**DEERWALK INSTITUTE OF TECHNOLOGY**

**Tribhuvan University**

**Faculties of Computer Science**

# Bachelors of Science in Computer Science and Information Technology (BSc. CSIT)

## Course: Computer Graphics (CSC209)

**Year/Semester: II/III**

## A Lab report on:

## Implementation of 3D Transformation Algorithm in C++


Submitted by:
Name: Arun Mainali
Roll: 1307

Submitted to:
Binod Sitaula
Department of Computer Science


Submission Date: 03/20/2025

# ❖ LAB 6

## OBJECTIVE:

Write a program in any high-level language to implement 3D Transformations:

- Translation
- Rotation
- Scaling
- Reflection
- Shearing

## THEORY:

## 3D Transformations:

3D transformations are used to change the position, size, orientation, and shape of objects in a three-dimensional space. These transformations are performed using matrix operations on the coordinates of the objects. The basic 3D transformations include:

### 1. Translation:

Translation is the process of moving an object from one position to another along the x, y, and z axes. If a point P(x, y, z) is translated by distances Tx, Ty, and Tz along the x, y, and z axes respectively, the new coordinates P'(x', y', z') are given by:

x' = x + Tx y' = y + Ty z' = z + Tz

In matrix form, this can be represented as:

[x'] [1 0 0 Tx] [x] [y'] = [0 1 0 Ty] × [y] [z'] [0 0 1 Tz] [z] [1 ] [0 0 0 1 ] [1]

### 2. Scaling:

Scaling is the process of changing the size of an object. If a point P(x, y, z) is scaled by factors Sx, Sy, and Sz along the x, y, and z axes respectively, the new coordinates P'(x', y', z') are given by:

x' = x * Sx y' = y * Sy z' = z * Sz

In matrix form, this can be represented as:

[x'] [Sx 0 0 0] [x] [y'] = [0 Sy 0 0] × [y] [z'] [0 0 Sz 0] [z] [1 ] [0 0 0 1] [1]

### 3. Rotation:

Rotation in 3D space is more complex than in 2D space, as it can occur around any of the three principal axes (x, y, or z) or any arbitrary axis.

Rotation around the x-axis by an angle θ:

[x'] [1 0 0 0] [x] [y'] = [0 cos(θ) -sin(θ) 0] × [y] [z'] [0 sin(θ) cos(θ) 0] [z] [1 ] [0 0 0 1] [1]

Rotation around the y-axis by an angle θ:

[x'] [cos(θ) 0 sin(θ) 0] [x] [y'] = [0 1 0 0] × [y] [z'] [-sin(θ) 0 cos(θ) 0] [z] [1 ] [0 0 0 1] [1]

Rotation around the z-axis by an angle θ:

[x'] [cos(θ) -sin(θ) 0 0] [x] [y'] = [sin(θ) cos(θ) 0 0] × [y] [z'] [0 0 1 0] [z] [1 ] [0 0 0 1] [1]

### 4. Reflection:

Reflection in 3D space can be about any of the principal planes (xy, yz, or xz).

Reflection about the xy-plane:

[x'] [1 0 0 0] [x] [y'] = [0 1 0 0] × [y] [z'] [0 0 -1 0] [z] [1 ] [0 0 0 1] [1]

Reflection about the yz-plane:

[x'] [-1 0 0 0] [x] [y'] = [0 1 0 0] × [y] [z'] [0 0 1 0] [z] [1 ] [0 0 0 1] [1]

Reflection about the xz-plane:

[x'] [1 0 0 0] [x] [y'] = [0 -1 0 0] × [y] [z'] [0 0 1 0] [z] [1 ] [0 0 0 1] [1]

### 5. Shearing:

Shearing in 3D space can be applied along any of the axes.

Shearing along the x-axis:

[x'] [1 Shxy Shxz 0] [x] [y'] = [0 1 0 0] × [y] [z'] [0 0 1 0] [z] [1 ] [0 0 0 1] [1]

Shearing along the y-axis:

[x'] [1 0 0 0] [x] [y'] = [Shyx 1 Shyz 0] × [y] [z'] [0 0 1 0] [z] [1 ] [0 0 0 1] [1]

Shearing along the z-axis:

[x'] [1 0 0 0] [x] [y'] = [0 1 0 0] × [y] [z'] [Shzx Shzy 1 0] [z] [1 ] [0 0 0 1] [1]

## Advantages of 3D Transformations:

1. They allow for realistic representation and manipulation of objects in 3D space.
2. They are essential for computer graphics, animation, and virtual reality applications.
3. They can be combined to create complex transformations.
4. They can be efficiently implemented using matrix operations.
5. They enable the creation of photorealistic 3D models and scenes.

