**Labset 1:Write a program to demonstrate the following OpenGL primitives with either RGB or indexed colors(as suggested by examiner): (CO1, PO3, 5)**

**i) GL\_POINTS ii)GL\_LINES ii)GL\_LINE\_STRIP iii)GL\_LINE\_LOOP iv)GL\_TRIANGLES v)GL\_TRIANGLE\_STRIP vi)GL\_TRIANGLE\_FAN vii)GL\_QUADS viii) GL\_QUAD\_STRIP ix)GL\_POLYGON**

#include <stdio.h>

#include <GL/glut.h>

int ch;

void drawPoints()

{

glColor3f(1.0,0.0,0.0);

glPointSize(3.0);

glBegin(GL\_POINTS);

glVertex2f(0.0,0.0);

glVertex2f(0.8,0.8);

glVertex2f(-0.7,-0.7);

glVertex2f(-0.2,-0.2);

glEnd();

}

void drawLines()

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2f(0.0,0.0);

glVertex2f(0.8,0.8);

glVertex2f(-0.7,-0.7);

glVertex2f(-0.2,-0.2);

glEnd();

}

void drawLineStrip()

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINE\_STRIP);

glVertex2f(0.0,0.0);

glVertex2f(0.8,0.8);

glVertex2f(-0.5,-0.7);

glVertex2f(-0.2,-0.8);

glEnd();

}

void drawLineLoop()

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(0.0,0.0);

glVertex2f(0.8,0.8);

glVertex2f(-0.5,-0.7);

glVertex2f(-0.2,-0.8);

glEnd();

}

void drawTriangles()

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_TRIANGLES);

glVertex2f(0.0,0.0);

glVertex2f(0.8,0.8);

glVertex2f(0.3,0.5);

glVertex2f(-0.2,-0.1);

glVertex2f(-0.5,-0.7);

glVertex2f(-0.2,-0.8);

glEnd();

}

void drawTriangle\_Strip()

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_TRIANGLE\_STRIP);

glVertex2f(-0.8,0.0);

glVertex2f(-0.5,0.5);

glVertex2f(-0.2,0.0);

glVertex2f(0.0, 0.5);

glVertex2f(0.2,0.0);

glEnd();

}

void drawTriangle\_Fan()

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_TRIANGLE\_FAN);

glVertex2f(0.0,0.0);

glVertex2f(0.2,0.8);

glVertex2f(0.4,0.6);

glVertex2f(0.6, 0.4);

glVertex2f(0.8,0.2);

glEnd();

}

void drawQuads()

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_QUADS);

glVertex2f(0.0,0.0);

glVertex2f(0.2,0.8);

glVertex2f(0.4,0.6);

glVertex2f(0.6, 0.4);

glVertex2f(0.8,0.2);

glEnd();

}

void drawQuad\_Strip()

{

glBegin(GL\_QUAD\_STRIP);

glVertex2f(0.0,0.0);

glVertex2f(0.2,0.8);

glVertex2f(0.4,0.6);

glVertex2f(0.6, 0.4);

glVertex2f(0.8,0.2);

glVertex2f(-0.2,0.2);

glEnd();

}

void drawPolygon()

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(0.0,-0.6);

glVertex2f(-0.2,0.3);

glVertex2f(0.2,0.3);

glVertex2f(0.0,0.6);

glVertex2f(0.2,-0.3);

glVertex2f(-0.2,-0.3);

glEnd();

}

void display()

{

while(1){

printf("Enter\n 1:GL\_POINTS\n2:GL\_LINES\n3:GL\_LINE\_STRIP\n4:GL\_LINE\_LOOP\n");

printf("5:GL\_TRIANGLES \n6:GL\_TRIANGLE\_STRIP\n7:GL\_TRIANGLE\_FAN\n8:GL\_QUADS\n");

printf("9:GL\_QUAD\_STRIP\n10:GL\_POLYGON\n11:exit\n");

scanf("%d",&ch);

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(1.0,1.0,1.0,0.0);

switch(ch)

{

case 1:drawPoints();

glFlush();

break;

case 2: drawLines();

glFlush();

break;

case 3: drawLineStrip();

glFlush();

break;

case 4: drawLineLoop();

glFlush();

break;

case 5: drawTriangles();

glFlush();

break;

case 6: drawTriangle\_Strip();

glFlush();

break;

case 7: drawTriangle\_Fan();

glFlush();

break;

case 8: drawQuads();

glFlush();

break;

case 9:drawQuad\_Strip();

glFlush();

break;

case 10:drawPolygon();

glFlush();

break;

case 11: exit(0);

}

}

}

int main(int argc, char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutCreateWindow("First Program");

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Labset 2:Write a program to demonstrate the two views, orthographic view and perspective view, of OpenGL by consider a 2D and a 3D with any two primitives.(Dimensions are as suggested by the examiner). (CO1, PO3, 5)**

#include <stdio.h>

#include<GL/glut.h>

void display2D()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(1.0,0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(0.0,0.0);

glVertex2f(0.8,0.2);

glVertex2f(0.5,0.5);

glVertex2f(0.8,0.7);

glEnd();

glFlush();

}

void display3D()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glClearColor(1.0,0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(0.0,0.0);

glVertex2f(0.8,0.2);

glVertex2f(0.5,0.5);

glVertex2f(0.8,0.7);

glEnd();

glFlush();

}

void init2D(int ch)

{

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

if(ch==1)

glutCreateWindow("2D Orthogonal Projection");

else

glutCreateWindow("2D Perspective Projection");

glutDisplayFunc(display2D);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(ch==1)

gluOrtho2D(-2.0,2.0,-2.0,2.0);

else

gluPerspective(90.0,2.0,0.0,0.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void init3D(int ch)

{

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

if(ch==1)

glutCreateWindow("3D Orthogonal Projection");

else

glutCreateWindow("3D Perspective Projection");

glutDisplayFunc(display3D);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(ch==1)

glOrtho(-2.0,2.0,-2.0,2.0,-2.0,2.0);

else

glFrustum(-2.0,2.0,-2.0,2.0,-2.0,2.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glEnable(GL\_DEPTH\_TEST);

}

int main(int argc, char \*\*argv)

{

int ch;

glutInit(&argc,argv);

printf("Enter the choice: \\n 1: 2D orthogoal Projection\n 2: 2D Perspective Projection\n 3: 3D orthogoal Projection\n 4: 3D Perspective Projection\n");

scanf("%d", &ch);

switch(ch)

{

case 1: init2D(1);

break;

case 2: init2D(2);

break;

case 3: init3D(1);

break;

case 4: init3D(2);

}

glutMainLoop();

return 0;

}

**Labset 3: Write a program to deisgn a scenery by using the following OpenGL buit-in shapes.**(CO1, PO3, 5)

|  |  |  |  |
| --- | --- | --- | --- |
| glutSolidSphere(float radius, int slices, int stacks) | Circle | Sphere | Choose an integer 20-100 for the slices and stacks arguments; the higher the number, the more accurate the sphere/circle. |
| glutSolidCube(double size) | Square | Cube |  |
| glutSolidCone(double base, double height, int slices, int stacks) | Triangle | Cone | The greater the slices argument is, the smoother the cone will appear. |
| glutSolidTorus(double innerRadius, double outerRadius, int nSides, int rings) |  | Torus |  |
| glutSolidTeapot(float radius) |  | | |

