// SOURCE CODE OF DRAWING EDITOR //

#include "stdafx.h"

#include <stdlib.h> //Needed for "exit" function

#include <GL/glut.h>

#include<math.h>

#include<iostream>

#include<time.h>

//Include OpenGL header files, so that we can use OpenGL

#ifdef \_\_APPLE\_\_

#include <OpenGL/OpenGL.h>

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

using namespace std;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Globle variables\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

const int width=900,height=650;

//GLdouble a,b,c,d;//variables for screen size control

GLint brush=5;

#define RED 1

#define GREEN 2

#define BLUE 3

#define WHITE 4

#define QUAD 5

#define LINE 6

#define PENCIL 7

#define TRIANGLE 8

#define CLEAR 9

#define CIRCLE 10

#define CIRCLE2 11

#define QUAD2 12

#define TRIANGLE2 13

#define TEXT 14

#define BLACK 15

#define ERASER 16

#define SPRAY 17

#define CALLIGRAPH 18

//initializing global variables

float R=0,G=0,B=0;

int X1=-1, X2=-1, X3=-1, Y1=-1, Y2=-1, Y3=-1, rad, LastX,LastY;

float DX,DY;

bool P1= false,P2=false, P3= false, Cleared = false, Cleared2 = false, Quad=false,

Quad2=false,Line=false, Point=false, Triangle=false,Triangle2=false ,Pencil=false,Eraser=false,spray=false,calli=false,

Circle = false, Circle2=false, checked=false, Text=false,flag=false;

char keyboardbuffer[64];

char \*shape=" ";

int keyboardindex;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class for Random Dots mode\*\*\*\*\*\*\*\*\*\*\*\*//

class GLintRandom{

public:

int random(int R)

{

return rand()%(R-1);

}};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class for paint mode\*\*\*\*\*\*\*\*\*\*\*\*//

class GLintPaint{

public:

static int brushsizeP,enbP;

void enableP(int P)

{

brushsizeP=5;

enbP=P;

}

};

int GLintPaint::brushsizeP=5;

int GLintPaint::enbP=0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class for calligraphy mode\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

class GLintCalligraphy3{

public:

static int enbC3;

static int brushsizeC3;

void enableC3(int C3)

{

enbC3=C3;

}

};

int GLintCalligraphy3::enbC3=0;

int GLintCalligraphy3::brushsizeC3=14;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class for Erase mode\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

class GLintErase{

public:

void clear(){

glClear(GL\_COLOR\_BUFFER\_BIT);

}

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class for Spray mode\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

class GLintSpray:public GLintRandom{

public:

static int enbS;

static int brushsizeS;

void enableS(int S)

{

enbS=S;

}

};

int GLintSpray::enbS=0;

int GLintSpray::brushsizeS=4;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class for Rubber mode\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

class GLintRubber{

public:

static int enbR;

static int brushsizeR;

void enableR(int R)

{

enbR=R;

}

};

int GLintRubber::enbR=0;

int GLintRubber::brushsizeR=5;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Function for window configuration\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

static void setwindow(float left,GLdouble right,float bottom,GLdouble top)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(left,right,bottom,top);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Function for Viewport configuration\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

static void setviewport(int left,int right,int width,int height)

{

glViewport(left,right,width,height);

}

//control function

void myInit(void)

{

glClearColor(121.0,1.0,1.0,120.0); //background color

glColor3f(0.0f,0.0f,0.0f); //draw color

glPointSize(0.1); //pixel size

setwindow(0.0,width,0.0,height);

}

//start of drawpanel() function

void drawpanel()

{

//panel backgound

glBegin(GL\_QUADS);

glColor3f(0,0,1);

glVertex2i(0,0);

glVertex2i(0,70);

glVertex2i(900,70);

glVertex2i(900,0);

glEnd();

