U18CO018
Shubham Shekhaliya
Assignment – 6

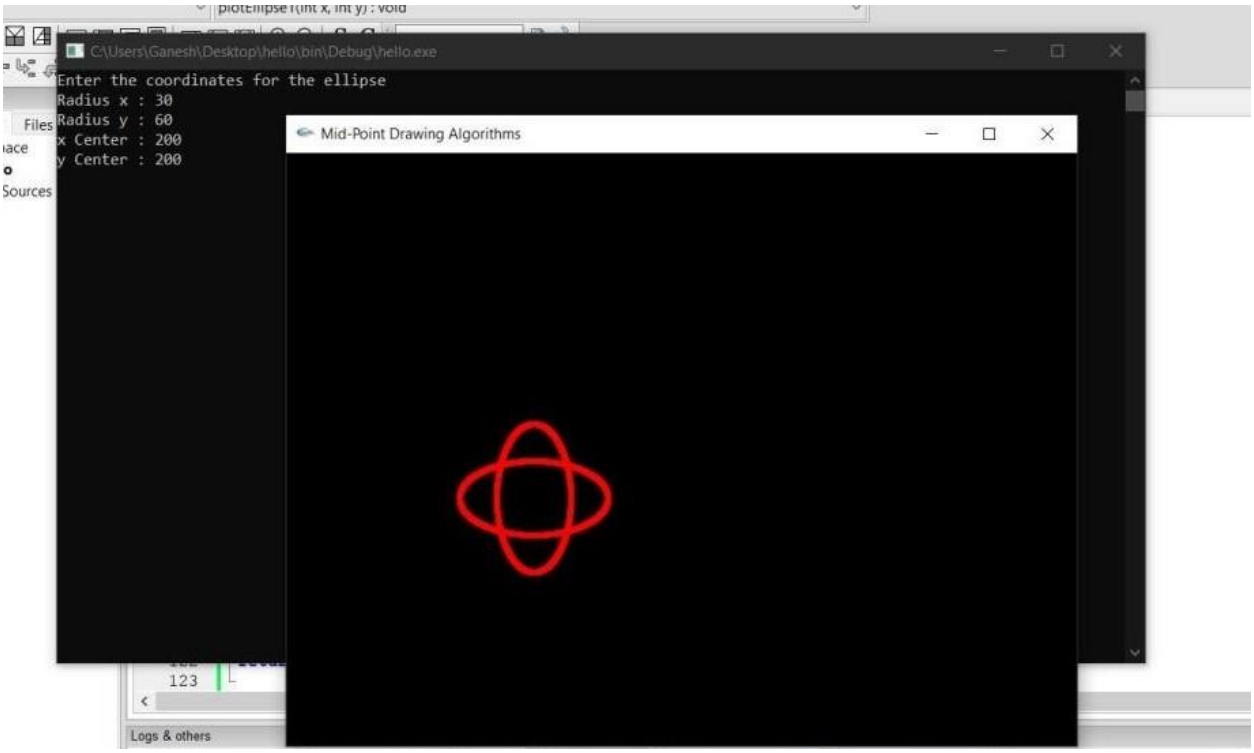
1 - > Code

```
#include <stdio.h>
#include <windows.h>
#include <iostream> #include <GL/glut.h> using namespace std;
int r, rx, ry, xc, yc;

void plotEllipse1(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2i(x + xc, y + yc);
    glEnd();
    glFlush();
}

void plotEllipse2(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2i(x + xc, y + yc);
    glEnd();
    glFlush();
}

void myInit(void) {
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(5.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void midPointEllipseAlgo() {
    float dx, dy, d1, d2, x, y;
    x = 0;
    y = ry;
    // Initial decision parameter of region 1
    d1 = (ry * ry) - (rx * rx * ry) + (0.25 * rx * rx);
    dx = 2 * ry * ry * x;
    dy = 2 * rx * rx * y; // For region 1
    while (dx < dy) {
        // Print points based on 4-way symmetry
```

```
        plotEllipse1(x, y);
        plotEllipse1(-x,
                     y);
        plotEllipse1(x, -y);
        plotEllipse1(-x, -y);
        plotEllipse2(y, x);
        plotEllipse2(-y,
                     x);
        plotEllipse2(y, -x);
        plotEllipse2(-y, -x);
        // Checking and updating value of // decision parameter based on algorith
m
        if (d1 < 0) {
            x++;
            dx = dx + (2 * ry * ry);
            d1 = d1 + dx + (ry * ry);
        }
        else {
            x++;
            y--;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            d1 = d1 + dx - dy + (ry * ry);
        }
    }
    // Decision parameter of region 2
    d2 = ((ry * ry) * ((x + 0.5) * (x + 0.5))) +
         ((rx * rx) * ((y - 1) * (y - 1))) -
         (rx * rx * ry * ry); // Plotting points of region 2
    while (y >= 0) {
        // Print points based on 4-way symmetry
        plotEllipse1(x, y);
        plotEllipse1(-x,
                     y);
        plotEllipse1(x, -y);
        plotEllipse1(-x, -y);
        plotEllipse2(y, x);
        plotEllipse2(-y,
                     x);
        plotEllipse2(y, -x);
        plotEllipse2(-y, -x);
        // Checking and updating parameter // value based on algorithm
        if (d2 > 0) {
            y--;
            dy = dy - (2 * rx * rx);
```

```
            d2 = d2 + (rx * rx) - dy;
        }
        else {
            y--;
            x++;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            d2 = d2 + dx - dy + (rx * rx);
        }
    }
}
void launcher(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Mid-Point Drawing Algorithms");
    glutDisplayFunc(midPointEllipseAlgo);
    myInit();
    glutMainLoop();
}
int main(int argc, char **argv) {
    cout << "Enter the coordinates for the ellipse" << endl;
    cout << "Radius x : ";
    cin >> rx;
    cout << "Radius y : ";
    cin >> ry;
    cout << "x Center : ";
    cin >> xc;
    cout << "y Center : ";
    cin >> yc;
    launcher(argc, argv);
    return 0;
}
```