#include<GL/gl.h>

#include<GL/glut.h>

#include<GL/glu.h>

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glColor3d(1,0,0);

glPushMatrix();

glTranslated(-1.0,1.2,-6);

glScalef(2,2,2);

glutSolidSphere(0.3,50,50);

glPopMatrix();

glPushMatrix();

glTranslated(-2.6,1.5,-6);

glRotatef(45,1.0,0.0,0.0);

glutWireSphere(0.9,50,50);

glPopMatrix();

glPushMatrix();

glTranslated(1.0,1.2,-6);

glutSolidTorus(0.25,0.5,50,50);

glPopMatrix();

glPushMatrix();

glTranslated(2.6,2.0,-6);

glRotatef(45,1.0,0.0,0.0);

glutWireTorus(0.25,0.5,50,50);

glPopMatrix();

glPushMatrix();

glTranslated(-2.6,-1.5,-6);

glRotatef(45,1.0,0.0,0.0);

glutWireCone(1,1.5,50,50);

glPopMatrix();

glPushMatrix();

glTranslated(-1.0,-1.5,-6);

glRotatef(90,1.0,0.0,0.0);

glutSolidCone(1,1.5,50,50);

glPopMatrix();

glPushMatrix();

glTranslated(1.0,-1.5,-6);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(2.6,-1.5,-6);

glutWireCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(2.0,0.0,-6);

glutSolidTeapot(0.6);

glPopMatrix();

glutSwapBuffers();

}

void resize(int w, int h)

{

const float ar=(float) w / (float) h;

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(-ar,ar,-1.0,1.0,2.0,100.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_RGB|GLUT\_DOUBLE|GLUT\_DEPTH);

glutInitWindowSize(640,500);

glutInitWindowPosition(0.0,0.0);

glutCreateWindow("Tea Pot");

glutReshapeFunc(resize);

glutDisplayFunc(display);

glClearColor(1,1,1,1);

glutMainLoop();

return 0;

}

**Labset 5:Marching Squares with midpoint for 62 cells and a= 0.49 and b=0.5**

#include <stdio.h>

#include <stdlib.h>

#include<GL/glut.h>

#define CMIN -3.0

#define CMAX 3.0

#define THS 0.0

#define NC 62

//global variables to keep track of 4 midpoint of a cell

double m1x,m1y,m2x,m2y,m3x,m3y,m4x,m4y;

//function to generate sample data

double CreateSD(double x,double y)

{

double a=0.49, b=0.5;

return (x\*x+y\*y+a\*a)\*(x\*x+y\*y+a\*a)-4\*a\*a\*x\*x-b\*b\*b\*b;

}

//function to identify a cell type

int celltype(double a, double b, double c, double d)

{

int n=0;

if(a>THS) n+=1;

if(b>THS) n+=2;

if(c>THS) n+=4;

if(d>THS) n+=8;

return n;

}

//function to draw lines in a cell type

void drawLines(int n, float x, float y)

{

void drawOne(int n, float x, float y);

void drawAdjacent(int n, float x, float y);

void drawOpposite(int n, float x, float y);

switch(n)

{

case 1:case 2:case 4:case 7:case 8: case 11: case 13:case 14:

drawOne(n,x,y);

break;

case 3:case 6:case 9:case 12:

drawAdjacent(n,x,y);

break;

case 5: case 10:

drawOpposite(n,x,y);

break;

case 0:case 15:

break;

}

}

void findmidpoints(float x, float y)

{

//dx=dy=0.1

m1x= x;

m1y = y+0.1/2.0;

m2x= x+0.1/2.0;

m2y= y;

m3x= x+0.1;

m3y= y+0.1/2.0;

m4x= x+0.1/2.0;

m4y= y+0.1;

}

void drawOne(int n, float x,float y)

{

findmidpoints(x,y);

glBegin(GL\_LINES);

switch(n)

{

case 1: case 14:

glVertex2d(m1x,m1y);

glVertex2d(m2x,m2y);

break;

case 2: case 13:

glVertex2d(m2x,m2y);

glVertex2d(m3x,m3y);

break;

case 4: case 11:

glVertex2d(m4x,m4y);

glVertex2d(m3x,m3y);

break;

case 7: case 8:

glVertex2d(m4x,m4y);

glVertex2d(m1x,m1y);

}

glEnd();

}

void drawAdjacent(int n, float x,float y)

{

findmidpoints(x,y);

glBegin(GL\_LINES);

switch(n)

{

case 3: case 12:

glVertex2d(m1x,m1y);

glVertex2d(m3x,m3y);

break;

case 6: case 9:

glVertex2d(m2x,m2y);

glVertex2d(m4x,m4y);

}

glEnd();

}

void drawOpposite(int n, float x,float y)

{

findmidpoints(x,y);

glBegin(GL\_LINES);

switch(n)

{

case 5:

glVertex2d(m1x,m1y);

glVertex2d(m4x,m4y);

glVertex2d(m2x,m2y);

glVertex2d(m3x,m3y);

break;

case 10:

glVertex2d(m1x,m1y);

glVertex2d(m2x,m2y);

glVertex2d(m4x,m4y);

glVertex2d(m3x,m3y);

}

glEnd();

}

void display()

{

//array of sample data generated

double SD[NC][NC];

int i,j;

float x,y;

int c;

glClear(GL\_COLOR\_BUFFER\_BIT);

//generate sample data and store on to an array

for(x=-3.0,i=0;x<3.0;i++,x+=0.1)

for(y=-3.0,j=0;y<3.0;j++,y+=0.1)

SD[i][j]= CreateSD(x,y);

//assign generated data to vertices in mesh and find cell type to draw lines

for(x=-3.0,i=0;x<3.0;i++,x+=0.1)

for(y=-3.0,j=0;y<3.0;j++,y+=0.1)

{

c=celltype(SD[i][j],SD[i+1][j],SD[i+1][j+1],SD[i][j+1]);

drawLines(c,x,y);

}

glFlush();

}

void myReshape(int w, int h)

{

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(w<=h)

gluOrtho2D(CMIN,CMAX,CMIN\*(GLfloat)h/(GLfloat)w,CMAX\*(GLfloat)h/(GLfloat)w);

else

gluOrtho2D(CMIN\*(GLfloat)w/(GLfloat)h,CMAX\*(GLfloat)w/(GLfloat)h,CMIN, CMAX);

glMatrixMode(GL\_MODELVIEW);

}

int main(int argc, char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutCreateWindow("Marching Squares");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glClearColor(0.0,0.0,0.0,0.0);

glColor3f(1.0,1.0,1.0);