//panel sepration lines

glBegin(GL\_LINES);

glColor3f(0,0,0);

glVertex2i(0,70);

glVertex2i(900,70);

glVertex2i(0,35);

glVertex2i(240,35);

glVertex2i(600,35);

glVertex2i(900,35);

glVertex2i(40,70);

glVertex2i(40,0);

glVertex2i(80,70);

glVertex2i(80,0);

glVertex2i(120,70);

glVertex2i(120,0);

glVertex2i(160,70);

glVertex2i(160,0);

glVertex2i(200,70);

glVertex2i(200,0);

glVertex2i(240,70);

glVertex2i(240,0);

glVertex2i(600,70);

glVertex2i(600,0);

glEnd();

//pencil button in panel

glBegin(GL\_POLYGON);

glColor3f(1,0,0);

glVertex2i(25,65);

glVertex2i(15,55);

glVertex2i(20,50);

glVertex2i(30,60);

glVertex2i(25,65);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(0,1,0);

glVertex2i(25,65);

glVertex2i(17,52);

glVertex2i(30,60);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(0,0,0);

glVertex2i(15,55);

glVertex2i(11,46);

glVertex2i(20,50);

glEnd();

//eraser button in panel

glBegin(GL\_POLYGON);

glColor3f(1,1,0);

glVertex2i(16,12);

glVertex2i(30,12);

glVertex2i(16,7);

glVertex2i(30,7);

glVertex2i(16,12);

glVertex2i(16,7);

glVertex2i(30,12);

glVertex2i(30,7);

glEnd();

glBegin(GL\_LINE\_STRIP);

glVertex2i(16,11);

glVertex2i(11,11);

glVertex2i(11,11);

glVertex2i(11,7);

glVertex2i(11,7);

glVertex2i(16,7);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3f(0,0,0);

glVertex2i(25,28);

glVertex2i(24,30);

glVertex2i(24,30);

glVertex2i(14,30);

glVertex2i(14,30);

glVertex2i(14,18);

glVertex2i(14,18);

glVertex2i(24,18);

glVertex2i(24,18);

glVertex2i(25,20);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3f(0,0,0);

glVertex2i(14,24);

glVertex2i(20,24);

glEnd();

//drawing color status

glBegin(GL\_POLYGON);

glColor3f(0,0,0);

glVertex2i(600,35);

glVertex2i(700,35);

glVertex2i(700,0);

glVertex2i(600,0);

glVertex2i(600,35);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(R,G,B);

glVertex2i(604,31);

glVertex2i(696,31);

glVertex2i(696,4);

glVertex2i(604,4);

glVertex2i(604,31);

glEnd();

// line button in panel

glBegin(GL\_LINE\_STRIP);

glColor3f(0,0,0);

glVertex2i(110,60);

glVertex2i(90,45);

glEnd();

// Text button in panel

glBegin(GL\_LINE\_STRIP);

glColor3f(1,0,0);

glVertex2i(100,25);

glVertex2i(93,10);

glVertex2i(100,25);

glVertex2i(107,10);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3f(1,0,0);

glVertex2i(97,17);

glVertex2i(103,17);

glEnd();

// spray button in panel

glBegin(GL\_POLYGON);

glColor3f(1,0,0);

glVertex2i(52,42);

glVertex2i(68,42);

glVertex2i(68,55);

glVertex2i(52,55);

glVertex2i(52,42);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(0,0,0);

glVertex2i(57,55);

glVertex2i(57,60);

glVertex2i(64,60);

glVertex2i(64,55);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3f(1,0,0);

glVertex2i(59,60);

glVertex2i(70,65);

glEnd();

//calligraphy button in panel

glBegin(GL\_LINE\_STRIP);

glColor3f(0,0,0);

glVertex2i(68,30);

glVertex2i(68,20);

glVertex2i(52,15);

glVertex2i(52,5);

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3f(0,0,0);

glVertex2i(70,30);

glVertex2i(70,20);

glVertex2i(54,15);

glVertex2i(54,5);

glEnd();

//empty quad button in panel

glBegin(GL\_LINE\_STRIP);

glColor3f(0,0,0);

glVertex2i(130,60);

glVertex2i(150,60);

glVertex2i(150,45);

glVertex2i(130,45);

glVertex2i(130,60);

glEnd();

