Graphics Assignment No:- 5

1. 3D Transformation:

1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shear

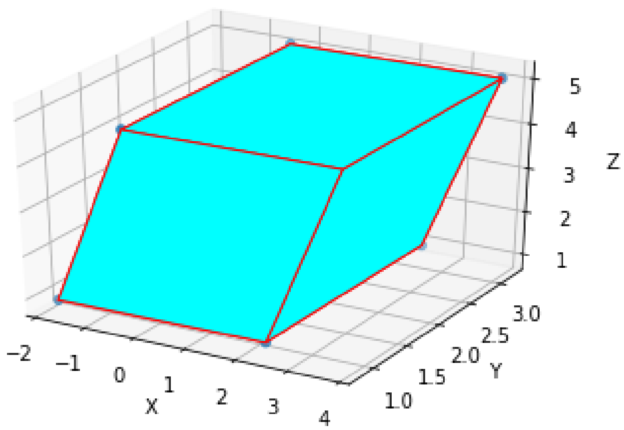
2. Translation

Python code

```
# translation
import warnings
import math
import numpy

def translation_matrix(direction):
    M = numpy.identity(4)
    M[:3, 3] = direction[:3]
    return M
T = translation_matrix((1, 2, 3))
print(T)

# translation
ZZ = np.concatenate([Z, np.ones((Z.shape[0], 1))], axis=1)
out = np.zeros((8,4))
out = np.dot(T, ZZ.T).T[:, :-1]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

r = [-1,1]

X, Y = np.meshgrid(r, r)
# plot vertices
ax.scatter3D(out[:, 0], out[:, 1], out[:, 2])

# list of sides' polygons of figure

verts = [[out[0],out[1],out[2],out[3]],
 [out[4],out[5],out[6],out[7]],
 [out[0],out[1],out[5],out[4]],
 [out[2],out[3],out[7],out[6]],
 [out[1],out[2],out[6],out[5]],
 [out[4],out[7],out[3],out[0]]]

# plot sides
ax.add_collection3d(Poly3DCollection(verts,
 facecolors='cyan', linewidths=1, edgecolors='r', alpha=.25))

ax.set_xlabel('X')
ax.set_ylabel('Y')
```

```
ax.set_zlabel('Z')

plt.show()
```

Result



3. Rotation

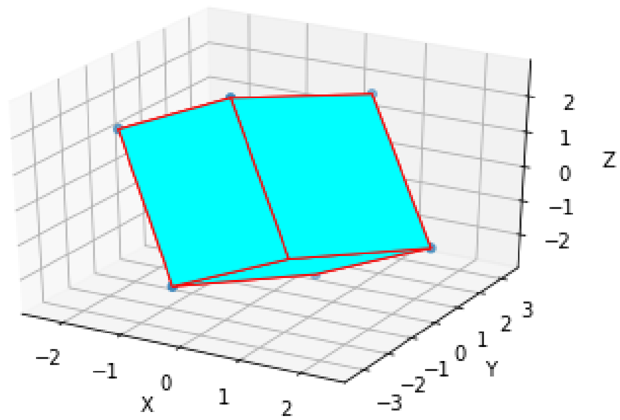
Python code

```
# rotation
_EPS = numpy.finfo(float).eps * 4.0
from __future__ import division

import warnings
import math
import numpy
def unit_vector(data, axis=None, out=None):
    if out is None:
        data = numpy.array(data, dtype=numpy.float64, copy=True)
        if data.ndim == 1:
            data /= math.sqrt(numpy.dot(data, data))
            return data
    else:
        if out is not data:
            out[:] = numpy.array(data, copy=False)
        data = out
    length = numpy.atleast_1d(numpy.sum(data*data, axis))
    numpy.sqrt(length, length)
    if axis is not None:
        length = numpy.expand_dims(length, axis)
    data /= length
    if out is None:
        return data
def quaternion_matrix(quaternion):
    q = numpy.array(quaternion, dtype=numpy.float64, copy=True)
    n = numpy.dot(q, q)
    if n < _EPS:
        return numpy.identity(4)
```

```
    q *= math.sqrt(2.0 / n)
    q = numpy.outer(q, q)
    return numpy.array([
        [1.0-q[2, 2]-q[3, 3],     q[1, 2]-q[3, 0],     q[1, 3]+q[2, 0], 0.0],
        [    q[1, 2]+q[3, 0], 1.0-q[1, 1]-q[3, 3],     q[2, 3]-q[1, 0], 0.0],
        [    q[1, 3]-q[2, 0],     q[2, 3]+q[1, 0], 1.0-q[1, 1]-q[2, 2], 0.0],
        [                0.0,                 0.0,                 0.0, 1.0]])

def random_quaternion(rand=None):
    if rand is None:
        rand = numpy.random.rand(3)
    else:
        assert len(rand) == 3
    r1 = numpy.sqrt(1.0 - rand[0])
    r2 = numpy.sqrt(rand[0])
    pi2 = math.pi * 2.0
    t1 = pi2 * rand[1]
    t2 = pi2 * rand[2]
    return numpy.array([numpy.cos(t2)*r2, numpy.sin(t1)*r1,
                        numpy.cos(t1)*r1, numpy.sin(t2)*r2])
def random_rotation_matrix(rand=None):
    return quaternion_matrix(random_quaternion(rand))

alpha, beta, gamma = 0.123, -1.234, 2.345
origin, xaxis, yaxis, zaxis = (0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1)
R = random_rotation_matrix(numpy.random.rand(3))
print(R)

# rotation
ZZ = np.concatenate([Z, np.ones((Z.shape[0], 1))], axis=1)
out = np.zeros((8,4))
out = np.dot(R, ZZ.T).T[:, :-1]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

r = [-1,1]

X, Y = np.meshgrid(r, r)
# plot vertices
ax.scatter3D(out[:, 0], out[:, 1], out[:, 2])

# list of sides' polygons of figure

verts = [[out[0],out[1],out[2],out[3]],
 [out[4],out[5],out[6],out[7]],
 [out[0],out[1],out[5],out[4]],
 [out[2],out[3],out[7],out[6]],
 [out[1],out[2],out[6],out[5]],
 [out[4],out[7],out[3],out[0]]]

# plot sides
ax.add_collection3d(Poly3DCollection(verts,
 facecolors='cyan', linewidths=1, edgecolors='r', alpha=.25))

ax.set_xlabel('X')
```

```
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    plt.show()
```

