In [1]:
```
import sys
import numpy as np
import cv2
```

In [2]:
```
def Map2Da(K, R, T, Vi):
    T_transpose = np.transpose(np.atleast_2d(T)) #numpy needs to treat 1D as 2D
    V_transpose = np.transpose(np.atleast_2d(np.append(Vi,[1])))
    RandTappended = np.append(R, T_transpose, axis=1)
    P = K @ RandTappended @ V_transpose #@ is the matrix mult operator for nump
    P = np.asarray(P).flatten() #just to make it into a flat array

    w1 = P[2]
    v= [None]*2 #makes an empty array of size 2

    #map Vi = (X, Y, Z) to v = (x, y)
    v[0]= P[0] / w1  #v[0] is the x-value for the 2D point v

    #MISSING: compute v[1], the y-value for the 2D point v
    # v[1] = ?????????????
    v[1]= P[1] / w1  # v[1] is the y-value for the 2D point v
    return v


'''function for mapping image coordinates in mm to
row and column index of the image, with pixel size p mm and
image center at [r0,c0]

u : the 2D point in mm space
[r0, c0] : the image center
p : pixel size in mm

@return : the 2D point in pixel space
'''
def MapIndex(u, c0, r0, p):
    v = [None]*2
    v[0] = round(r0 - u[1] / p)
    #calculating v[1]
    v[1] = round(c0 + u[0] / p)
    return v
```

In [3]:
```
'''
Wrapper for drawing line cv2 draw line function
Necessary to flip the coordinates b/c of how Python indexes pixels on the scree

A : matrix to draw a line in
vertex1 : terminal point for the line
vertex2 : other terminal point for the line
thickness : thickness of the line(default = 3)
color : RGB tuple for the line to be drawn in (default = (255, 255, 255) ie whi

@return : the matrix with the line drawn in it

NOTE: order of vertex1 and vertex2 does not change the line drawn
'''
#MISSING : Replace the function below with another one that does not call
# cv2.line(.) but does all calculations within itself.
#def drawLine(A,vertex1, vertex2, color = (255, 255, 255), thickness=3):
```

```
#   v1 = list(reversed(vertex1))
#   v2 = list(reversed(vertex2))
#   return cv2.line(A, v1, v2,  color, thickness)  #replace this =>
def drawLine(A,vertex1, vertex2, color = (255, 255, 255), thickness=3):
     v1 = (int(vertex1[1]), int(vertex1[0]))
     v2 = (int(vertex2[1]), int(vertex2[0]))

    # Create a copy of the input image
     image_with_line = np.copy(A)

    # Calculate the slope and intercept of the line
     x1, y1 = v1
     x2, y2 = v2
     slope = (y2 - y1) / (x2 - x1) if x2 != x1 else np.inf

     if slope == np.inf:  # Vertical line
         for y in range(min(y1, y2), max(y1, y2) + 1):
             for x in range(x1 - thickness // 2, x1 + thickness // 2 + 1):
                 image_with_line[y, x] = color
     else:
         for x in range(min(x1, x2), max(x1, x2) + 1):
             y = int(y1 + slope * (x - x1))
             for dx in range(-thickness // 2, thickness // 2 + 1):
                 dy = int(dx * slope)
                 image_with_line[y + dy, x + dx] = color

     return image_with_line
```

