Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavrepalanchowk

A Lab Report
On
"COMP 342"

Submitted by: Aaditya K.C
Roll No:05
Level:UNG CS(III/I)

Submitted to: Dhiraj Shrestha
Department of Computer Science and Engineering
Submission Date: 01/20/2026

1. **Implement the following 3D transformations using the 3D shapes provided by Opengl:**

a. Translation

3D Translation is the process of moving an object from one position to another in three-dimensional space without altering its size or orientation.

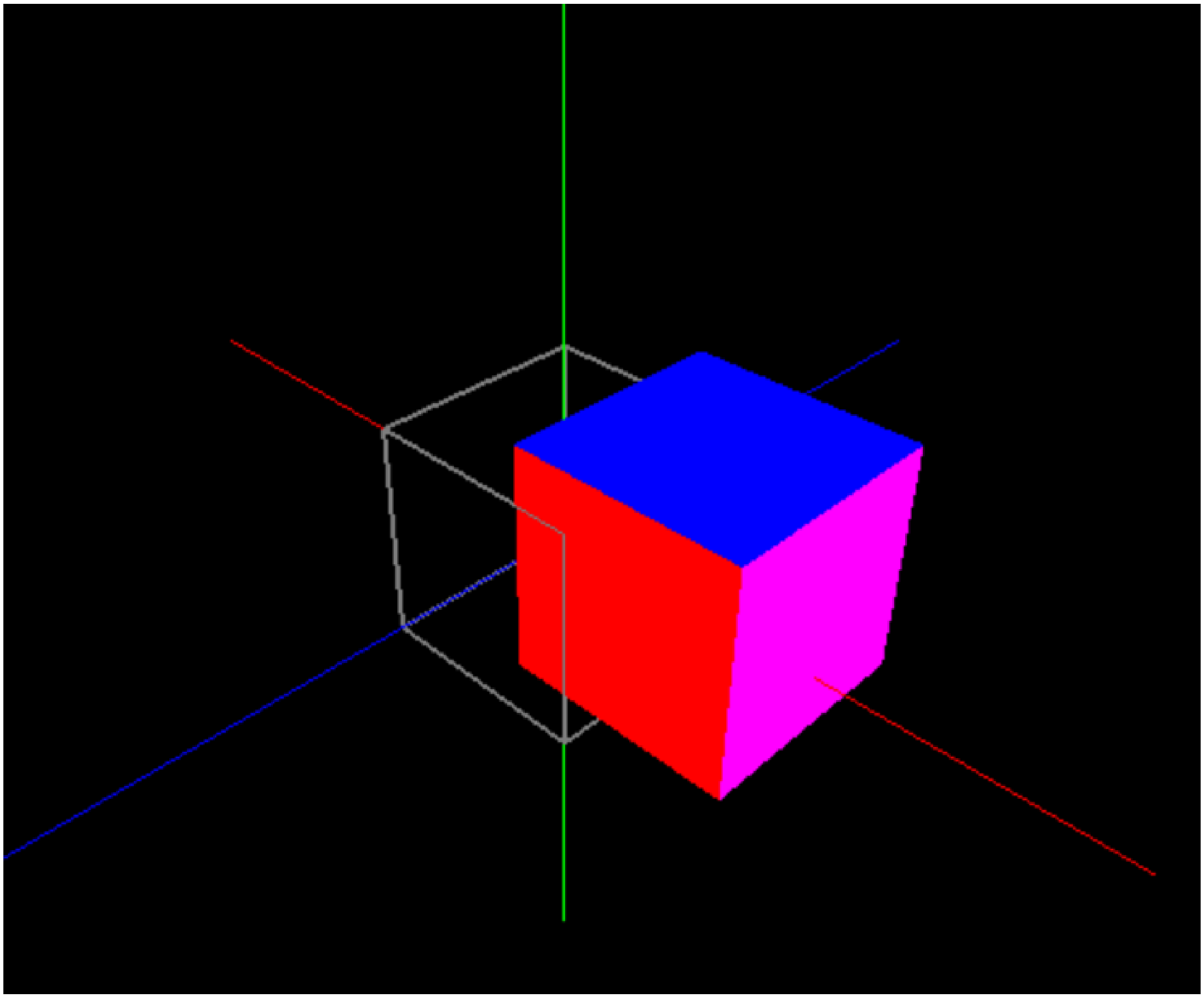
It is achieved by adding a displacement vector [t_x, t_y, t_z] to the original coordinates.