Result



4. Shear

Python code

```
# shear
_EPS = numpy.finfo(float).eps * 4.0
from __future__ import division

import warnings
import math
import numpy
def unit_vector(data, axis=None, out=None):
    if out is None:
        data = numpy.array(data, dtype=numpy.float64, copy=True)
        if data.ndim == 1:
            data /= math.sqrt(numpy.dot(data, data))
            return data
    else:
        if out is not data:
            out[:] = numpy.array(data, copy=False)
        data = out
    length = numpy.atleast_1d(numpy.sum(data*data, axis))
    numpy.sqrt(length, length)
    if axis is not None:
        length = numpy.expand_dims(length, axis)
    data /= length
    if out is None:
        return data

def shear_matrix(angle, direction, point, normal):
    normal = unit_vector(normal[:3])
    direction = unit_vector(direction[:3])
```

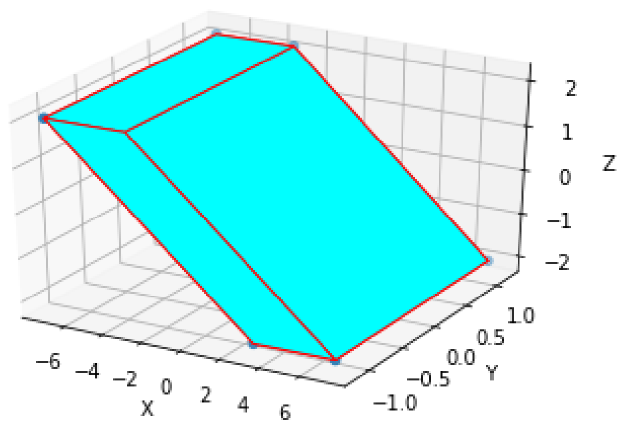
```
        if abs(numpy.dot(normal, direction)) > 1e-6:
            raise ValueError("direction and normal vectors are not orthogonal")
        angle = math.tan(angle)
        M = numpy.identity(4)
        M[:3, :3] += angle * numpy.outer(direction, normal)
        M[:3, 3] = -angle * numpy.dot(point[:3], normal) * direction
        return M

alpha, beta, gamma = 0.123, -1.234, 2.345
origin, xaxis, yaxis, zaxis = (0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1)
S_M = shear_matrix(beta, xaxis, origin, zaxis)
print(S_M)

# shear
ZZ = np.concatenate([Z, np.ones((Z.shape[0], 1))], axis=1)
out = np.zeros((8,4))
out = np.dot(S_M, ZZ.T).T[:, :-1]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

r = [-1,1]

X, Y = np.meshgrid(r, r)
# plot vertices
ax.scatter3D(out[:, 0], out[:, 1], out[:, 2])

# list of sides' polygons of figure

verts = [[out[0],out[1],out[2],out[3]],
 [out[4],out[5],out[6],out[7]],
 [out[0],out[1],out[5],out[4]],
 [out[2],out[3],out[7],out[6]],
 [out[1],out[2],out[6],out[5]],
 [out[4],out[7],out[3],out[0]]]

# plot sides
ax.add_collection3d(Poly3DCollection(verts,
 facecolors='cyan', linewidths=1, edgecolors='r', alpha=.25))

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

Result

5. Scale

Python code

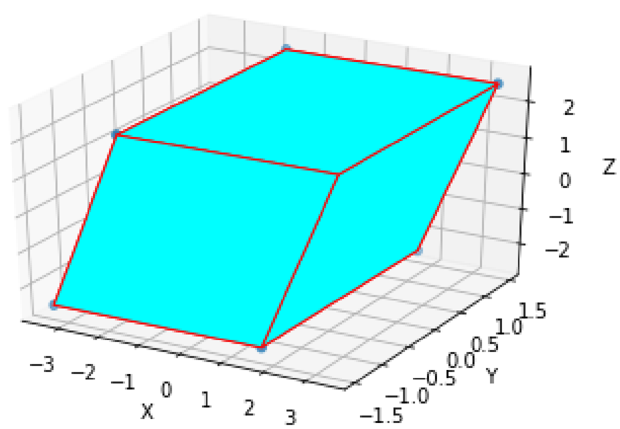
```
# scale
 _EPS = numpy.finfo(float).eps * 4.0
 from __future__ import division
```




```
import warnings
import math
import numpy
def scale_matrix(factor, origin=None, direction=None):
