

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

Name: Vikraman S

Reg No: 185001195

Lab Exercise 6: 2D Composite Transformations and Windowing in C++ using OpenGL

a) To compute the composite transformation matrix for any 2 transformations given as input by the user and applying it on the object.

The transformation can be any combination of the following.

- 1) Translation
- 2) Rotation
- 3) Scaling
- 4) Reflection
- 5) Shearing

Display the original and the transformed object.

Calculate the final transformation matrix by multiplying the two individual transformation matrices and then apply it to the object.

Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis)

b) Create a window with any 2D object and a different sized viewport. Apply window to viewport transformation on the object. Display both window and viewport.

Ex6a.cpp:

```
#include<bits/stdc++.h>
#include<GL/glut.h>
using namespace std;

constexpr auto PI=3.14;
int n;
vector<pair<int,int>> coords;

int tx,ty;
int xr,yr;
int xf,yf;
double sx,sy;
double ang,angRad;
double shx,shy;
int opr1,opr2,rfl,sh,shd;
```

```

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

void myInit(void) {
    glClearColor(1.0,1.0,1.0,1.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-500,500,-500,500);
}

void DrawCartesianPlane() {
    glBegin(GL_LINES);
    glColor3f(0.0,0.0,0.0);
    glVertex2d(-500,0);
    glVertex2d(500,0);
    glVertex2d(0,-500);
    glVertex2d(0,500);
    glEnd();
}

void drawPolygon()
{
    glBegin(GL_LINE_LOOP);
    glColor3f(0.0,0.0,0.0);
    for (int i=0; i < n; i++)
    {
        glVertex2d(coords[i].first,coords[i].second);
    }
    glEnd();
}

vector<vector<double>> drawPolygonTrans()
{
    vector<vector<double>> transMatrix(3,vector<double>(3,0));
    transMatrix[0][0]=1;
    transMatrix[1][1]=1;
    transMatrix[2][2]=1;
    transMatrix[0][2]=tx;
    transMatrix[1][2]=ty;
    return transMatrix;
}

vector<vector<double>> rotatePolygonFixed() {
    vector<vector<double>> rotatingMatrix(3,vector<double>(3,0));
    rotatingMatrix[0][0]=cos(angRad);
    rotatingMatrix[1][1]=cos(angRad);
    rotatingMatrix[2][2]=1;

```

```

    rotatingMatrix[1][0]=sin(angRad);
    rotatingMatrix[0][1]=-1*sin(angRad);
    rotatingMatrix[0][2]=xr*(1-cos(angRad))+yr*sin(angRad);
    rotatingMatrix[1][2]=yr*(1-cos(angRad))-xr*sin(angRad);
    return rotatingMatrix;
}

vector<vector<double>> scalePolygonFixed() {
    vector<vector<double>> scalingMatrix(3,vector<double>(3,0));
    scalingMatrix[0][0]=sx;
    scalingMatrix[1][1]=sy;
    scalingMatrix[2][2]=1;
    scalingMatrix[0][2]=xf*(1-sx);
    scalingMatrix[1][2]=yf*(1-sy);
    return scalingMatrix;
}

vector<vector<double>> reflection_Xaxis() {
    vector<vector<double>> reflectMatrix(3,vector<double>(3,0));
    reflectMatrix[0][0]=1;
    reflectMatrix[1][1]=-1;
    reflectMatrix[2][2]=1;
    return reflectMatrix;
}

vector<vector<double>> reflection_Yaxis() {
    vector<vector<double>> reflectMatrix(3,vector<double>(3,0));
    reflectMatrix[0][0]=-1;
    reflectMatrix[1][1]=1;
    reflectMatrix[2][2]=1;
    return reflectMatrix;
}

vector<vector<double>> reflection_origin() {
    vector<vector<double>> reflectMatrix(3,vector<double>(3,0));
    reflectMatrix[0][0]=-1;
    reflectMatrix[1][1]=-1;
    reflectMatrix[2][2]=1;
    return reflectMatrix;
}

vector<vector<double>> reflection_XeqYline() {
    vector<vector<double>> reflectMatrix(3,vector<double>(3,0));
    reflectMatrix[0][1]=1;
    reflectMatrix[1][0]=1;
    reflectMatrix[2][2]=1;
    return reflectMatrix;
}

vector<vector<double>> x_directionShear() {

```

```

        vector<vector<double>> shearMatrix(3,vector<double>(3,0));
        shearMatrix[0][0]=1;
        shearMatrix[1][1]=1;
        shearMatrix[2][2]=1;
        shearMatrix[0][1]=shx;
        return shearMatrix;
    }

vector<vector<double>> y_directionShear() {
    vector<vector<double>> shearMatrix(3,vector<double>(3,0));
    shearMatrix[0][0]=1;
    shearMatrix[1][1]=1;
    shearMatrix[2][2]=1;
    shearMatrix[1][0]=shy;
    return shearMatrix;
}

void executeTransformMatrix(int opr,int oprn) {
    vector<vector<double>> mat;
    if(opr==1) {
        mat=drawPolygonTrans();
    }
    else if(opr==2) {
        mat=rotatePolygonFixed();
    }
    else if(opr==3) {
        mat=scalePolygonFixed();
    }
    else if(opr==4) {
        if(rfl==1) mat=reflection_Xaxis();
        else if(rfl==2) mat=reflection_Yaxis();
        else if(rfl==3) mat=reflection_origin();
        else if(rfl==4) mat=reflection_XeqYline();
    }
    else {
        if(shd==1) mat=x_directionShear();
        else mat=y_directionShear();
    }
    if(oprn==1) T=mat;
    else {
        vector<vector<double>> res(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++) {
                    res[i][j] += T[i][k]*mat[k][j];
                }
            }
        }
    }
}

```

```

        T=res;
    }
}

void drawTransformedPolygon() {
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0,0.0,0.5);
    vector<pair<int,int>> newCoords;
    vector<double> curpoint(3,0),matProduct(3,0);
    for (int i=0; i < n; i++) {
        curpoint[0]=coords[i].first;
        curpoint[1]=coords[i].second;
        curpoint[2]=1;
        matProduct[0]=0;
        matProduct[1]=0;
        matProduct[2]=0;
        for (int j=0; j < 3; j++) {
            for (int k=0; k < 3; k++) {
                matProduct[j] += T[j][k]*curpoint[k];
            }
        }
        newCoords.push_back(make_pair(round(matProduct[0]),round(matProduct[1]
)));
    }

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

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,0.0,0.0);
    DrawCartesianPlane();
    drawPolygon();
    executeTransformMatrix(opr1,1);
    executeTransformMatrix(opr2,2);
    drawTransformedPolygon();
    glFlush();
}

void get(int opr) {
    switch(opr) {
        case 1: {
            cout<<"Translation factor : ";
            cin>>tx>>ty;

```

```

        break;
    }
    case 2: {
        cout<<"Angle of rotation : ";
        cin>>ang;
        angRad=ang*PI/180;
        cout<<"Rotate about : ";
        cin>>xr>>yr;
        break;
    }
    case 3: {
        cout<<"Scaling factor : ";
        cin>>sx>>sy;
        cout<<"Scale about : ";
        cin>>xf>>yf;
        break;
    }
    case 4: {
        cout<<"Reflect about : "<<endl;
        cout<<"[1] X-axis"<<endl;
        cout<<"[2] Y-axis"<<endl;
        cout<<"[3] origin"<<endl;
        cout<<"[4] X=Y line"<<endl;
        cin>>rfl;
        break;
    }
    default: {
        cout<<"Shear about : "<<endl;
        cout<<"[1] X-direction"<<endl;
        cout<<"[2] Y-direction"<<endl;
        cin>>shd;
        cout<<"Enter shear parameter: ";
        cin>>sh;
        if(shd==1) sh=shx;
        else sh=shy;
    }
}
}

int main(int argc, char** argv) {
    cout<<"Polygon "<<endl;
    cout<<"No. of vertices : ";
    cin>>n;
    int x,y;
    for (int i=0; i < n; i++) {
        cout<<"V"<<i+1<<" : ";
        cin>>x>>y;
        coords.push_back(make_pair(x,y));
    }
}

