Interactive Input Methods and Graphical User Interfaces, Computer Animation

Syllabus:

Interactive Input Methods and Graphical User Interfaces: *Graphical Input Data, Logical Classification of Input Devices, Input Functions for Graphical Data, Interactive Picture-Construction Techniques, Virtual-Reality Environments, OpenGL Interactive Input-Device Functions, OpenGL Menu Functions, Designing a Graphical User Interface.*

Computer Animation: *Design of Animation Sequences, Traditional Animation Techniques, General Computer-Animation Functions, Computer-Animation Languages, Character Animation ,Periodic Motions, OpenGL Animation Procedures.*

Resources:

1. *Donald D Hearn, M Pauline Baker and Warren Carithers: Computer Graphics with OpenGL 4th Edition, Pearson, 2014*

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 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, although most input devices can handle some data types better than others.
- Devices can be described either by

  1. Physical properties(Physical Device)-Mouse, Keyboard, Trackball etc.,

  2.Logical Properties(Logical Device)-What is returned to program via API? A position,An object identifier

Logical Classification of Input Devices

- Logical input device refers to input devices classified according to data type. The standard logical input-data classifications are:

  1. LOCATOR

  – A device for specifying one coordinate position.

  – i.e., return a position (mouse etc.,)

  2. STROKE

  – A device for specifying a set of coordinate positions.

  – i.e., return array of positions(mouse)

  3. STRING

  – A device for specifying text input.

  – i.e., return strings of characters(Keyboard)

4. **VALUATOR**
- A device for specifying a scalar value.
- i.e., return floating point number(widgets)

5. **CHOICE**
- A device for selecting a menu option.
- i.e., return one of n items(widgets)

6. **PICK**
- A device for selecting a component of a picture.
- i.e., return ID of an object (mouse etc.,)

LOCATOR- A device that allows the user to specify one coordinate position. Different methods can be used, such as a mouse cursor, where a location is chosen by clicking a button, or a cursor that is moved using different keys on the keyboard. Touch screens can also be used as locators; the user specifies the location by inducing force onto the desired coordinate on the screen.

STROKE-A device that allows the user to specify a set of coordinate positions. The positions can be specified, for example, by dragging the mouse across the screen while a mouse button is kept pressed. On release, a second coordinate can be used to define a rectangular area using the first coordinate in addition.

STRING-A device that allows the user to specify text input. A text input widget in combination with the keyboard is used to input the text. Also, virtual keyboards displayed on the screen where the characters can be picked using the mouse can be used if keyboards are not available to the application.

VALUATOR-A device that allows the user to specify a scalar value. Similar to string inputs, numeric values can be specified using the keyboard. Often, up-down-arrows are added to increase or decrease the current value. Rotary devices, such as wheels can also be used for specifying numerical values. Often times, it is useful to limit the range of the numerical value depending on the value.

CHOICE-A device that allows the user to specify a menu option. Typical choice devices are menus or radio buttons which provide various options the user can choose from. For radio buttons, often only one option can be chosen at a time. Once another option is picked, the previous one gets cleared.

PICK-A device that allows the user to specify a component of a picture. Similar to locator devices, a coordinate is specified using the mouse or other cursor input devices and then back-projected into the scene to determine the selected 3-D object. It is often useful to allow a certain “error tolerance”so that an object is picked even though the user did not exactly onto the object but close enough next to it. Also, highlighting objects within the scene can be used to traverse through a list of objects that fulfill the proximity criterion.

Certain applications do not allow the use of mouse or keyboard. In particular, 3-D environments, where the user roams freely within the scene, mouse or keyboard would unnecessarily bind the user to a certain location. Other input methods are required in these cases, such as a wireless gamepad or a 3-D stylus, that is tracked to identify its 3-D location.

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:
  - 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.
  - **Selection of a physical device** that is to provide input within a particular logical classification (for example, a mouse used as a stroke device).
  - **Selection of the input time and device** for a particular set of data values.

Input Modes

The manner by which Input devices provides Input to an application program can be described in terms of 2 entities:

1. A measure process- The measure of a device is what the input devices return information (their ***measure***) to the program.

Eg. Mouse returns position information, Keyboard returns ASCII code

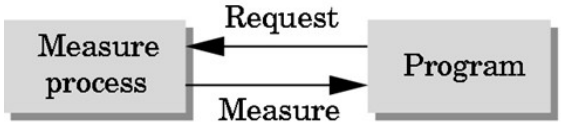
2. A device trigger- The trigger of a device is a physical Input on the device with which the user can send a signal to the operating system.

Eg. Button on mouse, Pressing or releasing a key.

The application program can obtain the measure of a device in **3 distinct modes**. Each mode is defined by the relationship between the measure process and the trigger.

1. Sample Mode

- Here the input is immediate.
- As soon as the function call in the user program is encountered, the measure is returned. Hence, no trigger is needed. This is as shown below:



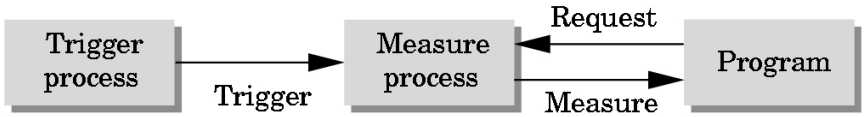
- In this mode, the user must have positioned the pointing device or entered data using keyboard before the function call, because the measure is extracted immediately from the buffer.

2. Request Mode

- Here, the measure of the device is not returned to the program until the device is triggered.
- The input mode is standard in non-graphical applications.
- Typical of keyboard input

  -Can erase (backspace), edit, correct until enter (return) key (the trigger) is depressed.

