COMUTER GRAPHICS AND VISUALIZATION

Module-5

5.1.1 INTERACTION

In the field of computer graphics, interaction refers to the manner in which the application program communicates with input and output devices of the system.

Ex: Image varying in response to the input from the user.

OpenGL doesn't directly support interaction in order to maintain portability. However, OpenGL provides the GLUT library. This library supports interaction with the keyboard, mouse etc and hence enables interaction. The GLUT library is compatible with many operating systems such as X windows, Current Windows, Mac OS etc and hence indirectly ensures the portability of OpenGL.

5.1.2 INPUT DEVICES

Input devices are the devices which provide input to the computer graphics application program. Input devices can be categorized in two ways:

- *Physical input devices*
- *Logical input devices*

PHYSICAL INPUT DEVICES

Physical input devices are the input devices which has the particular hardware architecture. The two major categories in physical input devices are:

***Keyboard devices** like standard keyboard, flexible keyboard, handheld keyboard etc. These are used to provide character input like letters.*

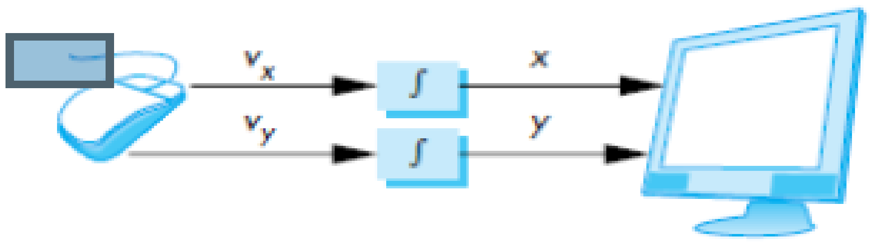
KEYBOARD

It is a general keyboard which has set of characters. We make use of ASCII value to represent the character i.e. it interacts with the programmer by passing the ASCII value of key pressed by programmer. Input can be given either single character of array of characters to the program.

***MOUSE AND TRACKBALL:** These are pointing devices used to specify the position. Mouse and trackball interact with the application program by passing the position of the clicked button. Both these devices are similar in use and construction.*

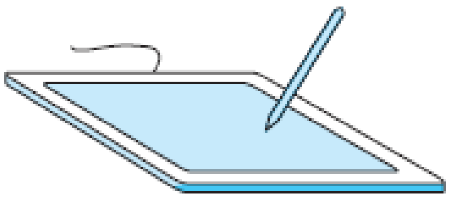
In these devices, the motion of the ball is converted to signal sent back to the computer by pair of encoders inside the device. These encoders measure motion in 2-orthogonal directions.

The values passed by the pointing devices can be considered as positions and converted to a 2-D location in either screen or world co-ordinates. Thus, as a mouse moves across a surface, the integrals of the velocities yield x,y values that can be converted to indicate the position for a cursor on the screen as shown below: These devices are relative positioning devices because changes in the position of the ball yield a position in the user program.



Data Tablets

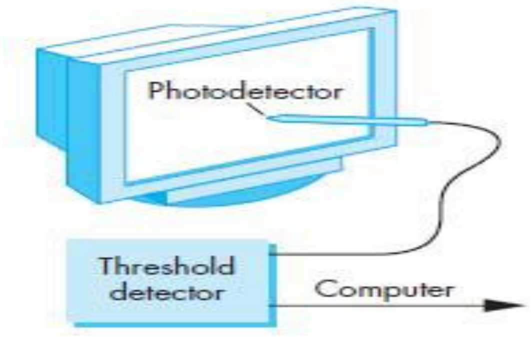
- *It provides absolute positioning.*
- *It has rows and columns of wires embedded under its surface.*
- *The position of the stylus is determined through electromagnetic interactions between signals travelling through the wires and sensors in the stylus.*



•

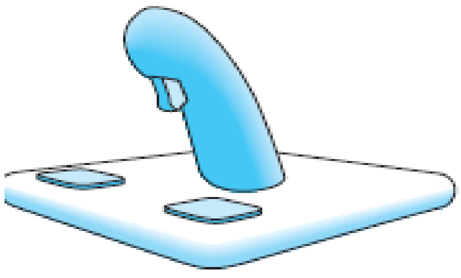
Light Pen

- *It consists of light-sensing device such as "photocell".*
- *The light pen is held at the front of the CRT.*
- *When the electron beam strikes the phosphor, the light is emitted from the CRT. If it exceeds the threshold, then light sensing device of the light pen sends a signal to the computer specifying the position*



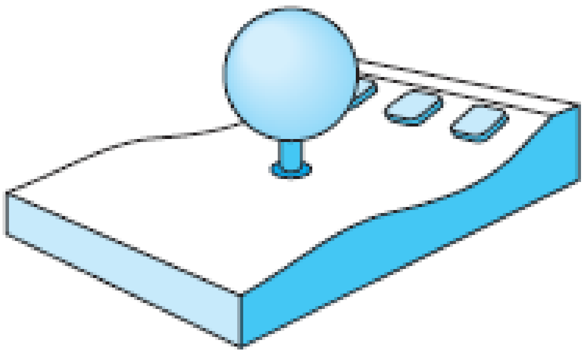
Joystick:

- *The motion of the stick in two orthogonal directions is encoded, interpreted as two velocities and integrated to identify a screen location.*
- *The integration implies that if the stick is left in its resting position, there is no change in cursor position.*
- *The faster the stick is moved from the resting position the faster the screen location changes.*
- *Joystick is variable sensitivity device.*
- *The advantage is that it is designed using mechanical elements such as springs and dampers which resist the user while pushing it.*
- *The mechanical feel is suitable for application such as the flight simulators, game controllers etc.*



Space Ball:

- *It is a 3-Dimensional input device which looks like a joystick with a ball on the end of the stick*
- *Stick doesn't move rather pressure sensors in the ball measure the forces applied by the user.*
- *The space ball can measure not only three direct forces (up-down, front-back, left-right) but also three independent twists.*
- *So totally device measures six independent values and thus has six degree of freedom.*



LOGICAL INPUT DEVICES

These are characterized by its high-level interface with the application program rather than by its physical characteristics. 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.

Consider the following fragment of C code:

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

The above code reads and then writes an integer. Although we run this program on workstation providing input from keyboard and seeing output on the display screen, the use of scanf() and printf() requires no knowledge of the properties of physical devices such as keyboard codes or resolution of the display.

These are logical functions that are defined by how they handle input or output character strings from the perspective of C program. From logical devices perspective inputs are from inside the application program.

Six classes of logical input devices

String: *A string device is a logical device that provides the ASCII values of input characters to the user program. This logical device is usually implemented by means of physical keyboard.*

glutKeyboardFunc()

Locator: *A locator device provides a position in world coordinates to the user program. It is usually implemented by means of pointing devices such as mouse or track ball.*

glutMouseFunc()

Pick: *A pick device returns the identifier of an object on the display to the user program. It is usually implemented with the same physical device as the locator but has a separate software interface to the user program. In OpenGL, we can use a process of selection to accomplish picking.*

bounding box technique

Choice: *A choice device allows the user to select one of a discrete number of options. In OpenGL, we can use various widgets provided by the window system. A widget is a graphical interactive component provided by the window system or a toolkit. The Widgets include menus, scrollbars and graphical buttons.*

A menu with n selections acts as a choice device, allowing user to select one of "n alternatives.