In [ ]:

```
def main():
    length = 10 #length of an edge in mm
    #the 8 3D points of the cube in mm:
    V1 = np.array([0, 0, 0])
    V2 = np.array([0, length, 0])
    V3 = np.array([length, length, 0])
    V4 = np.array([length, 0, 0])
    V5 = np.array([length, 0, length])
    V6 = np.array([0, length, length])
    V7 = np.array([0, 0, length])
    V8 = np.array([length, length, length])

    '''
    Find the unit vector u81 (N0) corresponding to the axis of rotation which i
    From u81, compute the 3x3 matrix N in Eq. 2.32 used for computing the rotat
    '''

    '''
    MISSING: the axis of rotation is to be u81, the unit vector which is (V8-V1
    Calculate u81 here and use it to construct 3x3 matrix N used later to compu
    Matrix N is described in Eq. 2.32, matrix R is described in Eq. 2.34
    '''

    # Calculate the vector V8-V1
    V8_minus_V1 = V8 - V1

    # Calculate the magnitude (length) of the vector
    magnitude = np.linalg.norm(V8_minus_V1)

    # Calculate the unit vector u81 (N0)
    u81 = V8_minus_V1 / magnitude
```

```
# Now, u81 represents the unit vector along the axis of rotation (V8-V1)
# Next, compute the 3x3 matrix N using u81

#N matrix
N = np.array([[0, -u81[2], u81[1]],
              [u81[2], 0, -u81[0]],
              [-u81[1], u81[0], 0]])

#Initialized given values (do not change unless you're testing something):
T0 = np.array([-20, -25, 500])  # origin of object coordinate system in mm
f = 40  # focal length in mm
velocity = np.array([2, 9, 7])  # translational velocity
acc = np.array([0.0, -0.80, 0])  # acceleration
theta0 = 0 #initial angle of rotation is 0 (in degrees)
w0 = 20  # angular velocity in deg/sec
p = 0.01  # pixel size(mm)
Rows = 600  # image size
Cols = 600  # image size
r0 = np.round(Rows / 2) #x-value of center of image
c0 = np.round(Cols / 2) #y-value of center of image
time_range = np.arange(0.0, 24.2, 0.2)

#MISSING: Initialize the 3x3 intrinsic matrix K given focal length f
# K = ???????????????

# Initialize the intrinsic matrix K
K = np.array([
    [f, 0, 0],
    [0, f, 0],
    [0, 0, 1]])


# This section handles mapping the texture to one face:

# You are given a face of a cube in 3D space specified by its
# corners at 3D position vectors V1, V2, V3, V4.
# You are also given a square graylevel image tmap of size r x c
# This image is to be "painted" on the face of the cube:
# for each pixel at position (i,j) of tmap,
# the corresponding 3D coordinates
# X(i,j), Y(i,j), and Z(i,j), should be computed,
# and that 3D point is
# associated with the brightness given by tmap(i,j).
#
# MISSING:
# Find h, w: the height and width of the face
# Find the unit vectors u21 and u41 which coorespond to (V2-V1) and (V4-V1)
# hint: u21 = (V2-V1) / h ; u41 = (V4 - V1) / w

# h = ??????????????
# w = ??????????????
# u21 =??????????????
# u41 =??????????????

# We use u21 and u41 to iteratively discover each point of the face below:

h = np.linalg.norm(np.array(V2) - np.array(V1))
w = np.linalg.norm(np.array(V4) - np.array(V1))
```

```
# Calculate unit vectors u21 and u41
u21 = (np.array(V2) - np.array(V1)) / h
u41 = (np.array(V4) - np.array(V1)) / w


# Finding the 3D points of the face bounded by V1, V2, V3, V4
# and associating each point with a color from texture:


tmap = cv2.imread('einstein132.jpg')  # texture map image
if tmap is None:
    print("image file can not be found on path given. Exiting now")
    sys.exit(1)

r, c, colors = tmap.shape
# We keep three arrays of size (r, c) to store the (X, Y, Z) points coores
# to each pixel on the texture
X = np.zeros((r, c), dtype=np.float64)
Y = np.zeros((r, c), dtype=np.float64)
Z = np.zeros((r, c), dtype=np.float64)
for i in range(0, r):
    for j in range(0, c):
        p1 = V1 + (i) * u21 * (h / r) + (j) * u41 * (w / c)
        X[i, j] = p1[0]
        #MISSING: compute the Y and Z for 3D point pertaining to this pixel