```

```

    }
    cout<<"What two operations do you want to perform?"<<endl;
    cout<<"[1] Translate"<<endl;
    cout<<"[2] Rotation"<<endl;
    cout<<"[3] Scaling"<<endl;
    cout<<"[4] Reflection"<<endl;
    cout<<"[5] Shearing"<<endl;

    cout<<"\nOperation 1 : ";
    cin>>opr1;
    get(opr1);
    cout<<"\nOperation 2 : ";
    cin>>opr2;
    get(opr2);

    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1000,1000);
    glutCreateWindow("Transformation");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 0;
}

```

Ex6b.cpp:

```

#include<GL/glut.h>
#include<vector>
#include<utility>
#include<iostream>
#include<math.h>
using namespace std;

int n;
vector<pair<int,int>> coords;
int xv_min,xv_max,yv_min,yv_max;
int xw_min=0,yw_min=0,xw_max=1000,yw_max=1000;
double sx,sy;

void myInit(void) {
    glClearColor(1.0,1.0,1.0,1.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,1000,0,1000);
}

```

```

void drawWVPort() {
    glBegin(GL_LINE_LOOP);
    glColor3f(0.0f,1.0f,0.0f);
    glVertex2d(xv_min,yv_min);
    glVertex2d(xv_min,yv_max);
    glVertex2d(xv_max,yv_max);
    glVertex2d(xv_max,yv_min);
    glEnd();
}

void drawWPolygon() {
    glBegin(GL_LINE_LOOP);
    for (int i=0; i < n; i++) {
        glVertex2d(coords[i].first,coords[i].second);
    }
    glEnd();
}

void drawVPolygon() {
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0f,0.0f,0.0f);
    for (int i=0; i < n; i++) {
        glVertex2d(xv_min+(coords[i].first-
xw_min)*sx,yv_min+(coords[i].second-yw_min)*sy);
    }
    glEnd();
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,0.0,0.0);
    drawWPolygon();
    drawVPolygon();
    drawWVPort();
    glFlush();
}

int main(int argc, char** argv) {
    cout<<"Polygon"<<endl;
    cout<<"No. of vertices : ";
    cin>>n;
    int x,y;
    for (int i=0; i < n; i++) {
        cout<<"V"<<i+1<<" : ";
        cin>>x>>y;
        coords.push_back(make_pair(x,y));
    }
    cout<<"Enter Viewport details"<<endl;