Valuators: *They provide analog input to the user program on some graphical systems, there are boxes or dials to provide value.*

Analog Input, Control Dials, Sensing Devices

Stroke: *A stroke device returns array of locations.*

pushing down a mouse button starts the transfer of data into specified array and releasing of button ends this transfer.

Input Modes

1.List and discuss three input modes with diagram.

How input devices provide input to an application program can be described in terms of two entities:

- ***Measure*** *of a device is what the device returns to the application program.*
- ***Trigger*** *of a device is a physical input on the device with which the user can send signal to the computer*

Example 1: *The measure of a keyboard is a single character or array of characters where as the trigger is the enter key.*

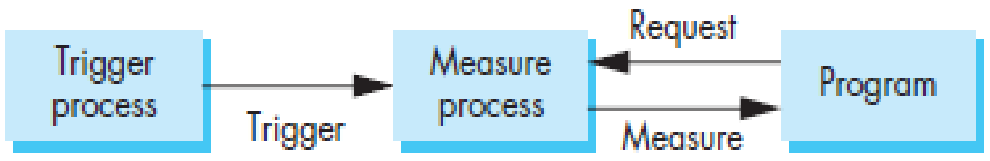
Example 2: *The measure of a mouse is the position of the cursor whereas the trigger is when the mouse button is pressed.*

The application program can obtain the measure and trigger in ***three distinct modes****:*

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

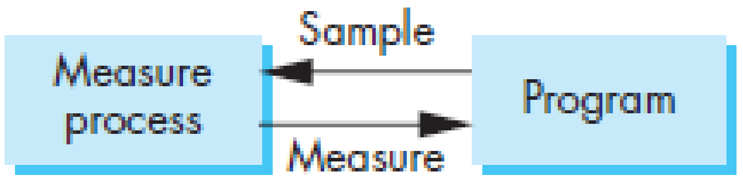
- *For example, consider a typical C program which reads a character input using scanf(). When the program needs the input, it halts when it encounters the scanf() statement and waits while user type characters at the terminal.*

- *The data is placed in a keyboard buffer (measure) whose contents are returned to the program only after enter key (trigger) is pressed.*
- *Another example, consider a logical device such as locator, we can move out pointing device to the desired location and then trigger the device with its button, the trigger will cause the location to be returned to the application program.*



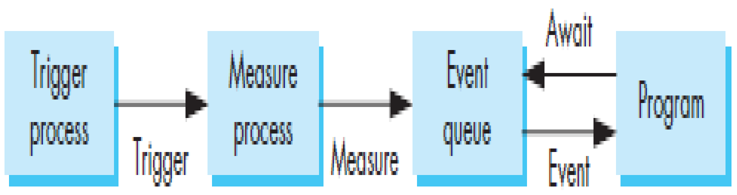
2. *SAMPLE MODE*

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



3. *EVENT MODE*

- *This mode can handle the multiple interactions.*
- *Suppose that we are in an environment with multiple input devices, each with its own trigger and each running a measure process.*
- *Whenever a device is triggered, an event is generated.*
- *The device measure including the identifier for the device is placed in an event queue.*
- *If the queue is empty, then the application program will wait until an event occurs.*
- *If there is an event in a queue, the program can look at the first event type and then decide what to do.*



5.1.3 CLIENT AND SERVER

The computer graphics architecture is based on the client-server model.
I.e., if computer graphics is to be useful for variety of real applications, it must function well in a world of distributed computing and network. In this architecture the building blocks are entities called as ***"servers"*** perform the tasks requested by the ***"client"*** **.**Servers and clients can be distributed over a network or can be present within a single system. Today most of the computing is done in the form of distributed based and network based as shown below:

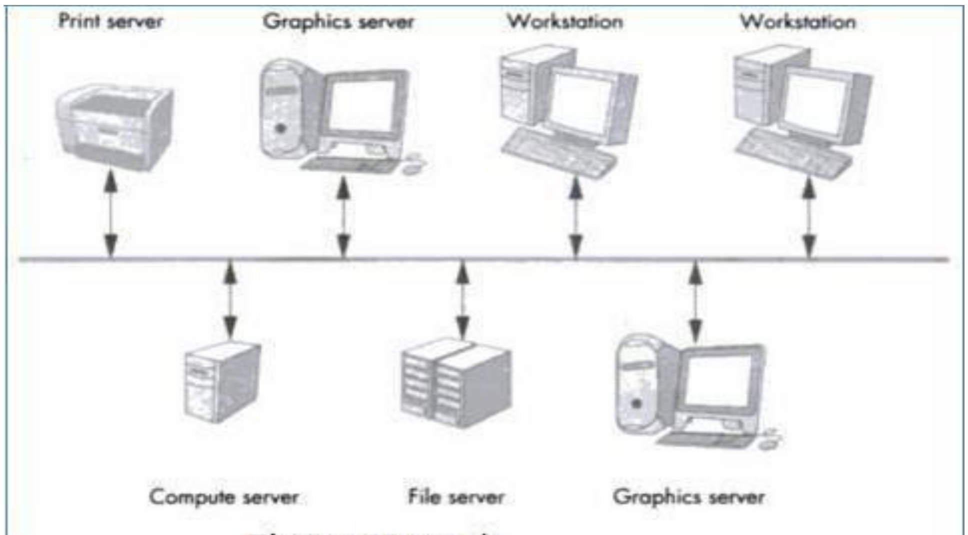


Figure: Network

Most popular examples of servers are print servers – which allow using high speed printer devices among multiple users. File servers – allow users to share files and programs. Users or clients will make use of these services with the help of user programs or client programs.

The OpenGL application programs are the client programs that use the services provided by the graphics server. Even if we have single user isolated system, the interaction would be configured as client-server model.

5.1.4 DISPLAY LISTS

The original architecture of a graphical system was based on a general-purpose computer connected to a display. The simple architecture is shown below.

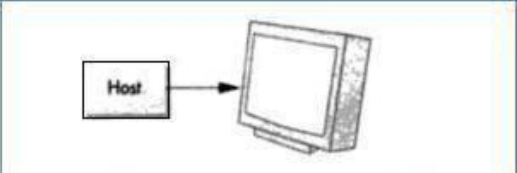
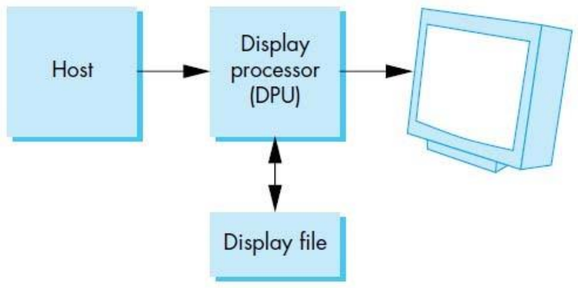


Figure: Simple graphics architecture

*At that time, the disadvantage is that system was slow and expensive. Therefore, a special purpose computer is build which is known as **"display processor"**.*
*The user program is processed by the host computer which results a compiled list of instruction that was then sent to the display processor, where the instruction are stored in a display memory called as **"display file"*** or ***"display list"***. Display processor executes its display list contents repeatedly at a sufficient high rate to produce flicker-free image.



