Experiment No: 01

Experiment Name: Write a program to draw some points.

Objective: The objective of this experiment is to write a program to draw some points using DDA line algorithm.

Theory: The digital differential analyzer (DDA) is an incremental scan conversion method. Suppose at step i we have calculated (x_i, y_i) to be a point on the line. Since that the next point (x_{i+1}, y_{i+1}) should satisfy $\Delta y / \Delta x = m$ where $\Delta y = y_{i+1} - y_i$ & $\Delta x = x_{i+1} - x_i$. We have,

$$y_{i+1} = y_i + m\Delta x$$

or, $$x_{i+1} = x_i + \Delta y / m$$

Algorithm:

Step 1 : Read the two end points $P_1(x_1, y_1)$ & $P_2(x_2, y_2)$

Step 2 : $dx = x_2 - x_1$ & $dy = y_2 - y_1$.

Step 3 : if abs $(dx) >$ abs (dy) then $K =$ abs (dx)
else $K =$ abs (dy)

Step 4 : $\nabla x = \frac{dx}{K}$ & $\nabla y = \frac{dy}{K}$

Step 5 : Initialize $x = x_1$ & $y = y_1$

Step 6 : Display pixel (x, y). $x = x + \nabla x$ & $y = y + \nabla y$

Step 7 : Display pixel round (x) & round (y).

Step 8 : Repeat step 6 & step 7 K times.

Step 9 : Stop.

Source Code:

```
#include <window.h>
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
void myInit (void){
glClearColor(1.0, 1.0, 1.0, 0.0);
```

```
glColor3f(0.0f, 0.0f, 0.0f);
glPointSize (4.0);
glLineWidth (4.0);
glMatrixMode (GL_PROJECTION);
glLoadIdentity();
glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
gluOrtho2D (0.0, 640.0, 0.0, 480.0); }
void myDisplay (void)}
glClear (GL_COLOR_BUFFER_BIT);
glPointSize (20.0);
glBegin (GL_POINTS); {
glEnd ();
glFlush (); }
int main (int argc, char** argv)}
glutInit(&argc, argv); }
glVertex2i (20,10);
glVertex2i (50,10);
glVertex2i (20,80);
glVertex2i (50,80); } }
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (640, 480);
glutInitWindowPosition (100, 150);
glutCreateWindow ("My first window");
glutDisplayFunc (myDisplay);
myInit ();
glutMainLoop ();
return 1;
}
```

Experiment No: 02

Experiment Name: Write a program to draw line using slope intersect method.

Objective: The objective of this experiment is to write a program to draw line using slope intersect method.

Theory: To graph a linear equation an slope intersect form we can use the information given by that forms.

Algorithm :

Step 1 : Read two end points $P_1(x_1, y_1)$ & $P_2(x_2, y_2)$

Step 2 : $dx = x_2 - x_1$ & $dy = y_2 - y_1$, $m = dy/dx$.

Step 3 : Set (x, y) to starting point if $dx > 0$ then $x = x_1$ and $y = y_1$ and $x_{end} = x_2$, if $dx < 0$ then $x = x_2$, $y = y_2$ and $x_{end} = x_1$

Step 4 : $c = y - mx$

Step 5 : Plot a point current (x, y) co-ordinates.

Step 6 : Increment the value of x; $x = x + 1$

Step 7 : Compute the value of y; $y = mx + c$.

Step 8 : Stop

Source code:

```
#include <windows.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
void myInit (void)
{ glClearColor (1.0, 1.0, 1.0, 0.0);
  glColor3f (0.0f, 0.0f, 255.0f);
  glPointSize (4.0);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
```

```
glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
gluOrtho2D (0.0, 640.0, 0.0, 480.0); }
void myDisplay (void) {
glClear(GL_Color-Buffer_BIT);
glPointsize (2.0);
GLint x,y, x1 = 150, x2= 100, y1 = 100, y2= 200);
gldouble xf, m, b;
x = x1 ; y = y1;
m=(y2 - y1) / (x2 - x1);
b = y1 - m * x1;
glBegin (GL_POINTS);}
glvertex2i(x,y); }
glEnd ();
while (y<y2) {
    y++;
    xf = (y-b) /m;
    x = floor (xf +0.5);
    glBegin (GL_POINTS); }
  glvertex2i (x,y); }
      glEnd (); }
glFlush(); }
int main (int argc, char** argv) {
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
 glutInitWindowSize (640, 480);
 glutInitWindowPosition (100, 150);
 glutCreateWindow ("My first window");
 glutDisplayFunc(myDisplay);
 myInit ();
 glutMainLoop ();
 return 1;
}
```

Experiment No: 03

Experiment Name: Write a program to draw a line using Brashenham Algorithm.