```



```

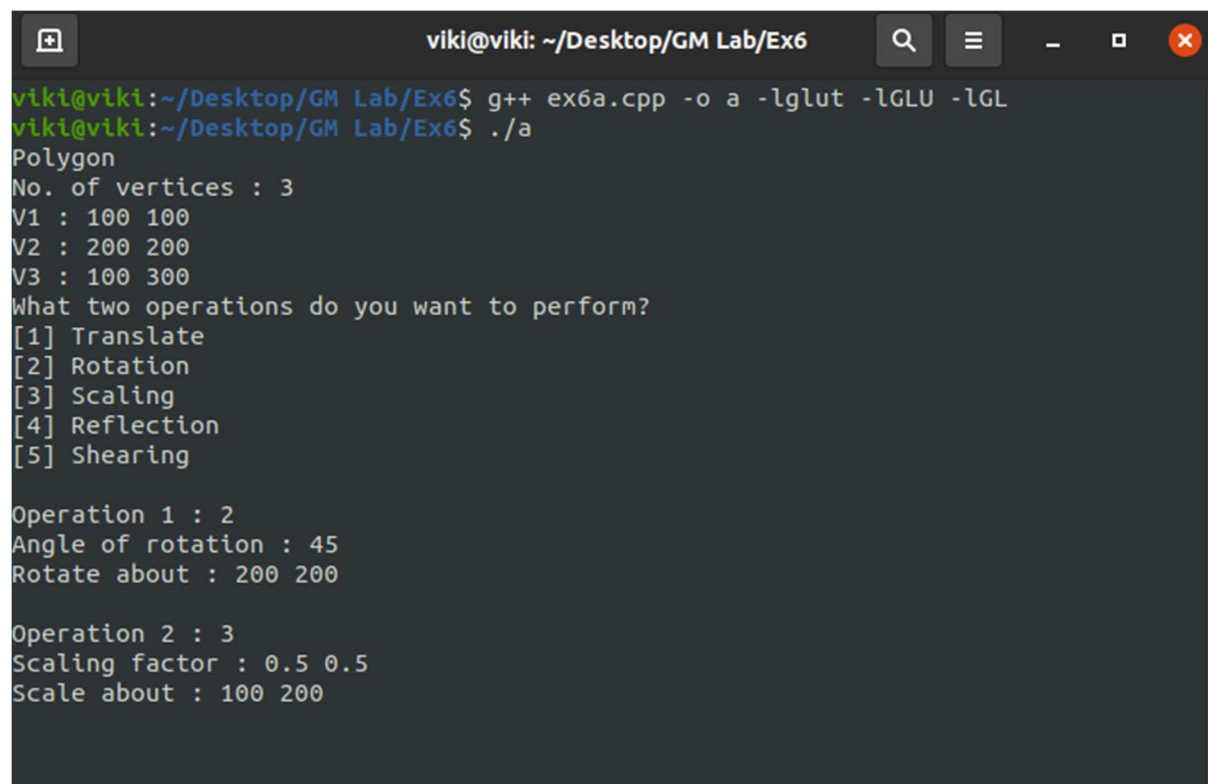
        cout<<"Min x & Max x : ";
        cin>>xv_min>>xv_max;
        cout<<"Min y & Max y : ";
        cin>>yv_min>>yv_max;

        sx=(xv_max-xv_min)*1.0/(xw_max-xw_min);
        sy=(yv_max-yv_min)*1.0/(yw_max-yw_min);

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(1000,1000);
        glutCreateWindow("Transformation");
        glutDisplayFunc(myDisplay);
        myInit();
        glutMainLoop();
        return 0;
}

```

Output:



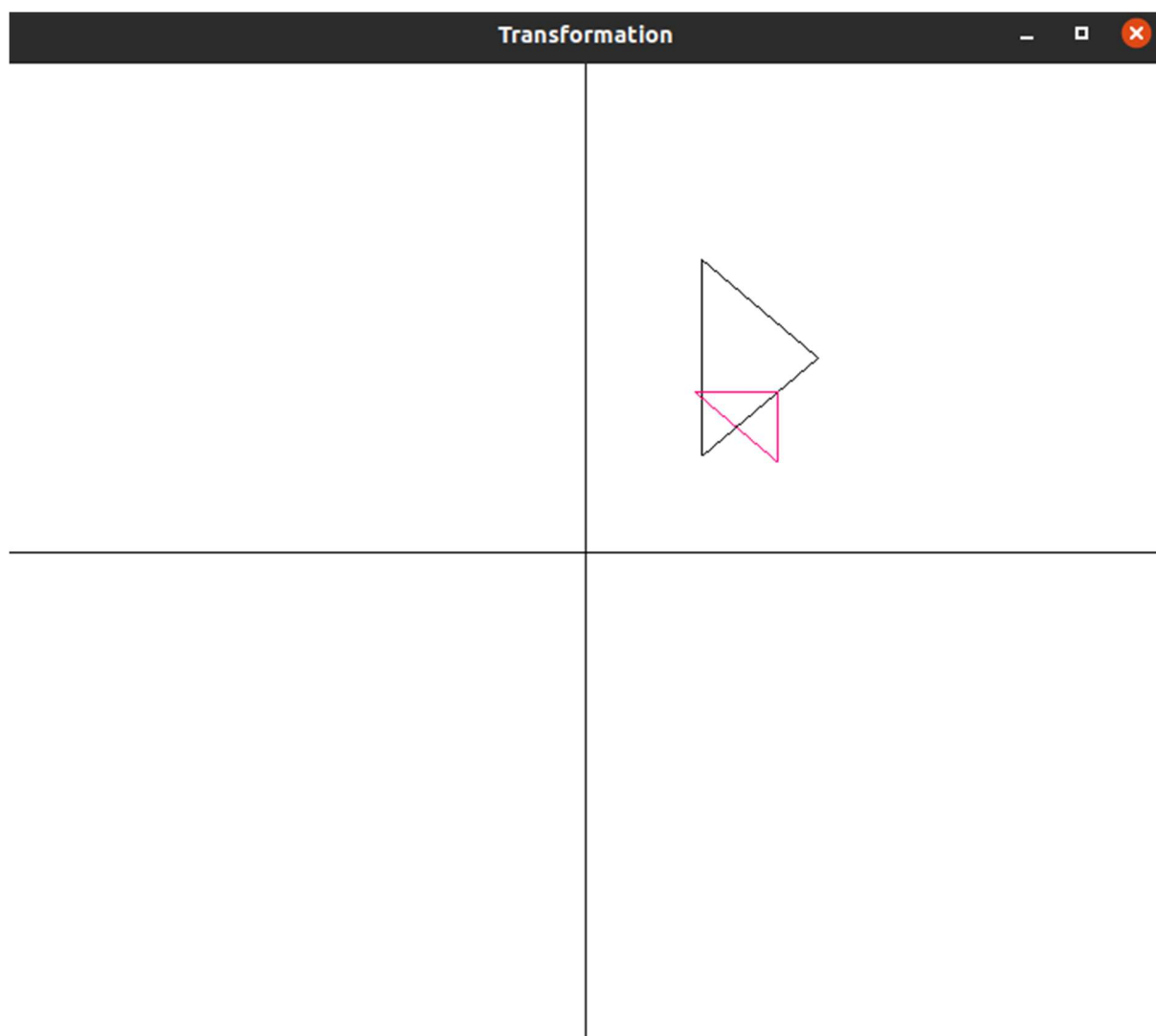
```

viki@viki: ~/Desktop/GM Lab/Ex6
viki@viki:~/Desktop/GM Lab/Ex6$ g++ ex6a.cpp -o a -lglut -lGLU -lGL
viki@viki:~/Desktop/GM Lab/Ex6$ ./a
Polygon
No. of vertices : 3
V1 : 100 100
V2 : 200 200
V3 : 100 300
What two operations do you want to perform?
[1] Translate
[2] Rotation
[3] Scaling
[4] Reflection
[5] Shearing

Operation 1 : 2
Angle of rotation : 45
Rotate about : 200 200

Operation 2 : 3
Scaling factor : 0.5 0.5
Scale about : 100 200

```



```
viki@viki: ~/Desktop/GM Lab/Ex6
viki@viki:~/Desktop/GM Lab/Ex6$ g++ ex6b.cpp -o a -lglut -lGLU -lGL
viki@viki:~/Desktop/GM Lab/Ex6$ ./a
Polygon
No. of vertices : 4
V1 : 100 100
V2 : 100 300
V3 : 300 300
V4 : 300 100
Enter Viewport details
Min x & Max x : 300 700
Min y & Max y : 400 900
```