$$\text{Transformation Matrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The code and the output for 3D translation is below:

```python
# Display function
def display():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()

    # Camera position
    gluLookAt(6, 6, 6, 0, 0, 0, 0, 1, 0)

    draw_axes()

    # Draw BEFORE: Original cube as wireframe
    glPushMatrix()
    draw_wireframe_cube()
    glPopMatrix()

    # Draw AFTER: Transformed cube with colors
    glPushMatrix()

    if transform_mode == "translate":
        # Translation matrix: move by (1.5, 0.5, 0)
        translation_matrix = [
            1, 0, 0, 0,
            0, 1, 0, 0,
            0, 0, 1, 0,
            1.5, 0.5, 0, 1
        ]
        glMultMatrixf(translation_matrix)
```

code



Output

b. Rotate

3D rotation in computer graphics is the transformation of an object in 3D space around a fixed axis (x, y, z) by a specific angle. I have taken 90 degrees as the angle of rotation.

The transformation matrix of rotation along respective axis are as follows:

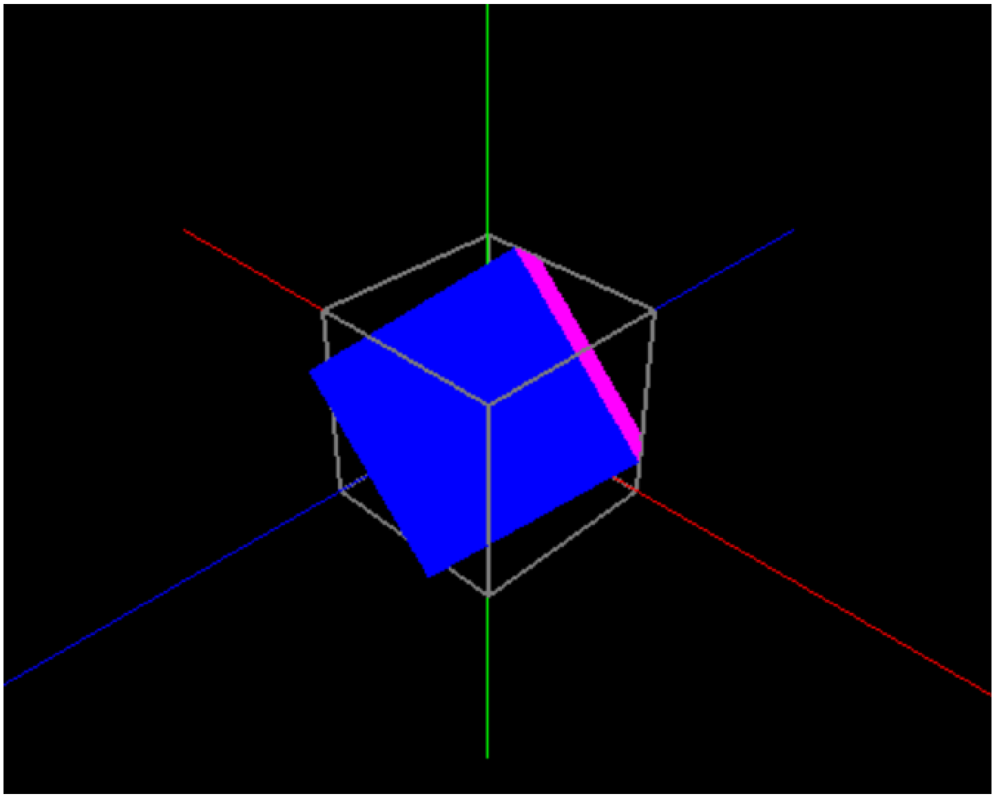
a. X-axis: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