There are two modes in which objects can be drawn on the screen:

✓***IMMEDIATE MODE:*** *This mode sends the complete description of the object which needs to be drawn to the graphics server and no data can be retained. i.e., to redisplay the same object, the program must re-send the information. The information includes vertices, attributes, primitive types, viewing details.*

✓***RETAINEDMODE:*** *This mode is offered by the display lists. The object is defined once and its description is stored in a display list which is at the server side and redisplay of the object can be done by a simple function call issued by the client to the server.*

NOTE: *The main disadvantage of using display list is it requires memory at the server architecture and server efficiency decreases if the data is changing regularly.*

DEFINITION AND EXECUTION OF DISPLAY LISTS

How to create and execute display list,explain with example.

Display lists are defined similarly to the geometric primitives. i.e., glNewList() at the beginning and glEndList() at the end is used to define a display list.

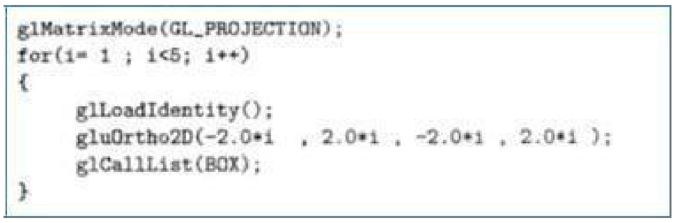
Each display list must have a unique identifier – an integer that is usually a macro defined in the C++ program by means of #define directive to an appropriate name for the object in the list.

The flag GL_COMPILE indicates the system to send the list to the server but not to display its contents. If we want an immediate display of the contents while the list is being constructed then GL_COMPILE_AND_EXECUTE flag is set.

Each time if the client wishes to redraw the box on the display, it need not resend the entire description. Rather, it can call the following function:

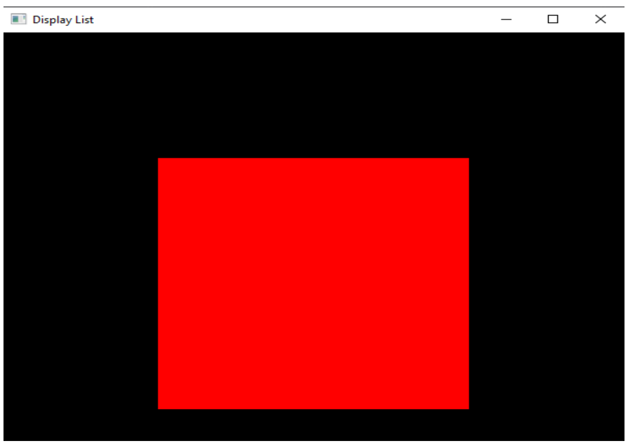
glCallList(Box)

The Box can be made to appear at different places on the monitor by changing the projection matrix as shown below:



```
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);
}
```

```
#include<stdio.h>
#include<gl/glut.h>
#define sq 10
void myinit()
{
            glMatrixMode(GL_PROJECTION_MATRIX);
            glLoadIdentity();
            gluOrtho2D(-100, 100, -100, 100);
            glMatrixMode(GL_MODELVIEW);
}
void display()
{
            glClearColor(0, 0, 0, 1);
            glClear(GL_COLOR_BUFFER_BIT);
            glColor3f(1, 0, 0);
            glNewList(sq, GL_COMPILE_AND_EXECUTE);
            glBegin(GL_POLYGON);
            glVertex2f(-50, -50);
            glVertex2f(-50, 50);
            glVertex2f(50, 50);
            glVertex2f(50, -50);
            glEnd();
            glEndList();
            glCallList(sq);
            glFlush();
}
void main()
{
            glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
            glutInitWindowSize(600, 600);
            glutInitWindowPosition(300, 150);
            glutCreateWindow("Display List");
            myinit();
            glutDisplayFunc(display);
            glutMainLoop();
}
```



We can save the present values of the attributes and the matrices by pushing them into the stack, usually the below function calls are placed at the beginning of the display list,

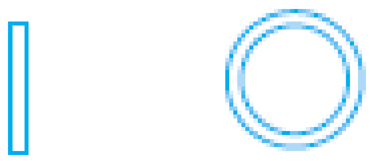
```
glPushAttrib(GL_ALL_ATTRIB_BITS);

glPushMatrix();
```

We can retrieve these values by popping them from the stack, usually the below function calls are placed at the end of the display list,

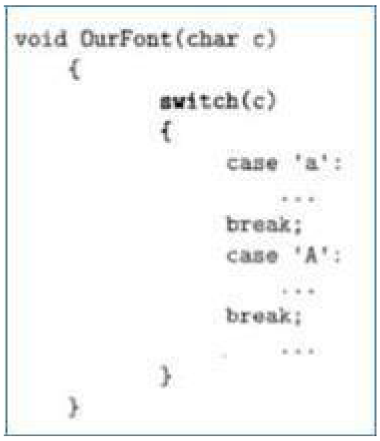
```
glPopAttrib();

glPopMatrix();
```

TEXT AND DISPLAY LISTS

*There are two types of text i.e., **raster text and stroke text** which can be generated. For example, let us consider a raster text character is to be drawn of size 8x13 pattern of bits. It takes 13 bytes to store each character. If we define a stroke font using only line segments, each character requires a different number of lines.*



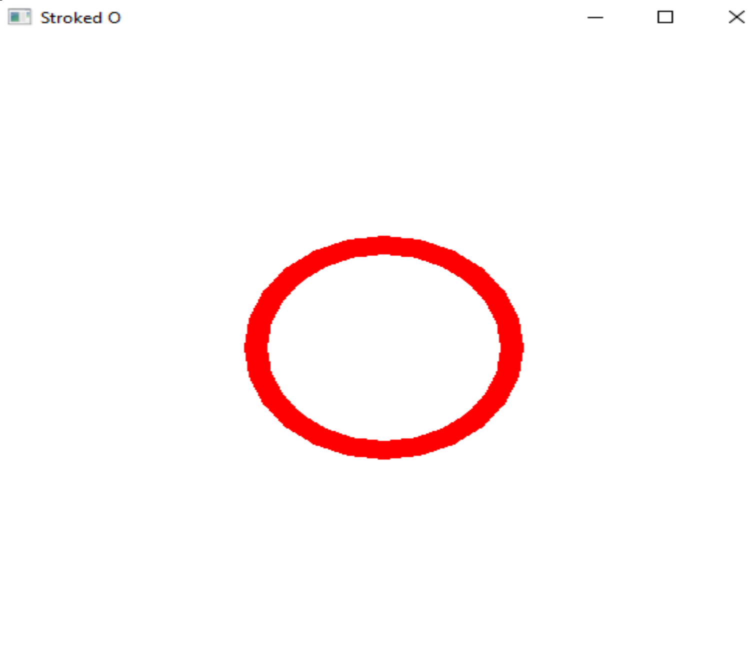
From the above figure we can observe to draw letter "I" we can use lines. Circle generation algorithm can be used for generating letter 'O'. Performance of the graphics system will be degraded for the applications that require large quantity of text if stroke text generation is used. A more efficient strategy is to define the font once, using a display list for each char and then store in the server. We define a function OurFont() which will draw any ASCII character stored in variable "c"



```
void OurFont(char c)
   {
        switch(c)
        {
            case 'a':
                ...
            break;
            case 'A':
                ...
            break;
                ...
        }
   }
```

