UNIT 3: INPUT AND INTERACTION

Syllabus.

* Interaction
* Input Devices
* clients and servers
* Display lists
* Display lists and modelling
* programming event-driven input
* Menus
* picking
* A simple CAD program
* Building interactive models
* Animating interactive programs
* Design of interactive programs
* Logic operations.

- 7 Hours.

INTERACTION

* Definition:
Interaction in the field of computer graphics refers to the process of enabling the users to interact with computer displays.
The image change in response to the input from the user.

How it is supported by open GL?

* Open GL does not support interaction directly. The main reason for this is to make open GL portable (ie to work on all types of systems irrespective of the hardware)
* However open GL provides the GLUT tool kit which supports minimum functionality such as opening of windows, use of keyboards and mouse and creation of pop up menus etc

[We discuss several interacting devices and the variety of ways that we can interact with them]

INPUT DEVICES

* We can think about input devices in two distinct ways
- ~~log~~ physical devices
- logical devices

(or) in computer graphics, ~~we can~~ the input devices (such as mouse, K/b etc) would be assesed from two perspectives.
- physical perspective: depending on their physical properties
- logical perspective: the way these devices appear to the application program.

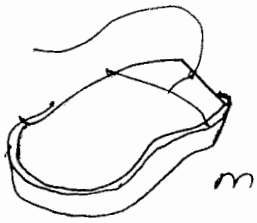
Explanations.

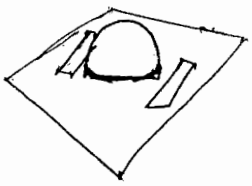
physical input Devices.

* From the physical perspective, each input device has properties that make it more suitable for certain tasks than for others.
* The two primary input devices are:
(or) two primary types of physical devices are

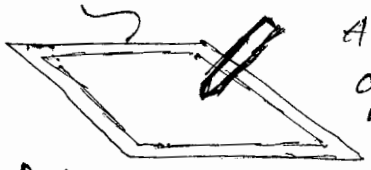
- pointing device.
  - Allows user to indicate a position on a display (ie allows user to return position)
- Keyboard device
  - Allows user to return ASCII codes.

* Mouse and trackball are two commonly used pointing devices.
Both are similar in use and in construction.
In both the devices, the motion of the ball is converted into signals and sent to the computer.
o/p of both is two independent values provided by the device, which are considered as positions and converted to a 2D locn in either screen / world coordinates.
In this mode, these devices are relative positioning devices because changes in the position of the ball yields a position in the user program; Absolute location of the ball (or mouse) is not used by the application program.


mouse

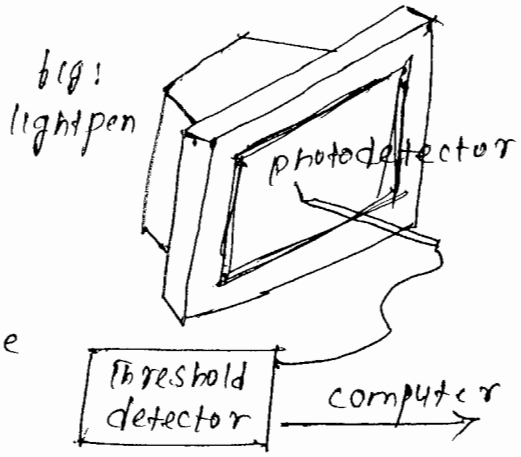

Track ball.

*

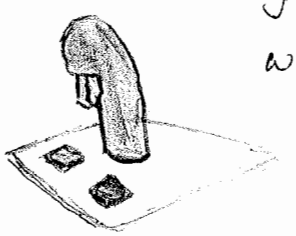

Data tablet

Data tablets provide for absolute positioning.
A typical data tablet has rows & columns of wires embedded under its surface.
The position of the stylus (pen) is determined through electromagnetic interactions b/w signals travelling through the wires and sensors in the stylus

* Light pen is one of the oldest input device in computer graphics.
It contains a light sensing device,
If the light pen is positioned on the face of the CRT at a location opposite where electron beam strikes the phosphor, the light emitted exceeds a threshold in the photodetector and a signal is sent to the computer.