Output :-



2 -> Code

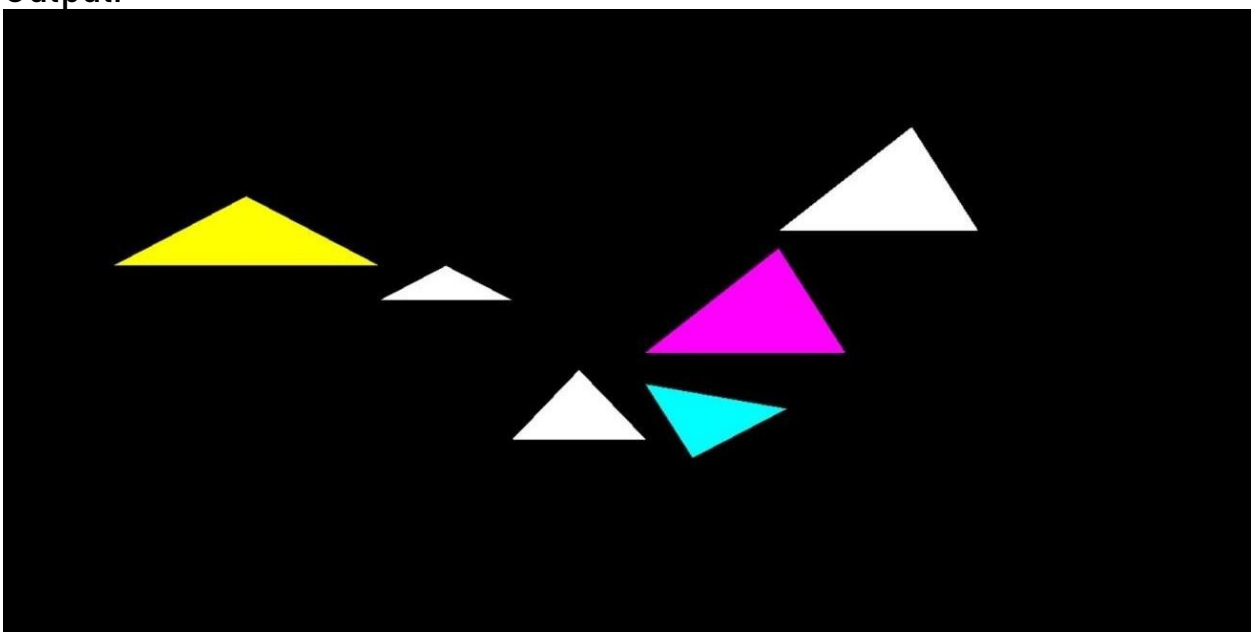
```
#ifdef APPLE
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <stdlib.h>
#include <bits/stdc++.h>
using namespace std;
#define PI 3.14159265
void init(void) {
    glClearColor(0, 0, 0, 0);
}
void put_pixel(float r, float g, float b, double x, double y) {
    glColor3f(r, g, b);
    glVertex2d(x,y);
}
void translate_points(int dx, int dy) {
    double points[3][2] = {{20, 30}, {50, 30}, {40, 60}};
    for (int i = 0; i < 3; ++i) {
        put_pixel(1, 1, 1, points[i][0], points[i][1]);
    }
```

```
    double tMatrix[3][3] = {
        {1, 0, 0},
        {0, 1, 0},
        {dx, dy, 1}};
    double pointMatrix[3][3] = {{20, 30, 1}, {50, 30, 1}, {40, 60, 1}};
    double resMatrix[3][3] = {0};
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            for (int k = 0; k < 3; ++k) {
                resMatrix[i][j] += pointMatrix[i][k] * tMatrix[k][j];
            }
        }
    }
    for (int i = 0; i < 3; ++i) {
        put_pixel(1, 0, 1, resMatrix[i][0], resMatrix[i][1]);
    }
}
void scale_points(int sx, int sy) {
    double points[3][2] = {{-40, 10}, {-20, 10}, {-30, 20}};
    for (int i = 0; i < 3; ++i) {
        put_pixel(1, 1, 1, points[i][0], points[i][1]);
    }
    double sMatrix[3][3] = {
        {sx, 0, 0},
        {0, sy, 0},
        {0, 0, 1}};
    double pointMatrix[3][3] = {
        {-40, 10, 1},
        {-20, 10, 1},
        {-30, 20, 1}};
    double resMatrix[3][3] = {0};
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            for (int k = 0; k < 3; ++k) {
                resMatrix[i][j] += pointMatrix[i][k] * sMatrix[k][j];
            }
        }
    }
    // // // glColor3i(1, 0, 0); for
    (int i = 0; i < 3; ++i) {
        put_pixel(1, 1, 0, resMatrix[i][0], resMatrix[i][1]);
    }
}
void rotate_points(double degree) {
    double points[3][2] = {{-20, -30}, {0, -30}, {-10, -10}};
```

```
    for (int i = 0; i < 3; ++i)
    {
        put_pixel(1, 1, 1, points[i][0], points[i][1]);
    }
    double rMatrix[3][3] = {
        {cos(degree * PI / 180), sin(degree * PI / 180), 0}, {sin(-
1 * degree * PI / 180), cos(degree * PI / 180), 0}, {0, 0, 1}};
    double pointMatrix[3][3] = {
        {-20, -30, 1},
        {0, -30, 1},
        {-10, -10, 1}};
    double resMatrix[3][3] = {0};
    for (int i = 0; i < 3; ++i)
    {
        for (int j = 0; j < 3; ++j)
        {
            for (int k = 0; k < 3; ++k)
            {
                resMatrix[i][j] += pointMatrix[i][k] * rMatrix[k][j];
            }
        }
    }
    // // glColor3i(0, 0, 1); for
    (int i = 0; i < 3; ++i)
    {
        put_pixel(0, 1, 1, resMatrix[i][0], resMatrix[i][1]);
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glPointSize(3.0);
    glBegin(GL_TRIANGLES);
    translate_points(-20, -35);
    scale_points(2, 2);
    rotate_points(45.0);
    glEnd();
    glFlush();
}
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

```
    gluOrtho2D(-100,
               100, -100, 100);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitWindowPosition(200, 100);
    glutInitWindowSize(500, 500);
    glutInitDisplayMode(GLUT_RGB);
    glutCreateWindow("Transformations");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}
```

Output:-



3 - > Code

```
#ifdef APPLE
#include <GLUT/glut.h>
#else
#include <windows.h>
#include <GL/glut.h>
#endif
```

```
#include <stdlib.h>
#include <bits/stdc++.h>
using namespace std;
#define PI 3.14159265
typedef vector<vector<double>> (*ScriptReflFunction)(void);
typedef vector<vector<double>> (*ScriptShearFunction)(int);
typedef vector<vector<double>> (*ScriptRotateFunction)(double);
typedef vector<vector<double>> (*ScriptRotateArbitrary)(double, int, int);
typedef vector<vector<double>> (*ScriptDoubleParamFunction)(int, int);
typedef unordered_map<string, ScriptReflFunction> script_map_refl;
typedef unordered_map<string, ScriptShearFunction> script_map_shear;
typedef unordered_map<string, ScriptRotateFunction> script_map_rot;
typedef unordered_map<string, ScriptRotateArbitrary> script_map_arbitrary;
typedef unordered_map<string, ScriptDoubleParamFunction> script_map_double;

script_map_refl mappingReflection;
script_map_shear mappingShear;
script_map_rot mappingRot;
script_map_double mappingDouble;
script_map_arbitrary mappingArbitrary;

void put_pixel(float r, float g, float b, double x, double y)
{
    glColor3f(r, g, b);
    glVertex2d(x,y);
}
vector<vector<double>> multiplyMatrix(vector<vector<double>> a, vector<vector<dou
ble>> b)
{
    vector<vector<double>> resMatrix(3, vector<double>(3, 0));
    for (int i = 0; i < 3; ++i)
    {
        for (int j = 0; j < 3; ++j)
        {
            for (int k = 0; k < 3; ++k)
            {
                resMatrix[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return resMatrix;
}
vector<vector<double>> set_translation_matrix(int dx, int dy)
{
    vector<vector<double>> tMatrix{
```

```
        {1, 0, 0},
        {0, 1, 0},
        {(double)dx, (double)dy, 1}};
    return tMatrix;
}
vector<vector<double>> set_scaling_matrix(int sx, int sy)
{
    vector<vector<double>> sMatrix{
        {(double)sx, 0, 0},
        {0, (double)sy, 0},
        {0, 0, 1}};
    return sMatrix;
}
vector<vector<double>> set_rotate_matrix(double degree)
{
    vector<vector<double>> rMatrix{
        {cos(degree * PI / 180), sin(degree * PI / 180), 0},
        {sin(-1 * degree * PI / 180), cos(degree * PI / 180), 0},
        {0, 0, 1}};
    return rMatrix;
}
vector<vector<double>> set_rotate_point_matrix(double degree, int px, int py) {
    // T(-px, -py).R(degree).T(px, py)
    vector<vector<double>> rPointMatrix(3, vector<double>(3, 0)); rPointMatrix =
multiplyMatrix(set_translation_matrix(-1 * px, -
1 * py), set_rotate_matrix(degree)); rPointMatrix = multiplyMatrix(rPointMatrix,
set_translation_matrix(px, py));
    return rPointMatrix;
}
vector<vector<double>> set_shearX_matrix(int shx)
{
    vector<vector<double>> shearXMatrix{
        {1, 0, 0},
        {(double)shx, 1, 0},
        {0, 0, 1}};
    return shearXMatrix;
}
vector<vector<double>> set_shearY_matrix(int shy)
{
    vector<vector<double>> shearYMatrix{
        {1, (double)shy, 0},
        {0, 1, 0},
        {0,0, 1}};
    return shearYMatrix;
}
```

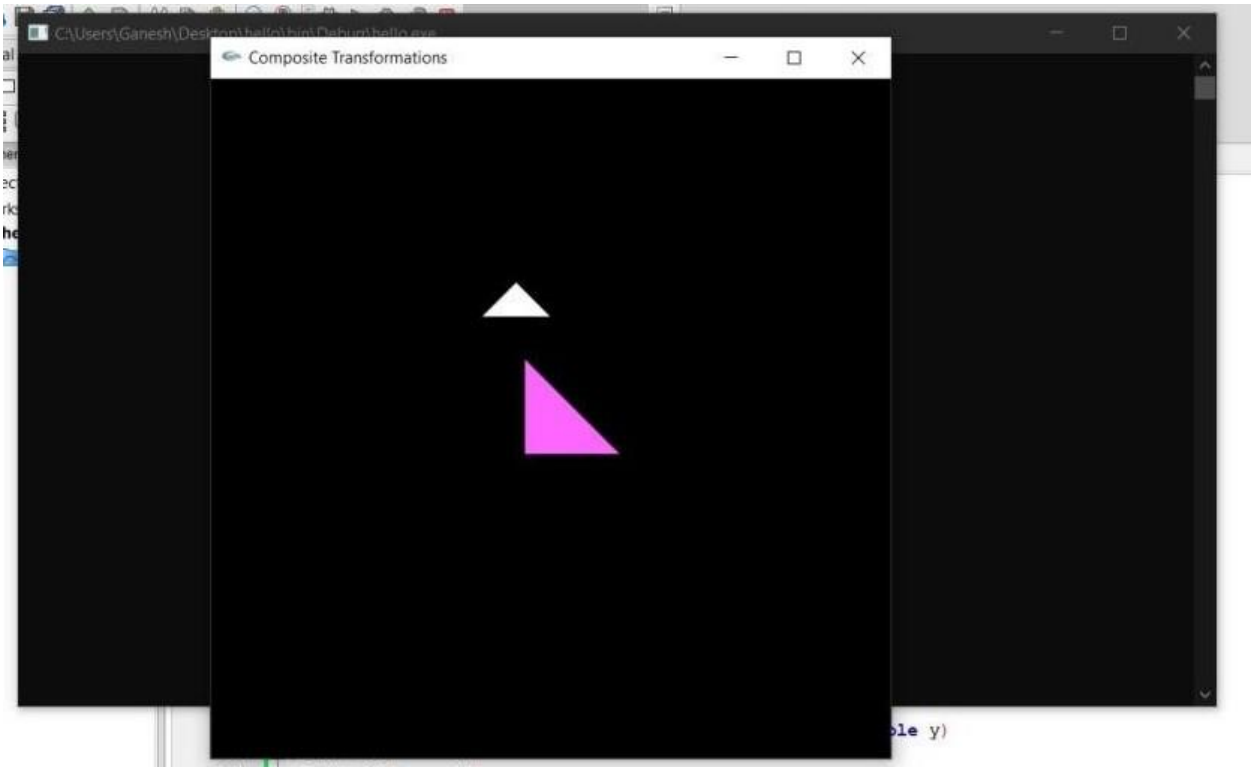
```
vector<vector<double>> set_reflection_xaxis()
{
    // y = 0
    vector<vector<double>> reflXMatrix{
        {1, 0, 0},
        {0, -1, 0},
        {0,0, 1}};
    return reflXMatrix;
}
vector<vector<double>> set_reflection_yaxis()
{
    vector<vector<double>> reflYMatrix{
        {-1, 0, 0},
        {0, 1, 0},
        {0, 0, 1}};
    return reflYMatrix;
}
vector<vector<double>> set_reflection_origin()
{
    vector<vector<double>> reflOriginMatrix{
        {-1, 0, 0},
        {0, -1, 0},
        {0,0, 1}};
    return reflOriginMatrix;
}
vector<vector<double>> set_reflection_line()
{
    // line => y = x
    vector<vector<double>> reflLineMatrix{
        {0, 1, 0},
        {1, 0, 0},
        {0,0, 1}};
    return reflLineMatrix;
}
vector<vector<double>> calc_composite_transformation(vector<string> sequence)
{
    vector<vector<double>> resMatrix(3, vector<double>(3, 0));
    for (auto seq : sequence)
    {
        auto itr1 = mappingRot.find(seq);
        auto itr2 = mappingDouble.find(seq);
        auto itr3 = mappingReflection.find(seq);
        auto itr4 = mappingArbitrary.find(seq);
        auto itr5 = mappingShear.find(seq);
        vector<vector<double>> tempMatrix(3,
```

```
                                        vector<double>(3, 0));
        if (itr1 !=
            mappingRot.end())
        {
            tempMatrix = (*itr1->second)(45);
        }
        if (itr2 != mappingDouble.end())
        {
            if (seq == "T")
                tempMatrix = (*itr2->second)(35, 30);
            if (seq == "S")
                tempMatrix = (*itr2 -
                              > second)(2, 2);
        }
        if (itr3 != mappingReflection.end())
        {
            tempMatrix = (*itr3->second)();
        }
        if (itr4 != mappingArbitrary.end())
        {
            tempMatrix = (*itr4->second)(45, 35, 30);
        }
        if (itr5 != mappingShear.end())
        {
            tempMatrix = (*itr5->second)(3);
        }
        if (seq == sequence.front())
            resMatrix = tempMatrix;
        if (seq != sequence.front())
            resMatrix = multiplyMatrix(resMatrix, tempMatrix);
    }
    return resMatrix;
}
void placePoints()
{
    vector<string> sequence = {"RAr", "S", "ReflX"};
    vector<vector<double>> res = calc_composite_transformation(sequence);
    vector<vector<double>>
        initialPoints{
            {{-20, 30}, {0, 30}, {-10, 40}}};
    vector<vector<double>> points{
        {{-20, 30, 1}, {0, 30, 1}, {-10, 40, 1}}};
    vector<vector<double>> finalPoints = multiplyMatrix(points, res);
    for (int i = 0; i < 3; ++i)
    {
```

```
        put_pixel(1, 1, 1, initialPoints[i][0], initialPoints[i][1]);
    }
    for (int i = 0; i < 3; ++i)
    {
        put_pixel(1, 0.4, 1, finalPoints[i][0], finalPoints[i][1]);
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glPointSize(3.0);
    glBegin(GL_TRIANGLES);
    placePoints();
    glEnd();
    glFlush();
}
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-100,
               100, -100, 100);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void init(void)
{
    glClearColor(0, 0, 0, 0);
}
int main(int argc, char **argv)
{
    mappingReflection.emplace("ReflX", &set_reflection_xaxis);
    mappingReflection.emplace("ReflY", &set_reflection_yaxis);
    mappingReflection.emplace("ReflOrigin", &set_reflection_origin);
    mappingReflection.emplace("ReflLine", &set_reflection_line);
    mappingShear.emplace("ShX", &set_shearX_matrix);
    mappingShear.emplace("ShY", &set_shearX_matrix);
    mappingRot.emplace("R", &set_rotate_matrix);
    mappingDouble.emplace("T",&set_translation_matrix);
    mappingDouble.emplace("S",&set_scaling_matrix);
    mappingArbitrary.emplace("RAr",&set_rotate_point_matrix);
    glutInit(&argc, argv);
    glutInitWindowPosition(200,100);
```

```
    glutInitWindowSize(500, 500);
    glutInitDisplayMode(GLUT_RGB);
    glutCreateWindow("Composite Transformations");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}
```

Output:-



4 - > Code

```
#ifdef APPLE
#include <GLUT/glut.h>
#else
#include <windows.h>
#include <GL/glut.h>
#endif
#include <stdlib.h>
#include <bits / stdc++.h>
using namespace std;
#define PI 3.14159265
float xLeftBoundary = -70;
```

```
float yBottomBoundary = -30;
float xRightBoundary = 70;
float upperBottomY = -1;
float xLeftUpper = -1, xRightUpper = -1;
void put_pixel(float x, float y)
{
    glVertex2d(x, y);
}
void createTriangle(float leftX, float rightX, float yCoord, float mid)
{
    glBegin(GL_LINE_LOOP);
    put_pixel(leftX - 5, yCoord);
    put_pixel(leftX + mid, yCoord + 15);
    put_pixel(rightX + 5, yCoord);
    put_pixel(rightX + 8, yCoord + 5);
    put_pixel(rightX - mid, yCoord + 23);
    put_pixel(leftX - 8, yCoord + 5);
    glEnd();
}
void slantUpperFoundation(vector<float> prevCoordLeft, vector<float> prevCoordRig
ht)
{
    put_pixel(prevCoordLeft[0], prevCoordLeft[1]);
    put_pixel(prevCoordLeft[0] + 5, prevCoordLeft[1] + 5);
    put_pixel(prevCoordRight[0] - 5, prevCoordRight[1] + 5);
    put_pixel(prevCoordRight[0], prevCoordRight[1]);
}
void createSlantFoundation()
{
    glBegin(GL_LINE_LOOP);
    slantUpperFoundation({xLeftBoundary, yBottomBoundary + 5 + 40}, {xRightBounda
ry, yBottomBoundary + 5 + 40});
    glEnd();
    float xLeft1 = xLeftBoundary + 5;
    float xRight1 = xRightBoundary - 5;
    float y1 = yBottomBoundary + 45 + 5;
    glBegin(GL_LINE_LOOP);
    slantUpperFoundation({xLeft1, y1}, {xRight1, y1});
    glEnd();
    float xLeft2 = xLeft1 + 5;
    float xRight2 = xRight1 - 5;
    float y2 = y1 + 5;
    glBegin(GL_LINE_LOOP);
    slantUpperFoundation({xLeft2, y2}, {xRight2, y2});
    glEnd();
```

```
    float xLeft3 = xLeft2 + 5;
    float xRight3 = xRight2 - 5;
    float y3 = y2 + 5;
    glBegin(GL_LINE_LOOP);
    slantUpperFoundation({xLeft3, y3}, {xRight3, y3});
    glEnd();
    upperBottomY = y3 + 5;
    xLeftUpper = xLeft3 + 5;
    xRightUpper = xRight3 - 5;
}
void verticalBaseFoundation(float distance)
{
    put_pixel(xLeftBoundary + distance, yBottomBoundary + 10);
    put_pixel(xLeftBoundary + distance, yBottomBoundary + 10 + 30);
    put_pixel(xLeftBoundary + 5 + distance, yBottomBoundary + 10 + 30);
    put_pixel(xLeftBoundary + 5 + distance, yBottomBoundary + 10);
}
void drawOutlineFoundation()
{
    glBegin(GL_LINE_LOOP);
    put_pixel(xLeftBoundary, yBottomBoundary);
    put_pixel(xLeftBoundary, yBottomBoundary + 10);
    put_pixel(xRightBoundary, yBottomBoundary + 10);
    put_pixel(xRightBoundary, yBottomBoundary);
    glEnd();
    glBegin(GL_LINE_LOOP);
    verticalBaseFoundation(5);
    verticalBaseFoundation(30);
    verticalBaseFoundation(xRightBoundary - xLeftBoundary - 35);
    verticalBaseFoundation(xRightBoundary - xLeftBoundary - 10);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(xLeftBoundary, yBottomBoundary + 40);
    put_pixel(xLeftBoundary, yBottomBoundary + 5 + 40);
    put_pixel(xRightBoundary, yBottomBoundary + 5 + 40);
    put_pixel(xRightBoundary, yBottomBoundary + 40);
    glEnd();
    createSlantFoundation(); // left upper box
    glBegin(GL_LINE_LOOP);
    put_pixel(xLeftUpper, upperBottomY);
    put_pixel(xLeftUpper, upperBottomY + 20);
    put_pixel(xLeftUpper + 30, upperBottomY + 20);
    put_pixel(xLeftUpper + 30, upperBottomY);
    glEnd();
    glBegin(GL_LINE_LOOP);
```

```
    put_pixel(xLeftUpper - 5, upperBottomY + 20);
    put_pixel(xLeftUpper - 5, upperBottomY + 20 + 5);
    put_pixel(xLeftUpper + 30, upperBottomY + 20 + 5);
    put_pixel(xLeftUpper + 30, upperBottomY + 20);
    glEnd(); // right upper box
    glBegin(GL_LINE_LOOP);
    put_pixel(xRightUpper, upperBottomY);
    put_pixel(xRightUpper, upperBottomY + 20);
    put_pixel(xRightUpper - 30, upperBottomY + 20);
    put_pixel(xRightUpper - 30, upperBottomY);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(xRightUpper + 5, upperBottomY + 20);
    put_pixel(xRightUpper + 5, upperBottomY + 20 + 5);
    put_pixel(xRightUpper - 30, upperBottomY + 20 + 5);
    put_pixel(xRightUpper - 30, upperBottomY + 20);
    glEnd();
    glBegin(GL_LINE_STRIP);
    put_pixel(xLeftUpper - 5, upperBottomY + 25);
    put_pixel(xLeftUpper + 7, upperBottomY + 25 + 15);
    put_pixel(xLeftUpper + 7 + 30, upperBottomY + 25 + 15);
    glEnd();
    glBegin(GL_LINE_STRIP);
    put_pixel(xRightUpper + 5, upperBottomY + 25);
    put_pixel(xRightUpper - 7, upperBottomY + 25 + 15);
    put_pixel(xRightUpper - 7 - 30, upperBottomY + 25 + 15);
    glEnd();
    float mid = ((xRightUpper - 30) - (xLeftUpper + 30)) / 2;
    createTriangle(xLeftUpper + 30, xRightUpper - 30, upperBottomY + 25,mid);
}
void verticalWindows(float xMiddle, float yMiddle)
{
    glBegin(GL_LINE_LOOP);
    put_pixel(xMiddle - 5, yMiddle - 8);
    put_pixel(xMiddle - 5, yMiddle + 8);
    put_pixel(xMiddle + 5, yMiddle + 8);
    put_pixel(xMiddle + 5, yMiddle - 8);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(xMiddle - 3, yMiddle + 2);
    put_pixel(xMiddle - 3, yMiddle + 6);
    put_pixel(xMiddle + 3, yMiddle + 6);
    put_pixel(xMiddle + 3, yMiddle + 2);
    glEnd();
    glBegin(GL_LINE_LOOP);
```

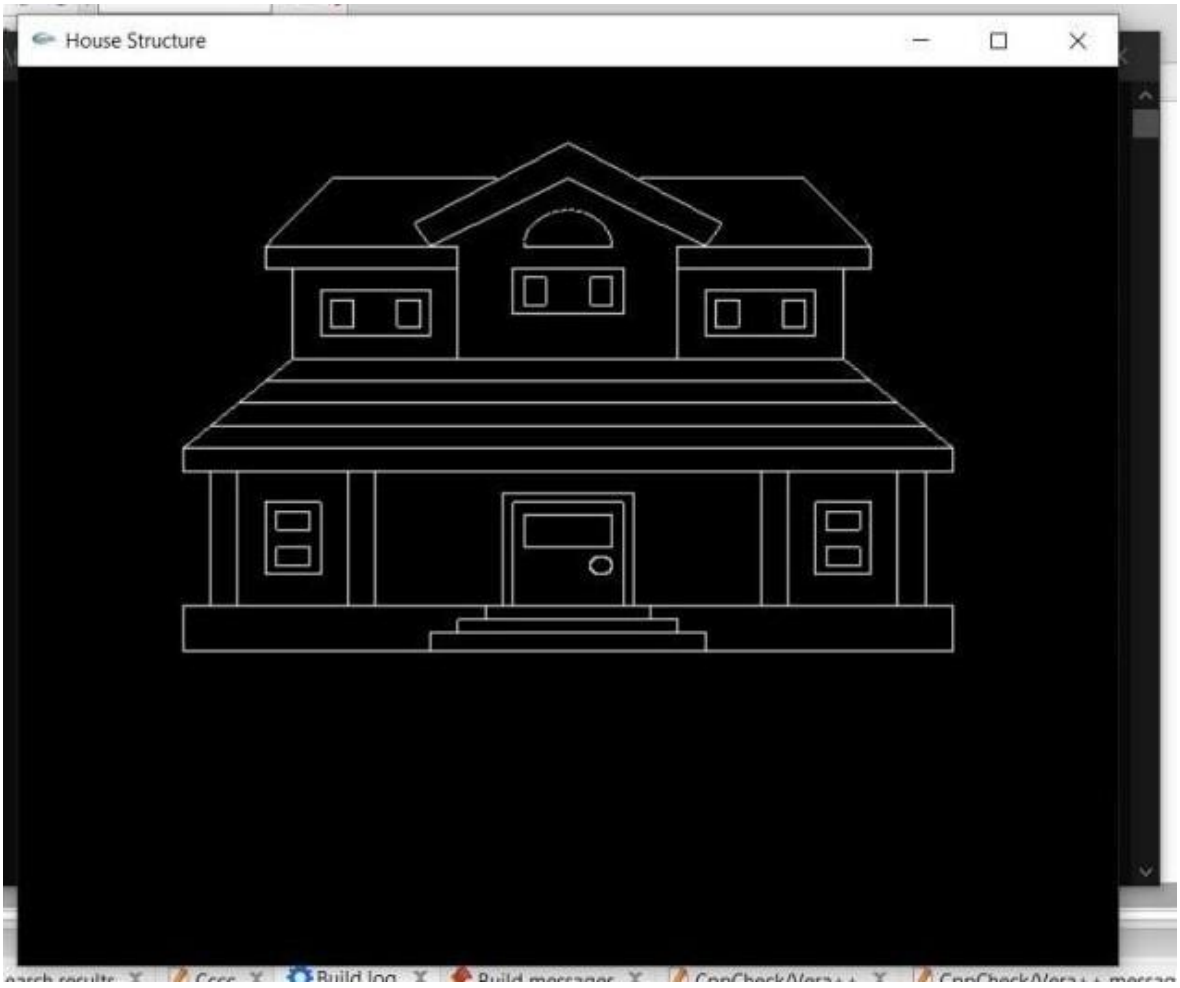
```
    put_pixel(xMiddle - 3, yMiddle - 2);
    put_pixel(xMiddle - 3, yMiddle - 6);
    put_pixel(xMiddle + 3, yMiddle - 6);
    put_pixel(xMiddle + 3, yMiddle - 2);
    glEnd();
}
void horizontalWindows(float xMiddle, float yMiddle)
{
    glBegin(GL_LINE_LOOP);
    put_pixel(xMiddle - 10, yMiddle - 5);
    put_pixel(xMiddle - 10, yMiddle + 5);
    put_pixel(xMiddle + 10, yMiddle + 5);
    put_pixel(xMiddle + 10, yMiddle - 5);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(xMiddle - 4, yMiddle - 3);
    put_pixel(xMiddle - 8, yMiddle - 3);
    put_pixel(xMiddle - 8, yMiddle + 3);
    put_pixel(xMiddle - 4, yMiddle + 3);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(xMiddle + 4, yMiddle + 3);
    put_pixel(xMiddle + 8, yMiddle + 3);
    put_pixel(xMiddle + 8, yMiddle - 3);
    put_pixel(xMiddle + 4, yMiddle - 3);
    glEnd();
}
void drawDoor(float xMiddle)
{
    float yCoord = yBottomBoundary + 10;
    glBegin(GL_LINE_LOOP);
    put_pixel(xMiddle - 12, yCoord);
    put_pixel(xMiddle - 12, yCoord + 25);
    put_pixel(xMiddle + 12, yCoord + 25);
    put_pixel(xMiddle + 12, yCoord);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(xMiddle - 10, yCoord);
    put_pixel(xMiddle - 10, yCoord + 23);
    put_pixel(xMiddle + 10, yCoord + 23);
    put_pixel(xMiddle + 10, yCoord);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(xMiddle - 8, yCoord + 23 - 10);
    put_pixel(xMiddle - 8, yCoord + 23 - 3);
```

```
    put_pixel(xMiddle + 8, yCoord + 23 - 3);
    put_pixel(xMiddle + 8, yCoord + 23 - 10);
    glEnd();
}
void drawCircle(float xMiddle)
{
    float yCoord = yBottomBoundary + 10;
    float yMiddle = yCoord + 23 - 10 - 4;
    float radius = 2;
    float pi = PI;
    glPointSize(1.0);
    glBegin(GL_POINTS);
    for (float i = 0.0; i <= 2 * pi; i += 0.05)
        put_pixel(xMiddle + (sin(i) * radius), yMiddle + (cos(i) * radius));
    glEnd();
}
void drawSemiCircle(float xMiddle, float yMiddle)
{
    float radius = 8;
    float pi = PI;
    glPointSize(1.0);
    glBegin(GL_POINTS);
    for (float i = -1 * pi / 2; i <= pi / 2; i += 0.05)
        put_pixel(xMiddle + (sin(i) * radius), yMiddle + (cos(i) * radius));
    glEnd();
    glBegin(GL_LINE_STRIP);
    put_pixel(xMiddle + (sin(-1 * pi / 2) * radius), yMiddle + (cos(-
1 * pi / 2) * radius));
    put_pixel(xMiddle + (sin(pi / 2) * radius), yMiddle + (cos(pi / 2) * radius))
;
    glEnd();
}
void drawInnerDesigns()
{
    float mid = (xRightBoundary + xLeftBoundary) / 2;
    glBegin(GL_LINE_LOOP);
    put_pixel(mid - 25, yBottomBoundary);
    put_pixel(mid - 25, yBottomBoundary + 4);
    put_pixel(mid + 25, yBottomBoundary + 4);
    put_pixel(mid + 25, yBottomBoundary);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(mid - 20, yBottomBoundary + 4);
    put_pixel(mid - 20, yBottomBoundary + 7);
    put_pixel(mid + 20, yBottomBoundary + 7);
```