generate stroked letter O using gl_quad_strip

```
#include<stdio.h>
#include<math.h>
#include<gl/glut.h>

void myinit()
{
        glMatrixMode(GL_PROJECTION_MATRIX);
        glLoadIdentity();
        gluOrtho2D(-50, 50, -50, 50);
        glMatrixMode(GL_MODELVIEW);
}

void display()
{
        glClearColor(1, 1, 1, 1);
        glClear(GL_COLOR_BUFFER_BIT);

        int i;
        float x1, x2, y1, y2, r1 = 15, r2 = 18, t;

        glColor3f(1, 0, 0);
        glBegin(GL_QUAD_STRIP);
        for (i = 0; i <= 24; i++)
        {
                t = 3.142 / 12 * i;
                x1 = r1 * cos(t);
                y1 = r1 * sin(t);
                x2 = r2 * cos(t);
                y2 = r2 * sin(t);
                glVertex2f(x1, y1);
                glVertex2f(x2, y2);
        }
        glEnd();

        glFlush();
}

void main()
{
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(300, 150);

        glutCreateWindow("Stroked O");
        myinit();
        glutDisplayFunc(display);
        glutMainLoop();
}
```



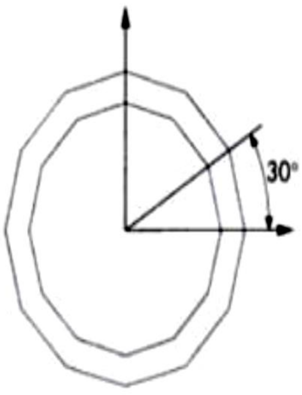
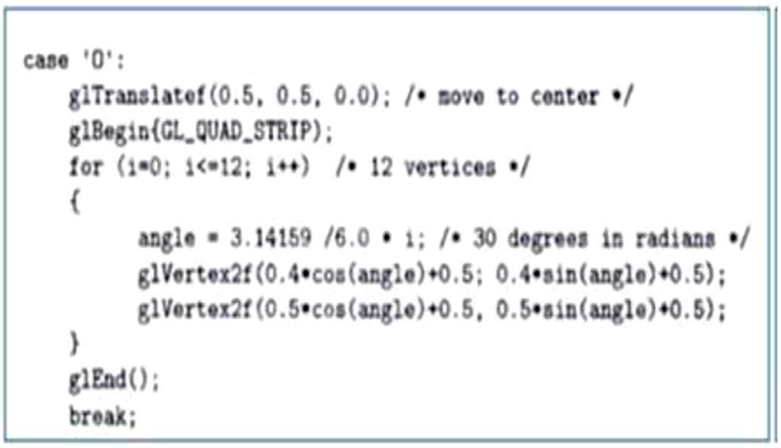


FIGURE Drawing of the letter "O."



```
case 'O':
    glTranslatef(0.5, 0.5, 0.0); /* move to center */
    glBegin(GL_QUAD_STRIP);
    for (i=0; i<=12; i++)  /* 12 vertices */
    {
        angle = 3.14159 /6.0 * i; /* 30 degrees in radians */
        glVertex2f(0.4*cos(angle)+0.5; 0.4*sin(angle)+0.5);
        glVertex2f(0.5*cos(angle)+0.5, 0.5*sin(angle)+0.5);
    }
    glEnd();
    break;
```

When we want to generate a 256-character set, the required code using OurFont() is as follows

```
base = glGenLists(256);
for(i=0;i<256;i++)
{
glNewList(base+i, GL_COMPILE);
  OurFont(i);
glEndList();

}
```

```
void OurFont(char c)
{
        switch(c)
        {
                case 'a':
                ... break;
                case 'A':
                ...
                break;
                ...
        }
}
```

When we wish to use these display lists to draw individual characters, rather than offsetting the identifier of the display lists by base each time, we can set an offset as follows:

glListBase(base);

*Finally, our drawing of a string is accomplished in the erver by the function call char *text_string;*

glCallLists((GLint) strlen(text_string), GL_BYTE, text_string);

which makes use of the standard C library function strlen to find the length of input string text_string. The first argument in the function glCallLists is the number of lists to be executed. The third is a pointer to an array of a type given by the second argument.

Fonts in Glut

In general, we prefer to use an existing font rather than to define our own. GLUT provides a few raster and stroke fonts.

glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN, int character)

We usually control the position of a character by using a translation before the character function is called.

glTranslatef(x, y, 0);

glutStrokeCharacter(GLUT_STROKE_ROMAN, 'a');

Raster and bitmap characters are produced in a similar manner. For example, a single 8 × 13 character is obtained using the following:

glutBitmapCharacter(GLUT_BITMAP_8_BY_13, int character)

Positioning of bitmap characters is considerably simpler than the positioning of stroke characters is because bitmap characters are drawn directly in the frame buffer.

The position where the next raster primitive to be placed can be set using the glRasterPos(). glutBitmapCharacter function automatically advances the raster position, so typically we do not need to manipulate the raster position until we want to define a string of characters elsewhere on the display.*

What is the Necessity of Event driven input?

Computer Graphic Applications require input from various input devices and each device either works in request mode or sample mode and hence programming event driven input is required.

Explain programming event driven input.Each with example program (mouse,keyboard,reshape,idle ,menu,callback)

PROGRAMMING EVENT DRIVEN INPUT

In this we understand various events that are recognized by the window system and based on application, we write callback functions that indicate how the application program responds to these events.

Using Pointing Devices

Pointing devices like mouse, trackball, data tablet allow programmer to indicate a position on the display. There are two types of events associated with pointing device.

*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—its measure—is made available to the application program.

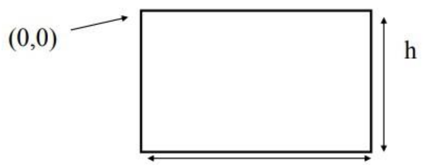
A mouse event occurs when one of the mouse buttons is either pressed or released. The information returned includes the button that generated the event(left button/right button/center button), the state of the button after the event (up or down), and the position of the cursor in window coordinates (with the origin in the upper-left corner of the window).We register the mouse callback function, usually in the main function as

glutMouseFunc(myMouse);

The mouse callback function must be defined in the form:

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

When a user presses and releases mouse buttons in the window, each press and each release generate a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON. The state parameter is either GLUT_UP or GLUT_DOWN. The value of x & y is according to window, measured from topmost left corner as follows.