Objective: The objective of this program to draw a line using Brashenham Algorithm.

Theory: Brashenham line algorithm is a highly efficient incremental method for scan converting lines. We start with pixel $P1(x_1', y_1')$ then select subsequent pixel as we work our way to the right, one pixel position at a time in the horizontal direction towards $P_2'(x_2', y_2')$.

Algorithm:

Step 1 : Read two endpoints $P_1(x_1, y_1)$ & $P_2(x_2, y_2)$

Step 2 : Calculate $dx = x_2 - x_1$ & $dy = y_2 - y_1$.

Step 3 : Calculate decision parameter, P. $P = 2dy - dx$.

Step 4 : Set initial point $x = x_1$, $y = y_1$ and $i = 0$

Step 5 : Now plot point (x, y). if $P < 0$ then $x = x+1$, $y = y$, $P = P + 2dy$. Otherwise, $x = x+1$, $y = y+1$, $P = P + 2dy - 2dx$

Step 6 : Repeat step 5 until $i \leq dx$.

Source Code:

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```

```
const int screenwidth = 640;
const int screenheight = 480;
GLdouble A, B, C, D;
void myInit (void){
glClearColor (1.0, 1.0, 1.0, 0.0);
glColor3f (0.0f, 0.0f, 0.0f);
glPointSize (4.0);
glLineWidth (4.0);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
A= screenwidth /4.0;
B =0.0;
C = D =screenHeight /2.0;
gluOrtho2D (0.0, 640.0, 0.0, 480.0); }
void myDisplay (void){
glClear (GL_COLOR_BUFFER_BIT);
glPointSize (2.0);
GLint (x, y, x1 = 50, x2 = 100, y1 =100, y2=200);
GLdouble xf, m, b;
x = x1 ; y = y1;
m = (y2 - y1)/ (x2 - x1);
b = y1 - m * x1;
glBegin (GL_POINTS);}
glVertex2i (x,y); }
glEnd ();
while (y< y2){
  y++;
xf= (y-b)/m;
x = floor (xf +0.5);
glBegin (GL_POINTS); }
```

```
glVertex2i (x,y);
}
glEnd ();
}
glFlush ();
}
int main (int argc, char** argv){
glutInit (&argc, argc);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (640, 480);
glutInitWindowPosition (100, 150);
glutCreatWindow ("My first window");
glutDisplayFunc (myDisplay);
myInit ();
glutMainLoop ();
return 1;
}
```

Experiment NO:04

Experiment Name: Write a program to show translate transformation.

Objective: The objective of this experiment is to a program to show triamslate transformation.

Theory: In translation, an object in displaced a given distance and direction from its original position. If the displacement is given by the vector $v = t_x I + t_y I$, the new object point $P'(x', y')$ can be found by applying the transformation T_y to $P(x, y)$. $P' = T_y(P)$; where, $x' = x + t_x$ & $y' = y + t_y$.

Source Code:

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
void init (void)
{
glClearColor (0.0, 0.0, 0.0, 0.0);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho (-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
}
void draw_triangle (void)
{
glBegin (GL_LINE_LOOP);
glVertex2f (0.0, 25.0);
glVertex2f (25.0, -25.0);
glVertex2f (-25.0, -25.0);
glEnd ();
}
```

```
void display (void)
{
 glClear (GL_COLOR_BUFFER_BIT);
 glColor3f (1.0, 1.0, 1.0);
 glLoadIdentity ();
 glColor3f (1.0, 1.0, 1.0);
 draw_triangle ();
 glEnable (GL_LINE_STIPPLE);
 glLineStipple (1, 0x0F0F);
 glLoadIdentity ();
 glTranslatef (-20.0, 0.0, 0.0);
 draw_triangle ();
 glFlush ();
}
int main (int argc, char **argv)
{
 glutInit (&argc, argv);
 glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
 glutInitWindowSize (500, 500);
 glutInitWindowPosition (100, 100);
 glutCreateWindow (argv[0]);
 init ();
 glutDisplayFunc (display);
 glutMainLoop ();
 return 0;
}
```

Experiment No: 05

Experiment Name: Write a program to show rotation transformation.

Objective: The objective of this experiment in to a program to show rotation transformation.

Theory: In rotation, the object is rotated θ° about the origin. The convention in that the direction of rotation is counterclockwise if θ is a positive angle & clockwise if θ is a negative angle. The transformation of rotation is -

$$P' = R_\theta(P)$$

where, $x' = x\cos(\theta) - y\sin(\theta)$ & $y' = x\sin(\theta) + y\cos(\theta)$

Source code:

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
  glMatrixMode(GL_MODEVIEW);
  glLoadIdentity();
}
void draw_triangle(void)
{
  glBegin(GL_LINE_LOOP);
  glVertex2f(0.0, 25.0);
  glVertex2f(25.0, -25.0);
  glVertex2f(-25.0, -25.0);
  glEnd();
}
```

```
void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(1.0, 1.0, 1.0);
 glLoadIdentity();
 glColor3f (1.0, 1.0, 1.0);
 draw_triangle();
glEnable (GL_LINE_STIPPLE);
glLineStipple (1, 0x8888);
glLoadIdentity();
gl Rotatef (90.0, 0.0, 0.0, 1.0);
draw_triangle();
glDisable (GL_LINE_STIPPLE);
glFlush();
}
~~glutInit~~
main (int argc, char** argv)
{ glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitwindow position (100, 100);
  glutCreatewindow (argv[0]);
  init();
  glutDisplayFunc (display);
  glutMainLoop();
  return 0;
}
```