*


Joystick

Joystick is an input device in which the stick would move in orthogonally two directions
- If the stick is left in the resting position, there is no change in the cursor position.
- The farther the stick is moved from the resting position, the faster the screen location changes.

Advantage of a joystick is that it is designed using mechanical elements such as springs and dampers which offer resistance to the user while pushing it. Such a mechanical feel is suitable for applications such as the flight simulators, game controllers etc.

* For 3D ~~applications~~ graphics, we might prefer to use 3D input devices (eg: spaceball, laser cameras)


space ball.

spaceball looks like joystick with the ball on the end of stick.
But, stick does not move, rather, pressure sensors in the ball measures the forces applied by the user.
It can measure not only 3 direct forces (up-down, front-back, right-left) but also three independent twists. ie the device measures 6 independent values & thus has six degrees of freedom.

Logical Devices.

+ From logical perspective, the input device is assesed by looking at it from inside the application program depending upon the "measurements" that the device returns to the user program and the "time" when the device returns those measurements.
+ From logical perspective, 6 classes of input forms can be identified such as.

- string : A string device is a logical device that provides ASCII strings to the user program. This logical device is usually implemented by means of a physical keyboard.
- Locator : A locator device provides position in world coordinates to the user program. It is usually implemented by means of pointing device (mouse or track ball)
- pick : A pick device returns an identifier of the object on the display to the user program. It is implemented with the same physical device as a locator, but has a separate s/w interface to the user program.
- choice : choice device allows the user to select one out of a discrete number of options. A various widgets (a graphical interactive device, provided either by window system or toolkit, eg: menus, scroll bars, & graphical buttons) can be used to select one of n alternatives. It can be implemented either using k/b or mouse.
- Valuators : valuator devices allow the user to provide analog i/p to the user prog. Dials and slide bars can be used for valuator inputs.
- stroke : A stroke device returns an array of locations. (ill° to ~~rotp~~ multiple use 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.

Input modes.

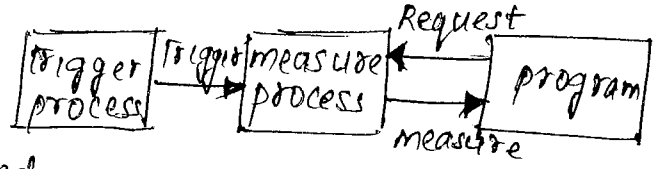
* There are 3 different modes in which an input device provides input to the application program
  - Request mode
  - sample mode
  - Event mode.

- note: the manner by which input devices provide i/p to an application program can be described in terms of two entities: a measure process & 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 ~~which~~ with which the user can signal the computer.

for eg: in K/b: measure - single char./strings of char.
trigger - Return or Enter key
in mouse: measure - position of cursor
trigger - press of a button.

Request mode.

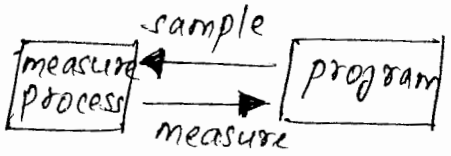
* In this mode, measure of the device is not returned to the program until the device is triggered.



eg: If a C program requires a string input, a scanf function is used. When the program encounters scanf function (statement) it waits while we type the characters at our terminal (or K/b). All the data that is entered is stored in the keyboard buffer and its contents are given to the program only after the enter key is pressed.
↳ the trigger.

* The relation b/w measure and trigger for request mode is as shown in above figure.

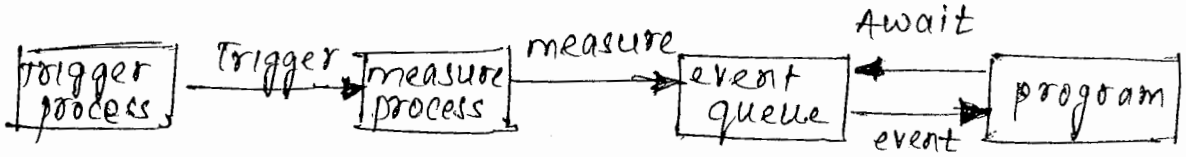
Sample Mode



* 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 in this mode (refer figure)
* In this mode the user must have positioned the pointing device or entered data using the kb before the function call, because the measure is extracted immediately from the buffer.