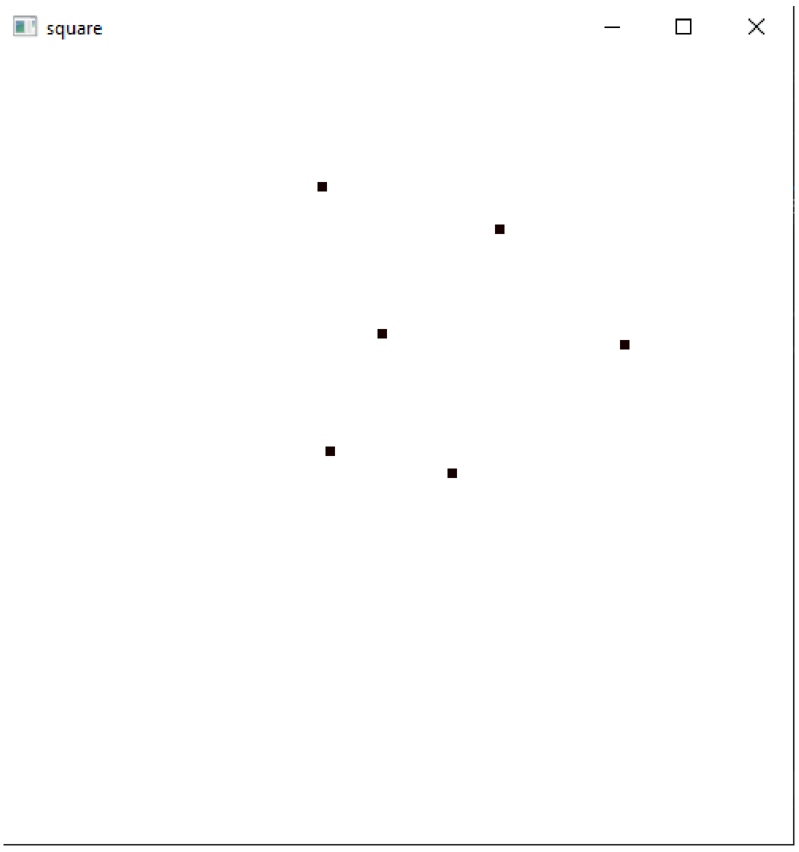


```
void myMouse(int button, int state, int x, int y)
{
if(button==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
exit(0);
}
```

The above code ensures whenever the left mouse button is pressed down, execution of the program gets terminated.

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

```
#include<stdio.h>
#include<stdlib.h>
#include<GL/glut.h>
int wh = 500, ww = 500; float siz = 3;
void myinit()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, wh, 0, ww);// xmin, xmax, ymin, ymax
        glMatrixMode(GL_MODELVIEW);
}
void drawsq(int x, int y)
{
        y = wh - y;
        glColor3f(0.1, 0.0, 0.0);
        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()
{
        glClearColor(1, 1, 1, 1);
        glClear(GL_COLOR_BUFFER_BIT);
}
void myMouse(int button, int state, int x, int y)
{
        if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
                drawsq(x, y);
        if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        {
                glClearColor(1, 1, 1, 1);
                glClear(GL_COLOR_BUFFER_BIT);
                glFlush();
        }//exit(0);
}
void main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
        glutInitWindowSize(wh, ww);
        glutCreateWindow("square");
        glutDisplayFunc(display);
        glutMouseFunc(myMouse);
        myinit();
        glutMainLoop();
}
```



Window Events

A window event is occurred when the corner of the window is dragged to new position or size of window is minimized or maximized by using mouse. The information returned to the program includes the height and width of newly resized window. The window callback function must be registered in the main function as

glutReshapeFunc(myReshape);

Window call back function must be defined as

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

Returns width and height of new window.
Code segment of reshape function

```
 void myReshape(GLsizei w, GLsizei h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION_MATRIX);
 glLoadIdentity();
float t1 = (float)w/(float)h;
float t2 = (float)h/(float)w;
 if(w>h)
gluOrtho2D(-100*t1,100*t1,-100,100);
else
 gluOrtho2D(-100,100,-100*t2,100*t2);
glMatrixMode(GL_MODELVIEW);
glutPostRedisplay();
}
```

Keyboard Events

Keyboard devices are input devices which return the ASCII value to the user program. Keyboard events are generated when the mouse is in the window and one of the keys is pressed or released.

The GLUT function ***glutKeyboardFunc*** *is the callback for events generated by pressing a key, whereas* ***glutKeyboardUpFunc*** *is the callback for events generated by releasing a key.*

When a keyboard event occurs, the ASCII code for the key that generated the event and the location of the mouse are returned
glutKeyboardFunc(myKey);

The keyboard callback function must be defined in the form:
void mykey (unsigned char key, int x, int y)

key is the ASCII value of the char entered. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. For example,

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

The above code ensures when “Q   or “q   key is pressed, the execution of the program gets terminated.

glutKeyboardFunc: registers callback handler for keyboard event.

*void glutKeyboardFunc (void (*func)(unsigned char key, int x, int y)*

// key is the char pressed, e.g., 'a' or 27 for ESC

// (x, y) is the mouse location in Windows' coordinates

glutSpecialFunc: registers callback handler for special key (such as arrow keys and function keys).

*void glutSpecialFunc (void (*func)(int specialKey, int x, int y)*

// specialKey: GLUT_KEY_* (for LEFT, RIGHT, UP, DOWN, HOME, END, PAGE_UP, PAGE_DOWN, F1,...F12).*

// (x, y) is the mouse location in Windows' coordinates

The Display and Idle Callbacks

Display callback is specified by GLUT using glutDisplayFunc(myDisplay). It is invoked when GLUT determines that window should be redisplayed. Re-execution of the display function can be achieved by using ***glutPostRedisplay().***

The idle callback is invoked when there are no other events. It is specified by GLUT using ***glutIdleFunc(myIdle).***

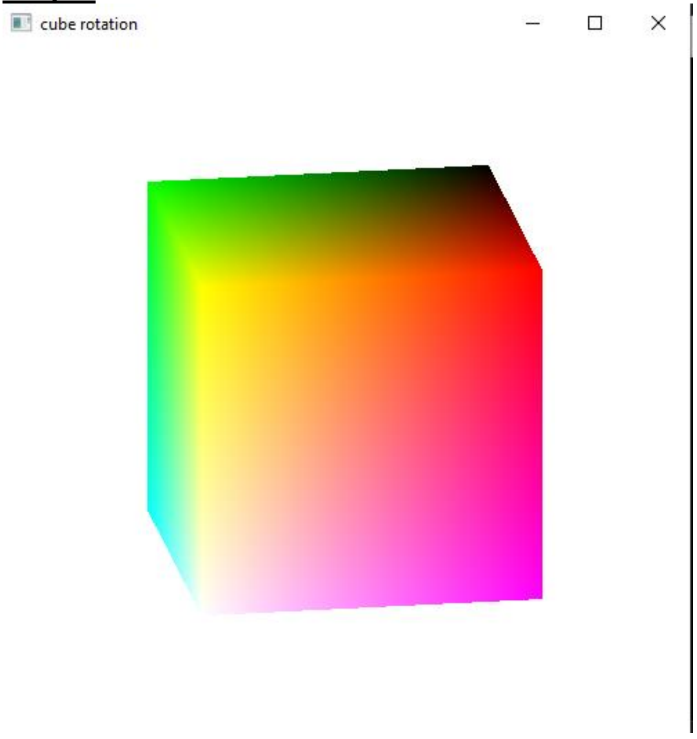
```
void spinCube()
{
/* Idle callback, spin cube 2 degrees about selected axis */
theta[axis] += 2.0;
if( theta[axis] > 360.0 )
   theta[axis] -= 360.0;
glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{
/* mouse callback, selects an axis about which to rotate */
if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}

glutIdleFunc(spinCube);
glutMouseFunc(mouse);
```

Draw a color cube and spin it using OpenGL transformation matrices.