//filled quad button in panel

glBegin(GL\_POLYGON);

glColor3f(1,1,0);

glVertex2i(130,25);

glVertex2i(150,25);

glVertex2i(150,10);

glVertex2i(130,10);

glVertex2i(130,25);

glEnd();

//empty circle button in panel

glBegin(GL\_POINTS);

glColor3f(0,0,0);

for (float i=0; i<360; i++)

{

int x= 180 + 10\*cos(i);

int y= 55 + 10\*sin(i);

glVertex2i(x,y);

}

glEnd();

//filled circle button in panel

glBegin(GL\_POLYGON);

glColor3f(1,1,0);

for (float j=0; j<360; j++)

{

int x= 180 + 10\*cos(j);

int y= 20 + 10\*sin(j);

glVertex2i(x,y);

}

glEnd();

// triangle button in panel

glBegin(GL\_LINE\_STRIP);

glColor3f(0,0,0);

glVertex2i(220,60);

glVertex2i(210,50);

glVertex2i(230,50);

glVertex2i(220,60);

glEnd();

// Filled triangle button in panel

glBegin(GL\_POLYGON);

glColor3f(1,1,0);

glVertex2i(220,25);

glVertex2i(210,15);

glVertex2i(230,15);

glVertex2i(220,25);

glEnd();

char \*a;

a="CLEAR";

glColor3f(R,G,B);

if(R >0.8 && G>0.8 && B < 0.4)

glColor3f(0,0,0);

glRasterPos3f(610, 45,0);

for (int count=0; count <= 5; count++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, a[count]);

}

char \*c2;

if(flag==false)

{

c2="No Shape Selected";

int tlength = strlen(c2);

glRasterPos3f(300,45,0);

for (int counter=0; counter <=tlength; counter++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, c2[counter]);

}

flag=true;

}

else

{

c2="You have selected";

int tlength = strlen(c2);

glRasterPos3f(300,45,0);

for (int counter=0; counter <=tlength; counter++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, c2[counter]);

}

}

//drawing status in panel, shows what shape is selected and is being drawn

int tlength;

glColor3f(R,G,B);

if(R >0.8 && G>0.8 && B < 0.4)

glColor3f(0,0,0);

tlength = strlen(shape);

glRasterPos3f(300, 15,0);

for (int counter=0; counter <=tlength; counter++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, shape[counter]);

}

//this is the color selection object in the panel

glBegin(GL\_QUADS);

glColor3f(0.6,0,0.6);

glVertex2i(700,0);

glColor3f(1,0,0);

glVertex2i(700,70);

glColor3f(1,0.5,0);

glVertex2i(783,70);

glColor3f(0,0,1);

glVertex2i(783,0);

glColor3f(1,0.5,0);

glVertex2i(783,70);

glColor3f(0,0,1);

glVertex2i(783,0);

glColor3f(0,0,1);

glVertex2i(806,0);

glColor3f(1,1,0);

glVertex2i(806,70);

glColor3f(0,0,1);

glVertex2i(806,0);

glColor3f(1,1,0);

glVertex2i(806,70);

glColor3f(0,1,0);

glVertex2i(900,70);

glColor3f(0,0,0);

glVertex2i(900,0);

glEnd();

};//end of drawpanel()

//start of drawline() function, used to draw lines by using

// 2 clicks on the drawing window

void drawline(){

if(P1==true && P2==true)

{

glBegin(GL\_LINES);

glColor3f(R,G,B);

glVertex2i(X1,Y1);

glVertex2i(X2,Y2);

glEnd();

P1=false;

P2=false;

}

};//end of drawline()

//start of drawfilledquad() function, used to draw a solid

// rectangle or square

void drawfilledquad(){

if(P1==true && P2==true)

{

glBegin(GL\_QUADS);

glColor3f(R,G,B);

glVertex2i(X1,Y1);

glVertex2i(X1,Y2);

glVertex2i(X2,Y2);

glVertex2i(X2,Y1);

glEnd();