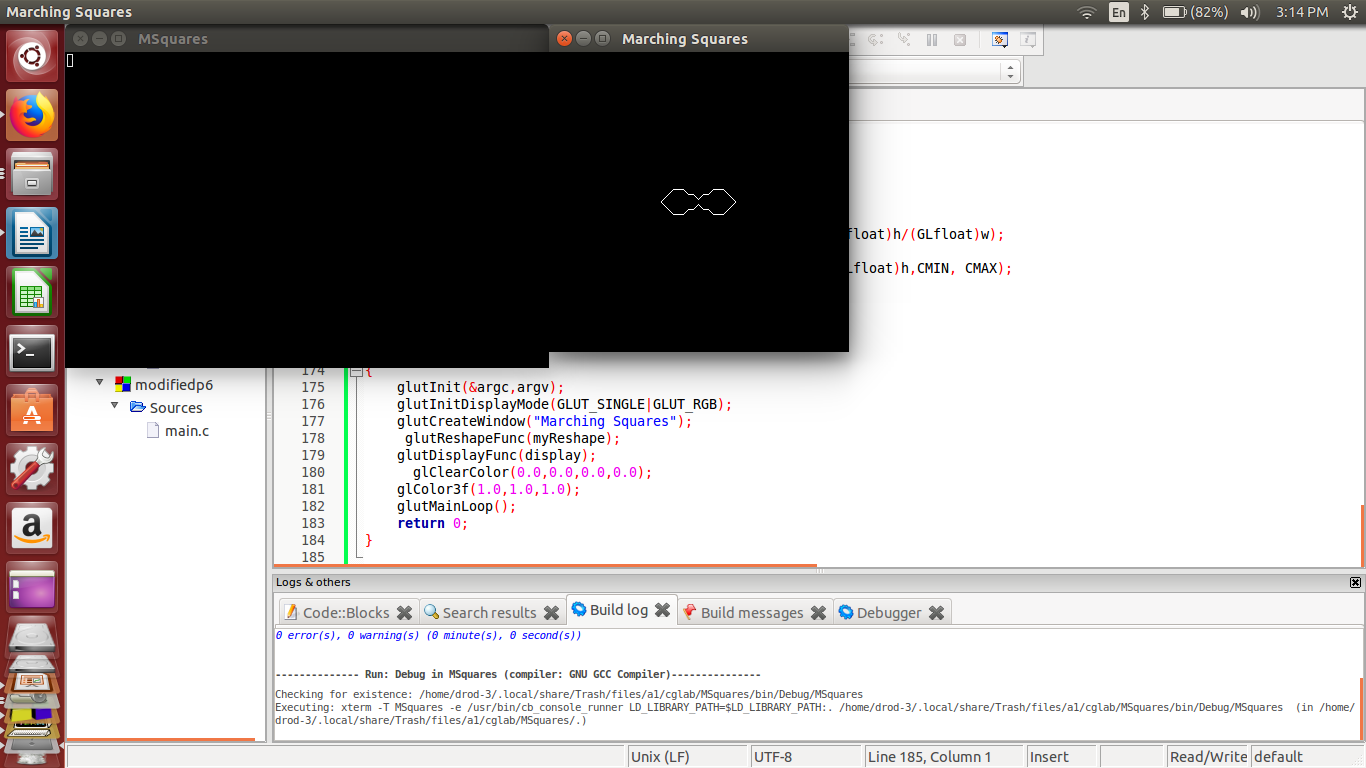
glutMainLoop();

return 0;

}

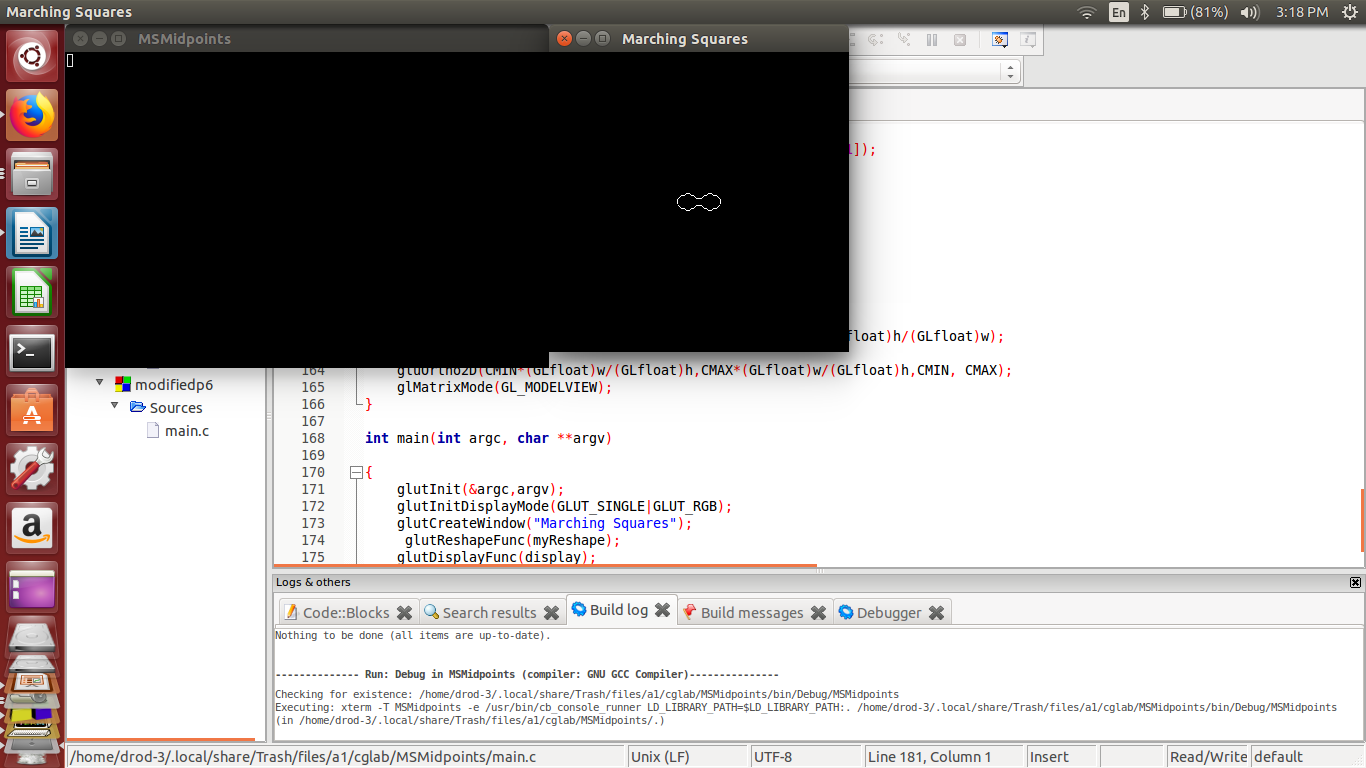
output1

number cells = 62, dx=dy=0.1, a=0.49 and b=0.5



output 2

number cells =150, dx=dy=0.05, a=0.29 and b=0.3



**Appendix Program:Polygon Modeler with the following operation**

1. creation of polygon

2. selection and deletion of a polygon

3. moving a polygon

4. changing the color of a polygon

#include <stdio.h>

#include <stdlib.h>

#include<GL/glut.h>

#define MAX\_POLY 8

#define MAX\_VER 10

#define TRUE 1

#define FALSE 0

typedef struct polygon

{

int color;

int used;

int xmin, xmax,ymin,ymax;

int xc, yc;

int nvtcs;

int x[MAX\_VER],y[MAX\_VER];

}polygon;

int picking = FALSE;

int moving =FALSE;

int in\_poly = -1;

int present\_color = 0;

GLsizei wh =500, ww = 500;

GLfloat colors[8][3]= {{0.0,0.0,0.0},{1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},{1.0,1.0,0.0},{1.0,0.0,1.0},{0.0,1.0,1.0},