## ALGORITHM:

## 1. Translation Algorithm:

**Step 1:** Take the input coordinates of the 3D object. **Step 2:** Take the translation factors Tx, Ty, and Tz. **Step 3:** For each point (x, y, z) in the object:

- Calculate x' = x + Tx
- Calculate y' = y + Ty

- Calculate z' = z + Tz **Step 4:** Return the new coordinates (x', y', z').

## 2. Scaling Algorithm:

**Step 1:** Take the input coordinates of the 3D object. **Step 2:** Take the scaling factors Sx, Sy, and Sz. **Step 3:** For each point (x, y, z) in the object:

- Calculate x' = x * Sx
- Calculate y' = y * Sy
- Calculate z' = z * Sz **Step 4:** Return the new coordinates (x', y', z').

## 3. Rotation Algorithm:

**Step 1:** Take the input coordinates of the 3D object. **Step 2:** Take the rotation axis (x, y, or z) and the rotation angle θ. **Step 3:** For each point (x, y, z) in the object:

- For rotation around the x-axis:
    - Calculate y' = y * cos(θ) - z * sin(θ)
    - Calculate z' = y * sin(θ) + z * cos(θ)
    - x' = x
- For rotation around the y-axis:
    - Calculate x' = x * cos(θ) + z * sin(θ)
    - Calculate z' = -x * sin(θ) + z * cos(θ)
    - y' = y
- For rotation around the z-axis:
    - Calculate x' = x * cos(θ) - y * sin(θ)
    - Calculate y' = x * sin(θ) + y * cos(θ)
    - z' = z **Step 4:** Return the new coordinates (x', y', z').

## 4. Reflection Algorithm:

**Step 1:** Take the input coordinates of the 3D object. **Step 2:** Choose the reflection plane (xy, yz, or xz). **Step 3:** For each point (x, y, z) in the object:

- For reflection about the xy-plane:
    - Calculate x' = x, y' = y, z' = -z
- For reflection about the yz-plane:
    - Calculate x' = -x, y' = y, z' = z
- For reflection about the xz-plane:
    - Calculate x' = x, y' = -y, z' = z **Step 4:** Return the new coordinates (x', y', z').

## 5. Shearing Algorithm:

**Step 1:** Take the input coordinates of the 3D object. **Step 2:** Choose the shearing axis (x, y, or z) and the shearing factors. **Step 3:** For each point (x, y, z) in the object:

- For shearing along the x-axis:
    - Calculate x' = x + y * Shxy + z * Shxz

        - Calculate y' = y
        - Calculate z' = z
    - For shearing along the y-axis:
        - Calculate x' = x
        - Calculate y' = y + x * Shyx + z * Shyz
        - Calculate z' = z
    - For shearing along the z-axis:
        - Calculate x' = x
        - Calculate y' = y
        - Calculate z' = z + x * Shzx + y * Shzy **Step 4:** Return the new coordinates (x', y', z').

## Implementation of 3D Transformation Algorithms in OpenGL:

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <iostream>
#include <vector>
#include <cmath>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

const unsigned int SCR_WIDTH = 2000;
const unsigned int SCR_HEIGHT = 1600;

// Vertex shader source code
const char* vertexShaderSource = R"(
#version 330 core
layout (location = 0) in vec3 aPos;
uniform mat4 transform;
uniform mat4 view;
uniform mat4 projection;
void main()
{
    gl_Position = projection * view * transform * vec4(aPos, 1.0);
}
)";

// Fragment shader source code
```

```
const char* fragmentShaderSource = R"(
#version 330 core
out vec4 FragColor;
uniform vec3 color;
void main()
{
    FragColor = vec4(color, 1.0f);
}
)";


// Function declarations
void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow* window);
unsigned int createShaderProgram();
void renderObject(unsigned int shaderProgram, const std::vector<float>&
vertices);
std::vector<float> applyTransformation(const std::vector<float>& vertices,
const glm::mat4& matrix);


int main()
{
    // Initialize GLFW
    if (!glfwInit())
    {
        std::cerr << "Failed to initialize GLFW" << std::endl;
        return -1;
    }


    // Configure GLFW
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);


#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif
```

```
    // Create window
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "3D Transformation", NULL, NULL);
    if (!window)
    {
        std::cerr << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    // Initialize GLAD
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cerr << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    // Enable depth testing
    glEnable(GL_DEPTH_TEST);

    // Define object vertices (a simple cube)
    std::vector<float> originalVertices = {
        // Front face
        -0.5f, -0.5f, 0.5f,
        0.5f, -0.5f, 0.5f,
        0.5f, 0.5f, 0.5f,
        0.5f, 0.5f, 0.5f,
        -0.5f, 0.5f, 0.5f,
        -0.5f, -0.5f, 0.5f,

