66. What is affine transformation? → 159
67. List the 3D OpenGL geometric transformations. → 168
68. What is color model? Explain the RGB color model. → 171
69. Explain the CMY and CMYK color models. → 173
70. What is light source? Explain the types of light source. → 175
71. Explain the PHONG model. → 182

MODULE 4

72. What is projection plane, parallel and perspective projection? → 184
73. What is depth cueing? → 188
74. Explain the 3D viewing pipeline with diagram. → 189
75. Explain the transformation from world to viewing coordinates. → 190
76. Explain the orthogonal projections. → 192
77. Explain the perspective projection transformation coordinates. → 194
78. Explain the OpenGL 3D viewing functions. → 197
79. Classify the visible surface detection algorithms. → 200
80. Explain the back-face detection algorithm. → 201
81. Explain the z-buffer/depth-buffer algorithm. → 203
82. Explain the OpenGL visibility detection functions. → 206
83. Explain in detail, Oblique and Symmetric perspective projection frustum. → 209
84. Explain vanishing points for perspective projections. → 212
85. Explain briefly the following: → 214
    a) Projections
    b) Depth Cueing
    c) Identifying visible lines and surfaces
    d) Surface rendering
    e) Exploded and cutaway views
    f) 3D and stereoscopic viewing
86. Explain viewup vector and uvn viewing coordinate reference frame → 220
87. Write short notes on axonometric and isometric orthogonal projections → 222.
88. Explain OpenGL functions with respect to: → 223
    a. Viewing Transformation functions
    b. Orthogonal Projection functions
    c. Symmetric Perspective Projection functions
    d. General Perspective Projection functions
    e. Viewport and Display Window
89. Imagine you have a 3D object in front of you. Illustrate how to Normalize the transformation for an Orthogonal Projection? → 227

MODULE 5

90. Explain how an event driven input can be performed for → 229
    (a) window events (b) pointing devices
91. Explain how an event driven input can be programmed for a keyboard device. → 232
92. List out any four characteristics of good interactive program. → 234
93. What are the major characteristics that describe the logical behavior of an input device? → 236
94. Explain how OpenGL provides the functionality of each of the classes of logical input devices. → 238

95. What is a display list? Explain the execution of display list. Give the OpenGL code segment that generates a display list defining a red triangle with vertices at (50,50), (150,50) and (100, 150). → 240

96. How pop-up menus are created using GLUT? Illustrate with an example. → 242

97. Explain the quadric surfaces. → 243

98. Explain the OpenGL Quadric-Surface and Cubic Surface Functions. → 246

99. Explain the Bezier Spline Curves. Give the equation representing control points of the Bezier spline curves. Discuss its properties. Also, draw Bezier curve with 4 and 3 control points. → 248

100. Explain the Bezier surfaces and its equations. → 251

101. Explain Request, sample and event Input modes with block diagram. → 252

102. Write the program snapshot, explain the creation of Menus in OpenGL → 254

103. With the role of glCallList() function in creating Display lists in OpenGL. Write OpenGL code for rendering a simple animated face. → 256

104. Demonstrate the OpenGL Visibility Detection Function. → 259

105. List and explain different input physical devices. → 261

106. Define measure & trigger. List and explain different input modes. → 264

107. Write short notes on: → 266

a) Client & Server
b) Display Lists
c) Texts & display lists
d) Fonts in GLUT

108. Explain briefly display lists and modelling. → 272

109. Explain menu creation and hierarchical menus with example code. → 274

110. Briefly explain different ways to overcome difficulty in picking. → 279

111. What are the features a good interactive program must have. → 281

112. With code snippet explain drawing erasable lines. → 282

113. Explain with OpenGL functioning double buffering and timer. → 285

114. Write a note on quadratic surface w.r.t sphere, ellipse and torus. → 286

115. Explain OpenGL functions to display sphere, cone, torus and teapot. → 289

116. Explain briefly quadratic surface functions. → 291

117. Implement OpenGL program to display wired cone, wired cylinder and wired sphere. → 294

118. Illustrate how an interactive program is animated. → 296

119. Represent simple graphics & display processor architecture. Explain 2 ways of sending graphical entities to a display and list the advantages and disadvantages. → 298

120. Discuss the following logical operations with suitable examples
(a)Copy mode (b)Exclusive OR mode © rubber-band effect (d) drawing erasable lines. → 300

MODULE - 5

90. Explain how an event driven input can be performed for (a) Window events.
(b) Pointing devices.

(a) Window events : A window event is occured 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. Programming the window event involves two steps :

1. Window call back function must be defined in the form :
Void myReshape (GLsizei w, GLsizei h) is written by the application programmer.

Let us consider drawing square as an example, the square of same size must be drawn regardless of window size.

```
Void myReshape (GLsizei w, GLsizei h) {
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluOrtho2D (0, (GLdouble) w, 0, (GLdouble) h);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glViewPort (0, 0, w, h)
/* save new window size in global
   Variables */ ww = w;
   wh = h;
}
```

2. The window call back function must be registered in the main function.

glutReshapeFunc (myReshape);

(b) Pointing devices :

Pointing devices like mouse, trackball, data tablet allow programmer to indicate a position on the display.

There are two types of event associated with pointing device, which is conventionally assomed to be mouse but could be trackball or data tablet also.

1. MOVE event - is generated when the mouse is moved with one of the button being pressed. If the mouse is moved without a button being pressed, this event is called as "passive move event".

2. MOUSE event - is generated when one of the mouse buttons is either pressed or released.

The information returned to the application program includes button that generated the event, state of the button after event (up or down), position (x, y) of the cursor.

Programming a mouse event involves two steps :

1. The mouse callback function must be defined in the form: void myMouse (int button, int state, int x, int y)

For example,

```
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.

2. Register the defined mouse callback function in the main function, by means of GLUT function:

```
glutMouseFunc (myMouse);
```

91) Explain how an event driven input can be programmed for a keyboard device.

Keyboard events are generated when the mouse is in the window and one of the keys is pressed or released.

With keyboard input, we use the following function to specify a procedure that is to be invoked when a key is pressed:

```
glutKeyboardFunc(keyfcn);
```

The specified procedure has three arguments:

```
void keyfcn (GLubyte key, GLint xMouse, GLint yMouse)
```

Parameter key is assigned a character value or the corresponding ASCII code. The display-window mouse location is returned as position (xMouse, yMouse) relative to the top-left corner of the display window.

When a designated key is pressed, we can use the mouse location to initiate some action, independently of whether any mouse buttons are pressed.

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

```
glutSpecialFunc (specialKeyfcn);
```

The specified procedure has same three arguments;

void specialKeyfcn (GLint specialKey, GLint xMouse, GLint yMouse).

Ex:

```
glutKeyboardFunc(dir);
void dir(unsigned char key, int x, int y)
{ if (key=='a'|| key=='A')
      rotate=1;
  if (key=='b' || key=='B')
      rotate=2;
}
```

9 92) List out any four characteristics of good interactive program.

The characteristics of good interactive program are:

* 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.
All information in the user dialogue is presented in the language of the application.

* Windows and Icons:

In addition to the standard display-window operations, 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 accomodated, each with its own window manager.

* 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 using the same combination of keyboard keys for action and always using the same color encoding so that a color does not create inconsistency.

* 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.

One key or button used for all delete operations is easier to remember than a number of different keys for different kinds of delete procedures.

* Backup and Error Handling

A mechanism for undoing a sequence of operations is a 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.

In addition, good diagnostics and error messages help a user to determine the cause of an error.

* Accommodating multiple 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 ~~pre~~ preferable.

Experienced users typically want speed. This means fewer prompts and more input from the keyboard.

An interface may be designed to provide different sets of options to users with different experience levels.

93) What are the major characteristics of That describe the logical behavior of an Input device ?

Answer

The main characteristics that describes the logical behavior of an Input device :

i) What measurements the device returns to the user program

ii) When the device returns those measurements

In general, there are six classes of logical input devices :

a) String – provides ASC II strings to the user progam(logically implemented via keyboard.)

b) Locator – Provides a position in world coordinates to to the user program (pointing devices and conversion may be needed)

c) Pick – return the identifier of an object to the user program (pointing devices and conversions may be needed)

d) choice – allows user to select one of the distinct number of options (widgets – menus, scrollbars and graphical buttons)

e) Dial – provides analog input to user program (widgets)

f) stroke – It returns an array of locations (similar to multiple use of a locator continuously)

Shankar R
Asst Professor,
CSE, BMSIT&M

Q94) Explain how Open GL provides the functionality of each of the classes of logical input devices ?

When input functions are classified according to data type, any device that is used to provide the specified data is referred to as a logical input device for that data type. The standard logical input-data classifications are-

LOCATOR - A device for specifying one coordinate position.

STROKE - A device for specifying a set of coordinate positions.

STRING - A device for specifying text input.

VALUATOR - A device for specifying a scalar value.

CHOICE - A device for selecting a menu option.

PICK - A device for selecting a component of a picture.

Locator device - Interactive selection of a co-ordinate point is usually accomplished by positioning the screen cursor at some location in a displayed scene, although other methods such as menu options, could be used in certain applicatn. We can use a mouse, joystick, trackball, spaceball, ~~thumbball~~, thumbwheel dial, hand cursor, or digitizer styles for screen-cursor positioning. And various buttons, keys, or switches can be used to indicate processing options for the selected location.

Stroke devices - This class of logical devices is used to input a sequence of co-ordinate position & the physical devices used for generating locator input are also used as stroke devices.

String devices - The primary physical device used for string input is, the keyboard. Character strings in CG applications are typically used for picture or graph labelling.

valuator devices - ~~to~~ A typical physical device used to provide valuator input is a panel of control dials. Dial settings are callibrated to produce numerical values within some predefined range. Any keyboard with a set of numeric keys can be used as a valuator device.

Choice devices - Menus are typically used in graphic programs to select processing options, parameter values & object shapes are used in constructing a picture. Commonly used choice devices for selecting a menu option are cursor positioning devices such as mouse, trackball, keyboard, touch panel or button box.

Pick devices - We use a pick device to select a part of a scene that is to be transformed or edited in some way.

Q95) What is display list? Explain the execution of display list. Give the OpenGL code segment that generates a display list defining a red triangle with vertices at (50,50), (150,50), (100,150).

Display list is a group of OPENGL commands that have been stored for later execution. Once a display list is created all vertex & pixel data are evaluated and copied into the display list memory on the server machine.

glNewList() and glEndList() are used to begin and end the definition of a display list which is then invoked by supplying its identifying index with glCallList().
a display list is created in the init routine this display list contains OpenGL command to create red triangle.

