Lab Report 5

Title: Matrix Operations for Translation, Rotation, Scaling, Reflection, and Shearing

Theory: Matrix Operations for Translation, Rotation, Scaling, Reflection, and Shearing are crucial in computer graphics for their efficiency and accuracy in transforming objects using precise numerical computations. Unlike earlier methods like manual transformation, which required complex calculations, these operations rely on matrices, simplifying the process for devices with limited computational power.

One of their primary strengths lies in their versatility across different transformations. They manipulate objects seamlessly, applying translations, rotations, scalings, reflections, and shearings simultaneously. By leveraging matrix multiplication, they compute transformations efficiently and maintain visual fidelity, essential for real-time graphics applications.

This approach optimizes performance by executing transformations in a single operation, conserving computational resources and ensuring precise rendering. Its versatility and computational efficiency make it indispensable for software and systems requiring rapid and accurate graphical transformations.

Midpoint Algorithm:

`Step 1`: Start

`Step 2`: Initialize the graphics window and setup for drawing.

`Step 3`: Draw the initial shape (e.g., a right-angled triangle) using predefined coordinates.

`Step 4`: Define a 3x3 matrix shape to store the initial shape's vertices and their transformation.

`Step 5`: Display the composite matrix of the initial figure.

`Step 6`: Prompt user for transformation options:

1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shearing
6. Exit

`Step 7`: Depending on the user's choice:

- Case 1: Perform translation:
    - Prompt user for translation distances dx and dy.
    - Create a translation matrix.
    - Print and apply the translation matrix to shape.
    - Display the transformed shape.
- Case 2: Perform rotation:
    - Prompt user for rotation angle theta in degrees.
    - Convert angle to radians.
    - Create a rotation matrix using trigonometric functions.
    - Print and apply the rotation matrix to shape.
    - Display the rotated shape.
- Case 3: Perform scaling:
    - Prompt user for scaling factors sx and sy.
    - Create a scaling matrix.
    - Print and apply the scaling matrix to shape.
    - Display the scaled shape.
- Case 4: Perform reflection:
    - Apply reflection along the x-axis using a reflection matrix.
    - Print and apply the reflection matrix to shape.
    - Display the reflected shape.
- Case 5: Perform shearing:
    - Prompt user for shearing factors shx and shy.
    - Create a shearing matrix.
    - Print and apply the shearing matrix to shape.
    - Display the sheared shape.
- Case 6: Exit the program.

Step 8: Stop

The code:

Using the application of matrix, all the transformation was implanted with pure matrix implementation and class representation and operator overloading for multiplication operation:

```
#include <iostream>
#include <conio.h>
#include <graphics.h>
#include <vector>
```

```
#include <cmath>
using namespace std;

class Matrix {
private:
    int row;
    int column;
    vector<vector<float> > data;

public:
    Matrix(int r, int c) : row(r), column(c), data(r, vector<float>(c, 0)) {}

    void setValue(int r, int c, float value) {
        if (r >= 0 && r < row && c >= 0 && c < column) {
            data[r][c] = value;
        } else {
            cout << "Index out of bounds" << endl;
        }
    }

    int getValue(int r, int c) const {
        if (r >= 0 && r < row && c >= 0 && c < column) {
            return data[r][c];
        } else {
            cout << "Out of index" << endl;
            return -1;
        }
    }

    Matrix operator*(const Matrix& other) const {
        if (column != other.row) {
            cout << "Error: Matrix dimensions do not allow multiplication." << endl;
            return Matrix(0, 0);
        }

        Matrix result(row, other.column);
        for (int i = 0; i < row; ++i) {
            for (int j = 0; j < other.column; ++j) {
                result.data[i][j] = 0;
                for (int k = 0; k < column; ++k) {
                    result.data[i][j] += data[i][k] * other.data[k][j];
                }
            }
        }
        return result;
    }
   void print() const {
        for (int i = 0; i < row; ++i) {
            for (int j = 0; j < column; ++j) {
                cout << data[i][j] << " ";
            }
            cout << endl;
        }
    }
};

Matrix shape(3, 3);

void display() {
    line(shape.getValue(0, 0), shape.getValue(1, 0), shape.getValue(0, 1), shape.getValue(1, 1));
    line(shape.getValue(0, 0), shape.getValue(1, 0), shape.getValue(0, 2), shape.getValue(1, 2));
    line(shape.getValue(0, 2), shape.getValue(1, 2), shape.getValue(0, 1), shape.getValue(1, 1));
```

```
}

int main() {
    int options, dx, dy;
    initwindow(1000, 800, "Lab-5");

    // Right Angled Triangle
    line(100, 200, 300, 400);
    line(100, 200, 101, 401);
    line(101, 401, 300, 400);

    shape.setValue(0, 0, 100);
    shape.setValue(0, 1, 300);
    shape.setValue(0, 2, 101);
    shape.setValue(1, 0, 200);
    shape.setValue(1, 1, 400);
    shape.setValue(1, 2, 401);
    shape.setValue(2, 0, 1);
    shape.setValue(2, 1, 1);
    shape.setValue(2, 2, 1);

    cout << "What to do with the shape formed with given points?\n";
    cout << "1. Translation\n";
    cout << "2. Rotation\n";
    cout << "3. Scaling\n";
    cout << "4. Reflection\n";
    cout << "5. Shearing\n";
    cout << "6. Exit\n";
    cout << "Composite Matrix of the figure:\n";
    shape.print();
    cout << "Enter choice: ";
```

