# SSN COLLEGE OF ENGINEERING
# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
## UCS1712 – GRAPHICS AND MULTIMEDIA LAB
## EX NO: 6B-Window to Viewport mapping

Name : Mohamed Hashim G
RegNo : 185001094

**AIM**

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

**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).

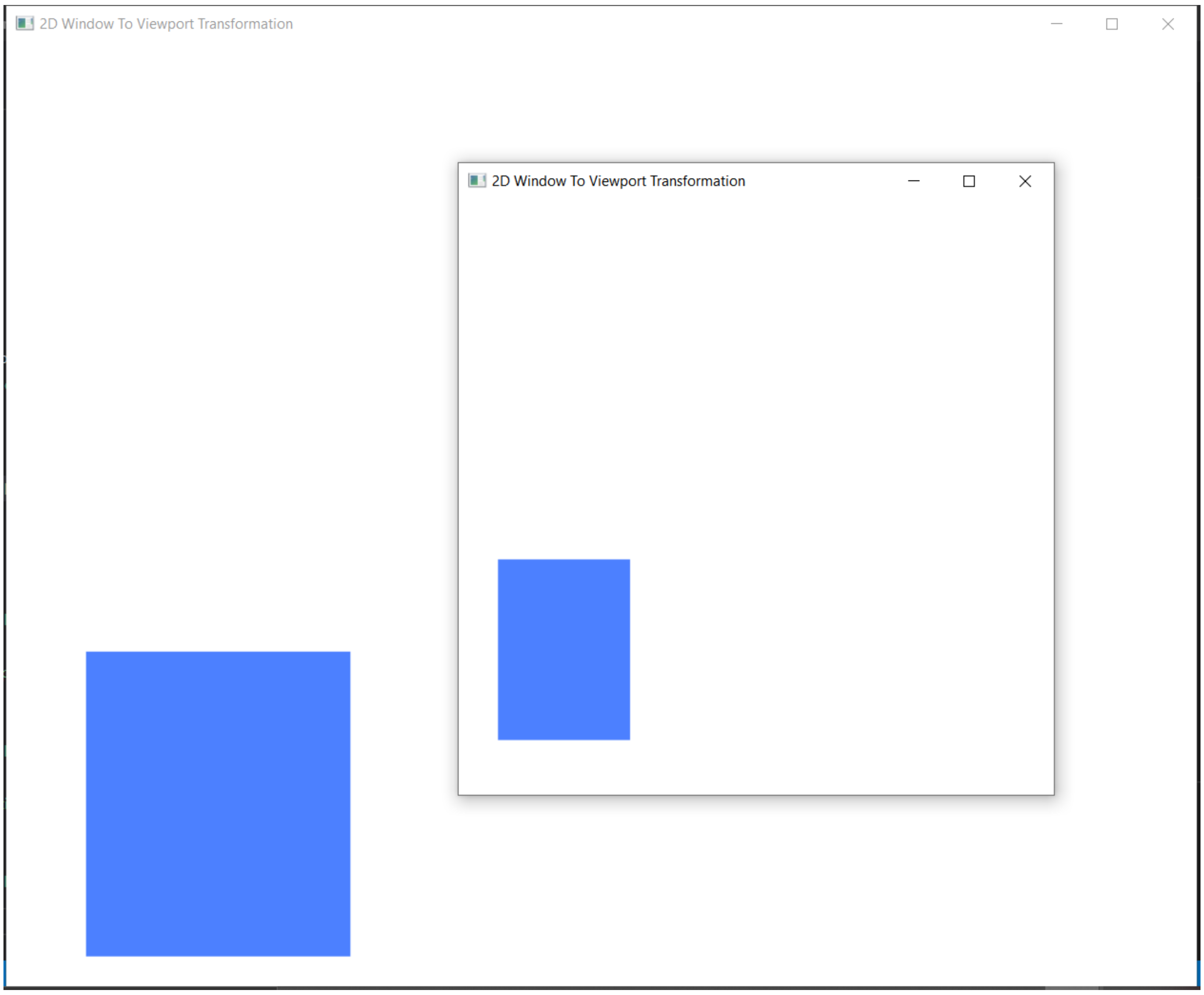
**CODE**

```
#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)
{
        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(0.0f,0.6f,0.3f,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);

vector<vector<pair<ll,ll>>> shapes = {
                {{0,0}, {100,0}, {100,100},{0,100}},
        };

void transformShape() {

    //Plot window;

    window.draw(0.7, 0.0, 1.0);
    for(auto shape: shapes) {
        glBegin(GL_POLYGON);
        glColor4f(0.3f,0.5f,1.0f,1.0f);

        for(auto point : shape) {
            glVertex2d(point.X,point.Y);
        }

        glEnd();
    }

}

pair<ld,ld> getPoint(vector<ld> point_matrix) {
    ll h = point_matrix[2];
    ld x = point_matrix[0];
    ld y = point_matrix[1];
```

```cpp
    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;
}
```

**OUTPUT**


2D Window To Viewport Transformation
2D Window To Viewport Transformation

RESULT

Thus a Polygon has been mapped to a viewport from the original window.