**OOPCGL (Mini project Report)**

**SE AIDS**

**BATCH : S-1 S-2 & S-3**

**Roll no. :**

**1. 22526**

**2. 22534**

**3. 22566**

**4. 22533**

**TITLE : [**Design and implement game / animation clip / Graphics Editor using open source graphics library. Make use of maximum features of Object Oriented Programming.**]**

**Aim :** To Implement basic methods of Computer Graphics and Object Oriented Programming In a Graphics Editor. The use of open source graphics libraries is recommended.

## Pre-requisite :

1. Basic programming skills of C++ & OpenGL
2. Machine with processor i5 and above , RAM
3. C++ IDE (Dev , Visual Studio etc)
4. Open-source programming tool like (G++, GCC,OpenGL)

## Introduction About Project

The Aim was to Implement the concepts of the Object Oriented Programming & Computer Graphics To create a simple program is a simple paint tool which is similar to the Paint of Microsoft . It is written in the language C++ with OpenGL/GLUT.

1. **Colour** -> You can choose Red, Green, Blue, Yellow or Random, the default color is Red.
2. **Shapes** -> The default shape is Point.
   * Point -> draw a dot at the point clicked with the mouse. Clicking and dragging will draw points constantly like free-form drawing.
   * Line -> draw a line between two subsequently clicked points. (Bresenham's algorithm is used here.)
   * Rectangle -> draw a rectangle with top-left corner specified by the first click and the bottom-right corner specified by a second click. If the second click is bottom-left, top-right or top-left comparing to the first click, a warning will show in the console.
   * Circle -> draw a circle centered at the position of the first click, with its radius set by a second click.
   * Airbrush -> draw multiple points as brush around the clicked point. There are four options of size.
3. **Radical Paint Brush** -> draw multiple points as brush around the centre point of the document. You can choose True or False. Only useful when shape is point. The default option is false. After choosing point in shape, it becomes False.
4. **Eraser** -> erase the points by clicking and dragging.
5. **Undo** -> undo, the history can keep maximum 20 records.
6. **Redo** -> redo, the history can keep maximum 20 records.
7. **Clear** -> clear all the points and clear the history.
8. **Quit** -> close the window.

**Keyboard events:**

1. **q** or **esc** -> close the window.
2. **c** -> clear all the points and clear the history.
3. **+** -> larger size of eraser or brush.
4. **-** -> smaller size of eraser or brush.
5. **u** -> undo, the history can keep maximum 20 records.
6. **r** -> redo, the history can keep maximum 20 records.

# Introduction to OpenGL:

1. **Header files explanation:**
   1. **#include<stdio.h>** : a header file which has the necesaary information to include the input/output related functions in our program. Ex : printf , scanf etc.
   2. **#include<time.h>** : a header file which contains definitions of functions to get and manipulate date and time information.
   3. **#include ”dot.h”** : a header file which contains a part of program.
   4. **.#include<iostream>** : tells the preprocessor to include these header file in the program
   5. **#include<cmath>** : a header file which contains numeric library.
   6. **#include<string.h>** : to manipulate string
   7. **#include<vector>** : a header file which is part of the containers library.
   8. **#include<list>** : a header file which is part of the containers library.
   9. **#include<windows.h>** : windows declaration file.

# 2)OpenGL functions :

## Open GL is a cross language appliation programming interface (API) for rendering 2D and 3D vector graphics, The API is typically used to interact with a graphics processing unit(GPU) to achieve hardware accelerated rendering.

a**.#include<openGL/openGL.h>** : refers to a set of utility functions that make some OpenGL operations easier to program.It is standard on all OpenGL b**.#include<GLUT/glut.h>** : is the base OpenGL headers which give you the OpenGL 1.1 version includes function and tokens declarations and many more

## Implementation (code) :

#include <time.h>

#include "Dot.h"

#include <iostream>

#include <cmath>

#include <string>

#include <vector>

#include <list>

#include <windows.h>

#include <OpenGL/gl.h>

#include <OpenGL/glu.h>

#include <GLUT/glut.h>

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/freeglut.h>

using namespace std;

float red = 1.0, green = 0.0, blue = 0.0;

int tmpx, tmpy; // store the first point when shape is line, rectangle or circle

int brushSize = 4;

int eraserSize = 1;

bool isSecond = false;

bool isRandom = false;

bool isEraser = false;

bool isRadial = false;

float window\_w = 500;

float window\_h = 500;

int shape = 1; // 1:point, 2:line, 3:rectangle, 4:circle, 5:brush

std::vector<Dot> dots; // store all the points until clear

std::list<int> undoHistory; // record for undo, maximum 20 shapes in history

std::list<int> redoHistory; // record for redo, maximum 20 shapes in history

std::vector<Dot> redoDots; // store the dots after undo temporaly

void display(void)