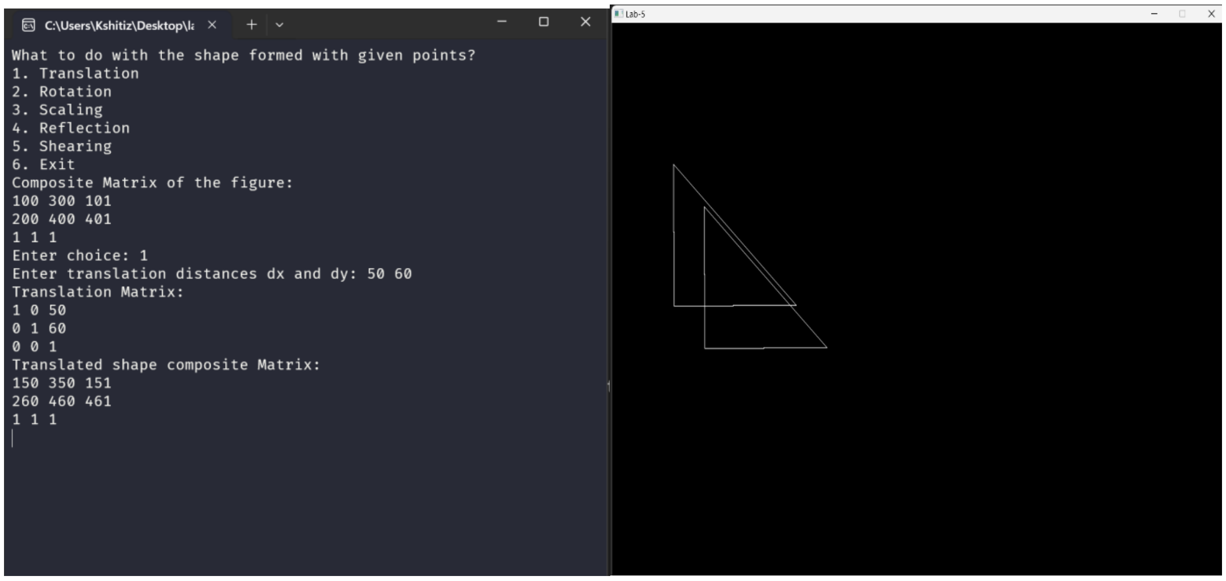
```
cin >> options;

switch (options) {
    case 1: {
        cout << "Enter translation distances dx and dy: ";
        cin >> dx >> dy;

        Matrix translationMatrix(3, 3);
        translationMatrix.setValue(0, 0, 1);
        translationMatrix.setValue(0, 1, 0);
        translationMatrix.setValue(0, 2, dx);
        translationMatrix.setValue(1, 0, 0);
        translationMatrix.setValue(1, 1, 1);
        translationMatrix.setValue(1, 2, dy);
        translationMatrix.setValue(2, 0, 0);
        translationMatrix.setValue(2, 1, 0);
        translationMatrix.setValue(2, 2, 1);
        cout << "Translation Matrix: \n";
        translationMatrix.print();
        shape = translationMatrix * shape;
        cout << "Translated shape composite Matrix: \n";
        shape.print();
        display();
        break;
    }

    case 2: {
        cout << "Enter the angle of rotation in degrees: ";
        int theta;
        cin >> theta;
        float radians = theta * M_PI / 180.0;
    Matrix rotationMatrix(3, 3);
    rotationMatrix.setValue(0, 0, cos(radians));
    rotationMatrix.setValue(0, 1, -sin(radians));
    rotationMatrix.setValue(0, 2, 0);
    rotationMatrix.setValue(1, 0, sin(radians));
    rotationMatrix.setValue(1, 1, cos(radians));
    rotationMatrix.setValue(1, 2, 0);
    rotationMatrix.setValue(2, 0, 0);
    rotationMatrix.setValue(2, 1, 0);
    rotationMatrix.setValue(2, 2, 1);
    cout << "Rotation Matrix with given angle of Rotation: \n";
    rotationMatrix.print();
    shape = rotationMatrix * shape;
    cout << "Rotated shape composite matrix: \n";
    shape.print();
    display();
    break;
}

case 3: {
    cout << "Enter the scaling you want in x and y: ";
    float sx , sy;
    cin >> sx >> sy;

    Matrix scalingMatrix(3, 3);
    scalingMatrix.setValue(0, 0, sx);
    scalingMatrix.setValue(0, 1, 0);
    scalingMatrix.setValue(0, 2, 0);
    scalingMatrix.setValue(1, 0, 0);
    scalingMatrix.setValue(1, 1, sy);
    scalingMatrix.setValue(1, 2, 0);
    scalingMatrix.setValue(2, 0, 0);
    scalingMatrix.setValue(2, 1, 0);
    scalingMatrix.setValue(2, 2, 1);
    cout << "Scaling Matrix: \n";
    scalingMatrix.print();
    shape = scalingMatrix * shape;
    cout << "Scaled shape composite Matrix: \n";
    shape.print();
    display();
    break;
}

case 4: {
    cout << "Reflection in x axis\n";
    Matrix reflectionMatrix(3, 3);
    reflectionMatrix.setValue(0, 0, 1);
    reflectionMatrix.setValue(0, 1, 0);
    reflectionMatrix.setValue(0, 2, 0);
    reflectionMatrix.setValue(1, 0, 0);
    reflectionMatrix.setValue(1, 1, -1);
    reflectionMatrix.setValue(1, 2, 0);
    reflectionMatrix.setValue(2, 0, 0);
    reflectionMatrix.setValue(2, 1, 0);
    reflectionMatrix.setValue(2, 2, 1);
    cout << "Reflection matrix: \n";
    reflectionMatrix.print();
    shape = reflectionMatrix * shape;
    cout << "Reflected shape composite Matrix: \n";
    shape.print();
    display();
    break;
}

case 5: {
    cout << "Enter the shearing you want in x and y: ";
    float shx , shy;
    cin >> shx >> shy;

    Matrix shearingMatrix(3, 3);
    shearingMatrix.setValue(0, 0, 1);
    shearingMatrix.setValue(0, 1, shx);
    shearingMatrix.setValue(0, 2, 0);
    shearingMatrix.setValue(1, 0, shy);
    shearingMatrix.setValue(1, 1, 1);
    shearingMatrix.setValue(1, 2, 0);
    shearingMatrix.setValue(2, 0, 0);
    shearingMatrix.setValue(2, 1, 0);
    shearingMatrix.setValue(2, 2, 1);
    cout << "Shearing matrix: \n";
    shearingMatrix.print();
    shape = shearingMatrix * shape;
    cout << "Sheared shape composite matrix: \n";
    shape.print();
    display();
    break;
}

case 6:
    cout << "Exiting..." << endl;
    break;
```

```
        default:
            cout << "Invalid option. Exiting..." << endl;
            break;
    }

    getch();
    closegraph();

    return 0;
}
```

