once and the results are stored in display list on the graphics server.
- Disadvantage: requires memory on the server.

Creating and Naming an OpenGL Display List

- A set of OpenGL commands is formed into a display list by enclosing the commands within the glNewList/glEndList pair of functions. For example,

```
glNewList (listID, listMode};
.
.
.
glEndList ( );
```

- This structure forms a display list with a positive integer value assigned to parameter listID as the name for the list.
- Parameter listMode is assigned an OpenGL symbolic constant that can be either **GL_COMPILE or GL_COMPILE_AND_EXECUTE.**
- If we want to save the list for later execution, we use GL_COMPILE. Otherwise, the commands are executed as they are placed into the list, in addition to allowing us to execute the list again at a later time.

- As a display list is created, expressions involving parameters such as coordinate positions and color components are evaluated so that only the parameter values are stored in the list.
- Any subsequent changes to these parameters have no effect on the list. Because display-list values cannot be changed, we cannot include certain OpenGL commands, such as vertex-list pointers, in a display list.
- We can create any number of display lists, and we execute a particular list of commands with a call to its identifier. Further, one display list can be embedded within another display list.
- We can let OpenGL generate an identifier for us, as follows:

  listID = glGenLists (1);

- This statement returns one (1) unused positive integer identifier to the variable listID. A range of unused integer list identifiers is obtained if we change the argument of glGenLists from the value 1 to some other positive integer.

- To determine whether a specific integer value has been used as a list name. The function to accomplish this is

  glIsList (listID};

- A value of GL_TRUE is returned if the value of listID is an integer that has already been used as a display-list name. If the integer value has not been used as a list name, the glIsList function returns the value GL_FALSE.

Executing OpenGL Display Lists

- We execute a single display list with the statement

  glCallList (listID);

- Several display lists can be executed using the following two statements:

  glListBase (offsetValue);

  glCallLists (nLists, arrayDataType, listIDArray);

Deleting OpenGL Display Lists

- We eliminate a contiguous set of display lists with the function call

  glDeleteLists (startID, nLists);

- ParameterstartIDgives the initial display-list identifier, and parameternLists specifies the number of lists that are to be deleted.
- For example, the statement

  glDeleteLists (5, 4);

- eliminates the four display lists with identifiers 5, 6, 7, and 8. An identifier value that references a nonexistent display list is ignored.

Programming Event Driven Input

- 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, we have 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, we specify a procedure (the **callback function**) that is to be 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

- A mouse event occurs when the mouse is moved with one of the buttons pressed, this event is called move event.
- If the mouse is moved without a button being held down, this event is called passive move event.
- After a move event, the position of the mouse – its measure – is made available to the application program.
- A mouse event occurs when one of the mouse buttons is either pressed or released.
- We register the mouse callback function, usually in the main function, by means of the GLUT function as follows:

  glutMouseFunc (mouseFcn);

- This mouse callback procedure, which we 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 :GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, and GLUT_RIGHT_BUTTON
- parameter **action** is assigned a symbolic constant that specifies which button action we want to use to trigger the mouse activation event : 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

- 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 in this case 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.
- Similarly, we can perform some action when we move the mouse within the display window without pressing a button:

  glutPassiveMotionFunc (fcnDoSomethingElse);

- Again, 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, we use the following function 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, we can use the mouse location to initiate some action, independently of whether any mouse buttons are pressed.

```
void myKey(unsigned char key, int x, int y)
{
        if (key=='q' || key=='Q')
        exit(0);
}
```

- For function keys, arrow keys, and other special-purpose keys, we can use the command

  glutSpecialFunc (specialKeyFcn);

- The specified procedure has the same 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: constants GLUT KEY F1 through GLUT KEY F12.
- For the arrow keys: constants such as GLUT KEY UP and GLUT KEY RIGHT.
- Other keys can be designated using GLUT KEY PAGE DOWN, GLUT KEY HOME, and similar constants for the page up, end, and insert keys.
- The backspace, delete, and escape keys can be designated with the glutKeyboardFunc routine using their ASCII codes, which are 8, 127, and 27, respectively.

Window Events

- Most windows system allows user to resize window.
- This is a window event and it poses several problems like
  - Do we redraw all the images
  - The aspect ratio
  - Do we change the size or attributes of the primitives to suit the new window

- To allow us to compensate for a change in display- window dimensions, the
- GLUT library provides the following routine:

  glutReshapeFunc (winReshapeFcn);

- We can include this function in the main procedure in our program, along with the other GLUT routines, and it will be activated whenever the display-window size is altered.
- The argument for this GLUT function is the name of a procedure that is to receive the new display-window width and height.
- We can then use the new dimensions to reset the projection parameters and perform any other operations, such as changing the display-window color.
- In addition, we could save the new width and height values so that they could be used by other procedures in our program

```
void winReshapeFnc(GLsizei w, GLsizei h)
{
/* first adjust clipping box */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,(GLdouble)w, 0.0, (GLdouble)h);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
/* adjust viewport */
glViewport(0,0,w,h);
}
```

glutPostRedisplay

- Interactive and animation programs might contain many calls for the reexecution of the display function.
- If we have set up multiple display windows, then we have to repeat the display process for each of the display windows or subwindows.
- Also, we may need to call glutDisplayFunc after the glutPopWindow command if the display window has been damaged during the process of redisplaying the windows.
- In this case, the following function is used to indicate that the contents of the current display window should be renewed:

  glutPostRedisplay ( );