{

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glPointSize(2);

glBegin(GL\_POINTS);

for (unsigned int i = 0; i < dots.size(); i++)

{

glColor3f(dots[i].getR(), dots[i].getG(), dots[i].getB());

glVertex2i(dots[i].getX(), dots[i].getY());

}

glEnd();

glutSwapBuffers();

}

void clear()

{

dots.clear();

undoHistory.clear();

redoDots.clear();

redoHistory.clear();

glClear(GL\_COLOR\_BUFFER\_BIT);

glutSwapBuffers();

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

std::cout << asctime(timeinfo)

<< "[Info] The window and the history are cleared successfully.\n";

}

void quit()

{

std::cout << "Thank you for using this Paint tool! Goodbye!" << std::endl;

exit(0);

}

void undo()

{

if (undoHistory.size() > 0)

{

if (undoHistory.back() != dots.size() && redoHistory.back() != dots.size())

{

redoHistory.push\_back(dots.size());

}

int numRemove = dots.size() - undoHistory.back();

for (int i = 0; i < numRemove; i++)

{

redoDots.push\_back(dots.back());

dots.pop\_back();

}

redoHistory.push\_back(undoHistory.back());

undoHistory.pop\_back();

}

else

{

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

std::cout << asctime(timeinfo)

<< "[Warning] Cannot undo. This is the first record in the history.\n";

}

}

void redo()

{

if (redoHistory.size() > 1)

{

undoHistory.push\_back(redoHistory.back());

redoHistory.pop\_back();

int numRemove = redoHistory.back() - dots.size();

for (int i = 0; i < numRemove; i++)

{

dots.push\_back(redoDots.back());

redoDots.pop\_back();

}

}

else

{

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

std::cout << asctime(timeinfo)

<< "[Warning] Cannot redo. This is the last record in the history.\n";

}

}

void drawDot(int mousex, int mousey)

{

Dot newDot(mousex, window\_h - mousey, isEraser ? 1.0 : red, isEraser ? 1.0 : green, isEraser ? 1.0 : blue);

dots.push\_back(newDot);

}

void drawBrush(int x, int y)

{

for (int i = 0; i < brushSize; i++)

{

int randX = rand() % (brushSize + 1) - brushSize / 2 + x;

int randY = rand() % (brushSize + 1) - brushSize / 2 + y;

drawDot(randX, randY);

}

}

void drawLine(int x1, int y1, int x2, int y2)

{

bool changed = false;

// Bresenham's line algorithm is only good when abs(dx) >= abs(dy)

// So when abs(dx) < abs(dy), change axis x and y

if (abs(x2 - x1) < abs(y2 - y1))

{

int tmp1 = x1;

x1 = y1;

y1 = tmp1;

int tmp2 = x2;

x2 = y2;

y2 = tmp2;

changed = true;

}

int dx = x2 - x1;

int dy = y2 - y1;

int yi = 1;

int xi = 1;

if (dy < 0)

{

yi = -1;

dy = -dy;

}

if (dx < 0)

{

xi = -1;

dx = -dx;

}

int d = 2 \* dy - dx;

int incrE = dy \* 2;

int incrNE = 2 \* dy - 2 \* dx;

int x = x1, y = y1;

if (changed)

drawDot(y, x);

else

drawDot(x, y);

while (x != x2)

{

if (d <= 0)

d += incrE;

else

{

d += incrNE;

y += yi;

}

x += xi;

if (changed)

drawDot(y, x);

else

drawDot(x, y);

}

}

/\*\*

\* We can use drawLine function to draw the rectangle

\*

\* Top-left corner specified by the first click,

\* and the bottom-right corner specified by a second click

\*/

void drawRectangle(int x1, int y1, int x2, int y2)

{

if (x1 < x2 && y1 < y2)

{

drawLine(x1, y1, x2, y1);

drawLine(x2, y1, x2, y2);

drawLine(x2, y2, x1, y2);

drawLine(x1, y2, x1, y1);

}

else

{

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

std::cout << asctime(timeinfo)

<< "[Warning] The first click should be the top-left corner, the second click should be bottom-right corner.\n";

}

}

/\*\*

\* Midpoint circle algorithm

\*/

void drawCircle(int x1, int y1, int x2, int y2)

