**SSN College of Engineering**

**Department of Computer Science and Engineering**

## UCS1712 – GRAPHICS AND MULTIMEDIA LAB

**EX6b - Window to Viewport Mapping**

SATHEESH KUMAR G R

185001136

**1]AIM**

To Create an object and window as given below. Create a view port of size smaller than the window.

**2]ALGORITHM**

* + A point at position (xw, yw) in window mapped into position (xv, yv) in the associated viewport.
  + with a set of transformation that converts the window or world coordinate area into the viewport or screen coordinate area.
  + Perform a scaling transformation using a fixed point position (xwmin,ywmin) that scales the window area to the size of the viewport.
  + Translate the scaled window area to the position of the viewport. Relative proportions of objects are maintained if the scaling factors are the same (sx=sy).
  + This mapping called workstation transformation (It is accomplished by selecting a window area in normalized space and a viewport area in the coordinates of the display device).
  + Scaling of the window to match its size to the viewport Sx=(Xymax-Xvmin)*\*(Xwmax- Xwmin) Sy=(Yvmax-Yvmin*\*(Ywmax-Ywmin).

**3]CODE**

#include <GL/glut.h>

#include <iostream>

#include <vector>

using namespace std;

using ld = long double;

using ll = long long;

#define X       first

#define Y       second

const int WINDOW\_WIDTH = 1000;

const int WINDOW\_HEIGHT = 1000;

const int VIEWPORT\_WIDTH = 500;

const int VIEWPORT\_HEIGHT = 500;

const int X\_MIN = -30;

const int X\_MAX = 420;

const int Y\_MIN = -30;

const int Y\_MAX = 300;

struct Display {

    ld X\_MIN, X\_MAX, Y\_MIN, Y\_MAX;

    Display(ld X\_MIN, ld X\_MAX, ld Y\_MIN, ld Y\_MAX):  X\_MIN(X\_MIN), X\_MAX(X\_MAX), Y\_MIN(Y\_MIN), Y\_MAX(Y\_MAX) {}

    void draw(ld Red = 0.0f, ld Green = 0.0f, ld Blue = 0.0f, ld Alpha = 1.0) {

//        glBegin(GL\_LINE\_LOOP);

        glColor4f(Red, Green, Blue, Alpha);

        glVertex2d(X\_MIN, Y\_MIN);

        glVertex2d(X\_MIN, Y\_MAX);

        glVertex2d(X\_MAX, Y\_MAX);

        glVertex2d(X\_MAX, Y\_MIN);

        glEnd();

    }

};

void myInit();

void myDisplay();

void initialize\_window();

ld multiply(vector<ld> a, vector<ll> b);

vector<ld> multiply(vector<vector<ld>> &a, vector<ll> b);

vector<vector<ld>> multiply(vector<vector<ld>> &a, vector<vector<ld>> &b);

pair<ld,ld> getPoint(vector<ld> point\_matrix);

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);

vector<vector<ld>> getTransformMatrix();

vector<vector<ld>> translate(ld tx=0, ld ty=0);

vector<vector<ld>> scale(ld sx=2, ld sy=2, pair<ll,ll> pivot=make\_pair(0,0));

void transformShape();

void drawViewport(Display window, Display viewport, vector<vector<pair<ll,ll>>> &shapes);

void initialize\_window(int width, int height){

    glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

    glutInitWindowSize(width, height);

    glutCreateWindow("2D Window To Viewport Transformation");

    glutDisplayFunc(myDisplay);

    myInit();

}

void myInit() {

    glClearColor(1.0,1.0,1.0,0.0);

    glColor3f(0.0f,0.0f,0.0f);

    glPointSize(5.0);

    glMatrixMode(GL\_PROJECTION);

    glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

    glEnable( GL\_BLEND );

    glLoadIdentity();

    gluOrtho2D(X\_MIN,X\_MAX,Y\_MIN,Y\_MAX);

}

void myDisplay() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    transformShape();

    glFlush();

}

vector<vector<ld>> getTransformMatrix(Display window, Display viewport) {

    vector<vector<ld>> transform\_matrix = {{1,0,0},{0,1,0},{0,0,1}};

    vector<vector<ld>> translate\_matrix = translate(viewport.X\_MIN - window.X\_MIN, viewport.Y\_MIN - window.Y\_MIN);

    transform\_matrix = multiply(translate\_matrix, transform\_matrix);

    vector<vector<ld>> scale\_matrix = scale(

                                        (viewport.X\_MAX - viewport.X\_MIN)/(window.X\_MAX - window.X\_MIN),

                                        (viewport.Y\_MAX - viewport.Y\_MIN)/(window.Y\_MAX - window.Y\_MIN),

                                        {viewport.X\_MIN, viewport.Y\_MIN}

                                    );

    transform\_matrix = multiply(scale\_matrix, transform\_matrix);

    return transform\_matrix;