The output:

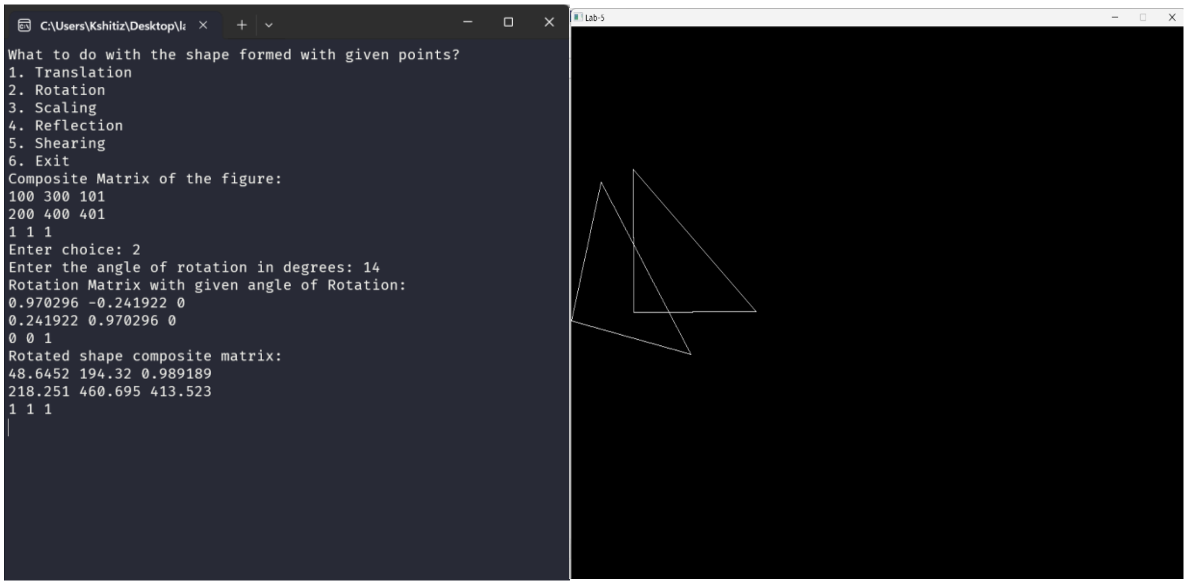
The output of the asked program is given below:

1. Translation



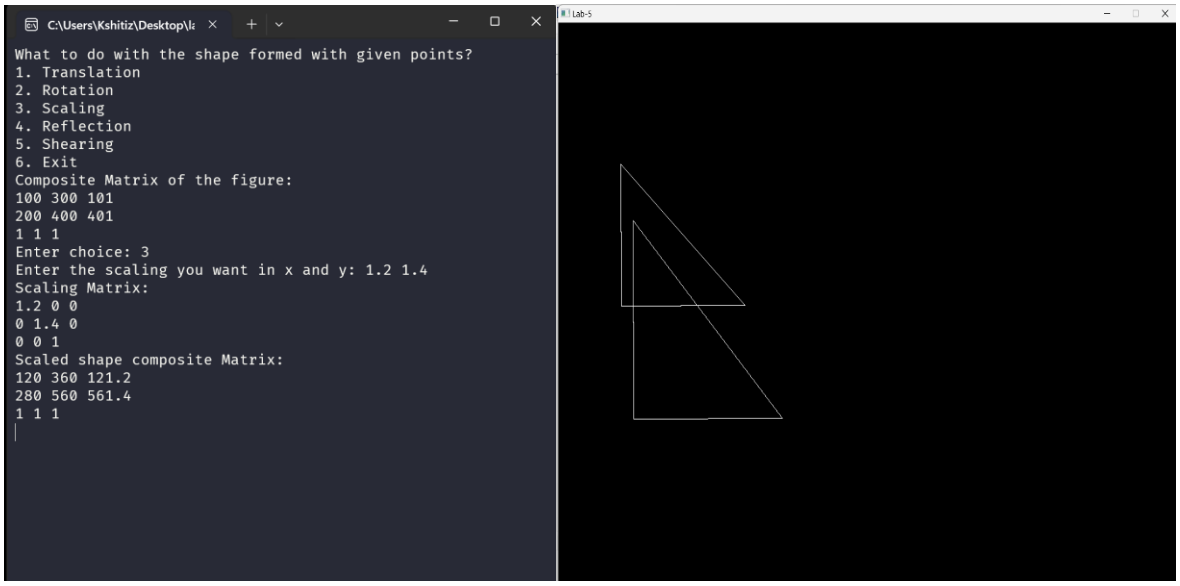
```
What to do with the shape formed with given points?
1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shearing
6. Exit
Composite Matrix of the figure:
100 300 101
200 400 401
1 1 1
Enter choice: 1
Enter translation distances dx and dy: 50 60
Translation Matrix:
1 0 50
0 1 60
0 0 1
Translated shape composite Matrix:
150 350 151
260 460 461
1 1 1
```

2. Rotation



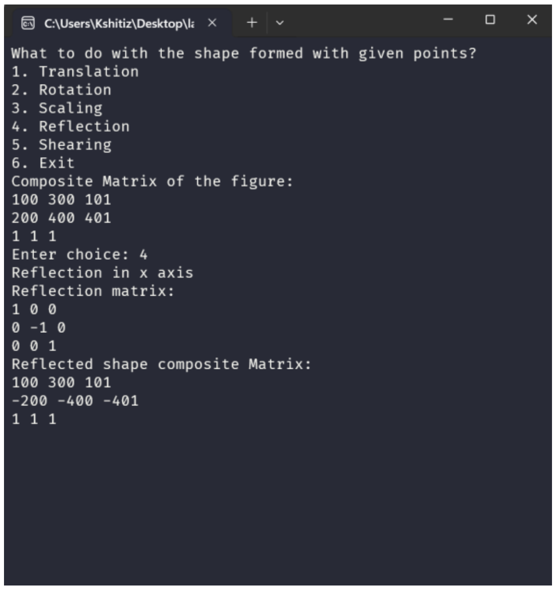
```
What to do with the shape formed with given points?
1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shearing
6. Exit
Composite Matrix of the figure:
100 300 101
200 400 401
1 1 1
Enter choice: 2
Enter the angle of rotation in degrees: 14
Rotation Matrix with given angle of Rotation:
0.970296 -0.241922 0
0.241922 0.970296 0
0 0 1
Rotated shape composite matrix:
48.6452 194.32 0.989189
218.251 460.695 413.523
1 1 1
```