- The relationship between measure and trigger for request mode is shown in figure below:

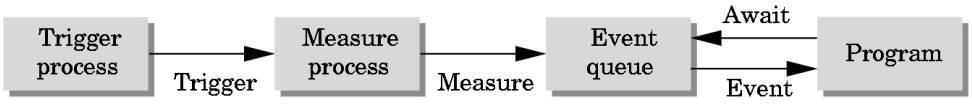


Note:

- One characteristic of both request and sample mode input in APIs is that the user must identify which device is to provide the input.
- Both request and sample modes are useful for situations where the program guides the user but are not useful in applications where the user controls the flow of the program.
- **For Eg.** : Consider a flight simulator, writing programs to control the simulator with only request and sample mode is impossible because we do not know what devices the pilot will use at any point in the simulation. Hence, these 2 modes are not sufficient in a modern computing environment.

Event Mode

- Used in the environment having more than one input device, each of which can be triggered at an arbitrary time by a user
- Each trigger generates an ***event*** whose measure is put in an ***event queue*** which can be examined by the user program.
- This is as shown in the figure:



- **For Eg.** If the first event is from the keyboard but the application program is not interested in keyboard input, the event can be discarded and the next event in the queue can be examined.
- Another approach is to associate a function called **callback** with a specific type of event. The OS queries or polls the event queue regularly and executes the **callbacks** corresponding to events in the queue.
- The different event types are:
    - Window: resize, expose, iconify.
    - Mouse: click one or more buttons.
    - Motion: move mouse.
    - Keyboard: press or release a key.
    - Idle: nonevent [Define what should be done if no other event is in queue].

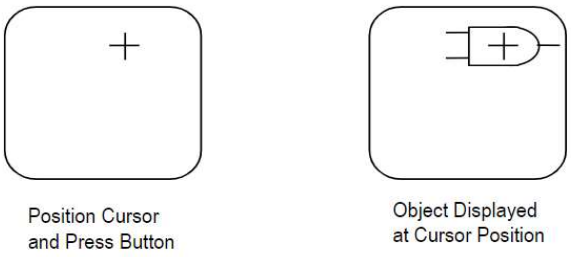
Interactive Picture-Construction Techniques

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.

Various Techniques are:

1. Basic Positioning Methods:

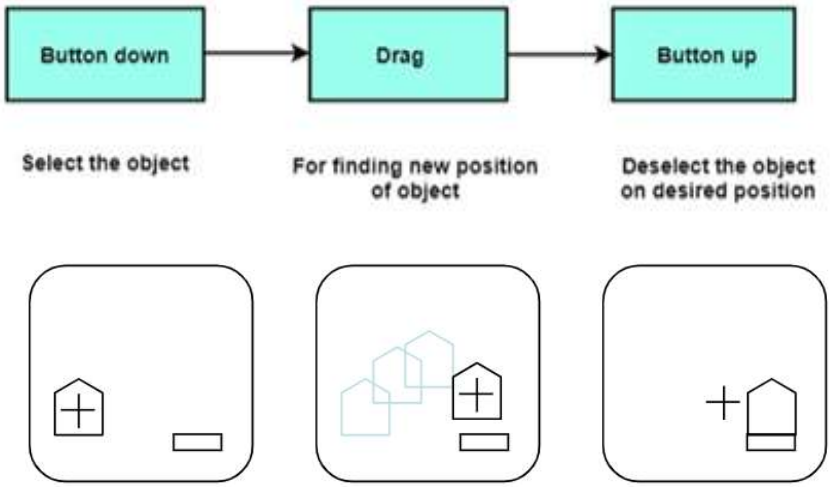
- It very basic technique of graphical input. It is also known as locating.
- With the help of input device, user indicates position on the screen. To display object this position marks location.
- **Example**: input position can be used to insert symbol to specify endpoint of line.
- The process of positioning involves two steps,
  - In first step user have to move cursor to desired spot on screen and
  - In second step user inform computer by pressing key or button.
- **Example:**



- The positioning is very often used in geometric modeling applications, where if user wants to define new element of model or to change position of already existing model.

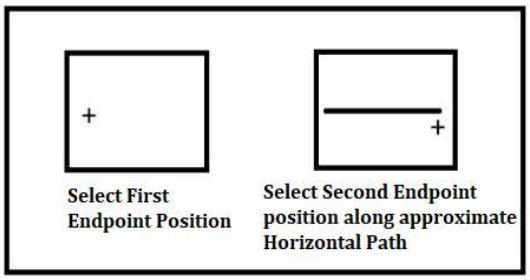
2. Dragging:

- 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.
- The following diagram represents the **dragging** procedure and an example.

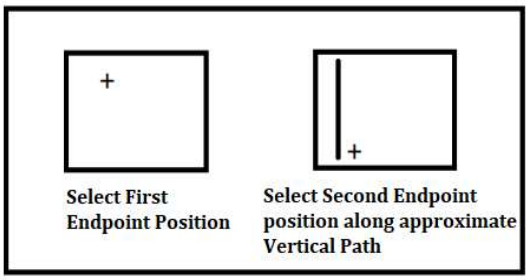


3. Constraints:

- 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 Figure Below:



Horizontal Line Constraint

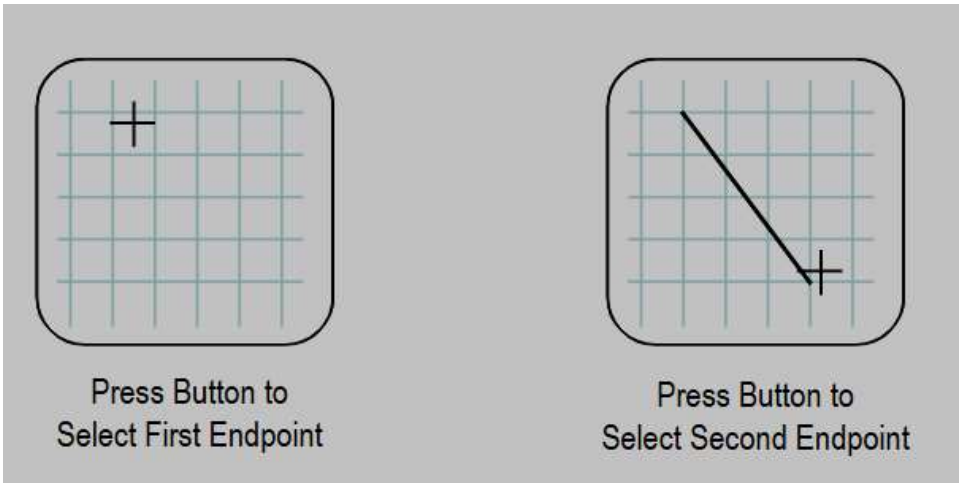


Vertical Line Constraint

- 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.
- The horizontal-vertical constraint is useful, for instance, in forming network layouts, and it eliminates the need for precise positioning of endpoint coordinates.
- 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 grid of rectangular lines displayed in some part of the screen area.
- When a grid is used, any input coordinate position is rounded to the nearest intersection of two grid lines.
- **Figure illustrates line drawing with a grid**. Each of the two cursor positions is shifted to the nearest grid intersection point, and the line is drawn between these grid points.



- 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.
- Grids can be turned on and off, and it is sometimes possible to use partials grids and grids

of different sizes in different screen areas.

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 below** 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, we construct a rubber-band line while pressing a mouse key. When the mouse key is released, the line display is completed.

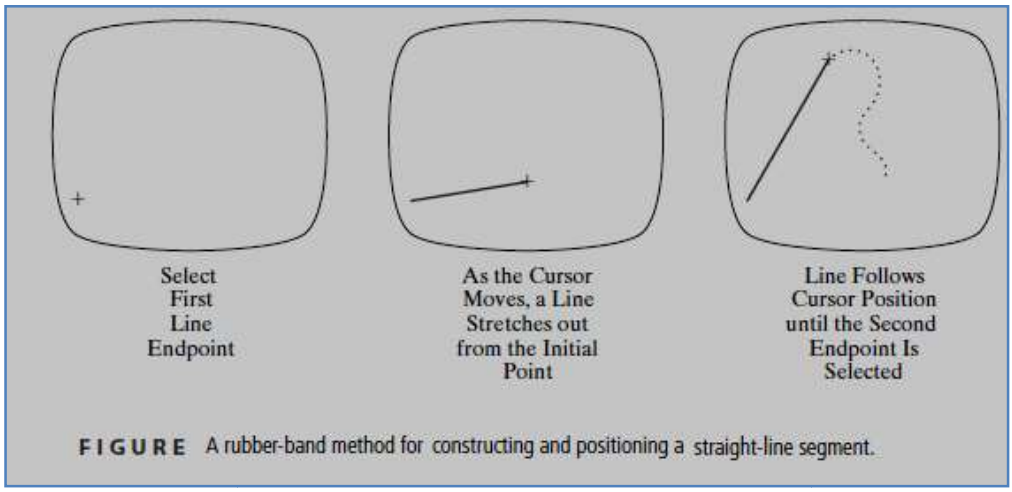


FIGURE A rubber-band method for constructing and positioning a straight-line segment.

- We can use similar rubber-band methods to construct rectangles, circles, and other objects.
- Figures below demonstrates rubber-band construction of a rectangle, and circle.

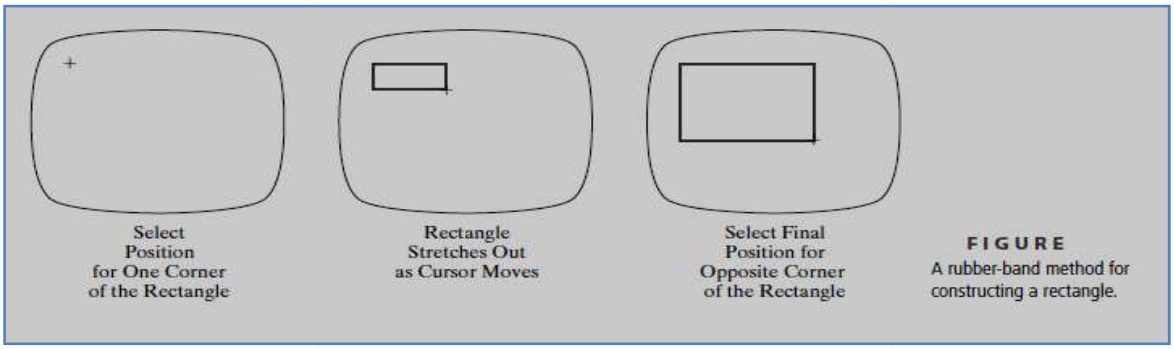


FIGURE
A rubber-band method for constructing a rectangle.

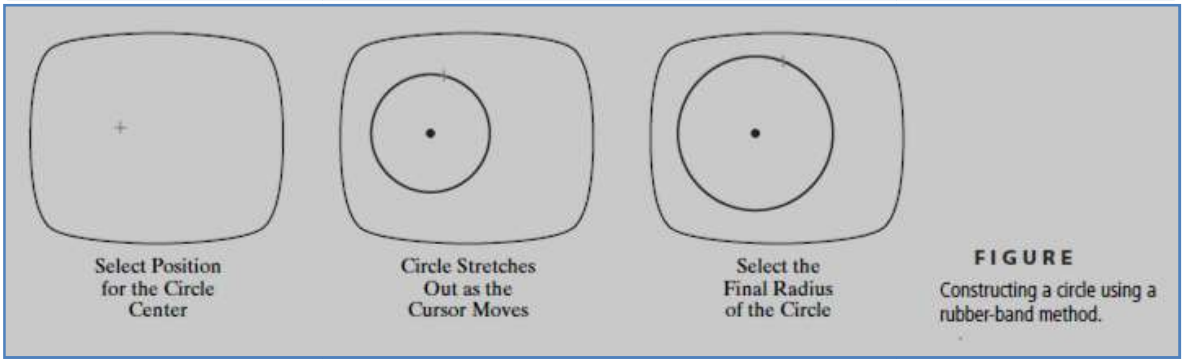


FIGURE
Constructing a circle using a rubber-band method.