{0.5,0.5,1.0}};

polygon polys[MAX\_POLY];

int pick\_poly(int x,int y);

void myReshape(int w, int h)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,(GLdouble)w, 0.0,(GLdouble)h);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glViewport(0,0,w,h);

ww = w;

wh= h;

}

void myinit()

{

int i;

glClearColor(1.0,1.0,1.0,0.0);

/\* mark all polygons unused \*/

for(i=0;i<MAX\_POLY;i++)

polys[i].used = FALSE;

}

void Mymouse(int btn, int st,int x, int y)

{

int i,j;

y=wh-y;

if(btn==GLUT\_LEFT\_BUTTON && st==GLUT\_DOWN && !picking && !moving)

{

/\* adding vertices\*/

moving = FALSE;

if(in\_poly>=0)

{

if(polys[in\_poly].nvtcs == MAX\_VER)

{

printf("Exceeds maximum number of vertices\n");

exit(0);

}

i= polys[in\_poly].nvtcs;

polys[in\_poly].x[i]=x;

polys[in\_poly].y[i]=y;

polys[in\_poly].nvtcs++;

}

}

if(btn==GLUT\_LEFT\_BUTTON && st==GLUT\_DOWN && picking && !moving)

{

/\*delete polygon\*/

picking= FALSE;

moving = FALSE;

j= pick\_poly(x,y);

if(j>=0)

{

polys[j].used = FALSE;

in\_poly = -1;

glutPostRedisplay();

}

}

}

int pick\_poly(int x, int y)

{

/\* find first polygon in which we are in bounding box \*/

int i;

for(i=0; i<MAX\_POLY; i++)

{

if(polys[i].used)

if((x>=polys[i].xmin) && (x<=polys[i].xmax) &&

(y>=polys[i].ymin) && (y<=polys[i].ymax))

{

in\_poly =i;

moving =TRUE;

return i;

}

}

printf("not in a polygon\n");

return -1;

}

void mymotion(int x, int y)

{

/\* find if we are inside a polygon \*/

float dx,dy;

int i,j;

if(moving)

{

y=wh-y;

j= pick\_poly(x,y);

if(j<0)

{

printf("not in a polygon\n");

return;

}

dx= x-polys[j].xc;

dy = y-polys[j].yc;

for(i=0;i<polys[j].nvtcs;i++)

{

polys[j].x[i]+=dx;

polys[j].y[i]+=dy;

}

/\* update the bounding box \*/

polys[j].xc+=dx;

polys[j].yc+=dy;

if(dx>0) polys[j].xmax+=dx;

else polys[j].xmin+=dx;

if(dy>0) polys[j].ymax+=dy;

else polys[j].ymin+=dy;

glutPostRedisplay();

}

}

void colormenu(int id )

{

present\_color = id;

if(in\_poly>=0) polys[in\_poly].color = id;

}

void mainmenu(int id)

{

int i;

switch(id)

{

case 1: /\*create a new polygon \*/

moving = FALSE;

/\*find a polygon which is not in use now \*/

for(i=0; i< MAX\_POLY;i++)

if(polys[i].used == FALSE) break;

if(i == MAX\_POLY)

{

printf("Exceeded maximum number of polygons\n");

exit(0);

}

/\*Give the details of the polygon\*/

polys[i].color =present\_color;

polys[i].used = TRUE;

polys[i].nvtcs =0;

in\_poly= i;

picking = FALSE;

break;

case 2: /\* end polygon and find bounding box and center\*/

moving =FALSE;

if(in\_poly>=0)

{

/\*initialize the bouning box and center to frist vertex \*/

polys[in\_poly].xmax = polys[in\_poly].xmin = polys[in\_poly].x[0];

polys[in\_poly].ymax = polys[in\_poly].ymin = polys[in\_poly].y[0];

polys[in\_poly].xc = polys[in\_poly].x[0];

polys[in\_poly].yc = polys[in\_poly].y[0];

/\* now find the actual center and limits of bounding box\*/

for(i=1;i<polys[in\_poly].nvtcs;i++)

{

if(polys[in\_poly].x[i] < polys[in\_poly].xmin)

polys[in\_poly].xmin = polys[in\_poly].x[i];

else if(polys[in\_poly].x[i] > polys[in\_poly].xmax)

polys[in\_poly].xmax = polys[in\_poly].x[i];

if(polys[in\_poly].y[i] < polys[in\_poly].ymin)

polys[in\_poly].ymin = polys[in\_poly].y[i];

else if(polys[in\_poly].y[i] > polys[in\_poly].ymax)

polys[in\_poly].ymax = polys[in\_poly].y[i];

polys[in\_poly].xc += polys[in\_poly].x[i];

polys[in\_poly].yc += polys[in\_poly].y[i];

}

polys[in\_poly].xc = polys[in\_poly].xc/ polys[in\_poly].nvtcs;

polys[in\_poly].yc = polys[in\_poly].yc/ polys[in\_poly].nvtcs;

}

in\_poly = -1;

glutPostRedisplay();

break;

case 3: /\* set picking mode \*/

picking = TRUE;

moving = FALSE;

break;

case 4: /\* set moving mode \*/

moving = TRUE;

break;

case 5: exit(0);

}

}

void display()

{

/\* display all active polygons \*/

int i,j;

glClear(GL\_COLOR\_BUFFER\_BIT);

for(i=0;i<MAX\_POLY;i++)

{

if(polys[i].used)

{

glColor3fv(colors[polys[i].color]);

glBegin(GL\_POLYGON);

for(j=0;j<polys[i].nvtcs;j++)

glVertex2i(polys[i].x[j], polys[i].y[j]);

glEnd();

}

}

glFlush();

}

int main(int argc,char \*\*argv)

{

int cm;

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutCreateWindow("Polygon Modeler");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

myinit();

cm= glutCreateMenu(colormenu);

glutAddMenuEntry("Black", 0);

glutAddMenuEntry("Red",1);

glutAddMenuEntry("Green",2);

glutAddMenuEntry("Blue",3);

glutAddMenuEntry("Yellow",4);

glutAddMenuEntry("Magenta",5);

glutAddMenuEntry("Cyan",6);

glutAddMenuEntry("unknown",7);

glutCreateMenu(mainmenu);

glutAddMenuEntry("new polygon", 1);

glutAddMenuEntry("end polygon",2);

glutAddMenuEntry("delete polygon", 3);

glutAddMenuEntry("move polygon",4);

glutAddMenuEntry("quit", 5);

glutAddSubMenu("Colors", cm);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMouseFunc(Mymouse);

glutMotionFunc(mymotion);

glutMainLoop();

return 0;

}

**Labset 6:Polygon Modeler with the following operation**

1. creation of polygon

2. selection of a polygon

3.deletion of a polygon

#include <stdio.h>

#include <stdlib.h>

#include<GL/glut.h>

#define MAX\_POLY 8

#define MAX\_VER 10

#define TRUE 1

#define FALSE 0

typedef struct polygon

{

int color;

int used;

int xmin, xmax,ymin,ymax;

int nvtcs;

int x[MAX\_VER],y[MAX\_VER];

}polygon;

int picking = FALSE;

int del = FALSE;

int in\_poly = -1;

int present\_color = 0;