    if direction is None:
        M = numpy.array(((factor, 0.0,     0.0,     0.0),
                         (0.0,    factor,  0.0,     0.0),
                         (0.0,    0.0,     factor,  0.0),
                         (0.0,    0.0,     0.0,     1.0)), dtype=numpy.float64)
        if origin is not None:
            M[:3, 3] = origin[:3]
            M[:3, 3] *= 1.0 - factor
    else:
        direction = unit_vector(direction[:3])
        factor = 1.0 - factor
        M = numpy.identity(4)
        M[:3, :3] -= factor * numpy.outer(direction, direction)
        if origin is not None:
            M[:3, 3] = (factor * numpy.dot(origin[:3], direction)) * direction
    return M
origin, xaxis, yaxis, zaxis = (0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1)
S = scale_matrix(1.23, origin)
print(S)

# scale
ZZ = np.concatenate([Z, np.ones((Z.shape[0], 1))], axis=1)
out = np.zeros((8,4))
out = np.dot(S, ZZ.T).T[:, :-1]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

r = [-1,1]

X, Y = np.meshgrid(r, r)
# plot vertices
ax.scatter3D(out[:, 0], out[:, 1], out[:, 2])

# list of sides' polygons of figure
```

```
verts = [[out[0],out[1],out[2],out[3]],
 [out[4],out[5],out[6],out[7]],
 [out[0],out[1],out[5],out[4]],
 [out[2],out[3],out[7],out[6]],
 [out[1],out[2],out[6],out[5]],
 [out[4],out[7],out[3],out[0]]]

# plot sides
ax.add_collection3d(Poly3DCollection(verts,
 facecolors='cyan', linewidths=1, edgecolors='r', alpha=.25))

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

Result



6. Scale

Python code

```
    # reflection
def unit_vector(data, axis=None, out=None):
    if out is None:
        data = numpy.array(data, dtype=numpy.float64, copy=True)
        if data.ndim == 1:
            data /= math.sqrt(numpy.dot(data, data))
            return data
    else:
        if out is not data:
            out[:] = numpy.array(data, copy=False)
        data = out
    length = numpy.atleast_1d(numpy.sum(data*data, axis))
    numpy.sqrt(length, length)
    if axis is not None:
        length = numpy.expand_dims(length, axis)
```

```
    data /= length
    if out is None:
        return data

def reflection_matrix(point, normal):
    normal = unit_vector(normal[:3])
    M = numpy.identity(4)
    M[:3, :3] -= 2.0 * numpy.outer(normal, normal)
    M[:3, 3] = (2.0 * numpy.dot(point[:3], normal)) * normal
    return M
```