```
#include <GL/glut.h>
GLuint triangle;
void draw_triangle()
{
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_TRIANGLES);
         glVertex(50, 50);
        glVertex(150, 50);
         glVertex(100, 150);
        glEnd();
}
```

```
void init ( )
{
        glMatrixMode (GL_PROJECTION);
        glLoad Identity ( );
        gluOrtho2D ( 0.0, 500, 0.0 ,500);
        glMatrixMode (GL_MODELVIEW);
         glClearColor (1.0, 1.0, 1.0, 1.0);
         triangle = glGenLists( 1);
         glNewList (triangle , GL_COMPILE);
          draw_triangle( );
         glEndList( );
    }
 void display( )
 {      glClear(GL_COLOR_BUFFER_BIT);
        glCallList(triangle);
          glFlush( );
 }
 int main ( int argc, char ** argv)
 {      glutInit (&argc ,argv);
        glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB);
        glutCreateWindow("Triangle Display List");
         init ( );
         glutDisplayFunc(display);
         glutMainLoop( );
          return 0;

 }
```

Q96: How pop-up menus are created using GLUT? Illustrate with an example.

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

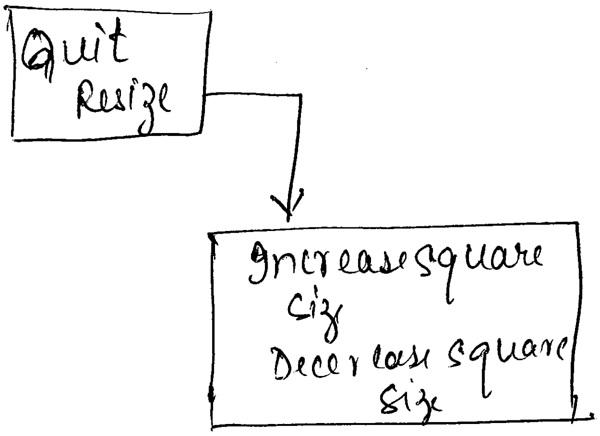
1. Define the actions corresponding to each entry in the menu.
2. Link the menu to a corresponding mouse button.
3. Register a callback fun^n for each entry in the menu.