GLsizei wh =500, ww = 500;

GLfloat colors[2][3]= {{1.0,0.0,0.0},{0.0,1.0,0.0}};

polygon polys[MAX\_POLY];

int pick\_poly(int x,int y);

void myReshape(int w, int h)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,(GLdouble)w, 0.0,(GLdouble)h);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glViewport(0,0,w,h);

ww = w;

wh= h;

}

void myinit()

{

int i;

glClearColor(1.0,1.0,1.0,0.0);

/\* mark all polygons unused \*/

for(i=0;i<MAX\_POLY;i++)

polys[i].used = FALSE;

}

void Mymouse(int btn, int st,int x, int y)

{

int i,j;

y=wh-y;

if(btn==GLUT\_LEFT\_BUTTON && st==GLUT\_DOWN && !picking )

{

/\* adding vertices\*/

if(in\_poly>=0)

{

if(polys[in\_poly].nvtcs == MAX\_VER)

{

printf("Exceeds maximum number of vertices\n");

exit(0);

}

i= polys[in\_poly].nvtcs;

polys[in\_poly].x[i]=x;

polys[in\_poly].y[i]=y;

polys[in\_poly].nvtcs++;

}

}

if(btn==GLUT\_LEFT\_BUTTON && st==GLUT\_DOWN && picking && !del)

{

picking= FALSE;

j= pick\_poly(x,y);

if(j>=0)

{

/\* highlight selected poylgon with select color\*/

polys[j].color = present\_color;

present\_color =0;

in\_poly = -1;

}

glutPostRedisplay();

}

if(btn==GLUT\_LEFT\_BUTTON && st==GLUT\_DOWN && picking && del)

{

/\*delete polygon\*/

picking= FALSE;

j= pick\_poly(x,y);

if(j>=0)

{

polys[j].used = FALSE;

del=FALSE;

in\_poly = -1;

}

glutPostRedisplay();

}

}

int pick\_poly(int x, int y)

{

/\* find first polygon in which we are in bounding box \*/

int i;

for(i=0; i<MAX\_POLY; i++)

{

if(polys[i].used)

if((x>=polys[i].xmin) && (x<=polys[i].xmax) &&

(y>=polys[i].ymin) && (y<=polys[i].ymax))

{

in\_poly =i;

return i;

}

}

printf("not in a polygon\n");

return -1;

}

void mainmenu(int id)

{

int i;

switch(id)

{

case 1: /\*create a new polygon \*/

/\*find a polygon which is not in use now \*/

picking = FALSE;

del = FALSE;

for(i=0; i< MAX\_POLY;i++)

if(polys[i].used == FALSE) break;

if(i == MAX\_POLY)

{

printf("Exceeded maximum number of polygons\n");

exit(0);

}

/\*Give the details of the polygon\*/

polys[i].color =present\_color;

polys[i].used = TRUE;

polys[i].nvtcs =0;

in\_poly= i;

break;

case 2: /\* end polygon and find bounding box and center\*/

if(in\_poly>=0)

{

/\*initialize the bouning box and center to frist vertex \*/

polys[in\_poly].xmax = polys[in\_poly].xmin = polys[in\_poly].x[0];

polys[in\_poly].ymax = polys[in\_poly].ymin = polys[in\_poly].y[0];

/\* now find the actual center and limits of bounding box\*/

for(i=1;i<polys[in\_poly].nvtcs;i++)

{

if(polys[in\_poly].x[i] < polys[in\_poly].xmin)

polys[in\_poly].xmin = polys[in\_poly].x[i];

else if(polys[in\_poly].x[i] > polys[in\_poly].xmax)

polys[in\_poly].xmax = polys[in\_poly].x[i];

if(polys[in\_poly].y[i] < polys[in\_poly].ymin)

polys[in\_poly].ymin = polys[in\_poly].y[i];

else if(polys[in\_poly].y[i] > polys[in\_poly].ymax)

polys[in\_poly].ymax = polys[in\_poly].y[i];

}

}

in\_poly = -1;

glutPostRedisplay();

break;

case 3: /\* set picking mode \*/

picking = TRUE;

present\_color =1;

del = FALSE;

break;

case 4: /\* delete a polygon \*/

picking = TRUE;

del = TRUE;

break;

case 5: exit(0);

}

}

void display()

{

/\* display all active polygons \*/

int i,j;

glClear(GL\_COLOR\_BUFFER\_BIT);

for(i=0;i<MAX\_POLY;i++)

{

if(polys[i].used)

{

glColor3fv(colors[polys[i].color]);

glBegin(GL\_POLYGON);

for(j=0;j<polys[i].nvtcs;j++)

glVertex2i(polys[i].x[j], polys[i].y[j]);

glEnd();

}

}

glFlush();

}

int main(int argc,char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutCreateWindow("Polygon Modeler");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

myinit();

/\* Create Menu\*/

glutCreateMenu(mainmenu);

glutAddMenuEntry("new polygon", 1);

glutAddMenuEntry("end polygon",2);

glutAddMenuEntry("Select polygon", 3);

glutAddMenuEntry("delete polygon", 4);

glutAddMenuEntry("quit", 5);

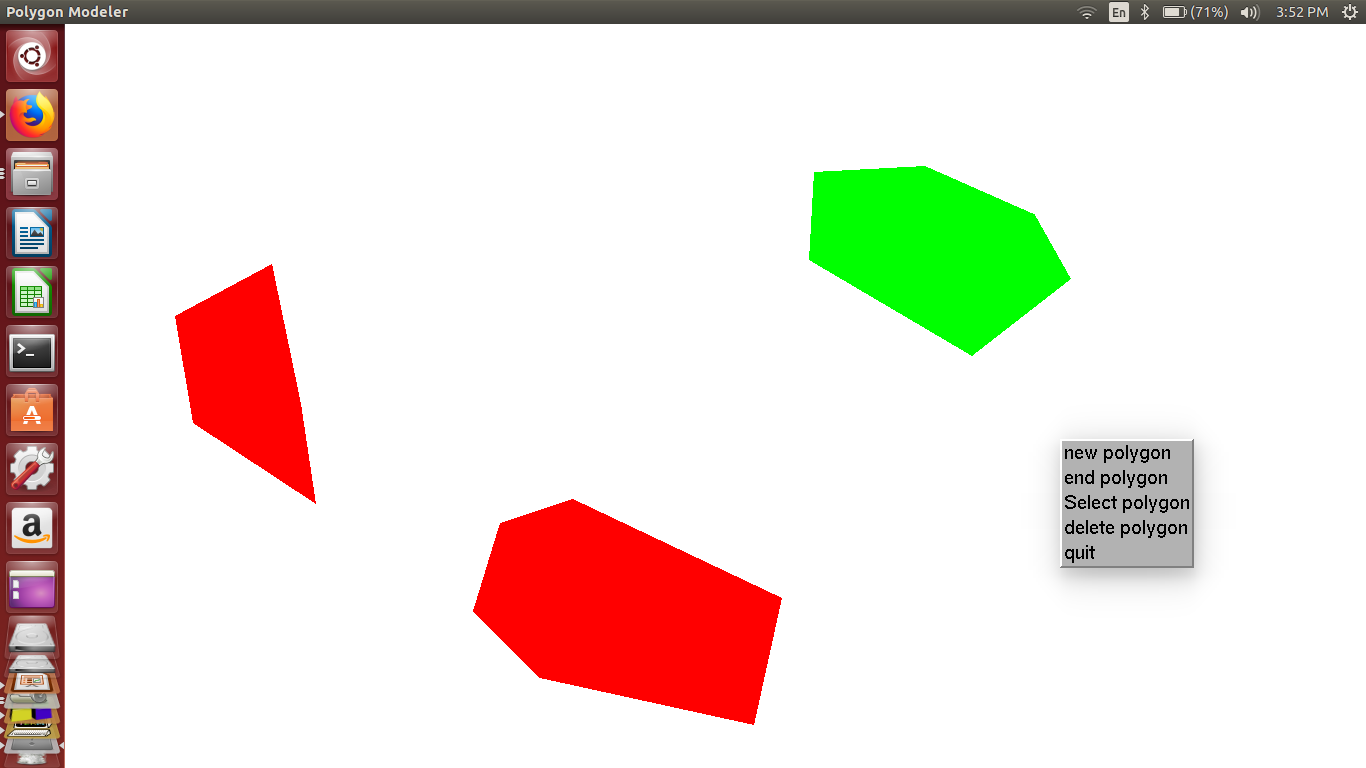
glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMouseFunc(Mymouse);