P1=false;

P2=false;

}

};//end of drawfilledquad()

//start of drawquad() function, used to draw a frame of a

// rectangle or a square

void drawquad(){

if(P1==true && P2==true)

{

glBegin(GL\_LINE\_STRIP);

glColor3f(R,G,B);

glVertex2i(X1,Y1);

glVertex2i(X1,Y2);

glVertex2i(X2,Y2);

glVertex2i(X2,Y1);

glVertex2i(X1,Y1);

glEnd();

P1=false;

P2=false;

}

};//end of drawquad()

//start of drawtriangle() function, used to draw a frame of a triangle

void drawtriangle(){

if(P1==true && P2==true && P3==true)

{

glBegin(GL\_LINE\_STRIP);

glColor3f(R,G,B);

glVertex2i(X1,Y1);

glVertex2i(X2,Y2);

glVertex2i(X3,Y3);

glVertex2i(X1,Y1);

glEnd();

P1=false;

P2=false;

P3=false;

}

};//end of drawtriangle()

//start of drawfilledtriangle() function, used to draw a solid trianlge

void drawfilledtriangle(){

if(P1==true && P2==true && P3==true)

{

glBegin(GL\_TRIANGLES);

glColor3f(R,G,B);

glVertex2i(X1,Y1);

glVertex2i(X2,Y2);

glVertex2i(X3,Y3);

glEnd();

P1=false;

P2=false;

P3=false;

}

};//end of drawfilledtriangle()

void drawfilledcircle(){

if(P1==true && P2==true)

{

glBegin(GL\_POLYGON);

glColor3f(R,G,B);

DX = X2 -X1;

DY = Y2 -Y1;

rad = sqrt((DX\*DX) + (DY\*DY));

for (float i=0; i<360; i=i+0.005)

{

int x= X1 + rad\*cos(i);

int y= Y1 + rad\*sin(i);

glVertex2i(x,y);

}

glEnd();

P1=false;

P2=false;

}

};//end of drawfilledcircle()

//start of drawcircle() function, used to create the frame of a

//circle, also uses lots of calculation for the quality of the

//output discarding the performance factor as stated in the

//drawfilledcircle() comment section

void drawcircle(){

if(P1==true && P2==true)

{

glBegin(GL\_POINTS);

glColor3f(R,G,B);

DX = X2 -X1;

DY = Y2 -Y1;

rad = sqrt((DX\*DX) + (DY\*DY));

for (float i=0; i<360; i=i+0.005)

{

int x= X1 + rad\*cos(i);

int y= Y1 + rad\*sin(i);

glVertex2i(x,y);

}

glEnd();

P1=false;

P2=false;

}

};//end of drawcircle()

void draw()

{

//the GL\_COLOR\_BUFFER\_BIT is cleared only once using this if

//statement, in order to keep the content of the drawing page

if(Cleared == false)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

Cleared = true;

}

//called everytime the draw function is called

drawpanel();

//in the statements below, the if statement checks if the object is selected

//using the boolean, and calls the appropriate function (the functions which are above)

//there if the functions have all the points needed to be drawn they r executed and

//drawn on the screen.

if(Circle == true)

drawcircle();

if(Circle2 == true)

drawfilledcircle();

if(Triangle == true)

drawtriangle();

if(Triangle2 == true)

drawfilledtriangle();

if (Quad==true)

drawquad();

if (Quad2==true)

drawfilledquad();

if (Line==true)

drawline();

glFlush();

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

}

void renderBitmapString(float x,float y,float z,void \*font, char \*string) {

char \*c;

c=string;

glColor3f(R,G,B);

glRasterPos3f(x, y,z);

for (int count=0; count <= keyboardindex; count++) {

glutBitmapCharacter(font, c[count]);

}

}//end of renderBitmapString

void processNormalKeys(unsigned char key, int x, int y) {

if(Text== true){

if (checked ==false)

{

LastX=X1;