6. Gravity Field:

- In the construction of figures, we sometimes need to connect lines at positions between endpoints.
- 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. This conversion of input position is accomplished by creating 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**.
- 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.

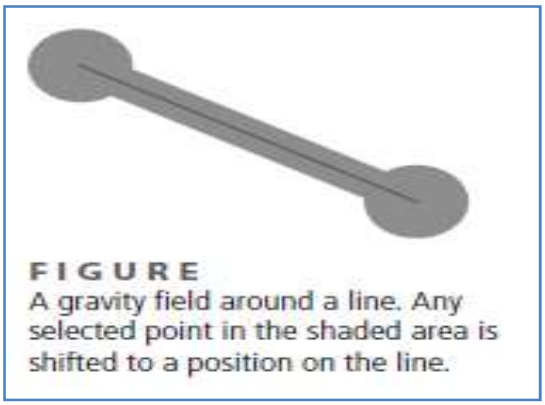


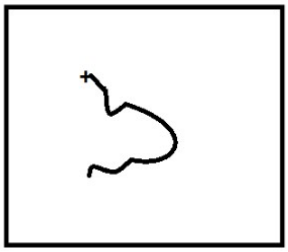
FIGURE
A gravity field around a line. Any selected point in the shaded area is shifted to a position on the line.

- 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.

7. Interactive Painting & Drawing Methods:

- Options for sketching, drawing, and painting come in a variety of forms.
- Straight lines, polygons, and circles can be generated with methods discussed in the previous sections.
- Curve drawing options can be provided using standard curve shapes, such as circular a rcs, or with freehand sketching procedures.
- Splines are interactively constructed by specifying a set of discrete screen points that give 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 posit ions 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 pattern are also available on many systems, particularly those designed as artist's system. Also allows an artist to select variations of a specified object shape, different surface textures, and a variety of lighting conditions for a scene.



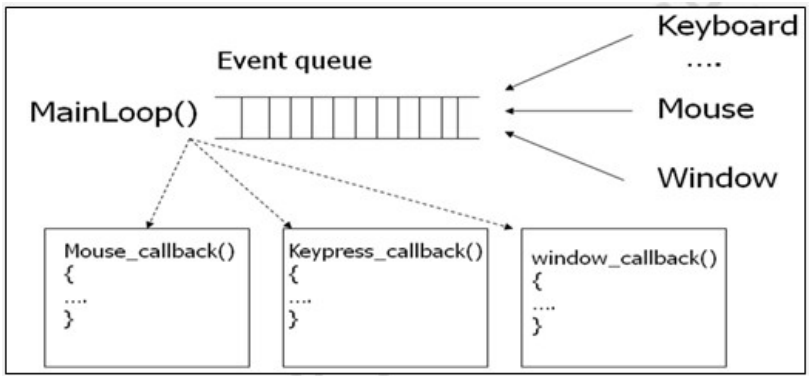
OpenGL Interactive Input-Device Functions

- 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, we have functions to accept input from standard devices, such as a mouse or a keyboard, etc.,. For each device, we specify a procedure (**the call back 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 recognizes a subset of the events recognized be any particular window system, as shown in the table below:

Events in OpenGL		
Event	**Example**	**OpenGL Callback Function Registered With**
Keypress	• General Keys • Speacial Keys	• glutKeyboardFunc • glutSpecialFunc
Mouse	Click of a mouse button	glutMouseFunc
Motion	• Motion With mouse press • Motion Without mouse press	• glutMotionFunc • glutPassiveMotionFunc
Window	• Moving, Resizing • Exposed	• glutReshapeFunc • glutDisplayFunc
System	• Idle • Timer	• glutIdleFunc • glutTimerFunc

GLUT Event Loop

Recall that the last line in **main( )** for a program using GLUT must be **glutMainLoop();** which puts the program in an infinite event loop.



In each pass through the event loop (Figure above), GLUT

- looks at the events in the queue
- for each event in the queue, GLUT executes the appropriate callback function if one is defined
- if no callback is defined for the event, the event is ignored

Window Event:

There are two types of event that occurs with window:

1. When a window is exposed to the user:(The display callback)

The display callback is executed whenever GLUT determines that the window should be refreshed, for example:

- When the window is first opened
- When a window is exposed
- When the user program decides it wants to change the display

In **main( ),**

glutDisplayFunc(mydisplay), identifies the callback function named **mydisplay** to be executed. Every GLUT program must have a **display callback**.

The prototype of display callback function is:

void mydisplay(void);

The display callback function is good place to put the code that generates most non-interactive output.

Example:

```
void display(void) {
        glClear( GL_COLOR_BUFFER_BIT );
        glClearColor(1,1,1,1);
        glColor3f(0,0,0);
        glBegin(GL_POLYGON);
        glVertex2f(x+size, y+size);
        glVertex2f(x-size, y+size);
         glVertex2f(x-size, y-size);
        glVertex2f(x+size, y-size);
        glEnd();
        glFlush();
        }
```

Posting redisplays

Many events may invoke the display callback function which can lead to multiple executions of the display callback on a single pass through the event loop. We can avoid this problem by instead using **glutPostRedisplay();** which sets a flag. GLUT checks to see if the flag is set at the end of the event loop If set then the display callback function is executed.