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
```

```
float v[8][3]={{-1,-1,1},{-1,1,1},{1,1,1},{1,-1,1},{-1,-1,-1},{-1,1,-1},{1,1,-1},{1,-1,-1}};
float p[8][3]={{0,0,1},{0,1,1},{1,1,1},{1,0,1},{0,0,0},{0,1,0},{1,1,0},{1,0,0}};
float theta[3]={0,0,0};
int flag=2;

void myinit()
{
       glMatrixMode(GL_PROJECTION);
       glLoadIdentity();
       glOrtho(-2,2,-2,2,-2,2);
       glMatrixMode(GL_MODELVIEW);
}
void idlefunc()
{
          theta[flag]++;
          if(theta[flag]>360)
          theta[flag]=0;
          glutPostRedisplay();
}
void mousefunc(int button,int status,int x,int y)
{
        if(button==GLUT_LEFT_BUTTON&&status==GLUT_DOWN)
            flag=2;
        if(button==GLUT_MIDDLE_BUTTON&&status==GLUT_DOWN)
            flag=1;
         if(button==GLUT_RIGHT_BUTTON&&status==GLUT_DOWN)
             flag=0;
}
void drawpoly(int a,int b,int c,int d)
{
       glBegin(GL_POLYGON);
           glColor3fv(p[a]);
           glVertex3fv(v[a]);
           glColor3fv(p[b]);
          glVertex3fv(v[b]);
          glColor3fv(p[c]);
          glVertex3fv(v[c]);
          glColor3fv(p[d]);
          glVertex3fv(v[d]);
     glEnd();
}
void colorcube()
{
        drawpoly(0,1,2,3);
        drawpoly(0,1,5,4);
        drawpoly(1,5,6,2);
        drawpoly(4,5,6,7);
        drawpoly(3,2,6,7);
        drawpoly(0,4,7,3);
}
void display()
{
       glClearColor(1,1,1,1);
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       glColor3f(1,0,0);
       glEnable(GL_DEPTH_TEST);
       glLoadIdentity();
       glRotatef(theta[0],1,0,0);//x
       glRotatef(theta[1],0,1,0);//y
      glRotatef(theta[2],0,0,1);//z
      colorcube();
      glFlush();
      glutSwapBuffers();
}
void main()
{
          glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |GLUT_DEPTH);
          glutInitWindowPosition(100,100);
          glutInitWindowSize(500,500);
          glutCreateWindow("cube rotation");
          myinit();
          glutDisplayFunc(display);
          glutMouseFunc(mousefunc);
          glutIdleFunc(idlefunc);
          glutMainLoop();
}
```

Output



Window Management

GLUT also supports multiple windows and Sub windows of a given window. We can open a second top-level window

id=glutCreateWindow("second window");

The returned integer value allows us to select this window as the current window into which objects will be rendered as follows:
glutSetWindow(id);

Furthermore, each window can have its own set of callback functions because callback specifications refer to the present window.

MENUS

how menus and submenus are created using glut?Illustrate with example.

Menus are an important feature of any application program. OpenGL provides a feature called "Pop-up-menus" using which sophisticated interactive applications can be created. Menu creation involves the following steps:

1. Define the actions corresponding to each entry in the menu.
2. Link the menu to a corresponding mouse button.
3. Register a callback function for each entry in the menu.

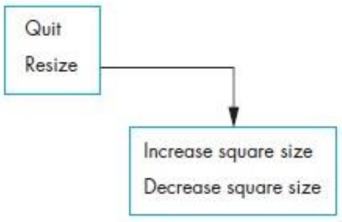
For example we can demonstrate simple menus with the example of a pop-up menu that has three entries. The first selection allows us to exit our program. The second and third change the size of the squares in our ***drawSquare*** *function. We name the menu callback demo_menu. The function calls to set up the menu and to link it to the right mouse button should be placed in our main function.*

```
glutCreateMenu(demo_menu);
glutAddMenuEntry("quit",1);
glutAddMenuEntry("increase square size", 2);
glutAddMenuEntry("decrease square size", 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

The glutCreateMenu() registers the callback function demo_menu. The function glutAddMenuEntry() adds the entry in the menu whose name is passed in first argument and the second argument is the identifier passed to the callback when the entry is selected.

```
void demo_menu(int id)
{
switch(id)
{
case 1: exit(0);
break;
case 2: size = 2 * size;
break;
case 3: if(size > 1) size = size/2;
break;
}
glutPostRedisplay( );
}
```

GLUT also supports hierarchical menus, as shown in the below Figure.



Suppose that we want the main menu that we create to have only two entries. The firstentry still causes the program to terminate, but now the second causes a submenu topop up. The submenu contains the two entries for changing the size of the square inour square-drawing program.

```
 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);
```

Picking

It is the logical input operation that allows the user to identify an object on the display.

The action of picking uses pointing device but the information returned to the application program is the identifier of an object not a position.

It is difficult to implement picking in modern system because of graphics pipeline architecture. Therefore, converting from location on the display to the corresponding primitive is not direct calculation.

There are at least three ways to deal with this difficulty

Selection: *It involves adjusting the clipping region and viewport such that we can keep track of which primitives lies in a small clipping region and are rendered into region near the cursor. These primitives are sent into a hit list that can be examined later by the user program.*

Bounding boxes or extents: *A simple approach is to use (axis-aligned) bounding boxes, or extents, for objects of interest. The extent of an object is the smallest rectangle, aligned with the coordinates axes, that contains the object.*

For two-dimensional applications, itis relatively easy to determine the rectangle in screen coordinates that corresponds toa rectangle point in object or world coordinates.

What is double buffer?Explain advantages of double buffering.

Usage of back buffer and extra rendering*: When we use double buffering it has two color buffers: front and back buffers. The contents present in the front buffer is displayed, whereas contents in back buffer is not displayed so we can use back buffer for other than rendering the scene. Suppose that we render our objects into the back buffer, each in a distinct color. The application program will identify the object based on the color saved.*

Picking can be performed in four steps that are initiated by user defined pick function in the application:

- *We draw the objects into back buffer with the pick colors.*
- *We get the position of the mouse using the mouse callback.*
- *Use glReadPixels() to find the color at the position in the frame buffer corresponding to the mouse position.*
- *We search table of colors to find the object corresponds to the color read.*

PICKING AND SELECTION MODE

The difficult problem in implementing picking within the OpenGL pipeline is that we cannot go backward directly from the position of the mouse to primitives that were rendered close to that point on the screen.