```
    put_pixel(mid + 20, yBottomBoundary + 4);
    glEnd();
    glBegin(GL_LINE_LOOP);
    put_pixel(mid - 15, yBottomBoundary + 7);
    put_pixel(mid - 15, yBottomBoundary + 10);
    put_pixel(mid + 15, yBottomBoundary + 10);
    put_pixel(mid + 15, yBottomBoundary + 7);
    glEnd();
    verticalWindows(xLeftBoundary + 10 + 10, yBottomBoundary + 25);
    verticalWindows(xRightBoundary - 10 - 10, yBottomBoundary + 25);
    drawDoor(mid);
    drawCircle(mid + 6);
    horizontalWindows(xLeftUpper + 15, upperBottomY + 10);
    horizontalWindows(xRightUpper - 15, upperBottomY + 10);
    float midUpperWindow = ((xRightUpper - 30) + (xLeftUpper + 30)) / 2;
    horizontalWindows(midUpperWindow, upperBottomY + 15);
    drawSemiCircle(midUpperWindow, upperBottomY + 25);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glPointSize(3.0);
    drawOutlineFoundation();
    drawInnerDesigns();
    glFlush();
}
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-100, 100, -100, 100);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void init(void)
{
    glClearColor(0, 0, 0, 0);
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitWindowPosition(200, 100);
    glutInitWindowSize(500, 500);
```

```
    glutInitDisplayMode(GLUT_RGB);
    glutCreateWindow("House Structure");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}
```

Output:-



5 - > Code

```
#ifdef APPLE
#include <GLUT/glut.h>
#else
#include <windows.h>
#include <GL/glut.h>
```

```
#endif
#include <stdlib.h>
#include <bits / stdc++.h>
using namespace std;
#define PI 3.14159265
double degree =0;
vector<vector<double>> cacheMatrix{};
void put_pixel(float r, float g, float b, double x, double y)
{
    glColor3f(r, g, b);
    glVertex2d(x, y);
}
vector<vector<double>> multiplyMatrix(vector<vector<double>> a, vector<vector<dou
ble>> b)
{
    vector<vector<double>> resMatrix(3, vector<double>(3, 0));
    for (int i = 0; i < 3; ++i)
    {
        for (int j = 0; j < 3; ++j)
        {
            for (int k = 0; k < 3; ++k)
            {
                resMatrix[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return resMatrix;
}
vector<vector<double>> set_translation_matrix(int dx, int dy)
{
    vector<vector<double>> tMatrix{
        {1, 0, 0},
        {0, 1, 0},
        {(double)dx, (double)dy, 1}};
    return tMatrix;
}
vector<vector<double>> set_rotate_matrix(double degree)
{
    vector<vector<double>> rMatrix{
        {cos(degree * PI / 180), sin(degree * PI / 180), 0},
        {sin(-1 * degree * PI / 180), cos(degree * PI / 180), 0},
        {0, 0, 1}};
    return rMatrix;
}
vector<vector<double>> set_rotate_point_matrix(double degree, int px, int py)
```

```
{
    // T(-px, -py).R(degree).T(px, py)
    vector<vector<double>> rPointMatrix(3, vector<double>(3, 0));
    rPointMatrix = multiplyMatrix(set_translation_matrix(-1 * px, -
1 * py), set_rotate_matrix(degree));
    rPointMatrix = multiplyMatrix(rPointMatrix, set_translation_matrix(px, py));
    return rPointMatrix;
}
void timer(int id)
{
    degree += 30;
    if (degree == 360)
    {
        cacheMatrix = {};
        degree = 0;
    }
    glutPostRedisplay();
}
void placePoints()
{
    vector<vector<double>> res = set_rotate_point_matrix(degree, 25, 40);
    vector<vector<double>> points{ {{-20, 30, 1}, {0, 30, 1}, {-10, 40, 1}}};
    points = multiplyMatrix(points, res);
    for (int i = 0; i < 3; ++i)
        cacheMatrix.push_back({points[i][0], points[i][1]});
    for (int i = 0; i < cacheMatrix.size(); ++i)
        put_pixel(0, 1, 1, cacheMatrix[i][0], cacheMatrix[i][1]);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glPointSize(3.0);
    glBegin(GL_TRIANGLES);
    placePoints();
    glEnd();
    glutTimerFunc(2000, timer, 1);
    glFlush();
}
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-100,100, -100, 100);
```

```
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
void init(void)
{
    glClearColor(0, 0, 0, 0);
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitWindowPosition(200, 100);
    glutInitWindowSize(500, 500);
    glutInitDisplayMode(GLUT_RGB);
    glutCreateWindow("Continuous Rotation");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}
```

Output:-