Event mode

* The request and sample mode can be used iff there is a single input device from which input is to be taken. They cannot be used if there are multiple input devices.
* Event mode must be used if there are inputs from a variety of input devices such as joystick, dials, buttons, switches etc



Here two approaches

i.) First Approach

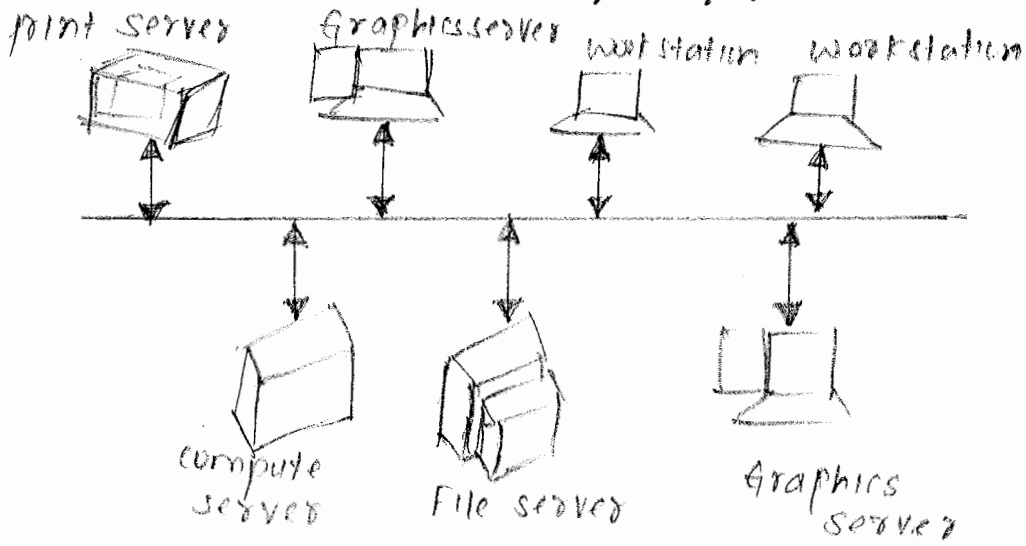
- Each time the device is triggered, an event is generated the device measure, including the identifier for the device, is placed in an event queue.
- The applicn program can examine the front ~~eq~~ event in the queue (or wait if queue empty) and then decide what ~~to~~ do. (it can discard the front queue & look for next event).

ii.) second approach.

- A specific callback function is associated with each type of input (event). The OS would regularly poll the event queue and executes the callbacks corresponding to the events in the .....

CLIENTS AND SERVERS

* The openGL application programs are treated as clients, and
the workstation (computer) with a display, k/b & pointing device is treated as a graphics server.
Even if we have a single user isolated system, the interaction would be configured as a simple client-server network (refer fig)

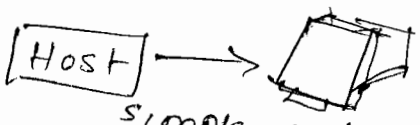


* This is to enable computer graphics to be useful under a variety of real applications.
Today most of the computing is distributed and network based. The building blocks are entities called "servers" that performs tasks for the "clients".
Clients and servers can be distributed over a n/w or can be contained entirely within a single computational unit.

DISPLAY LISTS

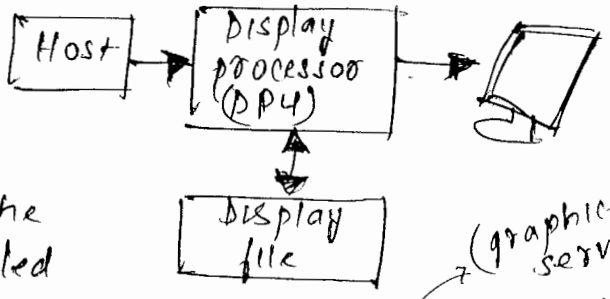
Display processor Architecture.