3. Scaling



```
What to do with the shape formed with given points?
1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shearing
6. Exit
Composite Matrix of the figure:
100 300 101
200 400 401
1 1 1
Enter choice: 3
Enter the scaling you want in x and y: 1.2 1.4
Scaling Matrix:
1.2 0 0
0 1.4 0
0 0 1
Scaled shape composite Matrix:
120 360 121.2
280 560 561.4
1 1 1
```

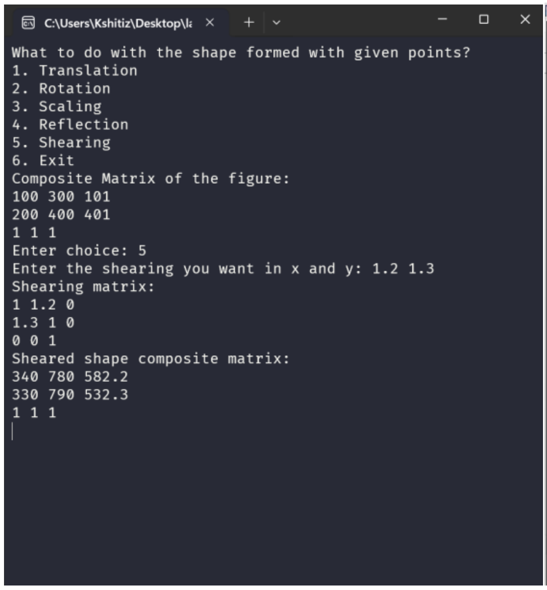
4. Reflection (X-axis)



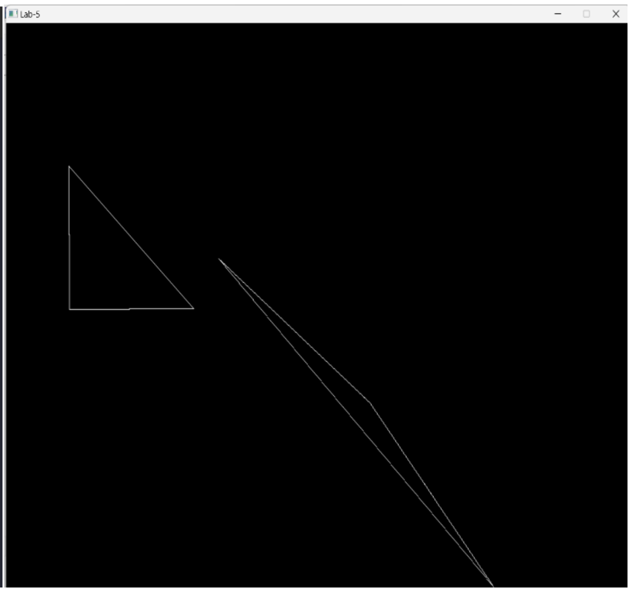
```
What to do with the shape formed with given points?
1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shearing
6. Exit
Composite Matrix of the figure:
100 300 101
200 400 401
1 1 1
Enter choice: 4
Reflection in x axis
Reflection matrix:
1 0 0
0 -1 0
0 0 1
Reflected shape composite Matrix:
100 300 101
-200 -400 -401
1 1 1
```

(The window didn't have enough space for plotting the negative coordinates as obtained)

5. Shearing



```
What to do with the shape formed with given points?
1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shearing
6. Exit
Composite Matrix of the figure:
100 300 101
200 400 401
1 1 1
Enter choice: 5
Enter the shearing you want in x and y: 1.2 1.3
Shearing matrix:
1 1.2 0
1.3 1 0
0 0 1
Sheared shape composite matrix:
340 780 582.2
330 790 532.3
1 1 1
```



Conclusion:

The matrix operations for transformation in computer graphics are crucial for their efficiency and precision. Unlike sequential methods, these operations handle transformations using matrix arithmetic, ensuring accuracy with integer calculations. The systematic application across all transformation types guarantees smooth rendering and precise adjustments, ideal for real-time graphics and interactive applications.