{

int r = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));

double d;

int x, y;

x = 0;

y = r;

d = 1.25 - r;

while (x <= y)

{

drawDot(x1 + x, y1 + y);

drawDot(x1 - x, y1 + y);

drawDot(x1 + x, y1 - y);

drawDot(x1 - x, y1 - y);

drawDot(x1 + y, y1 + x);

drawDot(x1 - y, y1 + x);

drawDot(x1 + y, y1 - x);

drawDot(x1 - y, y1 - x);

x++;

if (d < 0)

{

d += 2 \* x + 3;

}

else

{

y--;

d += 2 \* (x - y) + 5;

}

}

}

void drawRadialBrush(int x, int y)

{

int xc = glutGet(GLUT\_WINDOW\_WIDTH) / 2;

int yc = glutGet(GLUT\_WINDOW\_HEIGHT) / 2;

int dx, dy;

dx = xc - x;

dy = yc - y;

drawDot(xc + dx, yc + dy);

drawDot(xc - dx, yc + dy);

drawDot(xc + dx, yc - dy);

drawDot(xc - dx, yc - dy);

drawDot(xc + dy, yc + dx);

drawDot(xc - dy, yc + dx);

drawDot(xc + dy, yc - dx);

drawDot(xc - dy, yc - dx);

}

void erase(int x, int y)

{

for (int i = -eraserSize; i <= eraserSize; i++)

{

for (int j = -eraserSize; j <= eraserSize; j++)

{

drawDot(x + i, y + j);

}

}

}

void keyboard(unsigned char key, int xIn, int yIn)

{

isSecond = false;

switch (key)

{

case 'q':

case 27: // 27 is the esc key

quit();

break;

case 'c':

clear();

break;

case '+':

if (shape == 5 && !isEraser)

{

if (brushSize < 16)

brushSize += 4;

else

{

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

std::cout << asctime(timeinfo)

<< "[Warning] Airbrush's size cannot be larger. It is already the largest.\n";

}

}

else if (isEraser)

{

if (eraserSize < 10)

eraserSize += 4;

else

{

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

std::cout << asctime(timeinfo)

<< "[Warning] Eraser's size cannot be larger. It is already the largest.\n";

}

}

break;

case '-':

if (shape == 5 && !isEraser)

{

if (brushSize > 4)

brushSize -= 4;

else

{

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

std::cout << asctime(timeinfo)

<< "[Warning] Airbrush's size cannot be smaller. It is already the smallest.\n";

}

}

else if (isEraser)

{

if (eraserSize > 2)

eraserSize -= 4;

else

{

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

std::cout << asctime(timeinfo)

<< "[Warning] Eraser's size cannot be smaller. It is already the smallest.\n";

}

}

break;

case 'u':

undo();

break;

case 'r':

redo();

break;

}

}

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

{

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

{

if (isRandom)

{

srand(time(NULL));

red = float(rand()) / float(RAND\_MAX);

green = float(rand()) / float(RAND\_MAX);

blue = float(rand()) / float(RAND\_MAX);

}

if (isEraser)

{

undoHistory.push\_back(dots.size());

erase(x, y);

}

else

{

if (shape == 1)

{

undoHistory.push\_back(dots.size());

if (isRadial)

drawRadialBrush(x, y);

else

drawDot(x, y);

}

else if (shape == 5)

{

undoHistory.push\_back(dots.size());

drawBrush(x, y);

}

else

{

if (!isSecond)

{

tmpx = x;

tmpy = y;

isSecond = true;

}

else

{

if (undoHistory.back() != dots.size())

undoHistory.push\_back(dots.size());

if (shape == 2)

drawLine(tmpx, tmpy, x, y);

else if (shape == 3)

drawRectangle(tmpx, tmpy, x, y);

else if (shape == 4)

drawCircle(tmpx, tmpy, x, y);

isSecond = false;

}

}

}

if (undoHistory.size() > 20)

{

undoHistory.pop\_front();

}

}

}

void motion(int x, int y)

{

if (isEraser)

erase(x, y);

else

{

if (shape == 1)

{

if (isRadial)

drawRadialBrush(x, y);

else

drawDot(x, y);

}

if (shape == 5)

drawBrush(x, y);

}

}

void reshape(int w, int h)

{

window\_w = w;

window\_h = h;

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, w, 0, h);

glMatrixMode(GL\_MODELVIEW);

glViewport(0, 0, w, h);

}

void processMainMenu(int value)

{

switch (value)

{

case 0:

quit();

break;

case 1:

clear();

break;

case 2:

undo();

break;

case 3:

redo();

break;