b. Y-axis: $\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

c. Z-axis: $\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

```python
elif transform_mode == "rotate":
    # Rotation matrix: 90 degrees around axis (1, 1, 0)
    # Normalize the axis
    ax, ay, az = 1, 1, 0
    length = math.sqrt(ax**2 + ay**2 + az**2)
    ax, ay, az = ax/length, ay/length, az/length

    # Convert angle to radians
    rad = math.radians(angle)
    c = math.cos(rad)
    s = math.sin(rad)
    t = 1 - c

    # Rodrigues' rotation formula in matrix form
    rotation_matrix = [
        t*ax*ax + c,    t*ax*ay + s*az, t*ax*az - s*ay, 0,
        t*ax*ay - s*az, t*ay*ay + c,    t*ay*az + s*ax, 0,
        t*ax*az + s*ay, t*ay*az - s*ax, t*az*az + c,    0,
        0,              0,              0,              1
    ]
    glMultMatrixf(rotation_matrix)
```

Code For Rotation



Output

c. Scaling

3D scaling is the transformation method in which the object is resized by multiplying the coordinates [x, y, z] by scaling factors [Sx, Sy, Sz] along each axis.

The transformation matrix of 3D shearing along respective axis are as follows:

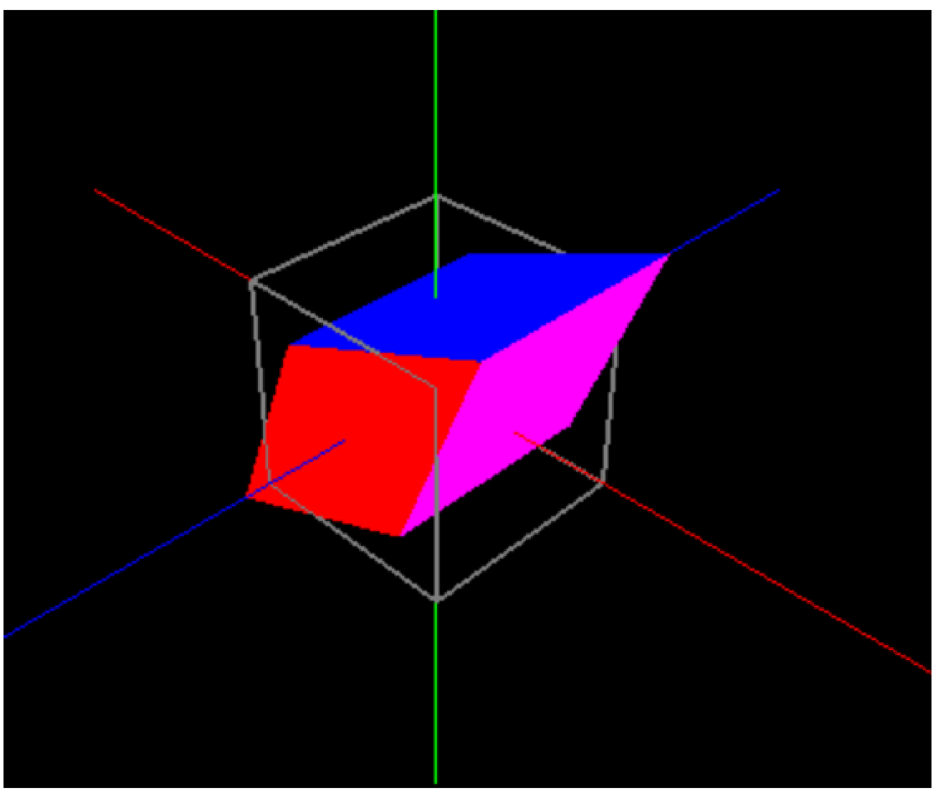
a. X-axis: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ S_{hy} & 1 & 0 & 0 \\ S_{hz} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

b. Y-axis: $\begin{bmatrix} 1 & S_{hx} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & S_{hz} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

c. Z-axis: $\begin{bmatrix} 1 & 0 & S_{hx} & 0 \\ 0 & 1 & S_{hy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

```
    # Shearing matrix
    shear_matrix = [
        1, 0.4, 0, 0,
        0.4, 1, 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1
    ]
    glMultMatrixf(shear_matrix)

draw_colored_cube()
glPopMatrix()
```