LastY=Y1;

checked=true;

}

if (LastX != X1 || LastY !=Y1)

{

keyboardindex=0;

LastX=X1;

LastY=Y1;

}

keyboardbuffer[keyboardindex] = key;

renderBitmapString(X1,Y1,0,GLUT\_BITMAP\_TIMES\_ROMAN\_24, keyboardbuffer);

keyboardindex++;

glutPostRedisplay();

}

}//end of pocessNormalKeys

void motion(int mousex,int mousey)

{

#define num 20

GLintPaint okP;

GLintCalligraphy3 okC3;

GLintSpray sp;

GLintRandom rand,s;

GLintRubber rub;

GLint x=mousex;

GLint y=height-mousey;

if(okP.enbP==1)//\*\*\*\*\*\*\*\*\*Check for enable paint mode

if(Pencil==true){

glColor3f(R,G,B);

glRecti(x,y,x+okP.brushsizeP,y+okP.brushsizeP);}

if(okC3.enbC3==1)

{

if(calli==true){

glColor3f(R,G,B);

glBegin(GL\_LINES);

glVertex2i(x,y);

glVertex2i(x+okC3.brushsizeC3,y-7);

glVertex2i(x+7,y+okC3.brushsizeC3);

// glVertex2i(x+sqrt(x),y);

glEnd();}

}

if(sp.enbS==1)

{

if(spray==true){

glBegin(GL\_POINTS);

glColor3f(R,G,B);

glVertex2i(x+s.random(sp.brushsizeS+4),y+s.random(sp.brushsizeS+6));

glVertex2i(x+s.random(sp.brushsizeS+6),y+s.random(sp.brushsizeS+5));

glVertex2i(x+s.random(sp.brushsizeS+9),y+s.random(sp.brushsizeS+12));

glEnd();}

}

if(rub.enbR==1)

{

if(Eraser==true){

glColor3f(1,1,1);

glRecti(x,y,x+rub.brushsizeR,y+rub.brushsizeR);

glColor3f(0,0,0);}

}

glFlush();

}

void processMenuEvents(int option)

{

GLintPaint okP;

GLintCalligraphy3 okC3;

GLintErase er;

GLintSpray sp;

GLintRandom rand,s;

GLintRubber rub;

if(Cleared2 == false)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

Cleared2 = true;

}

switch (option) {

case RED :

R = 1.0;

G = 0.0;

B = 0.0;

break;

case GREEN :

R = 0.0;

G = 1.0;

B = 0.0;

break;

case BLUE :

R = 0.0;

G = 0.0;

B = 1.0;

break;

case WHITE :

R = 1.0;

G = 1.0;

B = 1.0;

break;

case BLACK :

R = 0.0;

G = 0.0;

B = 0.0;

break;

case QUAD :

Quad = true;

Pencil=false;

spray=false;

Eraser=false;

calli=false;

Quad2 = false;

Triangle= false;

Triangle2= false;

Line=true;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Quadrangle shape";

break;

case LINE :

Quad = false;

Pencil=false;

spray=false;

Eraser=false;

calli=false;

Quad2 = false;

Triangle= false;

Triangle2= false;

Line=true;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="line";

break;

case TRIANGLE :

Quad = false;

Pencil=false;

spray=false;

Eraser=false;

calli=false;

Quad2 = false;

Line= false;

Point= false;

Triangle= true;

Triangle2= false;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Triangle shape";

break;

case CIRCLE :

Quad = false;

Pencil=false;

spray=false;

Eraser=false;

calli=false;

Quad2 = false;

Line= false;

Triangle= false;

Triangle2= false;

Circle= true;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Circle shape";

break;

case CIRCLE2 :

Quad = false;

Pencil=false;

spray=false;

Eraser=false;

calli=false;

Quad2 = false;

Line= false;

Triangle= false;

Triangle2= false;

Circle= false;

Circle2= true;

P1=false;P2=false;P3=false;

Text=false;

shape="Circle shape ( Filled )";

break;

case QUAD2 :