Experiment NO: 06

Experiment Name: Write a program to draw a circle using Breshenham algorithm.

Objective: The objective of this experiment is to draw a circle using Breshenham algorithm.

Theory: We will use Breshenham's circle algorithm for Calculation of the locations of the pixels in the first octant of 45 degrees. It assumes that the circle is centered on the origin.

Algorithm:

Step 1 : Here, $x^2 + y^2 = r^2$

Step 2 : Decision parameter, $P = 3 - 2r$

Step 3 : if $P < 0$, then $x_{i+1} = x_i + 1$,

$$y_{i+1} = y_i$$

$$P_{i+1} = P_i + 4(x_{i+1}) + 6$$

Otherwise $x_{i+1} = x_i + 1$,

$$y_{i+1} = y_i - 1,$$

$$P_{i+1} = P_i + 4(x_{i+1} - y_{i+1}) + 10$$

Source Code:

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
void myInit (void)
{
```

```
glClearColor (1.0, 1.0, 1.0, 0.0);
glColor3f (0.0f, 0.0f, 255.0f);
glPointSize (4.0);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluOrtho2D (0.0, 640.0, 0.0, 480.0);
}
void myDisplay (void)
{
glClear (GL_COLOR_BUFFER_BIT);
glPonitionsize (2.0);
GLint h=200, k=200, x, y, r, d;
y = r = 100;
d = 3 - 2*r;
for (x=0; x<=y; x++) {
    glBegin (GL_POINTS); }
glvertex2i (x+h, y+k);
glvertex2i (x+h, k-y);
glvertex2i (y+h, x+k);
glvertex2i (y+h, k-x);
glvertex2i (-y+h, x+k);
glvertex2i (-y+h, k-x);
glvertex2i (-x+h, y+k);
glvertex2i (-x+h, k-y); }
glEnd ();
    if (d<0)
        d = d + 4*x + 6;
    else { d = d + 4*(x-y)+10; } }
glFlush ();
}
int main (int argc, char** argv)
{ glutInit (& argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
```

```
glutInitWindowSize (640, 480);
glutInitWindowPosition (100, 150);
glutCreateWindow (my first Display);
glutDisplayFunc (myDisplay);

myInit ();
glutMainLoop ();

return 1;

}
```

Experiment No: 07

Experiment Name: Write a program to show scaling transformation.

Objective: The objective of this experiment is to show scaling transformation.

Theory: Scaling is the process of expanding or Compressing the dimensions of an object. Positive scaling constants S_x and S_y are used to describe changes in length with respect to the x-direction & y-direction, respectively. Scaling transformation $S_{sx=sy}$ is given by $P' = S_{sx=sy}(P)$ where $x' = S_x x$ and $y' = x_y y$.

Source Code:

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
void init (void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
 glMatrixMode (GL_PROJECTION);
 glLoadIdentity ();
 glOrtho (-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
 glMatrixMode (GL_MODELVIEW);
 glLoadIdentity ();
}
void draw_triangle (void)
   glVertex (GL_LINE_LOOP);
   glVertex2f (0.0, 25.0);
   glVertex2f (25.0, -25.0);
   glVertex2f (-25.0, -25.0);
 glEnd ();
```

```
void display (void)
{ glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);
  glLoadIdentity ();
  glColor3f (1.0, 1.0, 1.0);
  draw_triangle ();
glEnable (GL_LINE_STIPPLE);
glLineStipple (1, 0xF0F0);
glLoadIdentity ();
glScalef (1.0, 0.5, 1.0);
draw_triangle ();
glDisable (GL_LINE_STIPPLE);
glFlush ();
}
int main (int argc, char** argv)
{ glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreationWindow (argv [0]);
  init ();
  glutDisplayFunc (display);
  glutMainLoop ();
  return 0;
}
```

Experiment No: 08

Experiment Name: Write a program to draw a circle inside square using mouse operation.