* original architecture of a graphics system was based on a gen. purpose comp (host) connected to display
* Disadv: comp were slow & expensive.



simple graphics architecture.

* Solution is to build a special purpose ~~process~~ comp called display processor with an org like the one shown below.
It had limited instruction set (most are oriented towards drawing primitives on the display).



* User prog was processed in the host comp, resulting in compiled list of instructions that was sent to display processor, where the instructions were stored in a display mem as a display file or display list.
* We can send the graphical entities to a display in one of two ways - Immediate mode
- Retained mode.

Immediate mode
- In this operation, as soon as the program executes a statement that defines a primitive, the primitive is sent to the graphics server for possible display and no memory of it is retained in the host.
- Disadv
To redisplay the primitive after clearing screen, or to display it in a new position on the screen, the host program must resend the information through the display process. This would cause considerable traffic blw client & server.

Retained mode
- In this mode, we define the object once and place its description (vertices, attributes, primitive types, viewing information etc.) in a display list.
- The display list is stored in the server and redisplayed by a simple function call issued from the client to the serv

Definition:
Display lists are used to store the description of the objects which are to be displayed. The description would

Definition and execution of display lists.

* ~~Def~~ Display lists are defined in the same way as any geometric primitive is defined. There is a glNewList at the beginning and a glEndList at the end, with the contents in b/w.
* Each display list must have a unique identifier - an integer that is usually macro defined in the C program by means of a #define directive to an appropriate name for the object in the list.
* For ex: following code defines an red box & stores it in display list

```
#define BOX 1
glNewList (BOX, GL_COMPILE);
  glBegin (GL_POLYGON);
      glColor3f (1.0, 0.0, 0.0);
      glVertex2f (-1.0, -1.0);
      glVertex2f (1.0, -1.0);
      glVertex2f (1.0, 1.0);
      glVertex2f (-1.0, 1.0);
  glEnd ();
glEndList ();
```

⇒ tells the system to send the list to the server but not to display its contents

* Each time we wish to draw the box, the client must execute a function

```
glCallList (BOX);
```

* If we change the model-view or projection matrices b/w executions of the display list, the box will appear in different places or will no longer appear, as the following code fragment demonstrates

```
glMatrixMode (GL_PROJECTION)
for(i=1; i<5; i++)
{ glLoadIdentity ();
  gluOrtho2D (-2.0*i, 2.0*i, -2.0*i, 2.0*i);
  glCallList (BOX);
}
```

* Each time the glCallList(Box) is executed, the box is redrawn with a larger clipping rectangle which would not have been possible in immediate mode graphics.

* However, each time the display list is executed, the drawing color is set to red.
Unless the color is set to some other values, primitives defined subsequently in the program also will be colored red.

* A standard and safe procedure to overcome the above problem is to always push both the attributes and matrices into their respective stacks when we enter a display list, and to pop them when we exit() as shown:

```
glPushAttrib(GL_ALL_ATTRIB_BITS);
glPushMatrix();

~~glCallList(Box);~~ at the beginning of a display list
                                                    &
glPopAttrib();
glPopMatrix();
                    at the end.
```

Adv.
- Reduced n/w traffic
- Allows much of the overhead in executing commands to be done once and have the results stored in the display list on the graphics server.

Disadv.
- Display lists require m/m on the server.
- There is an overhead involved in creating a display list.

PROGRAMMING EVENT DRIVEN INPUT

using the pointing devices

(How an event driven input can be programmed for a pointing device ?)