Quad = false;

Pencil=false;

spray=false;

Eraser=false;

calli=false;

Quad2 = true;

Line= false;

Triangle= false;

Triangle2= false;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Quadrangle shape ( Filled )";

break;

case TRIANGLE2 :

Quad = false;

Pencil=false;

spray=false;

Eraser=false;

calli=false;

Quad2 = false;

Line= false;

Triangle= false;

Triangle2= true;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Triangle shape ( Filled )";

break;

case PENCIL:

Pencil=true;

Quad = false;

spray=false;

Eraser=false;

calli=false;

Quad2 = false;

Line= false;

Triangle= false;

Triangle2= false;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Pencil";

glColor3f(R,G,B);

okC3.enableC3(0);

sp.enableS(0);

rub.enableR(0);

okP.enableP(1);

break;

case ERASER:

Quad = false;

Pencil=false;

spray=false;

Eraser=true;

calli=false;

Quad2 = false;

Line= false;

Point= false;

Triangle= false;

Triangle2= false;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Eraser";

okP.enableP(0);

okC3.enableC3(0);

sp.enableS(0);

rub.enableR(1);

case '9':

rub.brushsizeR++;

break;

case '0':

if(rub.brushsizeR>=5)

rub.brushsizeR--;

break;

break;

case SPRAY:

Quad = false;

Pencil=false;

spray=true;

Eraser=false;

calli=false;

Quad2 = false;

Line= false;

Triangle= false;

Triangle2= false;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Sprayer";

okP.enableP(0);

okC3.enableC3(0);

rub.enableR(0);

sp.enableS(1);

break;

case CALLIGRAPH :

Quad = false;

Pencil=false;

spray=false;

Eraser=false;

calli=true;

Quad2 = false;

Line= false;

Triangle= false;

Triangle2= false;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=false;

shape="Calligraphy";

okP.enableP(0);

rub.enableR(0);

sp.enableS(0);

okC3.enableC3(1);

break;

case TEXT :

Quad = false;

Pencil=false;

spray=false;

Eraser=false;

calli=false;

Quad2 = false;

Line= false;

Triangle= false;

Triangle2= false;

Circle= false;

Circle2= false;

P1=false;P2=false;P3=false;

Text=true;

shape="Text mode";

break;

case CLEAR :

glClear(GL\_COLOR\_BUFFER\_BIT);

P1= false,P2=false, P3= false;

break;

}

glutPostRedisplay();

}

void mouse (int btn, int state, int x, int y)

{

//the if statment that listens to events by the left mouse click in

//the drawing area

if (btn == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && 650-y > 70)

{

//this if statement is entered if we select any of the shapes in the if condition

//they all share the same if coz they all need 2 points to be drawn

if(Quad == true || Line == true || Circle == true || Circle2==true || Quad2 == true)

{

if(P1==true && P2==false)

{

X2=x;Y2=650-y;

P2=true;

}

if(P1==false)

{

X1=x;Y1=650-y;

P1=true;

}

}//end of quad,quad2,line,circle,circle2 point selection

//start of point if

if (Pencil == true)

{

if(P1==false)

{

X1=x;Y1=650-y;

printf("x=%d,y=%d",x,y);

P1=true;

}

}//of point point selection

//start of text if

if (Text==true)

{X1=x;

Y1=650-y;}//end of text if

//both solid and frame triangles need 3 points that they get using this

//if statements

if (Triangle == true || Triangle2 == true)

{

if(P1==true && P2==true && P3==false)

{

X3=x;Y3=650-y;

P3=true;

}

if(P1==true && P2==false)

{

X2=x;Y2=650-y;

P2=true;

}

if(P1==false)

{

X1=x;Y1=650-y;

P1=true;