}

void drawViewport(Display window, Display viewport, vector<vector<pair<ll,ll>>> &shapes) {

    vector<vector<ld>> transform\_matrix;

    //Plot viewport;

    transform\_matrix = getTransformMatrix(window, viewport);

    viewport.draw(1.0, 0.0, 0.7);

    for(auto shape: shapes) {

        glBegin(GL\_POLYGON);

        glColor4f(1.0f,0.0f,0.7f,1.0f);

        for(auto point : shape) {

            pair<ld,ld> viewpoint = getPoint(multiply(transform\_matrix, getHomogeneousPointCoords(point)));

            glVertex2d(viewpoint.X, viewpoint.Y);

        }

        glEnd();

    }

}

Display window = Display(0, 400, 0, 250);

//    Display viewport2 = Display(180, 370, 170, 250);

//    Display viewport3 = Display(290, 370, 10, 90);

vector<vector<pair<ll,ll>>> shapes = {

//                {{20,60}, {60,60}, {60,20}, {20,20}},

                {{0,0}, {100,0}, {100,100},{0,100}},

//            {{40,200}, {60,230}, {140,190}, {80,160}, {60,140}},

//                {{140,20}, {120,220}, {140,240}}

        };

void transformShape() {

//    Display window = Display(10, 150, 10, 250);

//

////    Display viewport2 = Display(180, 370, 170, 250);

////    Display viewport3 = Display(290, 370, 10, 90);

//

//    vector<vector<pair<ll,ll>>> shapes = {

////                {{20,60}, {60,60}, {60,20}, {20,20}},

////                {{30,30}, {120,30}, {75,120}},

//         {{40,200}, {60,230}, {140,190}, {80,160}, {60,140}},

////                {{140,20}, {120,220}, {140,240}}

//            };

    //Plot window;

    window.draw(0.7, 0.0, 1.0);

    for(auto shape: shapes) {

        glBegin(GL\_POLYGON);

        glColor4f(0.7f,0.0f,1.0f,1.0f);

        for(auto point : shape) {

            glVertex2d(point.X,point.Y);

        }

        glEnd();

    }

//    Display viewport1 = Display(180, 260, 10, 140);

//    drawViewport(window, viewport1, shapes);

//    drawViewport(window, viewport2, shapes);

//    drawViewport(window, viewport3, shapes);

}

pair<ld,ld> getPoint(vector<ld> point\_matrix) {

    ll h = point\_matrix[2];

    ld x = point\_matrix[0];

    ld y = point\_matrix[1];

    return {x/h,y/h};

}

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {

    vector<ll> point\_matrix;

    point\_matrix.push\_back(h\*point.first);

    point\_matrix.push\_back(h\*point.second);

    point\_matrix.push\_back(h);

    return point\_matrix;

}

vector<vector<ld>> translate(ld tx, ld ty) {

    vector<vector<ld>> translate\_matrix = {

                                            {1,0,tx},

                                            {0,1,ty},

                                            {0,0,1}

                                        };

    return translate\_matrix;

}

vector<vector<ld>> scale(ld sx, ld sy, pair<ll,ll> pivot) {

    ll xf = pivot.X;

    ll yf = pivot.Y;

    vector<vector<ld>> scale\_matrix = {

                                {sx, 0, xf\*(1-sx)},

                                {0, sy, yf\*(1-sy)},

                                {0,  0, 1}

                            };

    return scale\_matrix;

}

vector<vector<ld>> multiply(vector<vector<ld>> &a, vector<vector<ld>> &b) {

    vector<vector<ld>> result;

    for(int i=0; i<a.size(); i++) {

        vector<ld> row;

        for(int j=0; j<b[0].size(); j++) {

            ld sum = 0;

            for(int k=0; k<a[0].size(); k++) {

                sum += a[i][k]\*b[k][j];

            }

            row.push\_back(sum);

        }

        result.push\_back(row);

    }

    return result;

}

vector<ld> multiply(vector<vector<ld>> &a, vector<ll> b) {

    vector<ld> result;

    for(int i=0;i<a.size();i++) {

        ll temp = multiply(a[i],b);

        result.push\_back(temp);

    }

    return result;

}

ld multiply(vector<ld> a, vector<ll> b) {

    ld result=0;

    for(int i=0;i<a.size();i++) {

        result+=(a[i]\*b[i]);

    }

    return result;

}

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

    glutInit(&argc,argv);

    initialize\_window(WINDOW\_WIDTH, WINDOW\_HEIGHT);

    Display viewport1 = Display(180, 210, 10, 110);

    drawViewport(window, viewport1, shapes);

    initialize\_window(VIEWPORT\_WIDTH, VIEWPORT\_HEIGHT);

    glutMainLoop();

    return 1;

}

**4]OUTPUT**