* Mouse, Trackball, Data tablet etc can be categorized as pointing device.
* There are two types of events associated with the pointing device
  - move event
  - mouse event
* A move event is generated when the mouse is moved with one of the buttons pressed.
  If the mouse is moved without a button being held down, this event is called a passive move event.
  After a move event, the position of the mouse (measure) is made available to the application program.
* A mouse event occurs when one of the mouse buttons is either pressed or released. A button being held down does not generate a mouse event until the button is released.
  The information returned to the program includes
  - The button that generated the event
  - The state of the button after the event (up or down)
  - position of the cursor in window coordinates (with origin in the upper left corner of the window)
* The mouse call back function must be registered in the main() function as shown below

```
glutMouseFunc ( myMouse );
```

* The mouse call back must have the form

```
void myMouse (int button, int state, int x, int y );
```

* The above function must be written by the application programmer acc. to his requirements.
Within the callback function, the programmer can define the actions that must take place if specified event occurs.

eg: If we want the pressing of the left mouse button to terminate the program, then the myMouse call back function must be designed as follows.

```
void myMouse (int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON &&
        state == GLUT_DOWN )
    {
        exit(0);
    }
}
```

{pressing of other buttons results in no response, since no action is defined for them}

Eg 2: program to draw a small Box at each location on the screen where the mouse cursor is located at the time that the left button is pressed. use the middle button to terminate the program.

```
GLsizei wh=500, ww=500; /* initial window width & height */
GLfloat size=3.0; /* one half of sidelength */

void myInit()
{
    glViewport(0,0, ww, hh);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)ww, 0.0(GLdouble) wh);
    glMatrixMode(GL_MODELVIEW);
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 0.0, 0.0); /* red squares */
}
```

```
void drawSquare (int x, int y)
{   y= wh-y;
    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();
}
void myDisplay()
{ glClear (GL_COLOR_BUFFER_BIT);
}
void myMouse (int ~~button~~ btn, int state, int x, int y)
{   if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        drawsquare(x, y);
    if (btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        exit(0);
}
int(main int argc, char **argv)
{   glutInit (&argc, argv);
    glutInitWindowSize (ww, wh);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow ("square");
    myInit();
    glutReshapeFunc(myReshape);
    glutMouseFunc (myMouse);
    glutDisplayFunc (myDisplay);
    glutMainLoop();
}
```

Window Events

(How an event driven input can be programmed for an window event?)

* Resizing a window is one of the example for window events.
  Resizing a window is done usually by using a mouse to drag a corner of the window to a new location.

* If such an event occurs, we have to consider three questions
  - Do we ~~draw~~ redraw all the objects that were in the window before it was resized?
  - What do we do if the aspect ratio of new window is different from that of the old window?
  - Do we change the sizes ~~of~~ or attributes of new primitives if the size of the new window is different from that of old?

* The window event must be registered in the main function using glutReshapeFunc (myReshape);

The window event (reshape event) returns its measure, the height, & width of the new window. So myReshape() must be of the form

```
~~void~~ myReshape ( GLsizei w, GLsizei h );
```

The above program must be written by the applicn programmer acc. to his requirements.

eg:
```
void myReshape (GLsizei w, GLsizei h)
{ glMatrixMode (GL_PROJECTION)                              } adjust
  glLoadIdentity();                                         } clippin
  gluOrtho2D ( 0.0, (GLdouble)w, 0.0, (GLdouble) h );       } box
  glMatrixMode (GL_MODELVIEW);
  glLoadIdentity();
  glViewport (0, 0, w, h );  --> adjust viewport & clear
```

WW = W; I ... new window size in

Keyboard Events

(How event driven input can be programmed for a keyboard device?)

* Keyboard is an input device.
  Keyboard events are generated when the mouse (cursor) is in the window and one of the keys is pressed or released
* The keyboard event must be registered in the main function using -

```
glutKeyboardFunc (myKey);
   (or)
glutKeyboardUpFunc (myKey);
```

The above given callback functions are for events generated by pressing a key and for releasing a key respectively.

* When the klb event occurs, the ASCII code for the key that generated the event and the location of the mouse are returned. So the myKey() must be of the following form

```
myKey (unsigned char key, int x, int y);
```

The above program must be written by the application programmer acc to his requirements.

eg: If we wish to use the klb to only exit from the program, then it can be done as shown.

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