}

}

void processBrushSizeMenu(int value)

{

shape = 5;

isEraser = false;

brushSize = value;

glutSetCursor(GLUT\_CURSOR\_CROSSHAIR);

}

void processColourMenu(int value)

{

isSecond = false;

isEraser = false;

isRandom = false;

switch (value)

{

case 1: // red

red = 1.0;

green = 0.0;

blue = 0.0;

break;

case 2: // green

red = 0.0;

green = 1.0;

blue = 0.0;

break;

case 3: // blue

red = 0.0;

green = 0.0;

blue = 1.0;

break;

case 4: // purple

red = 0.5;

green = 0.0;

blue = 0.5;

break;

case 5: // yellow

red = 1.0;

green = 1.0;

blue = 0.0;

break;

case 6: // random

isRandom = true;

break;

}

}

void processShapeMenu(int value)

{

shape = value;

isEraser = false;

isSecond = false;

isRadial = false;

switch (shape)

{

case 1:

glutSetCursor(GLUT\_CURSOR\_RIGHT\_ARROW);

break;

case 2:

case 3:

case 4:

glutSetCursor(GLUT\_CURSOR\_CROSSHAIR);

break;

}

}

void processEraserSizeMenu(int value)

{

glutSetCursor(GLUT\_CURSOR\_RIGHT\_ARROW);

eraserSize = value;

isEraser = true;

}

void processRadicalBrushMenu(int value)

{

isRadial = value == 1 ? true : false;

}

void createOurMenu()

{

int colourMenu = glutCreateMenu(processColourMenu);

glutAddMenuEntry("Red", 1);

glutAddMenuEntry("Green", 2);

glutAddMenuEntry("Blue", 3);

glutAddMenuEntry("Purple", 4);

glutAddMenuEntry("Yellow", 5);

glutAddMenuEntry("Random", 6);

int sizeMenu = glutCreateMenu(processBrushSizeMenu);

glutAddMenuEntry("4px", 4);

glutAddMenuEntry("8px", 8);

glutAddMenuEntry("12px", 12);

glutAddMenuEntry("16px", 16);

int shapeMenu = glutCreateMenu(processShapeMenu);

glutAddMenuEntry("Point", 1);

glutAddMenuEntry("Line", 2);

glutAddMenuEntry("Rectangle", 3);

glutAddMenuEntry("Circle", 4);

glutAddSubMenu("Airbrush", sizeMenu);

int eraserSizeMenu = glutCreateMenu(processEraserSizeMenu);

glutAddMenuEntry("Small", 2);

glutAddMenuEntry("Medium", 6);

glutAddMenuEntry("Large", 10);

int radicalBrushMenu = glutCreateMenu(processRadicalBrushMenu);

glutAddMenuEntry("True", 1);

glutAddMenuEntry("False", 2);

int main\_id = glutCreateMenu(processMainMenu);

glutAddSubMenu("Colour", colourMenu);

glutAddSubMenu("Shapes", shapeMenu);

glutAddSubMenu("Radical Paint Brush", radicalBrushMenu);

glutAddSubMenu("Eraser", eraserSizeMenu);

glutAddMenuEntry("Undo", 2);

glutAddMenuEntry("Redo", 3);

glutAddMenuEntry("Clear", 1);

glutAddMenuEntry("Quit", 0);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

void init(void)

{

/\* background color \*/

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(red, green, blue);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, window\_w, 0.0, window\_h);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void FPS(int val)

{

glutPostRedisplay();

glutTimerFunc(0, FPS, 0);

}

void callbackInit()

{

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutMouseFunc(mouse);

glutMotionFunc(motion);

glutTimerFunc(17, FPS, 0);

}

void printGuide()

{

std::cout << "#########################################################################\n"

<< "# Welcome to use this Paint tool! #\n"

<< "#########################################################################\n"

<< "A list of commands:\n"

<< "Right click\t"

<< "-> show menu\n"

<< "Left click\t"

<< "-> choose option\n"

<< "Menu \"Colour\"\t"

<< "-> You can choose Red, Green, Blue, Yellow or Random, the default color is Red.\n"

<< "Menu \"Shapes\"\t"

<< "-> The default shape is Point.\n"

<< "\tPoint\t\t"

<< "-> draw a dot at the point clicked with the mouse. Clicking and dragging will draw points constantly like free-form drawing.\n"

<< "\tLine\t\t"

<< "-> draw a linebetween two subseauently clicked points. (Bresenham's algorithm is used here.)\n"

<< "\tRectangle\t"