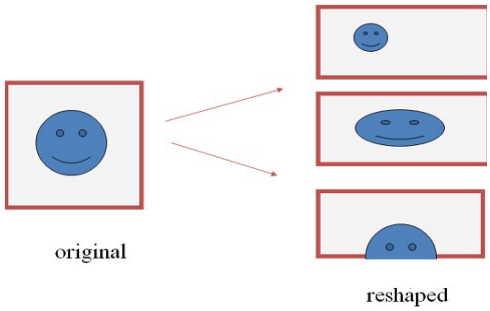
2. When window is resized or reshaped or moved:

The Reshape Callback

We can reshape and resize the OpenGL display window by pulling the corner of the window. When this event occurs the display may have two possibilities:

- Display part of world
- Display whole world but force to fit in new window

This is due to mismatch in the aspect ratio as show in the figure:



The reshape callback is good place to put viewing functions because it is invoked when the window is first opened. We can register the reshape callback using:

glutReshapeFunc(myreshape);

The prototype of the callback function is:

void myreshape(int w, int h)

- Returns width and height of new window (in pixels)
- A redisplay is posted automatically at end of execution of the callback
- GLUT has a default reshape callback but you probably want to define your own

Example Reshape

- This reshape preserves shapes by making the viewport and world window have the same aspect ratio

```
void myReshape(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION); /* switch matrix mode */
  glLoadIdentity();
  if (w <= h)
    gluOrtho2D(-2.0, 2.0, -2.0 * (float) h / (float) w,
      2.0 * (float) h / (float) w);
  else  gluOrtho2D(-2.0 * (float) w / (float) h, 2.0 *
      (float) w / (float) h, -2.0, 2.0);
  glMatrixMode(GL_MODELVIEW); /* return to modelview mode */
}
```

Mouse Event:

There are 3 types of events with mouse:

1. Click of mouse button

The information returned includes the button that generates event, the state of the button that generates the event (Up or Down), and the position of the cursor tracking the mouse in window coordinates (with origin in the upper-left corner of the window).

We register the mouse callback function, using

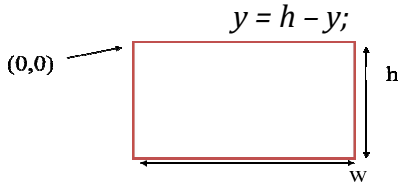
glutMouseFunc(mymouse);

The prototype of the callback function is:

void mymouse(int button, int state, int x, int y)

- Returns which button (**GLUT_LEFT_BUTTON**, **GLUT_MIDDLE_BUTTON**, **GLUT_RIGHT_BUTTON**) caused event
- state of that button (**GLUT_UP**, **GLUT_DOWN**)
- x, y:Position in window

OpenGL uses a world coordinate system with origin at the bottom left Must invert *y* coordinate returned by callback by height of window:



Example:

Terminating a program

- In our original programs, there was no way to terminate them through OpenGL
- We can use the simple mouse callback

```
void mouse(int btn, int state, int x, int y)
{
  If (btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        exit(0);
}
```

2. Click and move of mouse

Using the motion callback

- We can draw squares (or anything else) continuously as long as a mouse button is depressed by using the motion callback

  glutMotionFunc(drawSquare)
- **prototype of callback function is**

  void drawSquare(int x,int y)

3. Move of mouse without a click

Using the passive motion callback

- We can draw squares without depressing a button using the passive motion callback

  glutPassiveMotionFunc(drawSquare)
- **prototype of callback function is**

  void drawSquare(int x,int y)

Example for both Click and move of mouse & Move of mouse without a click:

- In this next example, we draw a small square at the location of the mouse moves
- This example does not use the display callback but one is required by GLUT; We can use the empty display callback function

mydisplay( ){ }

Drawing squares at cursor location

```
void mymouse(int btn, int state, int x, int y)
{
  if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        exit(0);
   if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        drawSquare(x, y);
}
void drawSquare(int x, int y)
{    y=w-y; /* invert y position */
   glColor3f( 1,0,0);/* red color */
   glBegin(GL_POLYGON);
      glVertex2f(x+size, y+size);
      glVertex2f(x-size, y+size);
      glVertex2f(x-size, y-size);
      glVertex2f(x+size, y-size);
    glEnd( );
glFlush( ); }
```

Keyboard event:

Keyboard events are generated when the mouse is in the window & on of the keys is pressed or released. When the keyboard event occurs, the ASCII code for the key that generated the event and the location of the mouse are returned. . There are 2 type of keys in the keyboard: **General** and **Special Keys.**

1. Keyboard Callbacks to handle General Keys:

We register the Keyboard Callback function to handle General Keys, using

void glutKeyboardFunc(myKey)

Prototype of the callback function is

void myKeys(unsigned char key, int x, int y)

-key holds the ASCII value of the key pressed

-(x, y) is always measured from top left corner of the window

Example : to terminate the program upon a press of key ‘q’ or ‘Q’

```
void mykey()
{
        if(key == ‘Q’ | key == ‘q’)
                exit(0);
}
```

2. Keyboard Callbacks to handle Special Keys:

We register the callback function that handles special keys (function,arrows, etc…), using

void glutSpecialFunc(myKeys)

Prototype of the callback function:

void myKeys(int key, int x, int y)

Example:

The callback registered by

glutSpecialFunc(myKeys) will have tests like:

```
void myKeys(int k, int x,int y){
if(key == GLUT_KEY_F1) .... // Function key F1
if(key == GLUT_KEY_F2) .... // Function key F2
}
```

The constants associated to the special keys are defined in glut.h

/*functional Keys*/	/* directional keys */
#define GLUT_KEY_F1 1 #define GLUT_KEY_F2 2 #define GLUT_KEY_F3 3 #define GLUT_KEY_F4 4 #define GLUT_KEY_F5 5 #define GLUT_KEY_F6 6 #define GLUT_KEY_F7 7 #define GLUT_KEY_F8 8 #define GLUT_KEY_F9 9 #define GLUT_KEY_F10 10 #define GLUT_KEY_F11 11 #define GLUT_KEY_F12 12	#define GLUT_KEY_LEFT 100 #define GLUT_KEY_UP 101 #define GLUT_KEY_RIGHT 102 #define GLUT_KEY_DOWN 103 #define GLUT_KEY_PAGE_UP 104 #define GLUT_KEY_PAGE_DOWN 105 #define GLUT_KEY_HOME 106 #define GLUT_KEY_END 107 #define GLUT_KEY_INSERT 108

We can also check one of the modifiers using:

int glutGetModifiers( )

Return one of **GLUT_ACTIVE_SHIFT, GLUT_ACTIVE_CTRL** or **GLUT_ACTIVE_ALT** if the key is pressed when a keyboard or mouse event are generated.