glutMainLoop();

return 0;

}



Labset 7: Demo of OpenGL transformations:

#include <stdio.h>

#include <stdlib.h>

#include <GL/glut.h>

#include<math.h>

/\* Demonstration of Transformations using APIs with Keyboard interfacing \*/

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POLYGON);

glVertex2f(-2.0, -2.0);

glVertex2f(2.0, -2.0);

glVertex2f(2.0, 2.0);

glVertex2f(-2.0, 2.0);

glEnd();

glFlush();

}

void init()

{

glViewport(0,0,500,500);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-30.0,30.0,-30.0,30.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glColor3f(0.5,0.5,1.0);

glClearColor(1.0,1.0,1.0,0.0);

}

/\*normal keys interaction for transformation

u -- up, d--down, r--right l--left

i -- increase size, D -- decrease size

RL- rotate left RR --rotate right

\*/

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

{

switch(key)

{

case 'l':

glTranslatef(-2.0,0.0f,0.0f);

break;

case 'r':

glTranslatef(2.0,0.0f,0.0f);

break;

case 'u': glTranslatef(0.0f,2.0,0.0f);

break;

case 'd':

glTranslatef(0.0,-2.0,0.0);

break;

case 'i':

glScalef(1.5,1.5,1.5);

break;

case 'D': ;

glScalef(-0.5,-0.5,-0.5);

break;

case 'R': glRotatef(10,1.0,1.0,0.0);

break;

case 'L': glRotatef(-10,1.0,0.0,0.0);

}

glutPostRedisplay();

}

int main(int argc,char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_RGB|GLUT\_SINGLE);

glutCreateWindow("Transformation with out APIs");

glutDisplayFunc(display);

//glutSpecialFunc(myskeys);

glutKeyboardFunc(mykeys);

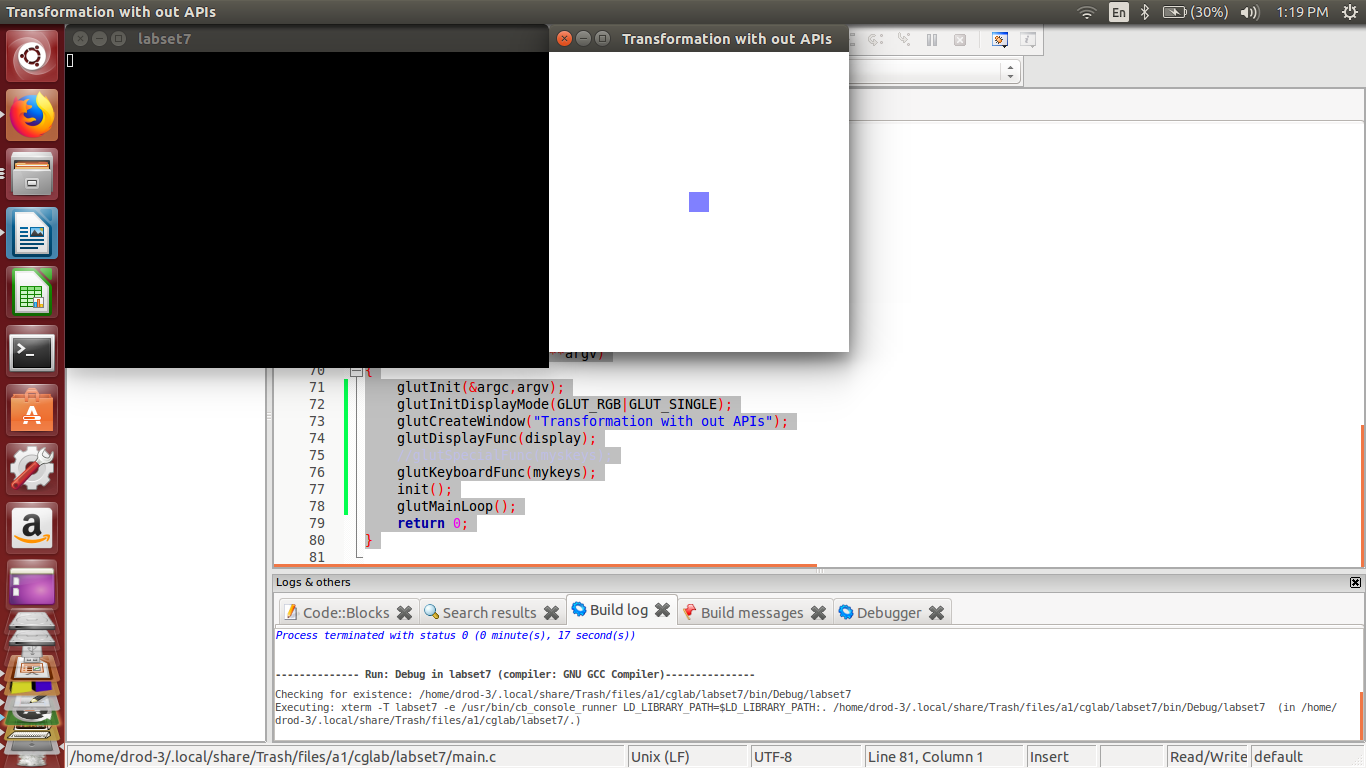
init();

glutMainLoop();