Code For Shearing



Output Of Scaling

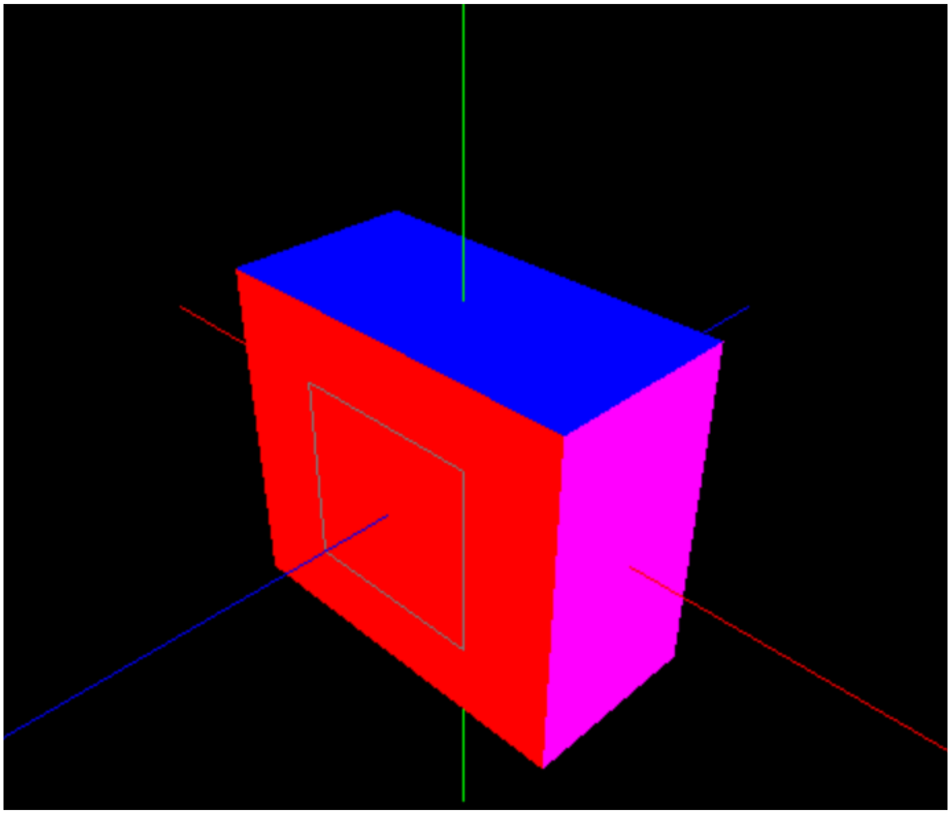
d. Scaling

3D scaling is the transformation method in which the object is resized by multiplying the coordinates [x, y, z] by scaling factors [Sx, Sy, Sz] along each axis.

$$\text{Transformation Matrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
transform_mode == "scale":
 Scaling matrix: scale by (1.2, 0.8, 1
caling_matrix = [
   1.2, 0, 0, 0,
   0, 0.8, 0, 0,
   0, 0, 1, 0,
   0, 0, 0, 1