        // Back face
        -0.5f, -0.5f, -0.5f,
        0.5f, -0.5f, -0.5f,
```

```
        0.5f, 0.5f, -0.5f,
        0.5f, 0.5f, -0.5f,
        -0.5f, 0.5f, -0.5f,
        -0.5f, -0.5f, -0.5f,

        // Left face
        -0.5f, -0.5f, -0.5f,
        -0.5f, 0.5f, -0.5f,
        -0.5f, 0.5f, 0.5f,
        -0.5f, 0.5f, 0.5f,
        -0.5f, -0.5f, 0.5f,
        -0.5f, -0.5f, -0.5f,

        // Right face
        0.5f, -0.5f, -0.5f,
        0.5f, 0.5f, -0.5f,
        0.5f, 0.5f, 0.5f,
        0.5f, 0.5f, 0.5f,
        0.5f, -0.5f, 0.5f,
        0.5f, -0.5f, -0.5f,

        // Top face
        -0.5f, 0.5f, -0.5f,
        0.5f, 0.5f, -0.5f,
        0.5f, 0.5f, 0.5f,
        0.5f, 0.5f, 0.5f,
        -0.5f, 0.5f, 0.5f,
        -0.5f, 0.5f, -0.5f,

        // Bottom face
        -0.5f, -0.5f, -0.5f,
        0.5f, -0.5f, -0.5f,
        0.5f, -0.5f, 0.5f,
        0.5f, -0.5f, 0.5f,
        -0.5f, -0.5f, 0.5f,
        -0.5f, -0.5f, -0.5f
```

```
    };

    // Create shader program
    unsigned int shaderProgram = createShaderProgram();

    // View and projection matrices
    glm::mat4 view = glm::lookAt(
        glm::vec3(2.0f, 2.0f, 2.0f),
        glm::vec3(0.0f, 0.0f, 0.0f),
        glm::vec3(0.0f, 1.0f, 0.0f)
    );
    glm::mat4 projection = glm::perspective(glm::radians(45.0f),
(float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);

    // Main loop
    int transformationChoice = 0;
    std::vector<float> transformedVertices;
    while (!glfwWindowShouldClose(window))
    {
        processInput(window);

        // Clear the screen
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Display menu
        if (transformationChoice == 0)
        {
            std::cout << "Choose a transformation:\n";
            std::cout << "1. Translation\n";
            std::cout << "2. Scaling\n";
            std::cout << "3. Rotation\n";
            std::cout << "4. Reflection\n";
            std::cout << "5. Shearing\n";
            std::cout << "6. Exit\n";
            std::cin >> transformationChoice;
```

```
            if (transformationChoice == 6)
            {
                glfwSetWindowShouldClose(window, true);
                continue;
            }
        }


        // Apply transformation
        glm::mat4 transformMatrix = glm::mat4(1.0f);
        switch (transformationChoice)
        {
        case 1: // Translation
        {
            float tx, ty, tz;
            std::cout << "Enter translation factors (tx ty tz): ";
            std::cin >> tx >> ty >> tz;
            transformMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(tx, ty,
tz));
            transformedVertices = applyTransformation(originalVertices,
transformMatrix);
            break;
        }
        case 2: // Scaling
        {
            float sx, sy, sz;
            std::cout << "Enter scaling factors (sx sy sz): ";
            std::cin >> sx >> sy >> sz;
            transformMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(sx, sy,
sz));
            transformedVertices = applyTransformation(originalVertices,
transformMatrix);
            break;
        }
        case 3: // Rotation
        {
            int axis;
            float angle;
```

```
            std::cout << "Choose rotation axis (1:x, 2:y, 3:z): ";
            std::cin >> axis;
            std::cout << "Enter rotation angle (in degrees): ";
            std::cin >> angle;

            if (axis == 1)
                transformMatrix = glm::rotate(glm::mat4(1.0f),
glm::radians(angle), glm::vec3(1.0f, 0.0f, 0.0f));
            else if (axis == 2)
                transformMatrix = glm::rotate(glm::mat4(1.0f),
glm::radians(angle), glm::vec3(0.0f, 1.0f, 0.0f));
            else if (axis == 3)
                transformMatrix = glm::rotate(glm::mat4(1.0f),
glm::radians(angle), glm::vec3(0.0f, 0.0f, 1.0f));

            transformedVertices = applyTransformation(originalVertices,
transformMatrix);
            break;
        }
        case 4: // Reflection
        {
            int reflectionType;
            std::cout << "Choose reflection plane (1:xy, 2:yz, 3:xz): ";
            std::cin >> reflectionType;

            if (reflectionType == 1) // xy-plane
                transformMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(1.0f,
1.0f, -1.0f));
            else if (reflectionType == 2) // yz-plane
                transformMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(-1.0f,
1.0f, 1.0f));
            else if (reflectionType == 3) // xz-plane
                transformMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(1.0f,
-1.0f, 1.0f));

            transformedVertices = applyTransformation(originalVertices,
transformMatrix);
            break;
        }
```