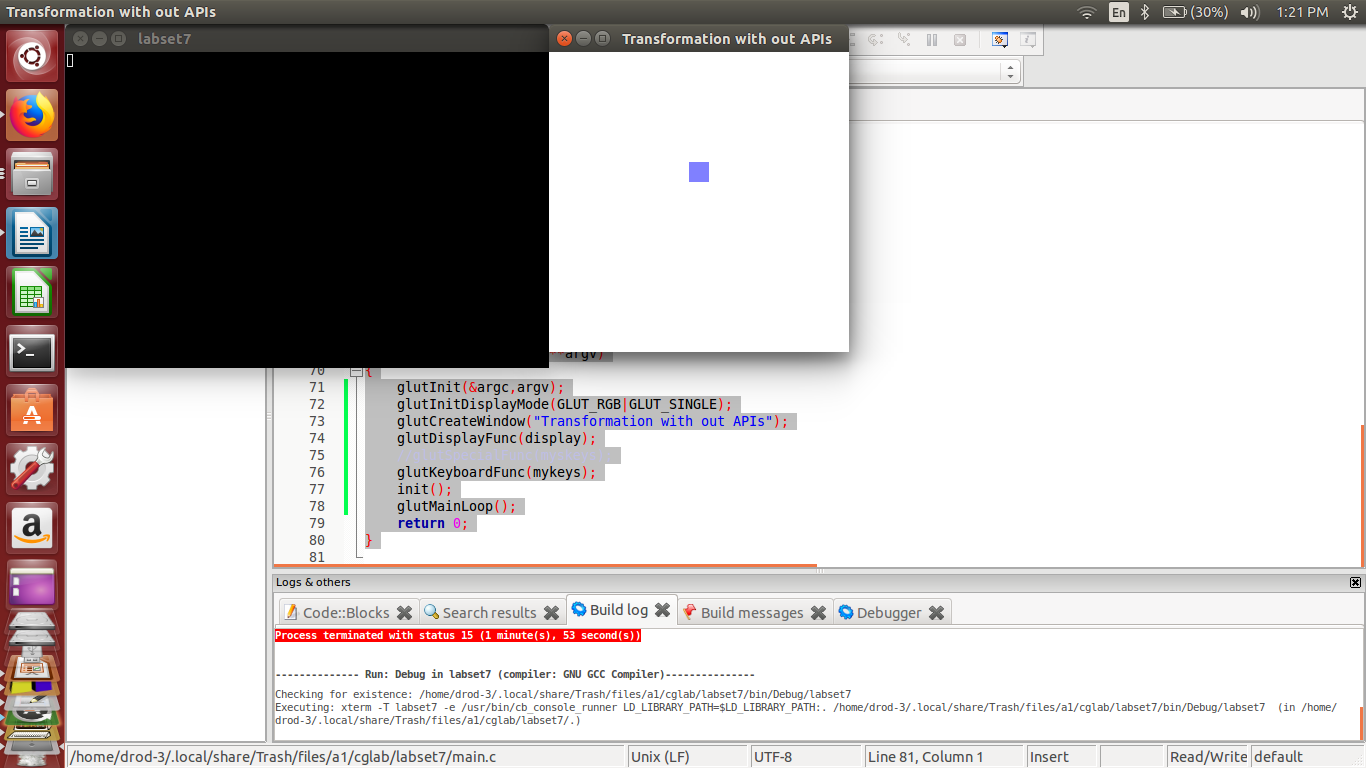
return 0;

}

output:



on pressing key = 'u'



and so on..

**Labset 8: CAR LIKE Structure with a menu of the following operations**

**i. start car**

**ii. stop car**

**iii. Turn Left**

**iv. turn right**

**v. increase speed**

**vi. decrease speed**

**vii. colors**

**viii. quit**

#include <stdio.h>

#include <stdlib.h>

#include <GL/glut.h>

#include<math.h>

//declare matrix of colors

float colors[4][3]={{1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},{1.0,1.0,0.0}};

int body\_color=0; //index to set body color

double s=0.0; // speed variable

double tr=0.0; // rotation with an arbitrary axis (tr,tr,tr);

void car()

{

//car wheels

glPushMatrix();

glTranslatef(-0.9,-0.02,-0.2);// move the torus to left

glutSolidTorus(0.01,0.03,25,25);// render a torus at the center

glPopMatrix();

glPushMatrix();

glTranslatef(-0.6,-0.02,-0.2);

glutSolidTorus(0.01,0.03,25,25);

glPopMatrix();

//car body

glColor3fv(colors[body\_color]);

glPushMatrix();

glTranslatef(-0.75,0.1,-0.3);

glScalef(2.5,2.0,1.0);

glutSolidCube(0.1);

glPopMatrix();

glColor3f(0.5,0.5,0.5);

glPushMatrix();

//front mirror

glTranslatef(-0.6,0.1,-0.4);

glScalef(0.5,2.0,1.0);

glutSolidCube(0.1);

glPopMatrix();

//back mirror

glPushMatrix();

glTranslatef(-0.9,0.1,-0.4);

glScalef(0.5,2.0,1.0);

glutSolidCube(0.1);

glPopMatrix();

//dicky

glColor3fv(colors[body\_color]);

glPushMatrix();

glTranslatef(-0.95,0.1,-0.4);

glScalef(1.2,2.0,1.0);

glutSolidCube(0.1);

glPopMatrix();

//banet

glPushMatrix();

glTranslatef(-0.55,0.1,-0.4);

glScalef(1.2,2.0,1.0);

glutSolidCube(0.1);

glPopMatrix();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glColor3f(0.4,0.4,0.4);

glPushMatrix();

glRotated(20,tr,tr,tr);

car();

glPopMatrix();

//road

glColor3f(0.0,0.0,0.0);

glPushMatrix();

glTranslated(-1.0,0.0,0.2);

glScalef(4.5,1.2,0.0);

glutSolidCube(1.0);

glPopMatrix();

glutSwapBuffers();

glFlush();

}

void init()

{

glClearColor(1.0,1.0,1.0,0.0);

}

void reshape(int w,int h)

{

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(w<=h)

glOrtho(-30.0,30.0,-30.0\*(GLfloat)w/h,30.0\*(GLfloat)w/h,-10.0,10.0);

else

glOrtho(-30.0\*(GLfloat)h/w,30.0\*(GLfloat)h/w,-30.0,30.0,-10.0,10.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void idle()

{

s=0.001;

glTranslated(s,0.0,0.0);

glutPostRedisplay();

}

void mainmenu(int id)

{

switch(id)

{

case 1: //start car in the current direction

glutIdleFunc(idle);

break;

case 2://stop car

glutIdleFunc(NULL);

break;

case 3: //Turn left

tr+=0.01;

break;

case 4: //Turn Right;

tr-=0.01;

break;

case 5://increase speed

if(s==1.01)

break;

s+=0.001;

break;

case 6: //decrease speed

if(s==0.0)

break;

s-=0.0001;

}

glutPostRedisplay();

}

void colormenu(int id)

{

body\_color =id;

}

int main(int argc,char \*\*argv)

{ int id;

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_RGB|GLUT\_DOUBLE|GLUT\_DEPTH);

glutCreateWindow("CAR APPLICATION");

glutDisplayFunc(display);

init();

glEnable(GL\_DEPTH\_TEST);

id= glutCreateMenu(colormenu);

glutAddMenuEntry("RED",0);

glutAddMenuEntry("GREEN",1);

glutAddMenuEntry("BLUE",2);

glutAddMenuEntry("YELLOW",3);

glutCreateMenu(mainmenu);

glutAddMenuEntry("Start Car",1);

glutAddMenuEntry("Stop Car",2);

glutAddMenuEntry("Turn left",3);

glutAddMenuEntry("Turn right",4);

glutAddMenuEntry("SPeed up",5);

glutAddMenuEntry("Speed down",6);

glutAddSubMenu("COLORS",id);

glutAddMenuEntry("Quit",7);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return 0;

}

output:



**Labset 9: Rotation of a cube with Z-buffer enabled**

#include <GL/glut.h>

double rotate\_y=0;

double rotate\_x=0;

void display(){

// Clear screen and Z-buffer

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

// Reset transformations

glLoadIdentity();

// Rotate when user changes rotate\_x and rotate\_y

glRotatef( rotate\_x, 1.0, 0.0, 0.0 );

glRotatef( rotate\_y, 0.0, 1.0, 0.0 );

//Multi-colored side - FRONT

glBegin(GL\_POLYGON);

glColor3f( 1.0, 0.0, 0.0 ); glVertex3f( 0.5, -0.5, -0.5 ); // P1 is red

glColor3f( 0.0, 1.0, 0.0 ); glVertex3f( 0.5, 0.5, -0.5 ); // P2 is green

glColor3f( 0.0, 0.0, 1.0 ); glVertex3f( -0.5, 0.5, -0.5 ); // P3 is blue

glColor3f( 1.0, 0.0, 1.0 ); glVertex3f( -0.5, -0.5, -0.5 ); // P4 is purple

glEnd();

// White side - BACK

glBegin(GL\_POLYGON);

glColor3f( 1.0, 1.0, 1.0 );

glVertex3f( 0.5, -0.5, 0.5 );

glVertex3f( 0.5, 0.5, 0.5 );

glVertex3f( -0.5, 0.5, 0.5 );

glVertex3f( -0.5, -0.5, 0.5 );

glEnd();

// Purple side - RIGHT

glBegin(GL\_POLYGON);

glColor3f( 1.0, 0.0, 1.0 );

glVertex3f( 0.5, -0.5, -0.5 );

glVertex3f( 0.5, 0.5, -0.5 );

glVertex3f( 0.5, 0.5, 0.5 );

glVertex3f( 0.5, -0.5, 0.5 );

glEnd();

// Green side - LEFT

glBegin(GL\_POLYGON);

glColor3f( 0.0, 1.0, 0.0 );

glVertex3f( -0.5, -0.5, 0.5 );

glVertex3f( -0.5, 0.5, 0.5 );

glVertex3f( -0.5, 0.5, -0.5 );

glVertex3f( -0.5, -0.5, -0.5 );

glEnd();

// Blue side - TOP

glBegin(GL\_POLYGON);

glColor3f( 0.0, 0.0, 1.0 );

glVertex3f( 0.5, 0.5, 0.5 );

glVertex3f( 0.5, 0.5, -0.5 );

glVertex3f( -0.5, 0.5, -0.5 );

glVertex3f( -0.5, 0.5, 0.5 );

glEnd();

// Red side - BOTTOM

glBegin(GL\_POLYGON);

glColor3f( 1.0, 0.0, 0.0 );

glVertex3f( 0.5, -0.5, -0.5 );

glVertex3f( 0.5, -0.5, 0.5 );

glVertex3f( -0.5, -0.5, 0.5 );

glVertex3f( -0.5, -0.5, -0.5 );

glEnd();

glFlush();

glutSwapBuffers();

}

// specialKeys() Callback Function

void specialKeys( int key, int x, int y ) {

// Right arrow - increase rotation by 5 degree

if (key == GLUT\_KEY\_RIGHT)

rotate\_y += 5;

// Left arrow - decrease rotation by 5 degree

else if (key == GLUT\_KEY\_LEFT)

rotate\_y -= 5;

else if (key == GLUT\_KEY\_UP)

rotate\_x += 5;

else if (key == GLUT\_KEY\_DOWN)

rotate\_x -= 5;

// Request display update

glutPostRedisplay();

}

// ----------------------------------------------------------

// main() function

// ----------------------------------------------------------

int main(int argc, char\* argv[]){

// Initialize GLUT and process user parameters

glutInit(&argc,argv);

// Request double buffered true color window with Z-buffer

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

// Create window

glutCreateWindow("Rotating Cube");

// Enable Z-buffer depth test

glEnable(GL\_DEPTH\_TEST);

// Callback functions

glutDisplayFunc(display);

glutSpecialFunc(specialKeys);

// Pass control to GLUT for events

glutMainLoop();

return 0;

}

**Labset 10: Scanline fiiling a Square**

#include<stdio.h>

#include <GL/glut.h>

float fgcolor[] = { 1.0,1.0,0.0};

float bgcolor[] = { 1.0,1.0,1.0};

void init() {

glClearColor(1, 1,1, 0.0);

glColor3fv(fgcolor);

glPointSize(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 640, 0, 480);

}

float\* getPixelColor(float x, float y)

{

float colr[3];

glReadPixels(x, y, 1, 1, GL\_RGB, GL\_FLOAT, colr);

return colr;

}

void setPixelColor(float x, float y) {

glColor3fv(fgcolor);

glBegin(GL\_POINTS);

glVertex2f(x, y);

glEnd();

glFlush();

}

void scanFill() {

float \*colr;

float x,y;

for(y=140.0;y<340;y++)

for(x=120.0;x<420;x++)

{

colr = getPixelColor(x, y);

if(\*(colr+0) == bgcolor[0] && \*(colr+1) == bgcolor[1] && \*(colr+2) == bgcolor[2])

setPixelColor(x, y);

}

return;

}

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

{

if(button==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN)

scanFill();

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_LINE\_LOOP);

glVertex2f(120.0,340.0);

glVertex2f(420.0,340.0);

glVertex2f(420.0,140.0);

glVertex2f(120.0,140.0);

glEnd();

glFlush();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutInitWindowSize(640, 480);

glutCreateWindow("Open GL");

init();

glutDisplayFunc(display);

glutMouseFunc(onMouseClick);

glutMainLoop();

return 0;

}

**Flood filling a Square**

#include<stdio.h>

#include <GL/glut.h>

float fgcolor[] = { 1.0,1.0,0.0};

float bgcolor[] = { 1.0,1.0,1.0};

void init() {

glClearColor(1, 1,1, 0.0);

glColor3fv(fgcolor);

glPointSize(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 640, 0, 480);

}

float\* getPixelColor(float x, float y)

{

float colr[3];

glReadPixels(x, y, 1, 1, GL\_RGB, GL\_FLOAT, colr);

return colr;

}

void setPixelColor(float x, float y) {

glColor3fv(fgcolor);

glBegin(GL\_POINTS);

glVertex2f(x, y);

glEnd();

glFlush();

}

void floodFill(float x,float y) {

float \*colr;

colr = getPixelColor(x, y);

if(\*(colr+0) == bgcolor[0] && \*(colr+1) == bgcolor[1] && \*(colr+2) == bgcolor[2])

{

//printf("current pixel\n");

setPixelColor(x, y);

floodFill(x+1, y);

floodFill(x-1, y);

floodFill(x, y-1);

floodFill(x, y+1);

}

}

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

{

if(button==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN)

floodFill(121.0,141.0);

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_LINE\_LOOP);

glVertex2f(120.0,340.0);

glVertex2f(420.0,340.0);

glVertex2f(420.0,140.0);

glVertex2f(120.0,140.0);

glEnd();

glFlush();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutInitWindowSize(640, 480);

glutCreateWindow("Open GL");

init();

glutDisplayFunc(display);

glutMouseFunc(onMouseClick);

glutMainLoop();

return 0;

}

output:

