SSN COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 6b – Window to viewport mapping

Name : Likhitha Verma A
Reg : 185001084
Date : 03/10/2021

AIM :

To create an object and window, to create a view port of size smaller than the window and apply Window to viewport transformation of the object.

ALGORITHM :

1. Read the no. of edges of the polygon from the user.
2. Read the vertices of the polygon.
3. Plot the original polygon.
4. Compute the scaling factor.
5. Using the above computed scaling factor, compute the transformed viewport coordinates
6. Plot transformed coordinates in the new viewport window.

CODE :

```
#include <math.h>
#include <stdio.h>
#include <iostream>
#include <vector>
#include <ctime>
#include <glut.h>

using namespace std;

const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_FPS = 60;
```

```
const int POINT_SIZE = 2;

double Sx, Sy;

pair<double, double> window_x_dims(0, 400), window_y_dims(0, 400);
pair<double, double> viewport_x_dims(0,200), viewport_y_dims(0, 200);

int edge_count;

vector<pair<double, double>> original_points, transformed_points;

void render();
void lineloop(double x1, double y1, double x2, double y2);
void setEdgeCount(int option);
void computeScaleFactor();
void computeTransformedPoints();
void drawWindow();
void drawWindowFigure();
void drawViewport();
void drawViewportFigure();

bool initGL() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-10.0, 640.0, -10.0, 480.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glClearColor(0.f, 0.f, 0.f, 1.f);

    glPointSize(POINT_SIZE);
    glEnable(GL_POINT_SMOOTH);

    GLenum error = glGetError();
    if (error != GL_NO_ERROR)
    {
        printf("Error initializing OpenGL! %s\n",
gluErrorString(error));
```

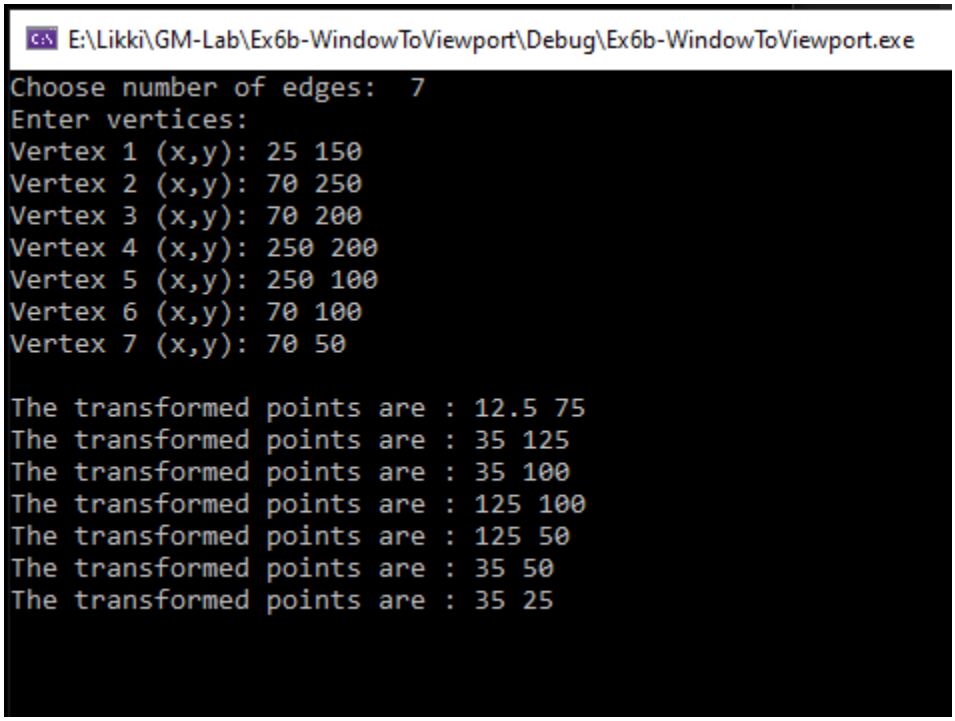
```
        return false;
    }

    return true;
}

void renderOG() {
    drawWindow();
    drawWindowFigure();
    glFlush();
}

void renderTrans() {
    drawViewport();
    computeScaleFactor();
    computeTransformedPoints();
    drawViewportFigure();
    glFlush();
}

void setEdgeCount(int option) {
    if (option == 0) {
        cout << "Invalid" << endl;
    }
    else if (option == 1 || option == 2) {
        edge_count = 2;
    }
    else {
        edge_count = option;
    }
}

void lineloop(double x1, double y1, double x2, double y2) {

    glBegin(GL_LINE_LOOP);

    glVertex2d(x1, y1);
    glVertex2d(x2, y1);
    glVertex2d(x2, y2);
```

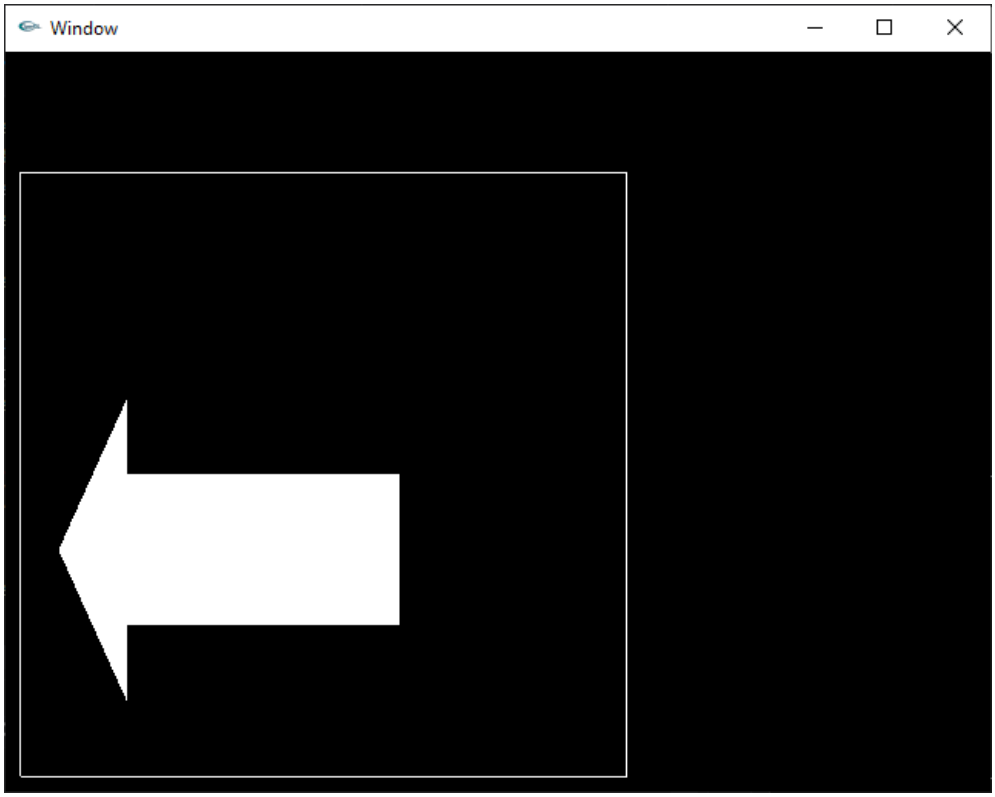
```
    glVertex2d(x1, y2);

    glEnd();
}

void drawWindow() {
    glColor3f(1.0, 1.0, 1.0);
    lineloop(window_x_dims.first, window_y_dims.first,
window_x_dims.second, window_y_dims.second);
}

void drawWindowFigure() {
    glColor3f(1.0, 1.0, 1.0);
    if (edge_count == 2)
        glBegin(GL_LINES);
    else
        glBegin(GL_POLYGON);

    for (int i = 0; i < edge_count; i++) {
        glVertex2d(original_points[i].first,
original_points[i].second);
    }
    glEnd();
    glFlush();
}

void drawViewport() {
    glColor3f(1.0, 1.0, 1.0);
    lineloop(viewport_x_dims.first, viewport_y_dims.first,
viewport_x_dims.second, viewport_y_dims.second);
}

void drawViewportFigure() {
    glColor3f(1.0, 1.0, 1.0);
    if (edge_count == 2)
        glBegin(GL_LINES);
    else
        glBegin(GL_POLYGON);
```

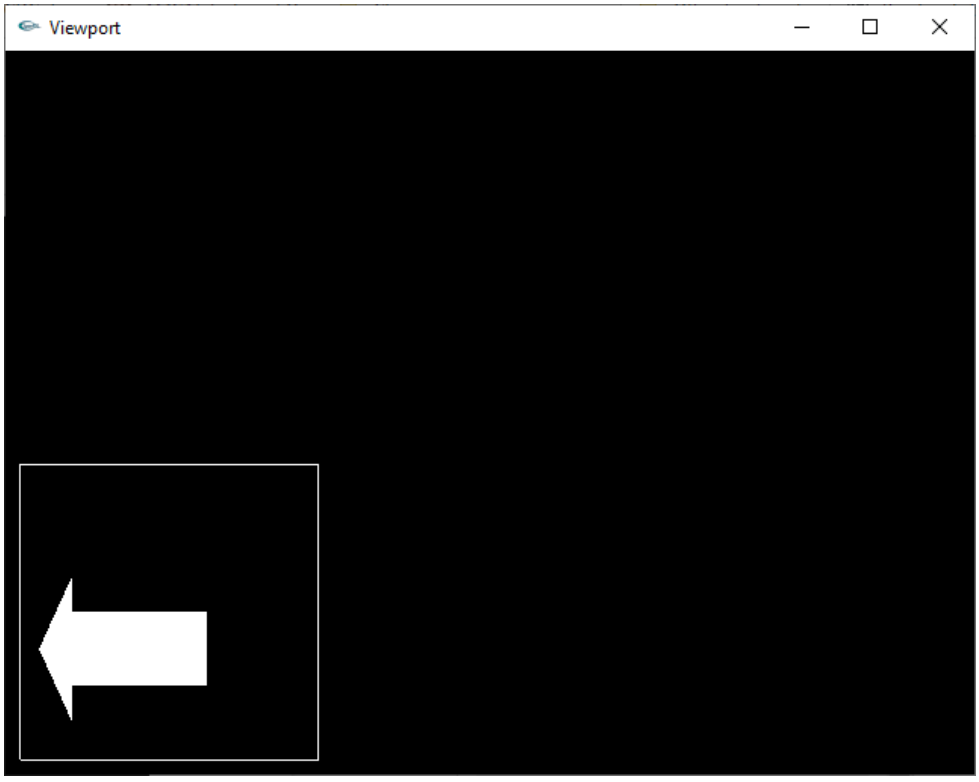
```
    for (int i = 0; i < edge_count; i++) {
        cout << "The transformed points are : " <<
transformed_points[i].first << " " << transformed_points[i].second;
        glVertex2d(transformed_points[i].first,
transformed_points[i].second);
    }
    glEnd();
    glFlush();
}

void computeScaleFactor() {

    double xNr = viewport_x_dims.second - viewport_x_dims.first;
    double xDr = window_x_dims.second - window_x_dims.first;

    Sx = xNr / xDr;

    double yNr = viewport_y_dims.second - viewport_y_dims.first;
    double yDr = window_y_dims.second - window_y_dims.first;

    Sy = yNr / yDr;
}

void computeTransformedPoints() {
    for (int i = 0; i < edge_count; i++) {
        pair<double, double> p = original_points[i];
        double xw = p.first;
        double yw = p.second;

        double xv = viewport_x_dims.first + (xw -
window_x_dims.first) * Sx;
        double yv = viewport_y_dims.first + (yw -
window_y_dims.first) * Sy;

        transformed_points.push_back(pair<double, double>(xv, yv));

    }
}
```

```
int main(int argc, char* args[]) {

    glutInit(&argc, args);
    GLint WindowID1, WindowID2;
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(SCREEN_WIDTH, SCREEN_HEIGHT);

    WindowID1 = glutCreateWindow("Window");
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);

    int option = 0;
    cout << "Choose number of edges:  ";
    cin >> option;

    setEdgeCount(option);
    cout << "Enter vertices: " << endl;
    for (int i = 0; i < edge_count; i++) {
        cout << "Vertex " << i + 1 << " (x,y): ";
        double x, y;
        cin >> x >> y;
        original_points.push_back(pair<double, double>(x, y));
    }

    drawWindowFigure();
    initGL();
    glutDisplayFunc(renderOG);
    WindowID2 = glutCreateWindow("Viewport");

    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glutDisplayFunc(renderTrans);
    initGL();

    glutMainLoop();

    return 0;
}
```

OUTPUT : Screenshots







RESULT :

Thus compiled and executed a C++ program using OPENGL to perform window to viewport transformation successfully.