- This routine is also used when an additional object, such as a pop-up menu, is to be shown in a display window.
- Renews the contents of the current display window

- glutPostRedisplay() – Calling this function sets a flag inside GLUT's main loop indicating that the display needs to be redrawn.
- At the end of each execution of the main loop, GLUT uses this flag to determine if the display function will be executed.
- The function ensures that the display will be drawn only once each time the program goes through the event loop.

```
void rotateHex (void)
{
        rotTheta += 3.0;
        if (rotTheta > 360.0)
                rotTheta -= 360.0;
        glutPostRedisplay ( );
}
```

Idle Callback

- Idle Callback is invoked when there are no other events to be performed.
- Its typical use is to continuously generate graphical primitives when nothing else is happening.

  glutIdleFunc(function name)

- The parameter for this GLUT routine could reference a background function or a procedure to update parameters for an animation when no other processes are taking place.
- Specifies a function to execute when the system is idle
- This procedure is continuously executed whenever there are no display-window events that must be processed.
- To disable the glutIdleFunc, set its argument to the value NULL or the value 0.

```
void mouseFcn (GLint button, GLint action, GLint x, GLint y)
{
        switch (button) {
                case GLUT_MIDDLE_BUTTON: // Start the rotation.
                        if (action == GLUT_DOWN)
                                glutIdleFunc (rotateHex);
                                break;
                case GLUT_RIGHT_BUTTON: // Stop the rotation.
                        if (action == GLUT_DOWN)
                                glutIdleFunc (NULL);
                                break;
                default:
                        break;
                }
}
```

GLUT Display-Window

GLUT Display-Window Identifier

- Multiple display windows can be created for an application, and each is assigned a positive-integer display-window identifier, starting with the value 1 for the first window that is created.
- At the time that we initiate a display window, we can record its identifier with the statement

  windowID = glutCreateWindow ("A Display Window");

- Once we have saved the integer display-window identifier in variable name windowID, we can use the identifier number to change display parameters or to delete the display window.

Deleting a GLUT Display Window

- The GLUT library also includes a function for deleting a display window that we have created. If we know the display window's identifier, we can eliminate it with the statement

glutDestroyWindow (windowID);

Current GLUT Display Window

- When we specify any display-window operation, it is applied to the current display window, which is either the last display window that we created or the one we select with the following command:

glutSetWindow (windowID);

- we can query the system to determine which window is the current display window:

  currentWindowID = glutGetWindow ( );

- A value of 0 is returned by this function if there are no display windows or if the current display window was destroyed.
- We can expand the current display window to fill the screen:

  glutFullScreen ( );

- The exact size of the display window after execution of this routine depends on the window-management system.
- To convert the current display window to an icon in the form of a small picture or symbol representing the window:

  glutIconifyWindow ( );

- The label on this icon will be the same name that we assigned to the window, but we can change this with the following command:

  glutSetIconTitle ("Icon Name");

- To change the name of the display window with a similar command:

  glutSetWindowTitle ("New Window Name");

- With multiple display windows open on the screen, some windows may overlap or totally obscure other display windows. We can choose any display window to be in front of all other windows by first designating it as the current window, and then issuing the “pop-window” command:

  glutSetWindow (windowID);

  glutPopWindow ( );

- In a similar way, we can “push” the current display window to the back so that it is behind all other display windows. This sequence of operations is

  glutSetWindow (windowID);

  glutPushWindow ( );

- We can also take the current window off the screen with

  glutHideWindow ( );

- In addition, we can return a “hidden” display window, or one that has been converted to an icon, by designating it as the current display window and then invoking the function

  glutShowWindow ( );

- We create a subwindow with the following function:

glutCreateSubWindow (windowID, xBottomLeft, yBottomLeft, width, height);

- Parameter windowID identifies the display window in which we want to set up the subwindow.
- With the remaining parameters, we specify the subwindow's size and the placement of its lower-left corner relative to the lower-left corner of the display window.

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.

- Once we have designated the menu function that is to be invoked when a menu item is selected, we must specify the options that are to be listed in the menu.
- We do this with a series of statements that list the name and position for each option. These statements have the general form

glutAddMenuEntry (charString, menuItemNumber);

- Parameter charString specifies text that is to be displayed in the menu, and parameter menuItemNumber gives the location for that entry in the menu.
- For example

```
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);
```

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. We can add the submenu 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);
```

Double Buffering

- Double Buffering is a method for producing a real-time animation with a raster system is to employ two refresh buffers.
- Initially, we create a frame for the animation in one of the buffers. Then, while the screen is being refreshed from that buffer, we construct the next frame in the other buffer.
- When that frame is complete, we switch the roles of the two buffers so that the refresh routines use the second buffer during the process of creating the next frame in the first buffer.
- This alternating buffer process continues throughout the animation.
- Graphics libraries that permit such operations typically have one function for activating the double buffering routines and another function for interchanging the roles of the two buffers.

- 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 ( );