}

}//end of triangle point collection

}//end of drawing area listener if statement

if( x > 0 && x < 40 && 650-y > 35 && 650-y < 70 )

processMenuEvents(PENCIL); // pencil button

if( x > 0 && x < 40 && 650-y > 0 && 650-y < 35 ){

processMenuEvents(ERASER); }// eraser button

if( x > 40 && x < 80 && 650-y > 35 && 650-y < 70 ){

processMenuEvents(SPRAY); }// spray button

if( x > 40 && x < 80 && 650-y > 0 && 650-y < 35 ){

processMenuEvents(CALLIGRAPH); }// caliigraphy button

if( x > 80 && x < 120 && 650-y > 35 && 650-y < 70 ){

processMenuEvents(LINE); }// caliigraphy button

if( x > 80 && x < 120 && 650-y > 0 && 650-y < 35 ){

processMenuEvents(TEXT); }// text button

if( x > 120 && x < 160 && 650-y > 35 && 650-y < 70 ){

processMenuEvents(QUAD); }// quadrangle button

if( x > 120 && x < 160 && 650-y > 0 && 650-y < 35 ){

processMenuEvents(QUAD2); }// filled quadrangle button

if( x > 160 && x < 200 && 650-y > 35 && 650-y < 70 ){

processMenuEvents(CIRCLE); }// circle button

if( x > 160 && x < 200 && 650-y > 0 && 650-y < 35 ){

processMenuEvents(CIRCLE2); }// filled circle button

if( x > 200 && x < 240 && 650-y > 35 && 650-y < 70 ){

processMenuEvents(TRIANGLE); }// triangle button

if( x > 200 && x < 240 && 650-y > 0 && 650-y < 35 ){

processMenuEvents(TRIANGLE2); }// filled triangle button

if( x > 600 && x < 700 && 650-y > 35 && 650-y < 70 ){

processMenuEvents(CLEAR); }// clear button

if (btn == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && 650-y < 70 && x > 700 )

{

glReadPixels(x,650-y,2,2,GL\_RED,GL\_FLOAT,&R);

glReadPixels(x,650-y,2,2,GL\_GREEN,GL\_FLOAT,&G);

glReadPixels(x,650-y,2,2,GL\_BLUE,GL\_FLOAT,&B);

}//end of color selection listener

glutPostRedisplay();

}

void createGLUTMenus() {

int menu,submenu,submenu2;

submenu = glutCreateMenu(processMenuEvents);

glutAddMenuEntry("Red",RED);

glutAddMenuEntry("Green",GREEN);

glutAddMenuEntry("Blue",BLUE);

glutAddMenuEntry("White",WHITE);

glutAddMenuEntry("Black",BLACK);

submenu2 = glutCreateMenu(processMenuEvents);

glutAddMenuEntry("Pencil",PENCIL);

glutAddMenuEntry("Eraser",ERASER);

glutAddMenuEntry("Sprayer",SPRAY);

glutAddMenuEntry("Calligarahy",CALLIGRAPH);

glutAddMenuEntry("Line",LINE);

glutAddMenuEntry("Text",TEXT);

glutAddMenuEntry("Quad",QUAD);

glutAddMenuEntry("Quad (Filled)",QUAD2);

glutAddMenuEntry("Circle",CIRCLE);

glutAddMenuEntry("Circle (Filled)",CIRCLE2);

glutAddMenuEntry("Triangle",TRIANGLE);

glutAddMenuEntry("Triangle (Filled)",TRIANGLE2);

menu = glutCreateMenu(processMenuEvents);

glutAddSubMenu("Colors",submenu);

glutAddSubMenu("Shapes",submenu2);

glutAddMenuEntry("Clear",CLEAR);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}//end of createGLUTMenus

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

{

glutInit(&argc, argv);//must include

glutInitWindowSize(900,650);

glutInitWindowPosition(5,10);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);//display mode. single for 2d. RGB colors

glutCreateWindow("Open Designer");

glutDisplayFunc(display); createGLUTMenus();

glutMotionFunc(motion);

glutMouseFunc(mouse);

glutKeyboardFunc(processNormalKeys);

glClearColor(1,1,1,1);

glutDisplayFunc(draw);

myInit();

glutMainLoop();

return EXIT\_SUCCESS;

}