OpenGL provides a somewhat complex process using a rendering mode called selection mode to do picking at the cost of an extra rendering each time that we do a pick.

Initially the application will be in GL_RENDER which is the default mode. When there is a click, we reduce the viewport to the region of click and change the mode to GL_SELECT. In GL_SELECT the objects primitives can be retrieved and It can be identified.

The function glRenderMode lets us select one of three modes: normal rendering to the color buffer (GL_RENDER),selection mode (GL_SELECT), or feedback mode (GL_FEEDBACK).

When we enter selection mode and render a scene, each primitive within the clipping volume generates a message called a hit that is stored in a buffer called the ***name stack****.*

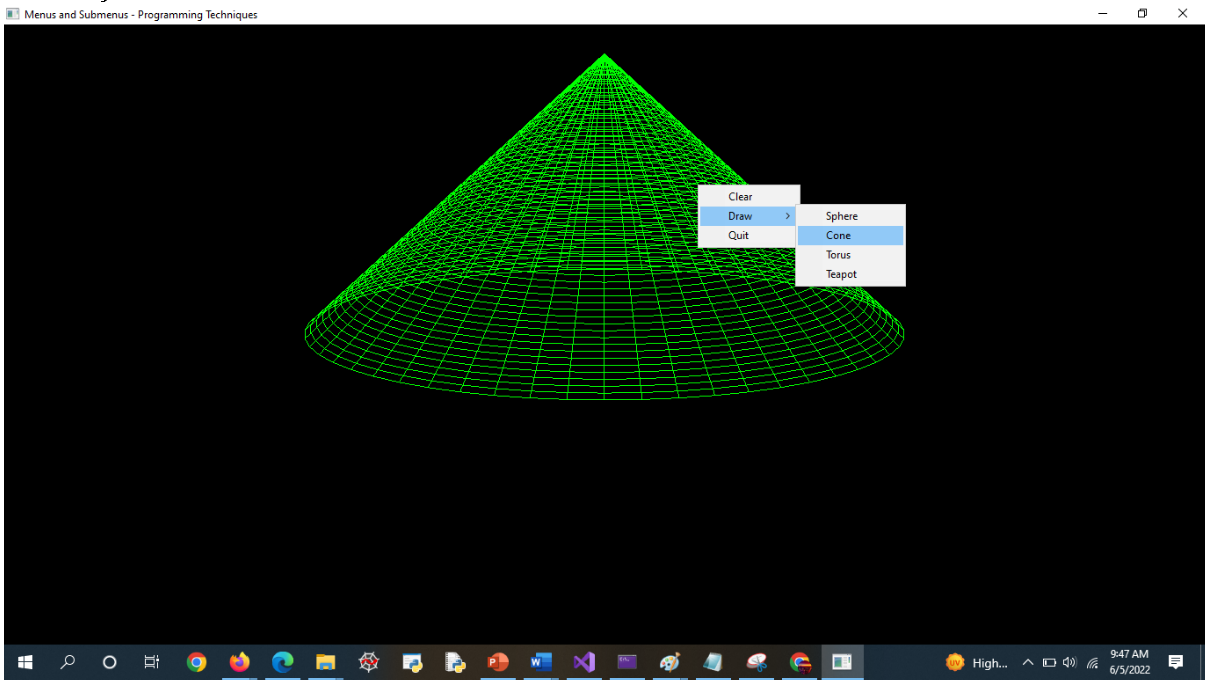
The following functions are used in selection mode:

- *void glInitNames() // initializes the name stack.*
- *void glPushName(GLuint name) // pushes name on the name stack.*
  void glPopName() // pops the top name from the name stack.
- *void glLoadName(GLuint name) // replaces the top of the name stack with name.*
- *OpenGL allow us to set clipping volume for picking using gluPickMatrix() which is applied before gluOrtho2D.*
- *gluPickMatrix(x,y,w,h,*vp) : creates a projection matrix for picking that restricts drawing to a w * h area and centered at (x,y) in window coordinates within the viewport vp.*

Open GL Menu & Submenu

```
#include <windows.h>
#include <GL/glut.h>
static int window;
static int menu_id;
static int submenu_id;
static int value = 0;
```

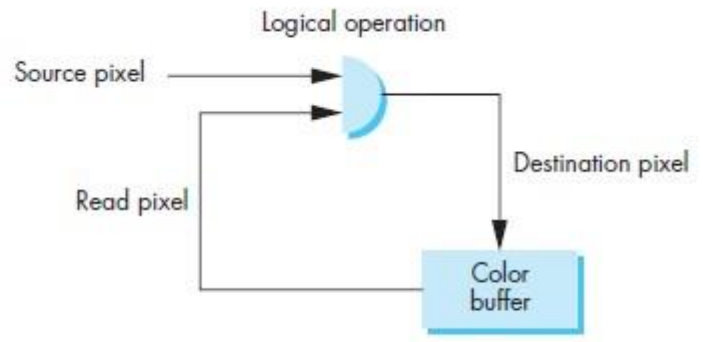
```
void menu(int num) {
        if (num == 0) {
                glutDestroyWindow(window);
                exit(0);
        }
        else {
                value = num;
        }
        glutPostRedisplay();
}
void createMenu(void) {
        submenu_id = glutCreateMenu(menu);
        glutAddMenuEntry("Sphere", 2);
        glutAddMenuEntry("Cone", 3);
        glutAddMenuEntry("Torus", 4);
        glutAddMenuEntry("Teapot", 5);
          menu_id = glutCreateMenu(menu);
        glutAddMenuEntry("Clear", 1);
        glutAddSubMenu("Draw", submenu_id);
        glutAddMenuEntry("Quit", 0);
          glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        if (value == 1) {
                glutPostRedisplay();
        }
        else if (value == 2) {
                glPushMatrix();
                glColor3d(1.0, 0.0, 0.0);
                glutWireSphere(0.5, 50, 50);
                glPopMatrix();
        }
        else if (value == 3) {
                glPushMatrix();
                glColor3d(0.0, 1.0, 0.0);
                glRotated(65, -1.0, 0.0, 0.0);
                glutWireCone(0.5, 1.0, 50, 50);
                glPopMatrix();
        }
        else if (value == 4) {
                glPushMatrix();
                glColor3d(0.0, 0.0, 1.0);
                glutWireTorus(0.3, 0.6, 100, 100);
                glPopMatrix();
        }
        else if (value == 5) {
                glPushMatrix();
                glColor3d(1.0, 0.0, 1.0);
                glutSolidTeapot(0.5);
                glPopMatrix();
        }
        glFlush();

}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        window = glutCreateWindow("Menus and Submenus");
        createMenu();
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glutDisplayFunc(display);
        glutMainLoop();
        }
```



LOGIC OPERATIONS Explain logical operators(rubberband technique)

Rubberbanding, a technique used for displaying line segments (and other primitives) that change as they are manipulated interactively.