```
glutCreateMenu( demo_menu);
glutAddMenuEntry ("quit", 1);
glutAddMenuEntry ("increase square size",2);
```

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 entry is selected.

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

GLUT also supports the creation of hierarchial menu which is given below.



: str. of hierarchial menus

Q97: Explain the quadric surfaces.

Ans: Frequently used class of objects with second-degree eqns. Includes spheres, ellipsoids, tori, paraboloids, hyperboloids.

Sphere

In Implicit/cartesian co-ordinates, a spherical surface with radius 'r' centered at co-ordinate origin is defined as set of point (x,y,z) that satisfy the eq^n

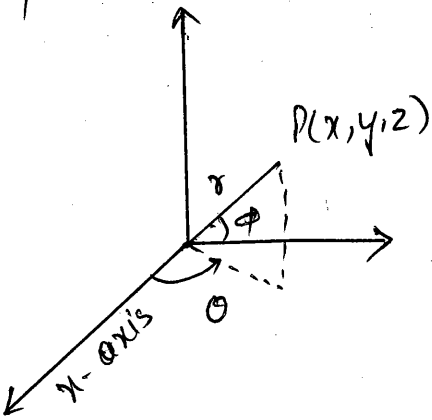
$$x^2+y^2+z^2=r^2$$

Parametric form

$$x = r\cos\phi\cos\theta \quad -\pi/2 \le \phi \le \pi/2$$

$$y = r\cos\phi\sin\theta \quad -\pi \le \theta \le \pi$$

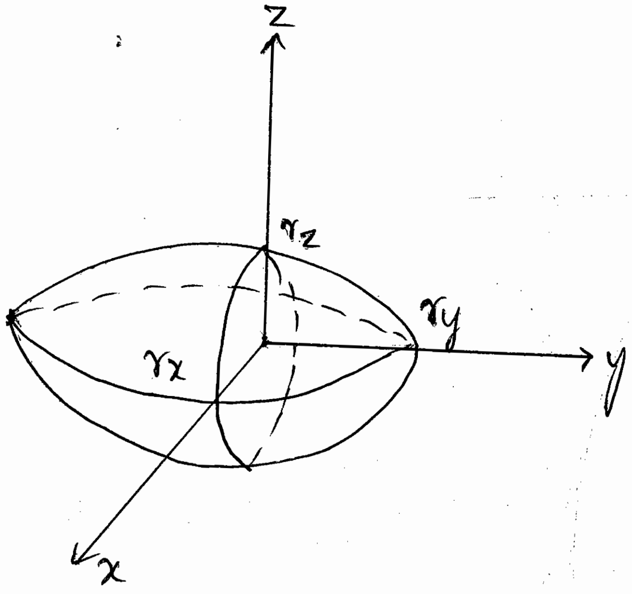
$$z = r\sin\phi$$



Ellipsoid

Cartesian representation on the origin

$$\left(\frac{x}{r_x}\right)^2+\left(\frac{y}{r_y}\right)^2+\left(\frac{z}{r_z}\right)^2=1$$



Parametric representation for the ellipsoid in terms of lattitude angle ϕ & longitude angle θ

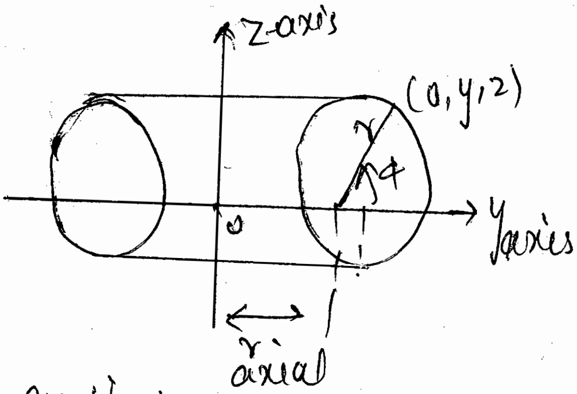
$$x=r_x\cos\phi\cos\theta \qquad -\pi/2\leq\phi\leq\pi/2$$

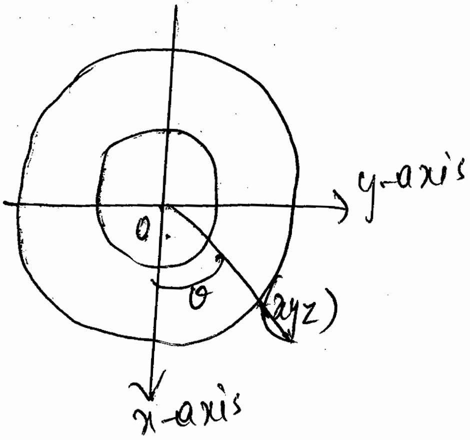
$$y=r_y\cos\phi\sin\theta \qquad -\pi\leq\theta\leq\pi$$

$$z=r_z\sin\phi$$

Torus: A doughnut shaped object is called a torus or anchor ring.

Parameters for a torus - the distance of conic center from the rotation axis.





Side View

The =n for the cross-sectional circle in side view fig is

$$(y-r_{axial})^2+z^2=r^2$$

Rotating this ⊙ about z-axis produces the torus with Cartesian =n

$$\left(\sqrt{x^2+y^2} - r_{axial}\right)^2 + z^2 = r^2$$

Parametric eqn for torus with ⊙ cross section are

$$x = (r_{axial} + r\cos\phi)\cos\theta \qquad -\pi \le \phi \le \pi$$

$$y = (r_{axial} + r\cos\phi)\sin\theta \qquad -\pi \le \theta \le \pi$$

$$z = r\sin\phi$$

We can generate a torus by rotating an ellipse instead of a ⊙ about z-axis.

For an ellipse in yz plane with semimajor & semiminor axis denoted by r_y & r_z

$$\left(\frac{y - r_{axial}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

Cartesian eqn

$$\Rightarrow \left(\frac{\sqrt{x^2+y^2} - r_{axial}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

Parametric eqn

$$x = (r_{axial} + r_y\cos\phi)\cos\theta \qquad -\pi \le \phi \le \pi$$

$$y = (r_{axial} + r_y\cos\phi)\sin\theta \qquad -\pi \le \theta \le \pi$$

$$z = r_z\sin\phi$$

98 Explain ~~the~~ OpenGL Quadric-Surface and Cubic Surface Functions

OpenGL Quadric Surface Functions :

- glutWireSphere (r, nLongitudes, nLatitudes);
  glutSolidSphere (r, nLongitudes, nLatitudes);

  r - Sphere radius in double precision point

  nLongitude & nLatitudes is number of longitude & latitude lines used to approx the sphere

- glutWireCone (rBase, height, nLongtitudes, n Latitudes)
  glutSolidCone (rBase, height, nLongtitudes, n Latitudes)

  rBase - radius of cone base

  height - height of cone

  nLongitudes & nLatitudes - assigned int values that specify the no. of orthogonal surface lines

OpenGL Cubic-Surface Functions :

- glutWireTeapot (size);
  glutSolidTeapot (size);

  - Teapot surface is generated using OpenGL Bezier curve
  - Size - sets the double precision floating-point value for maximum radius of teapot bowl.
  - Teapot is centered on world-co-ordinate origin to with its vertical axis along the y-axis.

99 Explain the Bezier Spline Curves. Give the equation representing control points of the Bezier spline curves. Discuss it's properties. Also, draw bezier curve with 4 and 3 control points.

Ans :

- Developed by French engineer Pierre Bezier for use in design of Renault automobile bodies.
- Bezier splines have a number of propoties that make them highly useful for curve and surface design. They are also easy to implement.
- Bezier curve section can be fitted to any number of control points.

Equation :-

$$P_k = (x_k, y_k, z_k) \qquad k \rightarrow 0 \text{ to } n$$

P_k = General $(n+1)$ control-point positions

$P(u)$ is the position vector which describes the path of an approximate Bezier polynomial function between P_0 and P_n

$$P(u) = \sum_{k=0}^{n} P_k \, BEZ_{k,n}(u) \qquad 0 \leq u \leq 1$$

$BEZ_{k,n}(u) = C(n,k)u^k(1-u)^{n-k}$ is the Bernstein Polynomial

where $C(n,k) = \dfrac{n!}{k!\,(n-k)!}$

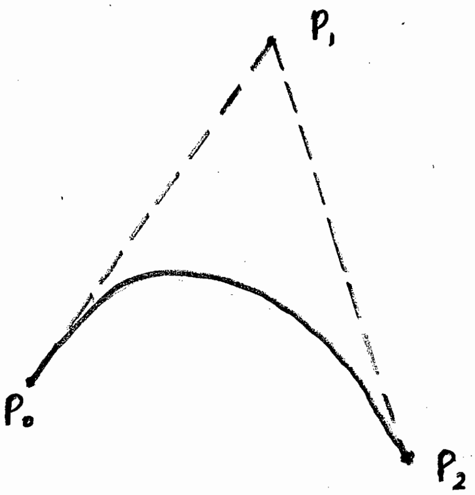
Set of equations for individual curve co-ordinates are -

$$x(u) = \sum_{k=0}^{n} x_k \, BEZ_{k,n}(u)$$

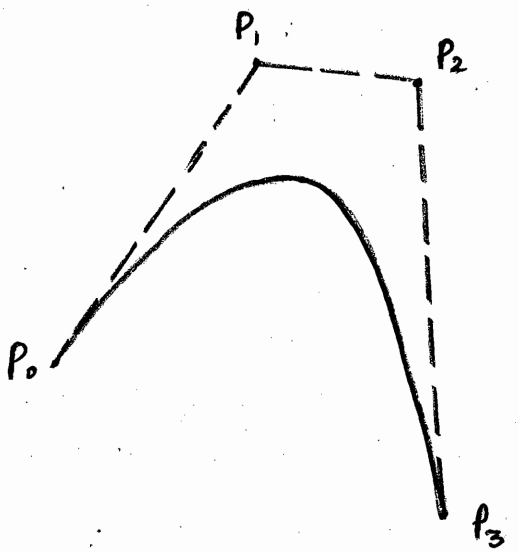
$$y(u) = \sum_{k=0}^{n} y_k \, BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k \, BEZ_{k,n}(u)$$

Bezier curves example :-



3 Control points



4 Control points

Properties :-

- Basic functions are real.
- Degree of polynomial defining the curve is one less than number of defining points.
- Curve generally follows the shape of defining polygon.

Shankar R
Asst Professor,
CSE, BMSIT&M

- Curve connects the first and last control points

  Thus $P(0) = p0$

  $P(1) = pn$

- Values for parametric first derivatives of a curve at endpoints are given by,

  $P(0) = -np_0 + np_1$

  $P(1) = -np_{o-1} + np_n$

- Second ~~derivates~~ derivatives at endpoints are given by

  $P''(0) = n(n-1)\,[(p_2 - p_1) - (p_1 - p_0)]$

  $P''(1) = n(n-1)\,[(p_{n-2} - p_{n-1}) - (p_{n-1} - p_n)]$

- Curve lies within the convex hull of the control points.

100: Explain Bezier surfaces & its equations

Ans: Two sets of orthogonal Bezier curves can be used to design an object surface. The parametric vector function for the Bezier surface is formed as the cartesian product of the Bezier blending functions

$$P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} p_{j,k} \, BEZ_{j,m}(v) \, BEZ_{k,n}(u)$$

with $p_{j,k}$ specifying the location of $(m+1)$ by $(n+1)$ control points.

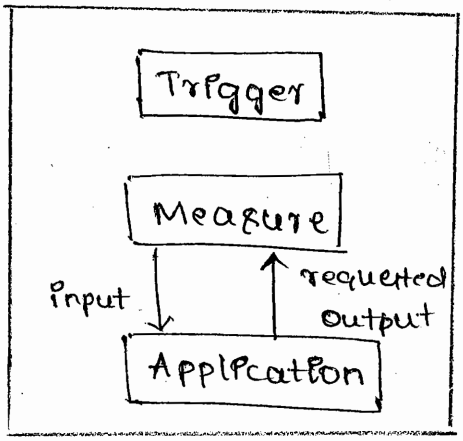
Bezier surfaces have the same properties as Bezier curves, and they provide a convenient method for interactive designs and applications. To specify the three-dimensional coordinate positions for the control points, we should first construct a regular rectangular grid in the x-y ground planee We choose elevations above the ground plane at grid intersections as the z coordinate values for the control points.

As with curves, a smooth transition from one section to the other is assured by establishing both zero order and first order continuity at the boundary line. ~~Zero~~ First order continuity is obtained by choosing control points along a straight line across the boundary. Zero order is maintained by choosing control points at the boundary.

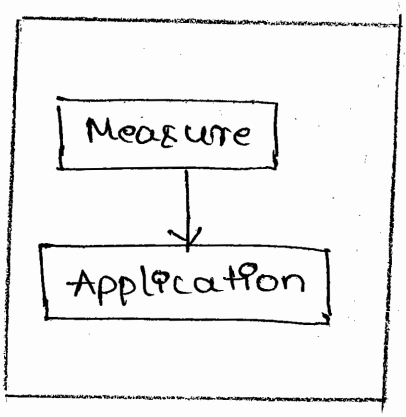
101: Explain request, sample and event input modes with a block diagram.

Ans:

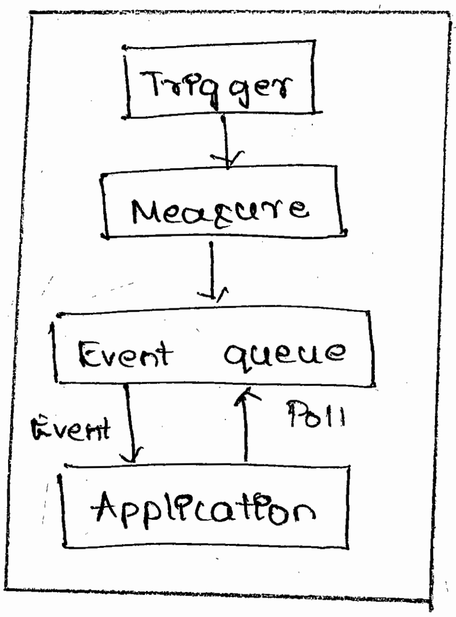
Request Mode Input



Sample mode input



Event mode Input



A program could request input at a particular time in the processing (request mode), or an input device could independently provide updated input (sample mode) or the device could independently store all collected data (event mode). In request mode: The application program initiates data entry, processing is suspended untill the required values are received. The program and i/p devices operate alternatively. Devices are put into a wait state untill the i/p request is made; then the program waits untill the data is delivered

In sample mode the application program and i/p devices operate independently. I/p devices may be operating at the same time that the program is processing other data. New values obtained replace previously i/p data values when the program requires new data it samples the ~~current~~ current values that have been stored from the device i/p.

Event Mode

In event mode the i/p devices initiate data i/p to the application program. the program and i/p devices operate concurrently but now the i/p devices deliver data to an input queue, called event queue. All i/p data is saved when the program requires new data it goes to the data queue.

2 Write the program snapshot, explain the creation of Menus in OpenGL

ANS) 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 in OpenGL :

1) Define the actions corresponding to each entry in the Menu.

2) Link the menu to a corresponding mouse button.

3) Register a call back function

for each entry in menu.