```
        case 5: // Shearing
        {
            int shearingAxis;
            std::cout << "Choose shearing axis (1:x, 2:y, 3:z): ";
            std::cin >> shearingAxis;

            if (shearingAxis == 1) // x-axis
            {
                float shxy, shxz;
                std::cout << "Enter shearing factors (shxy shxz): ";
                std::cin >> shxy >> shxz;
                transformMatrix[1][0] = shxy;
                transformMatrix[2][0] = shxz;
            }
            else if (shearingAxis == 2) // y-axis
            {
                float shyx, shyz;
                std::cout << "Enter shearing factors (shyx shyz): ";
                std::cin >> shyx >> shyz;
                transformMatrix[0][1] = shyx;
                transformMatrix[2][1] = shyz;
            }
            else if (shearingAxis == 3) // z-axis
            {
                float shzx, shzy;
                std::cout << "Enter shearing factors (shzx shzy): ";
                std::cin >> shzx >> shzy;
                transformMatrix[0][2] = shzx;
                transformMatrix[1][2] = shzy;
            }

            transformedVertices = applyTransformation(originalVertices,
transformMatrix);
            break;
        }
        default:
```

```
            transformedVertices = originalVertices;
            break;
        }

        // Render the original object
        glUseProgram(shaderProgram);
        glUniform3f(glGetUniformLocation(shaderProgram, "color"), 1.0f, 0.0f, 0.0f); // Red color

        // Pass view and projection matrices
        glUniformMatrix4fv(glGetUniformLocation(shaderProgram, "view"), 1, GL_FALSE, glm::value_ptr(view));
        glUniformMatrix4fv(glGetUniformLocation(shaderProgram, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

        // Identity matrix for original object
        glm::mat4 identityMatrix = glm::mat4(1.0f);
        glUniformMatrix4fv(glGetUniformLocation(shaderProgram, "transform"), 1, GL_FALSE, glm::value_ptr(identityMatrix));
        renderObject(shaderProgram, originalVertices);

        // Render the transformed object
        glUniform3f(glGetUniformLocation(shaderProgram, "color"), 0.0f, 1.0f, 0.0f); // Green color
        glUniformMatrix4fv(glGetUniformLocation(shaderProgram, "transform"), 1, GL_FALSE, glm::value_ptr(transformMatrix));
        renderObject(shaderProgram, originalVertices);

        // Swap buffers and poll events
        glfwSwapBuffers(window);
        glfwPollEvents();

        // Reset transformation choice
        transformationChoice = 0;
    }

    // Clean up
    glDeleteProgram(shaderProgram);
```

```
    glfwTerminate();
    return 0;
}

void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}

void processInput(GLFWwindow* window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

unsigned int createShaderProgram()
{
    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    int success;
    char infoLog[512];
    glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
    if (!success)
    {
        glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
        std::cerr << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" << infoLog
<< std::endl;
    }

    unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShader);
    glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
    if (!success)
    {
```

```
        glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);
        std::cerr << "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n" << infoLog
<< std::endl;
    }

    unsigned int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
    glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
    if (!success)
    {
        glGetProgramInfoLog(shaderProgram, 512, NULL, infoLog);
        std::cerr << "ERROR::SHADER::PROGRAM::LINKING_FAILED\n" << infoLog <<
std::endl;
    }

    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);
    return shaderProgram;
}

void renderObject(unsigned int shaderProgram, const std::vector<float>&
vertices)
{
    if (vertices.empty())
    {
        return;
    }

    unsigned int VBO, VAO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(float),
vertices.data(), GL_STATIC_DRAW);
```

```
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
    glDrawArrays(GL_TRIANGLES, 0, vertices.size() / 3);
    glBindVertexArray(0);
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
}


std::vector<float> applyTransformation(const std::vector<float>& vertices, const glm::mat4& matrix)
{
    std::vector<float> result;
    for (size_t i = 0; i < vertices.size(); i += 3)
    {
        float x = vertices[i];
        float y = vertices[i + 1];
        float z = vertices[i + 2];

        // Apply transformation matrix
        glm::vec4 point = matrix * glm::vec4(x, y, z, 1.0f);

        result.push_back(point.x);
        result.push_back(point.y);
        result.push_back(point.z);
    }
    return result;
}
```

## CONCLUSION:

Hence, in this lab work we were able to implement 3D transformation using C programming as the high- level language.