When a program specifies a primitive that is visible, OpenGL renders it into a set of colored pixels that are written into the present drawing buffer. In the default mode of operation, these pixels simply replace the corresponding pixels that are already in the frame buffer, this mode, called copy, or replacement, mode, it does not matter what color the original pixels under the rectangle were before.



The pixel that we want to write is called the source pixel. The pixel in the drawing buffer that the source pixel will affect is called the destination pixel. In copy mode, the source pixel replaces the destination pixel.

The second mode is the exclusive OR or XOR mode in which corresponding bits in each pixel are combined using the exclusive or logical operation. If s and d are corresponding bits in the source and destination pixels, we can denote the new destination bit as d^{i}, and it is given by

d'= d xor s,

The most interesting property of the xor operation is that if we apply it twice, we return to the original state. That is,

d'= (d xor s) xor s.

Thus, if we draw something in xor mode, we can erase it by simply drawing it a second time.

To change mode, we must enable logic operation, glEnable(GL_COLOR_LOGIC_OP) and then it can change to XOR mode glLogicOp(GL_XOR). GL_COPY is the default operator.

Drawing Erasable Lines

We use the mouse to get the first endpoint and store this point in object coordinates as follows:

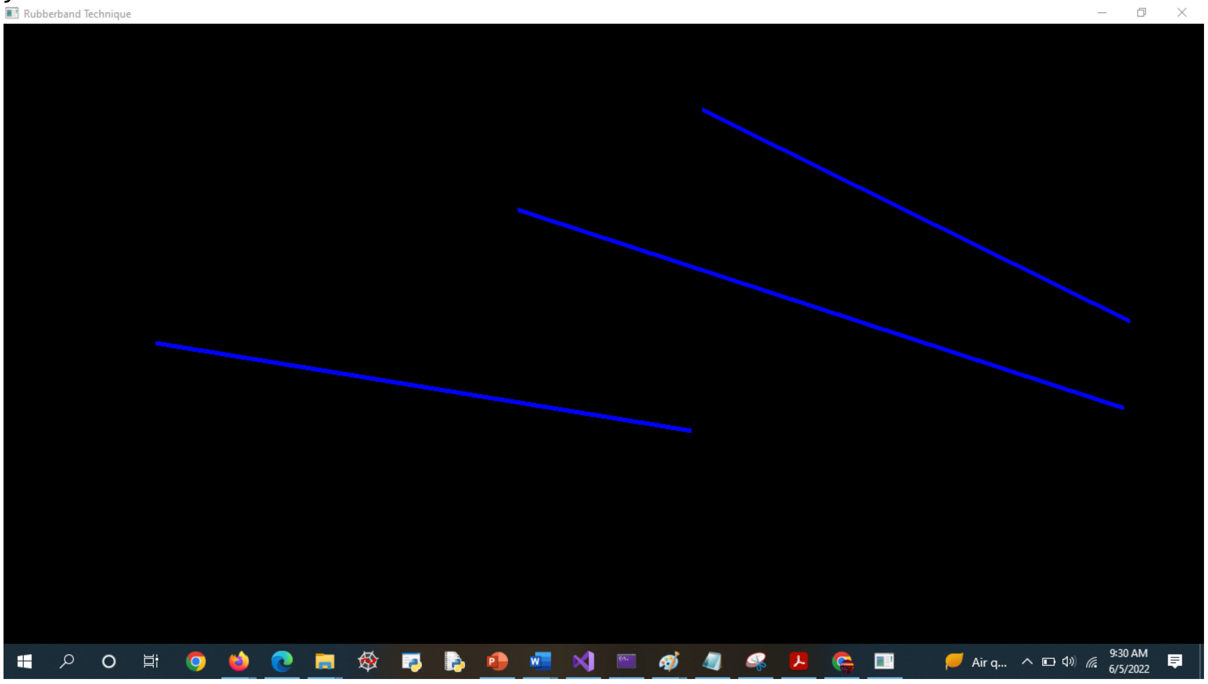
```
xm = x/500.;
ym = (500-y)/500.;
```

We then can get the second point and draw a line segment in xor mode:

```
xmm = x/500.;
ymm = (500-y)/500.;
glLogicOp(GL_XOR);


glBegin(GL_LINES);
glVertex2f(xm, ym);
 glVertex2f(xmm, ymm);
glLogicOp(GL_COPY);
 glEnd();
glFlush();
```

OpenGL -Rubberbanding

```
#include<stdio.h>
#include<gl/glut.h>
float xm, ym, xmm, ymm;
int first = 0, w = 600, h = 600;
void init()
{
        glMatrixMode(GL_PROJECTION_MATRIX);
        glLoadIdentity();
        gluOrtho2D(0, w, 0, h);
        glMatrixMode(GL_MODELVIEW);
}
void disp()
{
        glClearColor(0, 0, 0, 1);
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0, 0, 1);
        glLineWidth(5);
        glFlush();
}
void mouse(int b, int s, int x, int y)
{
        glColor3f(0, 0, 1);
        y = h - y;
        if (b == GLUT_LEFT_BUTTON && s == GLUT_DOWN)
        {
                xm = x;
                ym = y;
                first = 0;
        }
        if (b == GLUT_LEFT_BUTTON && s == GLUT_UP)
        {
                glLogicOp(GL_XOR);
                glBegin(GL_LINES);
                glVertex2f(xm, ym);
                glVertex2f(xmm, ymm);
                glEnd();
                glFlush();
                glLogicOp(GL_COPY);
                glBegin(GL_LINES);
                glVertex2f(xm, ym);
                glVertex2f(xmm, ymm);
                glEnd();
                glFlush();
        }
        if (b == GLUT_RIGHT_BUTTON && s == GLUT_DOWN)
        {
                glClearColor(0, 0, 0, 1);
                glClear(GL_COLOR_BUFFER_BIT);
                glFlush();
        }
        glFlush();
}
void move(int x, int y)
{
        y = h - y;
        if (first == 1)
        {
                glLogicOp(GL_XOR);
                glBegin(GL_LINES);
                glVertex2f(xm, ym);
                glVertex2f(xmm, ymm);
                glEnd();
        }
        xmm = x;
        ymm = y;
        glLogicOp(GL_XOR);
        glBegin(GL_LINES);
        glVertex2f(xm, ym);
        glVertex2f(xmm, ymm);
        glEnd();
        glFlush();
        first = 1;
}
void main()
{
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(600, 600);
        glutInitWindowPosition(300, 150);
        glutCreateWindow("Rubberband Technique");
        init();
        glEnable(GL_COLOR_LOGIC_OP);
        glutDisplayFunc(disp);
        glutMouseFunc(mouse);
        glutMotionFunc(move);
        glutMainLoop();
}
```



List the various Features that a good Interactive program should include

1. A smooth display, showing neither flicker nor any artifacts of the refresh process
2. A variety of interactive devices on the display
3. A variety of methods for entering and displaying information
4. An easy-to-use interface that does not require substantial effort to learn
5. Feedback to the user

6. Tolerance for user errors
7. A design that incorporates consideration of both the visual and motor properties of the human.