```
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

103) With the role of glCallList() function in creating Display lists in OpenGL.

ANS)

glCallList causes the named display list to be executed. The commands saved in the display lists are executed in order, just as if they were called without using a display list. If list has not been defined as the display list, glCallList is ignored.

glCallList can appear inside a display list. To avoid the possiblity of infinite recursion resulting in display list calling one another, a limit is placed on the nesting 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 ();
}
```

level of display lists during display list execution. The limit is atleast 64, and it depends on the Implementation.

GL state is not saved and restored across a call to glCalllist. Thus, changes made to GL state during the execution of a display list is completed

Specification

```
void glCallList(GLuint list);
```

104. Demonstrate the OpenGL visibility detection function.

We can apply both back-face removal and the depth-buffer visibility-testing method to our scenes using functions that are provided in the basic library of OpenGL.

OpenGL Polygon-Culling functions:

Back-face removal is accomplished with the functions

```
glEnable (GL_CULL_FACE);
glCullFace (mode);
```

where parameter mode is assigned the value GL-BACK. This function can also be used to remove front faces. To do this we can set the parameter mode to GL-FRONT, or we could change the definition of front facing polygons using the glFrontFace function. To eliminate all polygon surfaces in a scene, we set parameter mode to the openGL symbolic constant GL_FRONT_AND_BACK. By default, parameter mode in the glCullFace function has the value GL-BACK. Therefore, if we activate culling with the glEnable function without explicity invoking function glCullFace, the back faces in a scene will be removed. The culling routine is turned off with

```
glDisable (GL-CULL-FACE);
```

YASHASWINI K

Depth buffer values can then be initialized with

glClear (GL_DEPTH_BUFFER_BIT);

The depth ~~f~~ buffer is normally initialized with the refresh buffer to the background color. But the buffer has to be cleared each time we want to display a new frame. In OpenGL, depth values are normalized in the range from 0 to 1.0.

The openGL depth-buffer visibility-detection routines are activated with the following function.

glEnable (GL_DEPTH_TEST);

And, we deactivate the depth-buffer routines with

glDisable (GL_DEPTH_TEST);

We can also apply depth-buffer visibility testing using some other initialization for the maximum depth, and this initial value is chosen ~~to~~ with the openGL function:

glClearDepth (maxDepth);

parameter maxdepth can be set to any value between 0 and 1.0. To load the initialization value into the depth ~~x~~ buffer, we next must ~~include~~ invoke the glClear (GL_DEPTH_BUFFER_BIT) function. Projection coordinates in OpenGL are normalized to the range from -1 to 1.0, and the depth values between the near and far clipping planes are further normalized to the range 0.0 to 1.0. We can adjust these normalization values with

glDepthRange (~~nearDep~~ nearNormDepth, farNormDepth);

By default, nearNormDepth = 0.0 and farNormDepth = 1.0. Using the glDepthRange function, we can restrict the depth-buffer testing to any regional the view volume. Another option available in OpenGL is the test condition that is to be used in the depth-buffer routines. The following function can be used:

glDepthFunc (testCondition) ;

Parameter testCondition can be assigned any one of the following eigth symbolic constants:

GL-LESS, GL_GREATER, GL_EQUAL, GL_NOTEQUAL, GL-LEQUAL, GL_GEQUAL, GL_NEVER (no points are processed), GL_ALWAYS.

We can also set the status of the depth buffer so that it is in a read-only state or in a read-write state. This is accomplised with :

glDepthMask (writeStatus);

105. List and explain different input physical devices.

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

→ Keyboard devices

→ Pointing devices

*. 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.

*. MOUSE AND TRACKBALL : These are pointing devices used to specify the position. Mouse and trackball interacts 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. The values passed by the pointing devices can be considered as positions and converted to a 2D location in either screen or world co-ordinates. 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~~ 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 devices such as "photo cell". The light pen is held at th front of the CRT. When the electron beam strikes

phosphor, the light is emitted from the CRT. If it exceeds the threshold then the 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. Joystick is the variable sensitivity device.

*. SPACEBALL : It is a 3-dimensional input device. Stick doesnt move rather pressure sensors in the ball measure the forces applied by the user. The space ball can measure not only three direct forces but also three independent twists. So, totally device measures six independent values and thus has six degrees of freedom.

106. Define measure & trigger. List and Explain different input modes.

Ans:
measure: The measure of device is what device returns to the user program.

Trigger: trigger which can be used to send a signal to the Operating system.

Different input modes are:
Request mode.
Sample mode.
Event mode.

Request mode:

=> The application initiate data entry.
=> when input values are requested Processing is suspended until the required values are recived.
=> This input mode corresponds to the typical input operation in a general Programming language.
=> The program and input devices operate alternative devices are put into wait state until an input request is made; the program waits until the data are delivered.

Sample mode:

=> The application program and input devices operate independently.
=> input devices may be operating at the same time that the program is Processing other data.

⇒ New value obtained from the input device replace previously input data values.

⇒ when program requires newdata, it samples the current values that have been stored from the device input.

Event mode:

⇒ The input devices initiate data input to the application program.

⇒ The program and the input devices again operate concurrently, but now the input devices deliver data toan input queue also called an event queue.

⇒ All input data is saved.

⇒ when the program requires new data it goes to the data queue.

Typically, any numberof devices ~~again~~ can be operating at the same time in sample and event modes. Some can be operating in sample mode, while others are operating in even mode. But only on deviceat a time cam deliver input in request mode.

Other functions in the input library are used to specif physical devices the logical data classes.

107. write short notes on:

a. Client and Server

b. Display lists

c. Texts and Display lists

d. Fonts in GLUT.

a. Client and Server.

Networks and multiuser computing have changed this picture dramatically, and to such an extent that, even if we had a single-user isolated system, its software probably would be configured as a simple client-server network.

If computer graphics is to be useful for a variety of real applications, it must function well in a world of distributed computing and networks. In this world, our building blocks are entities called servers that can perform tasks for clients. Clients and servers can be distributed over a network or contained entirely within a single computational unit. Familiar examples of servers include print servers, which can allow sharing of a high-speed printer among users; compute servers, such as remotely located high-performance computers, accessible from user programs; file servers that allow users to share files and programs, regardless of the machine they are logged into; and terminal servers that handle dial-in access. Users and user programs that make use of these services are clients or client programs. Servers can also exist at a lower level of granularity within a single operating system.

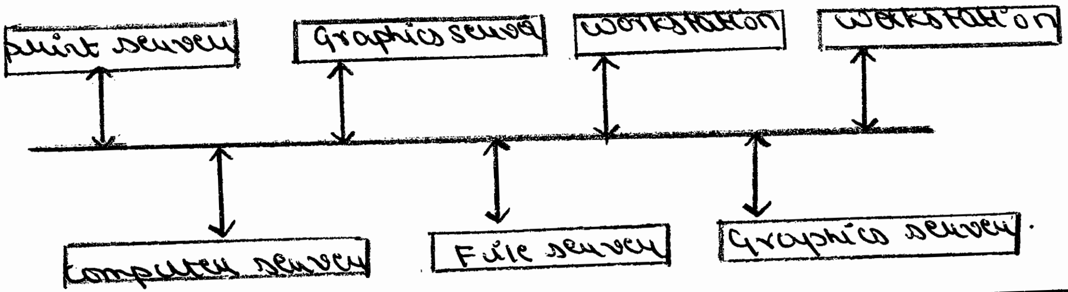
It is less obvious what we should call a workstation connected to the network: It can be both a client and a server, or perhaps more to the point, a workstation may run client programs and server programs concurrently.

The model that we use here was popularized by the X window System. We use much of that system's terminology which is now common to most window system and fits well with graphical applications.

A workstation with a raster displays, a keyboard and a pointing device, such as a mouse is a graphical server. The server can provide output services on its display and input services through the keyboard and pointing device. These services are potentially available to clients anywhere on the network.

Our OpenGL application programs are clients that use the graphics server. within an isolated system, this distinction may not be apparent as we write, compile, and run the software on a single machine. However, we also can run the same application program using other graphics servers on the network.

The network of client - server is shown below:



b. Display lists.

Display lists illustrate how we can use clients and servers on a network to improve interactive graphics performance. Display lists have their origin in the early days of computer graphics. The original architecture of a graphics system was based on a general - purpose computer (or host) connected to a display. The computer would send out the necessary information to redraw the display at a rate sufficient to avoid noticeable flicker. At that time (1960), computers were slow and expensive, so that cost of keeping even a simple display refreshed was prohibitive for all but a few applications.

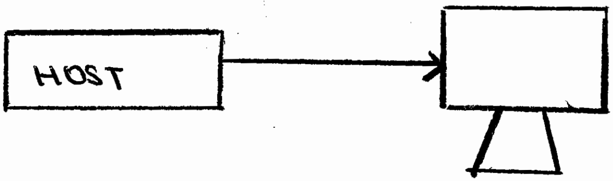
The solution to this promblem was to build a special - purpose computer, called a display processor, with an organization. The display processor, with an organization had a limited instruction set, most of which was oriented towards drawing primitives on the display. The user program was processed in the host computer, resulting in a compiled list of instructions that was then sent to the display processor, where the instructions were stored in a display memory as a display file or list. For an small or simple noninteractive application, once the display list was sent to the display processor, the host was free for other tasks, and the display lists was sent to the display processor, the host was free for other tasks, and the display list was and the processor would execute its display list repeatedly at a rate sufficient to avoid flicker. In addition to resolving the bottleneck due to burdening the host, the display processor introduced the advantages of special - purpose rendering hardware.

We can send graphical entities to a display lists on of 2 ways. We can send the complete description of our objects to the graphical server. For our typical geometric primitives, this transfer entails sending vertices, attributes and primitives types, in addition to viewing information

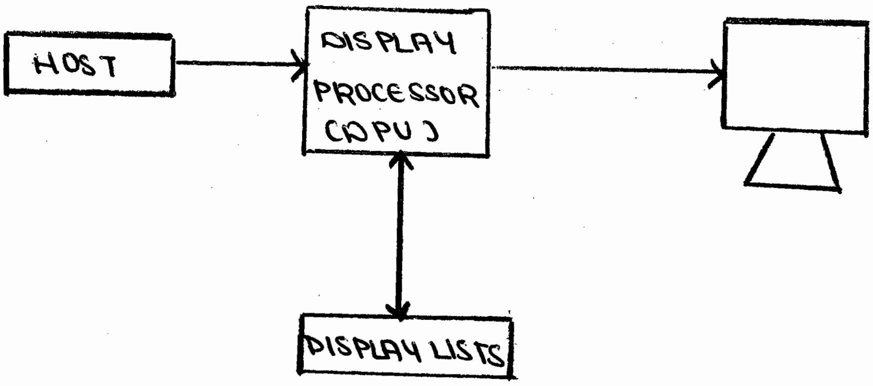
In our fundamental mode of operation, immediate mode as soon as the program executes a statement that defines a primitive, that primitive is sent to the server for possible display and no memory of it is retained in the system.

Display lists offer an alternative to this methode of operation called retained mode graphics. we define the object once, then put its description 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 server.

Disadvantages - It requires memory on the server, and there is the overhead of creating a display lists. Although this overhead is often offset by the efficiency to the execution of the display list, it might not be if the data are changing.



Simple Graphics architecture



Display - processor architecture.

c. Texts and Display lists.

The stroke and raster text, is used regardless of which type we choose to use, we need a reasonable amount of code to describe as set of characters.

Example - Suppose that we use a raster font in which each character is stored as an 8x13 pattern of bits. It takes 13 bytes to store each character. If we want to display a string by the most straight-forward method, we can send each characters to the server each time that we want it displayed. This transfer requires the movement of atleast 13 bytes/character. For application that display large quantities of text, sending each character to the display every time that it is needed can place a significant burden on our graphics system.

A more efficient strategy is to define the font once, using a display list for each character, and then to store the font on the server using these display lists. This solution is similar to what is done for bitmap fonts on std alphanumeric display terminals. The patters are stored in Rom in the terminals, and each character is selected and display based on a single byte: its ASCII code.

The basics of defining and displaying a character string (1 byte/char) using a stroke font and display list provide a simple but important eg of the use of display list in openGL. The procedure is essentially the same for a raster font. we can define a function OurFont(char c), which will draw any ASCII character c that can appear in our string. The function might have a form like the following

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

suppose that we are defining the letter 'O' and we wish it to fit in a unit square. The corresponding part of OurFont might be as follows

