**SSN COLLEGE OF ENGINEERING, KALAVAKKAM**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UCS1712 – GRAPHICS AND MULTIMEDIA LAB**

**VISHAL N**

**185001198**

**CSEC**

**13.10.2021**

**3. Draw a scene depicting sunrise and apply appropriate 2D composite transformations to show the positions of the sun every three hours with its reflection.**

Test\_file.cpp:

# pragma warning(disable:4996)

# include <GL/glut.h>

# include <stdlib.h>

# include <stdio.h>

# include <vector>

# include <iostream>

using namespace std;

void myInit(void) {

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor3f(0.0f, 1.0f, 0.0f);

glPointSize(1);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-600, 600, -600, 600);

}

void drawAxis() {

glClearColor(1.0, 1.0, 1.0, 0.0);

glColor3f(0.0f, 1.0f, 0.0f);

glBegin(GL\_LINES);

glVertex2d(-600, 0);

glVertex2d(600, 0);

glEnd();

glBegin(GL\_LINES);

glVertex2d(0, -600);

glVertex2d(0, 600);

glEnd();

}

void draw\_pixel(int x, int y) {

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

vector<double> multiply(vector<vector<double>> mat1, vector<double> mat2) {

// vector<vector<double>> product (3, vector<double> (3, 0));

vector<double> product(3, 0);

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

product[i] += mat1[i][j] \* mat2[j];

}

}

return product;

}

vector<double> translate(vector<double> P, double tx, double ty) {

vector<vector<double>> T(3, vector<double>(3, 0));

vector<double> product;

for (int i = 0; i < 3; i++) {

T[i][i] = 1;

}

T[0][2] = tx;

T[1][2] = ty;

product = multiply(T, P);

return product;

}

vector<double> reflection(vector<double> P) {

vector<vector<double>> R(3, vector<double>(3, 0));

vector<double> product;

for (int i = 0; i < 3; i++) {

R[i][i] = 1;

}

R[1][1] = -1;

product = multiply(R, P);

return product;

}

void drawCircle(vector<double> centre, int r) {

glClearColor(1.0, 1.0, 1.0, 0.0);

glColor3f(0.0f, 0.0f, 1.0f);

int x\_centre = centre[0];

int y\_centre = centre[1];

int x = r, y = 0;

if (r > 0)

{

draw\_pixel(x + x\_centre, -y + y\_centre);

draw\_pixel(y + x\_centre, x + y\_centre);

draw\_pixel(-y + x\_centre, x + y\_centre);

}

int P = 1 - r;

while (x > y)

{

y++;

if (P <= 0)

P = P + 2 \* y + 1;

else

{

x--;

P = P + 2 \* y - 2 \* x + 1;

}

if (x < y)

break;

draw\_pixel(x + x\_centre, y + y\_centre);

draw\_pixel(-x + x\_centre, y + y\_centre);

draw\_pixel(x + x\_centre, -y + y\_centre);

draw\_pixel(-x + x\_centre, -y + y\_centre);

if (x != y)

{

draw\_pixel(y + x\_centre, x + y\_centre);

draw\_pixel(-y + x\_centre, x + y\_centre);

draw\_pixel(y + x\_centre, -x + y\_centre);

draw\_pixel(-y + x\_centre, -x + y\_centre);

}

}

}

void myDisplay(void) {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawAxis();

int radius = 50;

vector<double> centre = { -450, 100, 1 };

// Circle 1

drawCircle(centre, radius);

vector<double> transformed = reflection(centre);

drawCircle(transformed, radius);

// Circle 2

vector<double> translated = translate(centre, 150, 150);

drawCircle(translated, radius);

transformed = reflection(translated);

drawCircle(transformed, radius);

// Circle 3

translated = translate(translated, 150, 150);

drawCircle(translated, radius);

transformed = reflection(translated);

drawCircle(transformed, radius);

// Circle 4

translated = translate(translated, 150, 150);

drawCircle(translated, radius);

transformed = reflection(translated);

drawCircle(transformed, radius);

// Circle 5

translated = translate(translated, 150, -150);

drawCircle(translated, radius);

transformed = reflection(translated);

drawCircle(transformed, radius);

// Circle 6

translated = translate(translated, 150, -150);

drawCircle(translated, radius);

transformed = reflection(translated);

drawCircle(transformed, radius);

// Circle 7

translated = translate(translated, 150, -150);

drawCircle(translated, radius);

transformed = reflection(translated);

drawCircle(transformed, radius);

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(600, 600);

glutCreateWindow("Sunrise");

glutDisplayFunc(myDisplay);

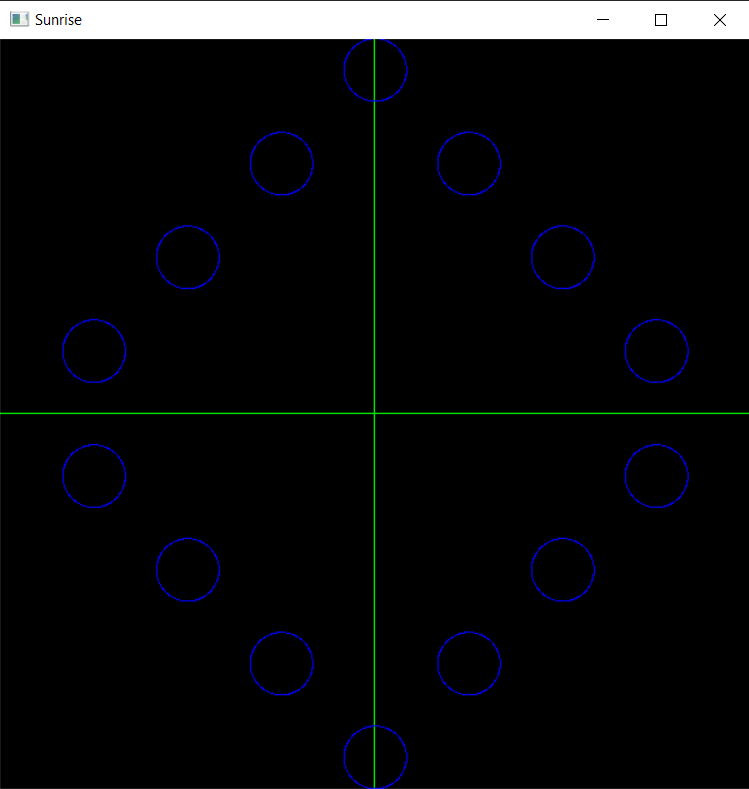
myInit();

glutMainLoop();

return 0;

}

**OUTPUTS:**

****