Objective: The objective of this experiment to draw a circle inside square using mouse operation.

Theory: In this algorithm, we divide the line clipping process two phases: (1) identify those lines which intersect the clipping window & so need to be clipped (2) perform the clipping.

Source code:

```
#include <windows.h>
#include <GL/glut.h>
void myInit (void)
{ glClearColor (1.0, 1.0, 1.0, 1.0);
  glColor3f (0.0f, 0.0f, 0.0f);
  glPointSize (4.0);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
  gluOrtho2D (0.0, 640, 0.0, 480.0);
}
void myMouse (int button, int state, int x, int y)
{ if (button == GLUT_LEFT_BUTTON &&
      state == GLUT_DOWN)
  {   glColor4f (0.0, 1.0, 0.0, 1.0);
      glRecti (200, 150, 500, 330);
      glFlush ();
  }
else if (button == GLUT_RIGHT_BUTTON &&
      state == GLUT_DOWN)
```

```
}
  glColor3f (1.0,0.0,0.0);
  GLint h= 350, k = 240, x, y, r, d;
  y= r= 60;
  d = 3 - 2*r;
  for (x=0; x<=y ; x++)
}
  glBegin (GL_POINTS);
  {
    glVertex2i (x+h, y+k);
    glVertex2i (x+h, k-y);
    glVertex2i (y+h, x+k);
    glVertex2i (y+h, k-x);
    glVertex2i (-y+h, x+k);
    glVertex2i (-y+h, k-x);
    glVertex2i (-x+h, y+k);
    glVertex2i (-x+h, k-y);
  }
  glEnd ();
  if (d<0)
      d = d + 4* x + 6;
    else
      } d = d + 4* (x-y)+10;
        y--;
      }
    }
  glFlush();
}}
void myDisplay (void)
  } glClear (GL_COLOR_BUFFER_BIT);
    glFlush();
}
int main (int argc, char** argv)
}
```

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(640, 480);
glutInitWindowPosition(100, 150);
glutCreateWindow("Mouse Operation");
glutDisplayFunc(myDisplay);
glutMouseFunc(myMouse);

myInit();
glutMainLoop();

return 1;

}
```

Experiment No: 09

Experiment Name: Write a program to show arm motion.

Objective: The objective of this experiment is to show arm motion.

Theory: Arm movements include reach patterns that have minimal movement changes at the elbow, forearm or wrist. These patterns require that the hand direct the task – distal initiation – while the shoulder pattern provides appropriate movement for stability or assist hand placement.

Source Code:

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
static int shoulder = 0, elbow = 0;
void init ()
{ glClearColor (0.0, 0.0, 0.0, 0.0);
} void display (void)
{  glClear (GL_COLOR_BUFFER_BIT);
   glPushMatrix ();
   glTranslatef (-1.0, 0.0, 0.0);
   glRotatef ((GLfloat) shoulder, 0.0, 0.0, 1.0);
   glTranslatef (1.0, 0.0, 0.0);
   glPushMatrix ();
   glScalef (2.0, 0.4, 1.0);
   glutWireCube (1.0);
   glPopMatrix ();
```

```
glPopMatrix ();
glutSwapBuffers();
}
void reshape (int w, int h)
{ glViewport (0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(65.0, (GLfloat)w / gluPerspective
        (GLfloat )h, 1.0, 20.0);
  glMatrixMode (GL_MODELVIEW);
  glLoadIdentity ();
  glTranslatef (1.0, 0.0, -5.0);
}
void keyboard (unsigned char key, int x, int y)
{
  switch (key)
  { case 's':
      shoulder = (shoulder +5) % 360;
      glutPostRedisplay ();
      break;
    case 'S':
      shoulder = (shoulder -5) % 360;
      glutPostRedisplay ();
      break;
    case 'e':
      elbow = (elbow +5) % 360;
      glutPostRedisplay();
      break;
    case 'E':
      elbow = (elbow -5) % 360;
      glutPostRedisplay();
      break;
```

```
case 27 :
    exit(0);
    break;
  default :
    break;
}
}

int main (int argc , char ** argv)
{
    glutInit (&argc , argv);
    glutInitDisplayMode (GLUT_SINGLE |
                GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv [0]);
    init ();
    glutDisplayFunc (display);
    glutReshapeFunc ( reshape);
    glutKeyboardFunc ( keyboard);
    glutMainLoop ()
    return 0;
}
```