```
case 'O';
   glTranslatef(0.5, 0.5, 0.0) /* move to centre*/
   glBegin{GL_QUAD_STRIP};
   for(i=0; i<=12; i++)   /* 12 vertices*/
  { angle = 3.14159/6.0 * i;   /* 30 degree 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;
```

This code approx the circle with 12 quadrilaterals. Each will be filled according to the current state. Here, each character is defined in the plane $z = 0$ and we can use whatever co-ordinate system we wish to define our characters. The usual strategy is to start at the lower lower-left corner of the first character in the string to draw one character at a time, drawing each characters such that we end at the lower-right corner of the character's box, which is the lower-left corner of the successor's box.

Although our code is indegant, its efficiency is of little consequence because the characters ale generated only once and then we sent to the graphics server as a compiled display lists.

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

```
glListBase(base);
```

Finally, our drawing of a string is accomplished in the server by the function call

```
char *text_string
glCallLists((GLint) strlen(text_string), GL_BYTE, text_string);
```

which make use of the std C library function strlen to find the length of input string text_string. The 1st 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 2nd argument. The identifier of the kth display list executed is the sum of the list base and the value of the kth character in the array of characters.

d. Fonts in GLUT.

GLUT provides a few raster and stroke fonts. They don't make use of display lists; in the example, we create display lists to contain one of these GLUT fonts. We can access a single character from a monotype, or evenly spaced, font by the following function call:

glutStrokeCharacter(GLUT_STROKE_ROMAN, int character)

GLUT_STROKE_ROMAN provides proportionally spaced characters. You should use these fonts with caution. Their size (approx 120 units max) may have little to do with the units of the rest of your program; thus they may have to be scaled. We usually control the position of a character by using a translation before the character function is called. In addition, each invocation of glutStrokeCharacter includes a translation to the bottom right of the character's box, to prepare for the next character. Scaling and translation affect the OpenGL state, so here we should be careful to use glPushMatrix and glPopMatrix as necessary to prevent undesirable positioning of objects defined later in the program.

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 obtained and considerably simpler than the positioning of stroke characters is because bitmap characters are drawn directly in the frame buffer and are not subject to geometric transformations, whereas stroke characters are. OpenGL keeps, within its state a raster position. This identifies where the next raster primitive will be placed; it can be set using the glRasterPos*() function. The user program typically moves the raster position to the desired location before the 1st character in a string defined by glutBitMapCharacter is invoked. This change doesnt affect subseqent rendering of geometric primitives. If characters have different widths, we can use the glutBitMapWidth(font, char) function to return the width of a particular character. However the glut-BitmapCharacter function automatically advances the raster position, so typically we don't need to manipulate the raster position until we want to define a string of characters elsewhere on the display.

108. Explain briefly display lists and modelling.

The user program is processed by the host computer which results a compiled list of instructions that was then sent to the display processor, where the instruction are stored in a display memory called as "display file" or "display lists".

glNewList() at the beginning and glEndList() at the end is used to define a display list. Each display list must have a unique identifier.

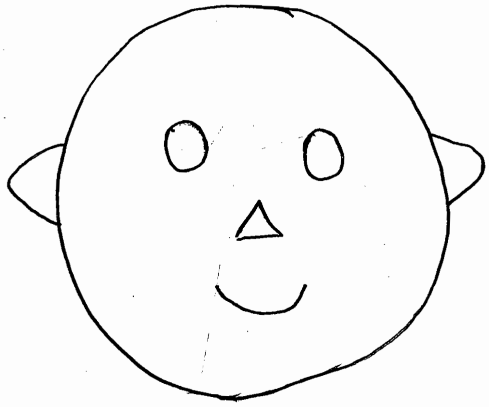
If we want an immediate display of the contents while the list is being constructed then GL_COMPILE_AND_EXECUTE flag is set.

Multiple lists with consecutive identifiers can be created more easily using glGenLists(number).

Multiple display lists can be displayed using glCallLists().

Display lists can call other display lists. Therefore, they are powerful tools for building hierarchical models that can incorporate relationships among parts of a model.

Consider a simple face modeling system that can produce images as follows:



SPOORTHY.M

Each face has two identical eyes, two identical ears, one nose, one mouth and an outline. we can specify these parts through display lists which is given below:

```
#define EYE 1
      glNewList (EYE);
        /*eye code*/
       glEndList ();
  // Similarly code for ears, nose, mouth, outline
   #define FACE 2
          glNewList (FACE);
     //code for outline
                glTranslatef(...);
                glCallList (EYE); //left-eye
                 glTranslatef (...);
                  glCallList (EYE); //right-eye
                  glTranslatef(...);
                   glCallList (NOSE);
        //similarly code for ears and mouth.
          glEndList ();
```

109. Explain menu creation and hierarchical menus with example code.

*The GLUT menu commands are placed in procedure main along with the other GLUT functions.

*A pop-up menu is created with the statement

```
glutCreateMenu (menuFcn);
```

where parameter menuFcn is the name of a procedure that is to be invoked when a menu entry is selected. This procedure has one argument which is the integer value corresponding to the position of a selected option.

```
void menuFcn(GLint menuItemNumber)
```

*The integer value passed to parameter menuItemNumber is then used by menuFcn to perform some operations. When a menu is created, it is associated with the current display window.

We must specify the options that are to be listed in the menu with a series of statements which have the general form:

```
glutAddMenuEntry (charString, menuItemNumber);
```

*Parameter charString specifies text that is to be displayed in the menu and menuItemNumber gives the location for that entry in the menu. The following statements creates a menu with two options:

```
glutCreateMenu (menuFcn);
    glutAddMenuEntry ("First Menu Item", 1);
    glutAddMenuEntry ("Second Menu Item", 2);
```

*Next we specify a mouse button that is to be used to select a menu option:

```
glutAttachMenu (button);
```

SPOORTHY.M

where parameter button is assigned one of the three GLUT symbolic constants referencing left, middle, or right mouse button.

* As each menu is created, it is assigned an integer identifier, starting with the value 1 for the first menu created. This identifier can be recorded as follows:

```
menuID = glutCreateMenu (menuFcn);
```

* To activate a menu for the current display window, we use glutSetMenu (menuID);

* We eliminate a menu with the command

```
glutDestroyMenu (menuID);
```

* To obtain the identifier for the current menu in the the current display window.