Example:

To have Control-c or Control-C terminate a program

```
void myKeys(unsigned char key, int x,int y)
{
if ((glutGetModifiers( ) == GLUT_ACTIVE_CTRL) &&((key == 'c') || (key == 'C'))) exit(0);
}
```

NOTE: The modifiers also work with mouse functions

No event: Idle Callback

- Sometimes it is convenient to designate a function that is to be executed when there are no other events for the system to process. We can do that with the idle callback function which is executed whenever there are no events in the event queue.
- Useful for animations
- The idle callback function is registered using:(in main())
  glutIdleFunc(myidle)
- Prototype of the callback function:

```
void myidle() {
/* change something */
t += dt
glutPostRedisplay();
}
void mydisplay() {
glClear();
/* draw something that depends on t */
glutSwapBuffers();
}
```

Note: This idle callback function 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.**

OpenGL Menu Functions

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

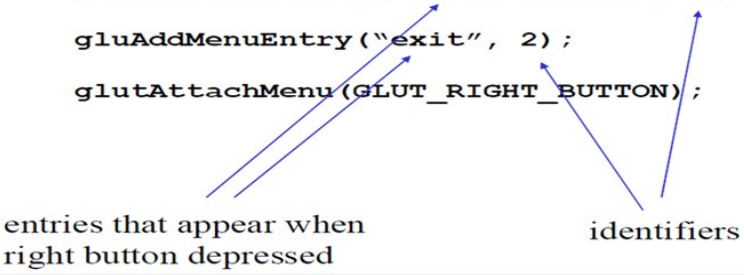
Creating a GLUT Menu

- A pop-up menu is created with the statement
  glutCreateMenu (menuFcn);
  - where parameter **menuFcn** is the name of a procedure(callback function) that is to be invoked when a menu entry is selected.
- Prototype of the callback function is:
  void menuFcn (int menuItemNumber)
  - This procedure has one argument, which is the integer value corresponding to the position of a selected option.
- Once the menu is created, we can add menu entries using:
  glutAddMenuEntry (charString, menuItemNumber);
- To attach the menu to a certain mouse button issue:
  glutAttachMenu(button);

Example:

- In main.c

```
glutCreateMenu(mymenu);
glutAddmenuEntry("clear Screen", 1);
gluAddMenuEntry("exit", 2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```



- Menu callback

```
void mymenu(int id)
{
        if(id == 1) glClear();
        if(id == 2) exit(0);
}
```

Creating and Managing Multiple GLUT Menus

- The function glutCreateMenu returns an identifier itself that can be used to reference the created menu:

  int menuID= glutCreateMenu(menuFcn);

- To activate a menu for the current display window, we use the statement:

  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 us pressed.

Creating GLUT Submenus

- A submenu can be associated with a menu just like a regular menu item.
- First, the submenu is created just like a regular menu and its identifier stored.
- Then, the submenu can be attached to another menu using:

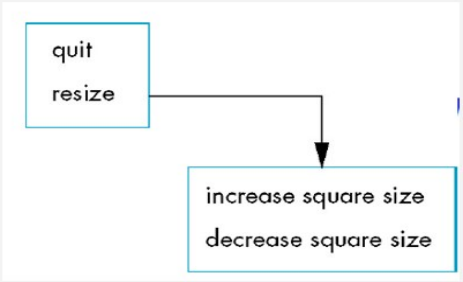
  glutAddSubMenu(char *submenu_name, submenu id)

Example:

```
in main()
int sub_menu=glutCreateMenu(size_menu);
glutAddMenuEntry("increase square size", 2);
glutAddMenuEntry("decrease square size", 3);

glutCreateMenu(top_menu);
glutAddMenuEntry("quit",1);
glutAddSubMenu("resize", sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);