lMultMatrixf(scaling_matrix)
```

Code For Scaling



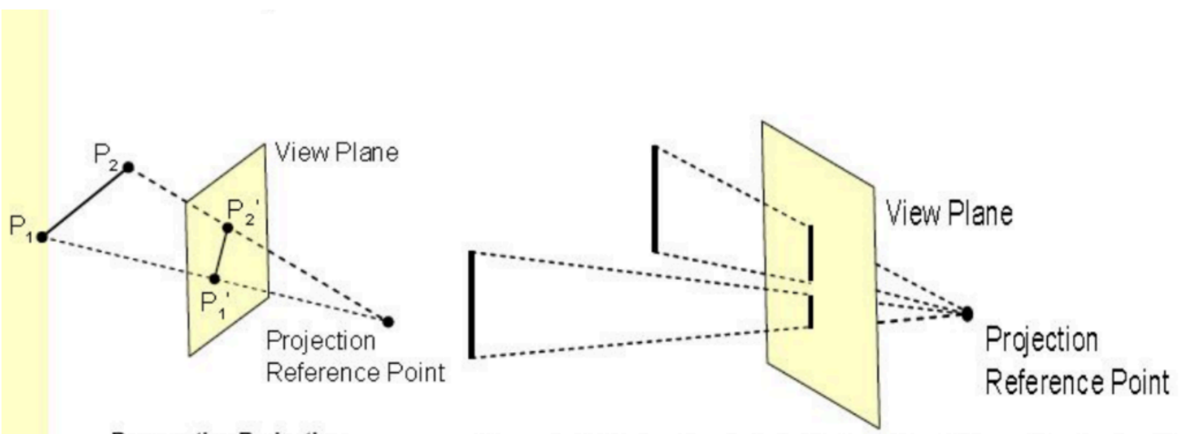
Output

2. Perspective Projection

Perspective Projection is a technique in computer graphics used to project 3D points onto a 2D view plane such that objects farther from the viewer appear smaller, mimicking human vision. It gives a sense of depth in the scene.

How it works:

- Each 3D point (x,y,z)(x, y, z)(x,y,z) is mapped to a 2D point (x′,y′)(x', y')(x′,y′) on the view plane.
- The farther the object along the Z-axis, the smaller it appears.
- The transformation depends on the viewing distance ddd (distance from the viewer to the projection plane).

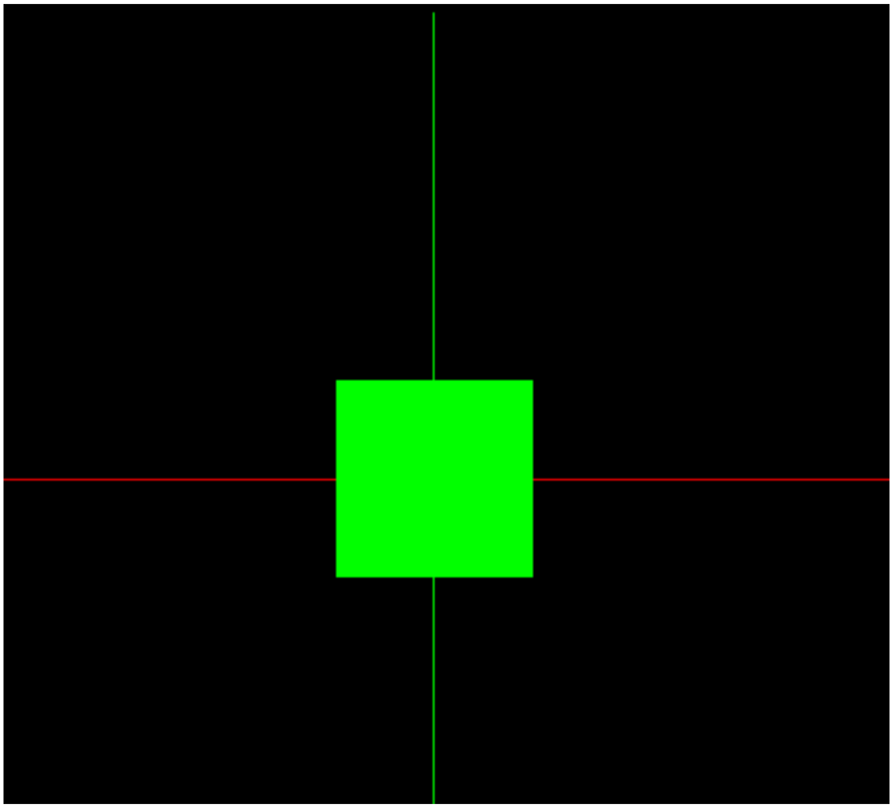


Perspective Projection

Perspective Projection of equal sized objects at different distances from the view plane