```
currentMenuID = glutGetMenu ( );
```

→ HIERARCHICAL MENUS:

* A submenu can be associated with a menu by first creating the submenu using glutCreateMenu, along with a list of suboptions and then listing the submenu as an additional option in the main menu. using a sequence of statements such as:

```
submenuID = glut CreateMenu (submenuFcn);
    glutAddMenuEntry ("First Submenu Item", 1);
    :
    :
glutCreateMenu (menuFcn);
    glutAddMenuEntry ("First Menu Item", 1);
    :
    :
    glutAddSubMenu ("SubMenu Option", submenuID);
```

The following program displays a submenu that provides three color choices (blue, green, white) for the first two vertices of the triangle.

```
#include <GL/glut.h>
  GLsizei winWidth = 400, winHeight=400;
   GLfloat red = 1.0, green= 1.0, blue =1.0;
   GLenum renderingMode = GL_SMOOTH;
void init (void)
 {
  glClearColor (0.6, 0.6, 0.6, 1.0);
  glMatrixMode (GL_PROJECTION);
   gluOrtho2D ( 0.0, 300.0, 0.0, 300.0);
 }
void mainMenu (GLint renderingOption)
{
 switch (renderingOption) {
   case1 : renderingMode = GL_FLAT; break;
   case2 : renderingMode = GL_SMOOTH; break;
  }
 glutPostRedisplay ( );
}
/* Set color values according to the submenu option
  selected. */
 void colorSubMenu(GLint colorOption)
{
 switch(colorOption) {
    case 1 :
       red = 0.0; green = 0.0; blue = 1.0;
       break;
     case 2 :
        red=0.0; green=1.0; blue = 0.0;
        break;
      case 3:
         red = 1.0; green = 1.0; blue =1.0;
    }
```

SPOORTHY.M.

```
glutPostRedisplay();
}
void displayTriangle(void)
 {
   glClear(GL_COLOR_BUFFER_BIT);
    glShadeModel(renderingMode);
     glColor3f(red, green, blue);
      glBegin(GL_TRIANGLES);
          glVertex2i(280,20);
            glVertex2i(160,280);
            ~~glVertex2f~~
             glColor3f(1.0, 0.0, 0.0); //set color of
                                    last vertex to red//
             glVertex2i(20,100);
        glEnd();
        glFlush();
    }
void reshapeFcn(GLint newWidth, GLint newHeight)
{
  glViewPort(0, 0, newWidth, newHeight);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
   gluOrtho2D(0.0, GLfloat(newWidth), 0.0, GLfloat(new
                                              Height));

   displayTriangle();
    glFlush();
 }
void main(int argc, char ** argv)
{
 GLint subMenu; // Identifier for submenu
  glutInit(&argc, argv);
   glutInditDisplayMode(GLUT_SINGLE|GLUT_RGB);
```

```
glutInitWindowPosition (200,200);
 glutInitWindowSize (winWidth, winHeight);
  glutCreateWindow ("Submenu Example");
  init();
   glutDisplayFunc (displayTriangle);
 Submenu = glutCreateMenu (colorSubMenu);
      glutAddMenuEntry ("Blue", 1);
       glutAddMenuEntry ("Green", 2);
       glutAddMenuEntry (" White", 3);
 glutCreateMenu (mainMenu); // create main pop-up menu
     glutAddMenuEntry ("Solid-color Fill", 1);
     glutAddMenuEntry ("color -Interpolation Fill", 2);
     ~~glutAddMenuEntry~~
      glutAddSubMenu ("color", subMenu); // Creating submenu under main menu.

   /* Select menu option using right mouse
            button */.
     glutAttachMenu (GLUT_RIGHT_BUTTON);
     glutReshapeFunc (reshapeFcn);
      glutMainLoop();
}
```

SPOORTHY.M.

Q. 110 Briefly explain different ways to overcome difficulty in picking

→ Picking is the logical input operation that allows the user to identify an object on the display. Although the action of picking uses the pointing devices, the information that the user wants to returned to the application program is not a position.

A pick device is considerably more difficult to implement on a modern system than is a ~~locator or~~. Locator.

Old display processors could accomplish picking easily by means of a lightpen. Each redisplay of the screen would start at a precise time The light pen would generate an interrupt with the time that the redisplay began, the processor could identify an exact place in the display list and subsequently could determine which object was being displayed.

One reason for the difficulty of picking in modern systems is the forward nature of rendering pipeline. Primitives are defined in an application program and move forward through sequence of geometric operations, rasterisation and fragment operations on their way to the frame buffer. But there is some difficulty as this process is reversible in a mathematical sense, hardware is not reversible.

This problem can be solved in 3 ways, One being the Selection. It involves adjusting the clipping region and viewport such that we can keep track of which primitives in a small clipping region are rendred into a region near the cursor.

These primitives go into a hit list that can be examined later by the user program. OpenGL supports this approach, and ~~we~~ there are 2 more simpler but less general strategies.

A simpler approach is to use bounding boxes, extents, for object of interest. The extent of an object is the smallest rectangle aligned with the co-ordinates axis, that contains the object. For 2-D applications, it is relatively easy to determine the rectangle in screen coordinates. But for 3-D applications, the bounding box is a right parallelpiped.

Another simple approach is the back buffer and an extra rendering. When we use double ~~f~~ buffering we use 2 color buffers: a front buffer and a back buffer. Since back buffer is not displayed, we can use it for purposes other than rendering the screen's scene. ~~that~~ Suppose, we render our objects into back buffer, each in a distinct color. The application programmer is free to determine an object's contents by simply changing colors wherever a new object definition appears in the program.

Q. What are the features a good interactive program must have?
III

→ A good interactive program must have the following features,

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 & displaying information.
4. An easy-to-use interface that doesnot 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.

The importance of these features and the difficulty of designing a good interactive program should never be underestimated.

112. With code snippet explain drawing erasable lines

* Mouse is used to get first end point and store this in object coordinates.

```
xm = x /500.0;
ym = (500-y) /500.0;
```

* Again mouse is used to get second point and draw a line segment in XOR mode.

```
xmm = x /500.0;
ymm = (500-y) /500.0;
glLogicOp(GL_XOR);
 glBegin(GL_LINES);
        glVertex2f(xm, ym);
        glVertex2f(xmm, ymm);
 glLogicOp(GL_COPY);
 glEnd();
glFlush();
```

In above code copy mode is used to switch back in order to draw other objects in normal mode. If we enter another point with mouse, we first draw line in XOR mode from 1st point to 2nd point and draw second line using 1st point to current point is as follows

```
glLogicOp(GL_XOR);
glBegin(GL_LINES);
        glVertex2f(xm, ym);
        glVertex2f(xmm, ymm);
 glEnd();
 glFlush();
xmm = x /500.0;
ymm = (500-y) /500.0;
 glBegin(GL_LINES);
        glVertex2f(xm, ym);
        glVertex2f(xmm, ymm);
  glEnd();
 glLogicOp(GL_COPY);
  glFlush();
```

Final form of code can be written as shown below:

```
        glLogicOp(GL-COPY);
        glBegin (GL-LINES);
              glVertex2f (xm,ym);
              glVertex2f (xmm,ymm);
        glEnd();
        glFlush();
        glLogicOp(GL-XOR);
```

In this example, we draw rectangle using same concept & code for callback function are given below

```
    float xm,ym,xmm,ymm;  /* corners of rectangle */
    int first = 0;        /* vertex the corner */
```

The callbacks are registered as follows:

```
     glutMouseFunc(mouse);
     glutMotionFunc(more);

     void mouse(int btn, int state, int x, int y)
     {    if (btn==GLUT_LEFT-BUTTON  &&  state == GLUT_DOWN)
          {  xm= x/500.0;
               ym= (500-y) /500.0;
               glColor3f ( 0.0, 0.0, 1.0);
               glLogicOp ( GL-XOR);
               first =0;
          }
          if (btn==GLUT_LEFT_BUTTON  &&  state == GLUT_UP)
          {  glRectf (xm,ym, xmm, ymm);
               glFush();
               glColor3f (0.0, 1.0, 0.0);
               glLogicOp(GL-COPY);
               xmm= x/500.0;
               ymm= (500-y) /500.0;
               glLogicOp (GL-COPY);
               glRectf (xm, ym, xmm, ymm);
               glFlush();
          }
     }
```

```
void move(int x, int y)
{    if (first ==1)
     {       glRectf (xm, ym, xmm, ymm);
             glFlush();