Callback function:
void size_menu(int id)
{
if (id==2) ...
if (id==3)...
}
void top_menu(int id)
{
if (id==1) exit(0);
}
```



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.**

Considerations for a User Interface

1. 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.
- All information in the user dialogue is presented in the language of the application.
- In an architectural design package, this means that all interactions are described only in architectural terms, without reference to particular data structures, computer-graphics terms, or other concepts that may be unfamiliar to an architect.

2. Windows and Icons:

- 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.

3. 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. Therefore, interfaces typically provide a means for masking the complexity of a package, so that beginners can use the system without being overwhelmed with too much detail.
- **Experienced users**, on the other hand, 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.
- An interface may be designed to provide different sets of options to users with different experience levels.
- This may be selectable by the user through an application preference setting, or suggested by the application itself as the user gains experience with it.
- Similarly, 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.

4. 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.
- Some other examples of consistency are 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, and always using the same color encoding so that a color does not have different meanings in different situations.

5. 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.

6. 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 now 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.
- In addition, 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.

7. Feedback:

i. 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.

ii. 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.
   - The system might also allow a user to input other commands or data while one instruction is being processed.

iii. 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.

iv. 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.
- Feedback can also be displayed in different colors to distinguish it from other displayed objects.

v. Echo feedback is often useful, particularly for keyboard input, so that errors can be detected quickly.
- Button and dial input can be echoed in the same way.
- Scalar values that are selected with dials or from displayed scales are usually echoed on the screen so that a user can check the input values for accuracy.
- Selection of coordinate points can be echoed with a cursor or other symbol that appears at the selected position.
- For more precise echoing of selected positions, the coordinate values could also be displayed on the screen.

Computer Animation

- Animation is a series or sequences of images drawn or created on computer which gives an illusion of movement.
- Although we tend to think of animation as implying object motion, the term computer animation generally refers to any time sequence of visual changes in a picture.
- In addition to changing object positions using translations or rotations, a computer-generated animation could display time variations in object size, color, transparency, or surface texture.
- We can also generate computer animations by varying camera parameters, such as position, orientation, or focal length, and variations in lighting effects or other parameters and procedures associated with illumination and rendering can be used to produce computer animations.
- Two basic methods for constructing a motion sequence are **real-time animation** and **frame-by-frame animation**.

Real-Time Animation

- In a **real-time computer-animation**, each stage of the sequence is viewed as it is created. Thus the animation must be generated at a rate that is compatible with the constraints of the refresh rate.
- **Simple animation** displays are generally produced in **real time.**
- However, some applications require real-time animation, regardless of the complexity of the animation.
- **A flight-simulator animation, for example**, is produced in **real time** because the video displays must be generated in immediate response to changes in the control settings.

Frame-by-Frame Animation

- For a **frame-by-frame animation**, each frame of the motion is separately generated and stored. Later, the frames can be recorded on film, or they can be displayed consecutively on a video monitor in "real-time playback" mode.
- More **complex animations** are constructed more slowly, **frame by frame**.

Design of Animation Sequences

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 such 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:

- Storyboard is basically the layout of what action is going to happen in the animation (outline of a action).
- It defines the motion sequences 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, hence the name ***"storyboard."***

  Example:

  a. In our animation one apple falls from a tree and bounces to the ground.
  b. Then a bird flies in and swirls around in the air to see if any danger is there.
  c. Then it sits on the apple picks it up and flies away.

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.

Example:

For the above storyline, Objects defined are:
a. A Tree
b. A Bird
c. An Apple
d. Mud on the ground

3) Key frame specifications:

- A key frame is detailed drawing of the scene at a particular moment in the animation sequence.
- In each key frame the objects are placed at exact specific position according to specific time in the animation sequence.
- Key frames are spaced at short time intervals so better animation sequence is developed, as it provides more information about position of objects at specific times intervals is available.

Example:

For the above storyline, Key frames are:

a. A Tree with an Apple on the tree

b. A Tree with an Apple fallen down

c. A Bird entering the scene

d. A Bird swirls around to see if no one there

e. A scene with bird sitting on the apple to pick it

f. A scene with bird flying away by taking an apple with it.

4) Generation of in-between frames:

- In between frames are frames that are in between the key frames.
- Depending on the medium, the number of in-between frames differs.
- For example:
  - Film requires 24 frames per second
  - For Television or graphic monitors, the frame rate is 30 to 60 frames per second.
- Typically, time intervals for the motion are set up so that there are 3 to 5 in-between frames between 2 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.

Example:

For the above storyline, all Key frames are displayed with a time delay of 1 sec or less, for creating the animation.

Traditional Animation Techniques

- One of the most important techniques for simulating acceleration effects, particularly for non-rigid objects, is **squash and stretch.**
- **Figure shows** how this 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.

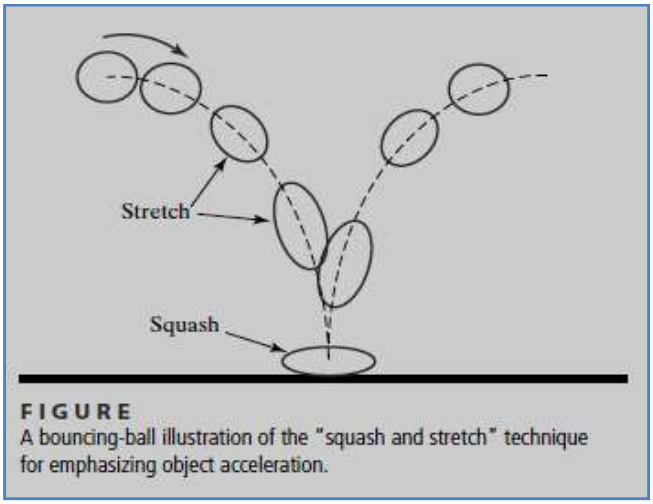


FIGURE
A bouncing-ball illustration of the "squash and stretch" technique for emphasizing object acceleration.

- Another technique used by film animators is **timing, which 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, where the position changes between frames increase as a bouncing ball moves faster.

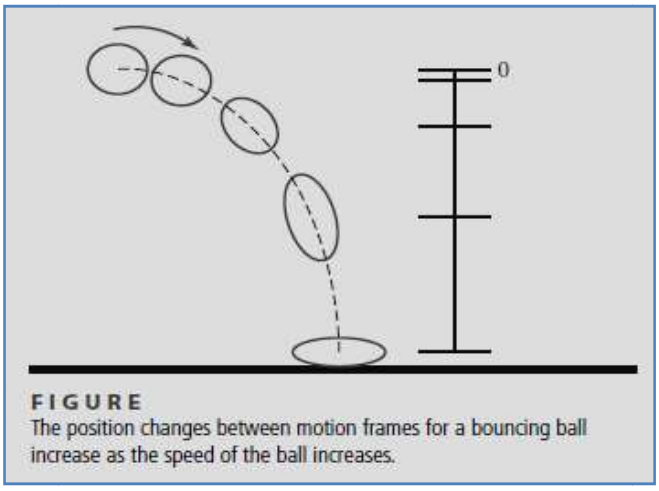


FIGURE