        # Y[i,j] = ??
        # Z[i,j] = ??
        Y[i, j] = p1[1] # Replace with your Y calculation
        Z[i, j] = p1[2]  # Replace with your Z calculation




for t in time_range:  # Generate a sequence of images as a function of time
    theta = theta0 + w0 * t
    theta = np.radians(theta)
    T = T0 + velocity * t + 0.5 * acc * t * t
    # MISSING: compute rotation matrix R as shown in Eq. 2.34
    # Warning: be mindful of radians vs degrees
    # Note: for numpy data, @ operator can be used for dot product
    # R = ?????????????
    R = np.identity(3) + np.sin(theta) * N + (1 - np.cos(theta)) * np.dot(N
    # find the image position of vertices

    #MISSING: given 3D vertices V1 to V8, map to 2D using Map2da
    #then, map to pixel space using mapindex
    #save all 2D vertices as v1 to v8

    #example for V1 -> v1:
    #v = Map2Da(K, R, T, V1)
    #v1 = MapIndex(v, c0, r0, p)
    v = Map2Da(K, R, T, V1)
    v1 = MapIndex(v, c0, r0, p)
    # v2, v3, ..., v8 = ?????????????????????????????
    v = Map2Da(K, R, T, V2)
    v2 = MapIndex(v, c0, r0, p)

    v = Map2Da(K, R, T, V3)
    v3 = MapIndex(v, c0, r0, p)
```

```
        v = Map2Da(K, R, T, V4)
        v4 = MapIndex(v, c0, r0, p)

        v = Map2Da(K, R, T, V5)
        v5 = MapIndex(v, c0, r0, p)

        v = Map2Da(K, R, T, V6)
        v6 = MapIndex(v, c0, r0, p)

        v = Map2Da(K, R, T, V7)
        v7 = MapIndex(v, c0, r0, p)

        v = Map2Da(K, R, T, V8)
        v8 = MapIndex(v, c0, r0, p)


        # Draw edges of the cube



        #color = (0, 0, 255) #note, CV uses BGR by default, not RGB. This is Re
        color = (0, 0, 255) #note, CV uses BGR by default, not gray=(R+G+B)/3.
        thickness = 2
        A = np.zeros((Rows, Cols, 3), dtype=np.uint8) #array which stores the i
        edges = [(v1, v2), (v2, v3), (v3, v4), (v4, v1),
         (v5, v7), (v6, v7), (v6, v8), (v8, v5),
         (v1, v7), (v2, v6), (v3, v8), (v4, v5)]

        for edge in edges:
            v_start, v_end = edge
            A = drawLine(A, v_start, v_end, color, thickness)



        #MISSING: use drawLine to draw the edges to draw a naked cube
        #there are 12 edges to draw

        #example drawing the v1 to v2 line:
        #A = drawLine(A, v1, v2, color, thickness)
        # ????????????????????????????


        # Now we must add the texture to the face bounded by v1-4:
        for i in range(r):
            for j in range(c):
                p1 = [X[i, j], Y[i, j], Z[i, j]]

                #p1 now stores the world point on the cubic face which
                #corresponds to (i, j) on the texture

                #MISSING: convert this 3D point p1 to 2D (and map to pixel spac
                #p1_homo = np.dot(K, p1)  # Apply the camera projection matrix
                #p1_homo /= p1_homo[2]    # Normalize the homogeneous coordinat
                #set ir to the x-value of this point
                # set jr to the y-value of this point
                #ir, jr, _ = p1_homo.astype(int)  # Cast to int to index the ar
                # This gives us a point in A to color in for the texture
                #note: cast ir, jr to int so it can index array A
                #(ir, jr) = ????????????????????????
```

```
                (ir, jr) = MapIndex(Map2Da(K, R, T, p1), c0, r0, p)

                if ((ir >= 0) and (jr >= 0) and (ir < Rows) and (jr < Cols)):
                    tmapval = tmap[i, j, 2]
                    A[ir ,jr] = [ 0, 0, tmapval ] # gray here, but [0, 0, tmpva


        cv2.imshow("Display Window", A)
        cv2.waitKey(1)
        #cv2.waitKey(0)
        # ^^^ uncomment if you want to display frame by frame
        # and press return(or any other key) to display the next frame
        #by default just waits 1 ms and goes to next frame



if __name__ == "__main__":
    main()
```

In [ ]:

```

```

In [ ]:

```

```