     }
     xmm= x/500.0;
     ymm= (500-y) /500;
     glRectf (xm, ym,xmm,ymm);
     glFlush();
     first =1;
}
```

For first time we draw rectangle in XOR mode. After that each time that we get vertex, we first erase existing rectangle by redrawing new rectangle using new vertex. Finally, when mouse button is released mouse callback is executed again which performs final erase and draw and go to replacement mode.

113. Explain with openGL functioning double buffering and timer.

Double buffering is a must in ~~see~~ animations where the primitives attributes and viewing condition are changing continously.

Double buffer consists of two buffers: front buffer and back buffer. Double buffering mode can be initialized

glutInitDisplayMode (GLUT_RGB | GLUT_DOUBLE);

Further in the display function, we have to include glutSwapBuffers() to exchange the contents of front and the back buffers.

~~USING~~ TIMER -

To understand the usage of timer, consider cube rotation program and its execution is done by using fast GPU.

The GLUT provides the following timer functions:

glutTimerFunc (int delay, void (* timer_func (int), int value)

Execution of this function starts timer in the event loop that delays for delay milliseconds. when timer has counted down, timer_func is executed the value parameter allow user to pass variable into the timer call back.

114 Write a note on quadratic surface wrt sphere, ellipse and torus.

A frequently used class of objects is the quadratic surfaces, which are described with second-degree equations.

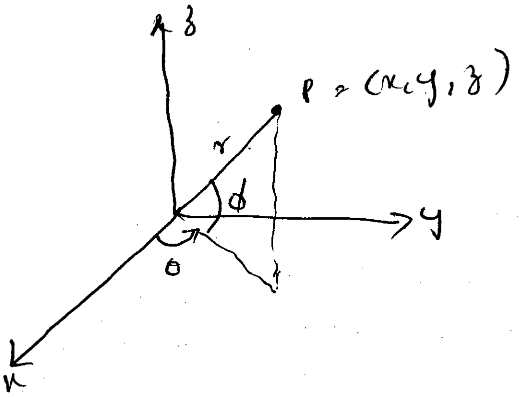
They include spheres, ellipsoids, tori. etc.

1) Sphere.

A spherical surface with radius r centered on the coordinate origin is defined as the set of points (x,y,z) that satisfy the equation

$$x^2+y^2+z^2=r^2.$$

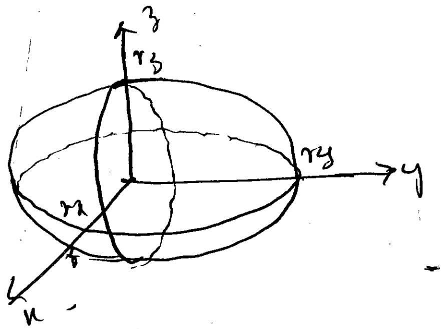
In parametric form,



$$x = r\cos\phi\cos\theta.$$
$$y = r\cos\phi\sin\theta.$$
$$z = r\sin\phi.$$

2) Ellipsoid

An ellipsoidal surface can be described as an extension of a spherical surface, where the radii in three mutually perpendicular directions can have different values.



The cartesian representation for points over the surface of an ellipsoid centered on the origin is

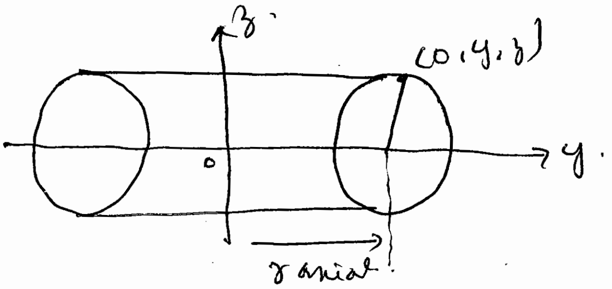
$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

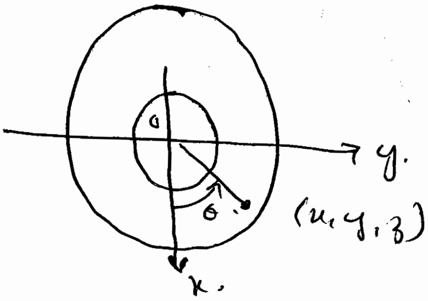
In parametric form,

$$x = r_x \cos\phi \cos\theta.$$
$$y = r_y \cos\phi \sin\theta.$$
$$z = r_z \sin\phi.$$

5) Torus.

A torus is a doughnut-shaped object, as shown





$$(y - r_{axial})^2 + z^2 = r^2.$$

Rotating this circle about the z-axis produces the torus whose surface positions are described with the cartesian equation

$$\left(\sqrt{x^2+y^2} - r_{axial}\right)^2 + z^2 = r^2.$$

In parametric form,

$$x = (r_{axial} + r\cos\phi)\cos\theta \quad , -\pi \le \phi \le \pi$$
$$y = (r_{axial} + r\cos\phi)\sin\theta \quad , -\pi \le \theta \le \pi.$$
$$z = r\sin\phi$$

$$\left(\frac{y - r_{axis}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1.$$

$$\left(\frac{\sqrt{x^2+y^2} - r_{axial}}{r_y}\right) + \left(\frac{z}{r_z}\right)^2 = 1.$$

$x = (r_{axial} + r_y \cos\phi)\cos\theta.$

$y = (r_{axial} + r_y \cos\phi)\sin\theta.$

$z = r_y \sin\phi.$

115 Explain with openGL functions to display sphere, cone, torus & teapot.

Sphere :

func^n :

glutWireSphere (r, nlongitudes, nLatitudes);

or

glut solidsphere (r, nlongitudes, nlatitudes);

where,

- r is the radius of sphere with double precision pt.
- nLongitude & nlatitudes is number of lines used to approximate the sphere.

Cone :

func^n :

glutWireCone (rBase, height, nlogitude, nlatitudes)

or

glutSolidCone (rBase, height, nlongitude, nlatitude);

where,

→ rBase is the radius of cone base

→ height is the height of cone

→ nLongitudes & nLatitudes are assigned integer values the specify the number of orthogonal surface lines for the quadrilateral mesh approximation.

Shankar R
Asst Professor,
CSE, BMSIT&M

Torus :

Funcⁿ :

```
glutWireTorus (rCrossSection, rAxial, nConcentrics,
               nRadialSlices);
    or
glutSolidTorus (rCrossSection, rAxial, nConcentrics,
               nRadialSlices);
```

where ,

→ rCrossSection radius about the coplanar z axis.

→ rAxial is the distance of the circle center from z-axis.

→ nConcentrics specifies the no of concentric circles.

→ nRadialSlices specifies the no of radial slices through torus surface.

Teapot :

funcⁿ :

```
glutWireTeapot (size);
    Or
glutSolidTeapot (size);
```

→ teapot is generated using Bézier curve funcⁿˢ

→ parameter size sets double-precision floating point value for the max radius the teapot bowl

→ teapot is centred on the world-coordinate origin co-ordinate origin with its vertical axis along y-axis

116. Explain GLUT Quadric-Surface Functions?

→ To generate a quadric surface using GLU functions

1. assign a name to the Quadric
2. activate the GLU quadric renderer &
3. designate values for the surface parameters.

→ The foll statements illustrate the basic seq of calls for displaying & frame sphere centered on the world-coordinate origin:

```
GLUquadricObj * sphere1;
Sphere1 = gluNewQuadric();
gluQuadricDrawStyle(sphere1, GLU-LINE);
gluSphere (sphere1, r, nLongitudes, nLatitudes);
```

where,

→ sphere1 is the name of the object.

→ the quadric renderer is activated with the gluNew Quadric function, then the display mode GLU_LINE is selected for sphere1 with the gluQuadricDrawStyle

→ Parameter r is assigned a double-precision value for the sphere radius.

→ nLongitudes and nLatitudes, number of lines.

the display modes available are:

GLU_POINT : quadric surface is displayed as plot:

GLU-SILHOUETTE : quadric surface displayed will not contain shared edges b/w 2 coplanar polygon facets.

GLU_FILL : quadric surface is displayed as patches of filled area.

→ To produce a view of cone, cylinder, cylinder we replace gluSphere func^n with

gluCylinder (quadricName, rBase, rTop, height, nLongitudes, nLatitudes);

→ The base of this object is in xy plane (z=0)

→ rBase is the base radius & rTop is Top radius.

→ if rTop=0 we get cone, rTop = rBase we get cylinder.

→ To get flat, circular ring is displayed in xy plane,

gluDisk (ringName, rInner, rOuter, nRadii, nRings);

→ double-precision values for an inner radius and an outer radius with parameters rInner & rOuter. If rInner=0, the disk is solid.

→ Otherwise, it is displayed with concentric hole in centre of disk.

→ The disk surface is divided into a set of facets with Integer parameters nRadii & nRings.

- gluPartialDisk (ringName, rInner, rOuter, nRadii, nRings, startAngle, sweepAngle);

- startAngle designates an angular position in degrees in the xy plane measured clockwise from +ve y axis

- parameter sweepAngle denotes an angular distance in degrees from startAngle position.

- Allocated memory for any GLU quadric surface can be reclaimed & the surface eliminated with

  gluDeleteQuadric (quadricName);

- to change front Back directions

  gluQuadricOrientation (quadricName, normalVector Direction);

Shankar R
Asst Professor,
CSE, BMSIT&M

117. Implement openGL program to display wried cone, wried cylender and wried sphere