```
lab5 > prospective.py > main
def draw_colored_cube(size):
    glVertex3f(-size, -size, size)

    glEnd()

# Perspective Projection Display
def display():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()

    # Camera position - Z-axis towards screen
    gluLookAt(0, 0, -10,    # Eye position (behind cube)
              0, 0, 0,      # Look-at point
              0, 1, 0)      # Up direction

    draw_axes()

    # Draw cube at user-specified Z position
    glPushMatrix()
    glTranslatef(0, 0, cube_z)
    draw_colored_cube(cube_size)
    glPopMatrix()

    glutSwapBuffers()

# --------------- INIT ---------------
def init():
    glClearColor(0, 0, 0, 1)
    glEnable(GL_DEPTH_TEST)

    # Perspective Projection
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60, 1, 0.1, 50)
    glMatrixMode(GL_MODELVIEW)

# Keyboard callback to move cube along Z axis
def keyboard(key, x, y):
    global cube_z
```

Code for Perspective projection



Normal Image

image as the object comes near

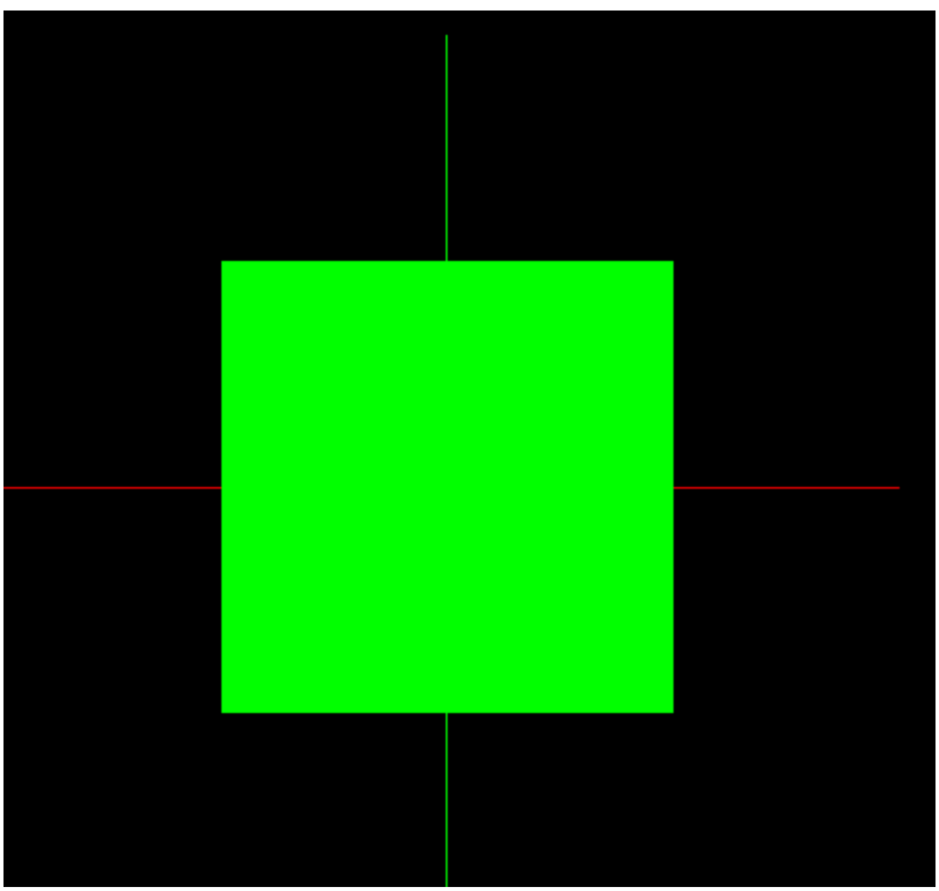


Image as the object moves away