The position changes between motion frames for a bouncing ball increase as the speed of the ball increases.

Computer-Animation Languages

- We can develop routines 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.
- These languages typically include a **graphics editor**, a **key-frame generator**, an **in-between generator**, and **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.
- This 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**, which involves the layout of motion paths for the objects and camera.
- We need the usual graphics routines: 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.

Specialized Animation Languages:

1. **Key-frame systems**
   a) Originally designed as a separate set of animation routines for generating the in-betweens from the user-specified key frames.
   b) These routines are often a component in a more general animation package.
   c) 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.
   d) As an example, the single-armed robot in Figure-1 has 6 degrees of freedom, which are

referred to as arm sweep, shoulder swivel, elbow extension, pitch, yaw, and roll.

e) We can extend the number of degrees of freedom for this robot arm to 9 by allowing 3D translations for the base (Figure 2).
f) If we also allow base rotations, the robot arm can have a total of 12 degrees of freedom.
g) The human body, in comparison, has more than 200 degrees of freedom.

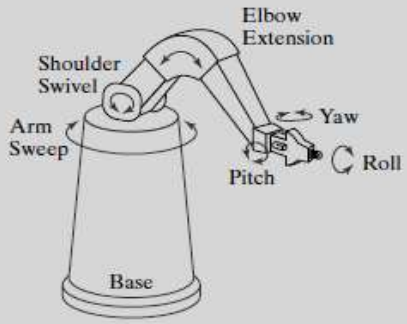


FIGURE 1
Degrees of freedom for a stationary, single-armed robot.

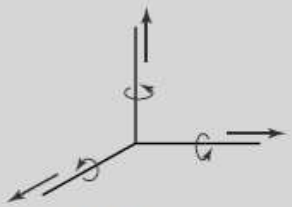


FIGURE 2
Translational and rotational degrees of freedom for the base of the robot arm.

2. Parameterized systems

- This system 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.

3. Scripting systems

- This system allows 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

- The animation of more complex figures such as humans or animals, however, it becomes much more difficult to create realistic animation.
- 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**, which are hierarchical structures composed of a set of rigid links that are connected at rotary joints **(Figure 17).**
- In less formal terms, this just means that we model animate objects 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.

  - 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.
  - Both kinematic-motion descriptions and inverse kinematics are used in figure animations.

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.

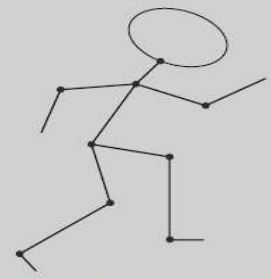


FIGURE 17
A simple articulated figure with nine joints and twelve connecting links, not counting the oval head.

- 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 3D. Typical motion capture systems will use up to two dozen cameras, but systems with several hundred cameras exist.
- Active systems can be constructed so that the markers illuminate in a pattern or sequence, which allows each marker to be uniquely identified in each frame of the recording, simplifying the tracking process.

Periodic Motions

- When we construct an animation with repeated motion patterns, such as a rotating object, we need to be sure that the motion is sampled frequently enough to represent the movements correctly.
- In other words, the motion must be synchronized with the frame-generation rate so that we display enough frames per cycle to show the true motion. Otherwise, the animation may be displayed incorrectly.
- In a computer-generated animation, we can control the sampling rate in a periodic motion by adjusting the motion parameters.
- When complex objects are to be animated, we also must take into account the effect that the frame construction time might have on the refresh rate.
- 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.

OpenGL Animation Procedures

- One method for producing a real-time animation with a raster system is to employ **two refresh buffers.**
- Initially,
  1. We create a frame for the animation in one of the buffers.
  2. Then, while the screen is being refreshed from that buffer, we construct the next frame in the other buffer.
  3. 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****,* 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);

-Returns **GL_TRUE 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.**

Questions on Unit-3

1. Explain the 6 logical classifications of input devices.7M
2. Explain the 7 interactive Picture Construction Techniques.10M
3. Explain any three programming event driven input with suitable examples. Explain the various input modes with neat diagram.10M
4. Discuss Logical Device and Hierarchical Menus.10M
5. Explain Request, Sample and Event mode with suitable diagram. 6M
6. Define double buffering. Explain how double buffering is implemented in OpenGL.(06 Marks)
7. Explain Menu creation in OpenGL. Write an interactive OpenGL program to display a square when the left button is pressed and to exit the program if right button is pressed.(08 Marks)
8. List and explain the various classes of logical input devices that are supported by OpenGL. With suitable diagrams, explain various input modes.(10 Marks)
9. Explain how keyboard events are recognized by GLUT. Give suitable example.(10)
10. Explain how window events are recognized by GLUT. Give suitable example.(10)
11. Explain how mouse events are recognized by GLUT. Give suitable example.(10)
12. How pop-up menus are created using GLUT? Illustrate with an example.(10 Marks)
13. What are the features of a good interactive program? Explain.(10 Marks)
14. Illustrate with an example the steps in construction of Animation Sequences (8M)
15. Explain the key factors to be considered when designing a user interface to ensure optimal user experience and accessibility? (10M)
16. What is computer animation? Explain the various stages in the development of animation sequences. (10M)
17. What is meant by measure and trigger of a device? Explain with the neat diagram, the various input mode models.(10M)
18. Explain the following: (i) Request Mode (ii)Sample Mode (iii) Event Mode (10M)
19. Differentiate event mode with request mode.(04M)
20. Write an OpenGL program to display square when a left button is pressed and to exit the program if right button is pressed. (10M)
21. Using OpenGL functions, explain the structure of hierarchical menus. (06M)
22. List out the characteristics of good interactive program. Explain in detail. (10M)
23. What is double buffering? How OpenGL implements double buffering? (5M)

Note: Answer to Q.15, Q.13, Q.22 is same

Write an OpenGL program to display square when a left button is pressed and to exit the program if right button is pressed.

```
#include<stdlib.h>
#include< glut.h>
 int wh=500, ww=500;
float siz=3;
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glViewPort(0,0,w,h)
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,(GLdouble) ww, 0, (GLdouble) wh);
glMatrixMode(GL_MODELVIEW);
glColor3f(1,0,0);
}
void drawsq (int x, int y)
{
y=wh-y;
glBegin(GL_POLYGON);
glVertex2f(x+siz, y+siz);
glVertex2f(x-siz, y+siz);
glVertex2f(x-siz, y-siz);
glVertex2f(x+siz, y-siz);
glEnd();
glFlush();
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
drawsq(x,y);
}
void myMouse(int button, int state, int x, int y)
{
if(button==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
display();
if(button==GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
exit(0);
}
void main( )
{  glutCreateWindow("square");
glutDisplayFunc(display);
glutMouseFunc(myMouse);
myinit();
glutMainLoop();
 }
```