<< "-> draw a rectangle with top-left corner specified by the first click and the bottom-right corner specified by a second click. If the second click is bottom-left, top-right or top-left comparing to the first click, a warning will show in the console.\n"

<< "\tCircle\t\t"

<< "-> draw a circle centered at the position of the first click, with its radius set by a second click.\n"

<< "\tAirbrush\t"

<< "-> draw multiple points as brush around the clicked point. There are four options of size.\n"

<< "Menu \"Radical..\""

<< "-> draw multiple points as brush around the centre point of the document. You can choose True or False. Only useful when shape is point. The default option is false. After choosing point in shape, it becomes False.\n"

<< "Menu \"Eraser\"\t"

<< "-> erase the points by clicking and dragging.\n"

<< "Menu \"Undo\"\t"

<< "-> undo, the history can keep maximum 20 records.\n"

<< "Menu \"Redo\"\t"

<< "-> redo, the history can keep maximum 20 records.\n"

<< "Menu \"Clear\"\t"

<< "-> clear all the points and clear the history.\n"

<< "Menu \"Quit\"\t"

<< "-> close the window.\n"

<< "Keyboard 'q'\t"

<< "-> close the window.\n"

<< "Keyboard 'esc'\t"

<< "-> close the window.\n"

<< "Keyboard 'c'\t"

<< "-> clear all the points and clear the history.\n"

<< "Keyboard '+'\t"

<< "-> larger size of eraser or brush.\n"

<< "Keyboard '-'\t"

<< "-> smaller size of eraser or brush.\n"

<< "Keyboard 'u'\t"

<< "-> undo, the history can keep maximum 20 records.\n"

<< "Keyboard 'r'\t"

<< "-> redo, the history can keep maximum 20 records.\n"

<< "################################# Paint #################################" << std::endl;

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE);

glutInitWindowSize(window\_w, window\_h);

glutInitWindowPosition(100, 100);

glutCreateWindow("Paint");

callbackInit();

init();

printGuide();

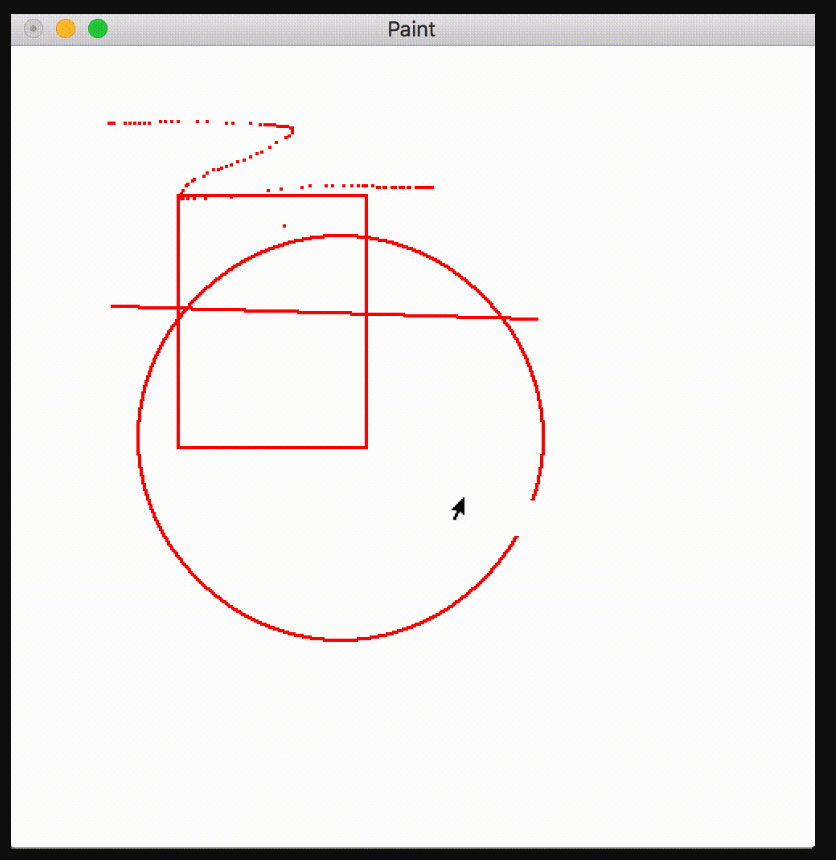
createOurMenu();

glutMainLoop();

return (0);

}

## Output :

****

**Conclusion :**

## Hence we have Implemented a Mini Project BitPaint for a Computer Graphics & Object Oriented Programming using Open GL and Graphics Library.