```
#include <GL/glut.h>
GLsizei winWidth =500, winHeight = 500;
void init( void )
{
  glClearColor (1.0, 1.0, 1.0, 1.0);
}
void wireQuadsurfs ( void)
{
  glClear ( GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 0.0, 1.0);
  gluLookAt (2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);
  glPushMatrix ();
  glTranslatef (1.0, 1.0, 0.0);
  glutWireSphere (0.75, 8, 6);
  glPopMatrix ();
  glPushMatrix ();
  glTranslatef ( 1.0, -0.5, 0.5);
  glutWireCone (0.7, 2.0, 7.6);
  glPopMatrix ();
  GLUquadricObj . cylinder;
  glPushMatrix ();
  glTranslatef (0.0, 1.2, 0.8);
  cylinder - gluNewQuadric ();
  gluQuadricDrawStyle (cylinder, GLU_LINE);
  gluCylinder (cylinder, 0.0, 0.0, 1.5, 6.4);
  glPopMatrix ();
  glFlush();
}
```

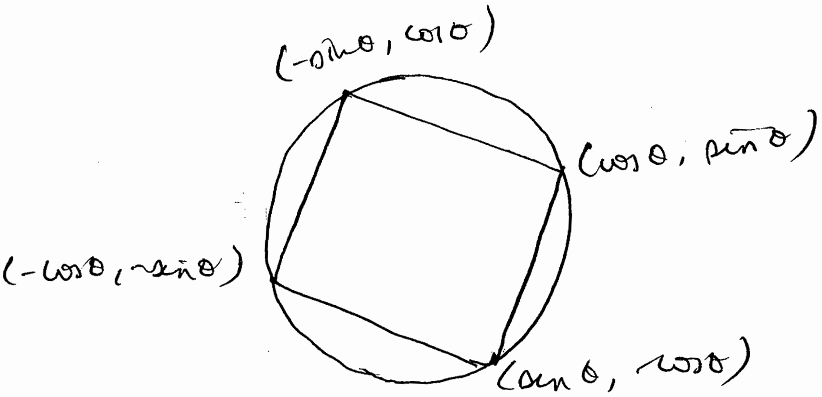
```
Void  winReshapeFunc( GLint newWidth,
                             GLint newHeight)
{
  glViewport ( 0, 0, newWidth, newHeight);
  glmatrixmode (GL_PROJECTION);
  glOrtho(-2.0,2.0, -2.0, 2.0, 0.0, 5.0);
   glmatrixmode (GL_MODELVIEW);
   glClear (GL_COLOR_BUFFER_BIT);

}

void  main (int argc, char ** argv)
{
  glutInit (&argc, argv);
  glutInitDisplaymode ( GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (100, 100);
  glutInitWindowSize ( winWidth, winHeight);
  glutCreateWindow (" Wire -Fram Quadratic surfaces");
  init ();
  glutDisplayFunc (wireQuadSurfs);
  glutReshapeFunc (winReshapeFcn);
  glutMainLoop ();
}.
```

118. Illustrate how an interactive program is animated.

Using Open GL, the programmar can design interactive programs. Programs in which objects are not static rather they appear to be moving or changing is consired as "Interactive programs". Consider the following diagram:



Consider a 2D point p(x,y) such that x=cosθ, y=sinθ. This point would lie on a unit circle regardless of the value of θ. Thus, if we connect the above given four points we get a square which has its center as the origin. The above square can be shown as.

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    thetar = theta / (3.14159/180.0);
    glVertex2f( cos(thetar), sin(thetar));
    glVertex2f(-sin(thetar), cos(thetar));
    glVertex2f(-cos(thetar), -sin(thetar));
  ~~glEn~~ glVertex2f(sin(thetar, -cos(thetar));
  glEnd();
}
```

```
void idle()
{
    theta += 2;
    if (theta >= 360.0)
         theta -= 360.0;
    glutPostRedisplay();
}.
```

The above Idle callback function must be registered in the main function:

glutIdleFunc(idle);

```
void mouse(int button, int state, int x, int y)
{
  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
   glutIdleFunc(idle);
  if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN);
    glutIdleFunc(idle);
}.
```

The above mouse callback function starts the rotation of the cube when the left mouse button and when the middle button is pressed it will halt.

It should be called in the main function as,

glutMouseFunc(mouse);

Shankar R
Asst Professor,
CSE, BMSIT&M

Q.119) Represent simple graphics and display processor architecture Explain two ways of sending graphical entities to a display and list the advantages and disadvantages.

Soln: The original architecture of a graphical system was based on a general-purpose computer connected to a display.

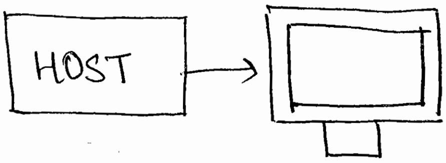


figure: Simple graphics Architecture

At that time, the disadvantage was that the system was slow and expensive.
Therefore, a special purpose computer has been built which is known as "Display Processor".

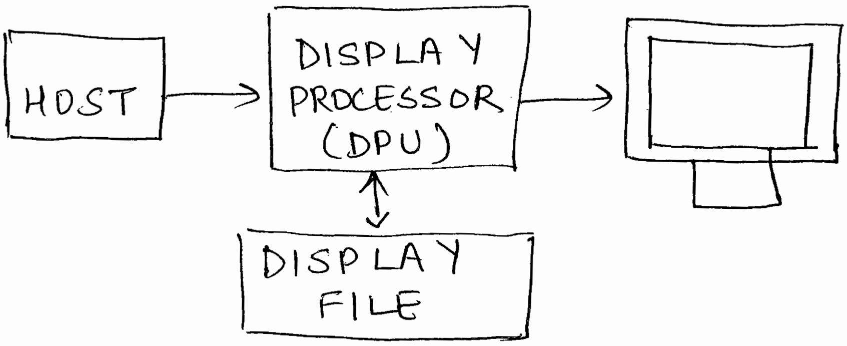


figure: Display processor Architecture

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 instructions 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 or ways of sending graphical entities to a display.

1> Immediate Mode

2> Retained Mode.

1> 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 ~~be~~ re-send the information. The information includes vertices, attributes, primitive types, viewing details.

2> Retained Mode - 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.

Advantages of display list

→ one time process

→ It can be shared with many clients.

→ minimizes data transmissions from client to server

→ reduces CPU cycles to perform actual data transfer.

Disadvantage of display list

→ The main disadvantage is it requiries memory at the server architecture and server efficiency decreases if the data is changing regularly.

Shankar R
Asst Professor,
CSE, BMSIT&M

Q120) Discuss the following logical operations with suitable examples.
a) Copy mode (b) Exclusive OR mode
c) Rubber-band effect (d) drawing erasable lines

Soln:

Two types of functions that define writing modes are:
1> Replacement mode
2> Exclusive OR (XOR)

a) Copy mode –

When a program specifies about visible primitive then OpenGL renders it into set of color pixels and stores it in the present drawing buffer.
In case of default mode, consider we start with a color buffer that has to be cleared to black. Later, we draw a blue color rectangle of size 10×10 pixels then 100 blue pixels are copied into the color buffer, replacing 100 black pixels. Therefore, this mode is called as "copy or replacement mode".

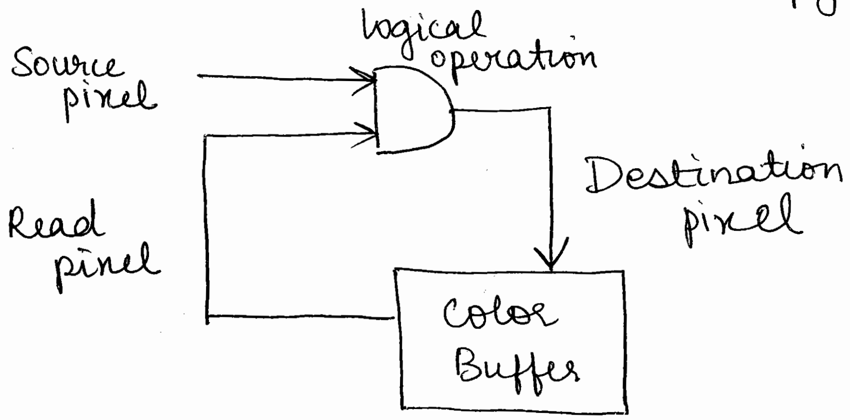


figure: Pixel writing model.

The pixel that we want to write is called as "source pixel".

The pixel in the drawing buffer which gets replaced by source pixel is called as "destination pixel".

b) Exclusive OR or (XOR) mode - In this mode, the corresponding bits in each pixel are combined using XOR logical operation

If s and d are corresponding bits in the source and destination pixels, we can denote the new destination bit as d', $\boxed{d' = d \oplus s}$.

One special property of XOR is that if we apply it twice, it returns the original state.

$$\boxed{d = (d \oplus s) \oplus s}$$

c) Rubber-band effect -

It is a technique used to define the elastic nature of pointing device to draw primitives.

Consider a paint application, if we want to draw a line, we indicate only two end points of our desired line segment.

Rubber band effect begins when mouse button is pressed and continues till button is released at the time final line segment is drawn.

d) Drawing Erasable Lines -

Mouse is used to get first end point and store this in object coordinates.

$$xm = x/500;$$
$$ym = (500-y)/500;$$

Again mouse is used to get second point and draw the line 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 ();
```

Copy mode is used to switch back in order to draw other objects in normal mode.
If we enter another point with mouse, we first draw line in XOR mode.

```
      glLogicOp (GL-XOR);
      glBegin (GL-LINES);
          glVertex2f (xm, ym);
          glVertex2f (xmm, ymm);
      glEnd ();
       glFlush ();
      xmm = x/500.0;
      ymm = (500-y)/(500.0);
      glBegin (GL-LINES);
          glVertex2f (xm ,ym);
          glVertex2f (xmm, ymm);
      glEnd ();
```

```
glLogicOp(GL_COPY);
glFlush();
```

Final form of code can be written as shown below.

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