World, Camera, Image Plane, and Image Pixel Coordinate Systems

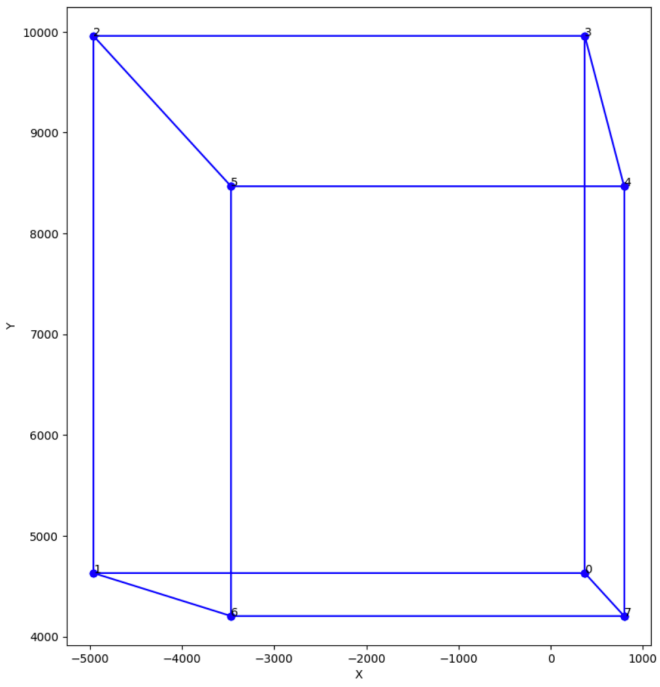
Modify the code in the Colab notebook for this module to produce the 2D image in pixel coordinates of the cube wireframe studied with the following parameters changed. The rest of the parameters remain unchanged. The rotation between the world and camera coordinates is just a 90-degree rotation around the z-axis. Generate a pdf file with screenshots of the code changes, and the resulting view. Explain what you observe, and upload it to Canvas.

```
#-----------------------------------------------------------------
# Generate 8 vertices of a 5 by 5 by 5 sized cube with one of the vertices at origin of the world coordinates (homogenous)
N = 8
p_tilde = np.array([[0, 0, 0, 1], [5, 0, 0, 1], [5, 5, 0, 1], [0, 5, 0, 1], [0, 5, 5, 1], [5, 5, 5, 1], [5, 0, 5, 1], [0, 0, 5, 1]])
p_tilde = np.expand_dims(p_tilde, 2) # the dimension for p_tilde will change from (N by 4) to (N by 4 by 1)
print('Dimensions of p_tilde:', p_tilde.shape)
#-----------------------------------------------------------------
# plot the cube, with respect to the world coordinates
fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(1,2,1, projection='3d')
plot_cube (ax, p_tilde)
ax.set_title('World coordinates')
#-----------------------------------------------------------------
# Setup the camera rotation and translation with respect to the world coordinates
# R = rotation_euler(30*np.pi/180, 0, 0)

# Modified to introduce a 90-degree rotation around the Z-axis
R_new = rotation_euler(0, 0, np.pi/2)


T = np.array([-2,2,20])
P = np.eye(4)
P[0:3,0:3] = R_new # R_new introduces a 90-degree rotation around the Z-axis
P[0:3,3] = T
print('R=\n', R_new, '\nT=\n', T, '\nP=\n', P) # R_new introduces a 90-degree rotation around the Z-axis
#-----------------------------------------------------------------
# Transform the 3D coordinates from world to camera coordinates
p_c_tilde = P @ p_tilde  # @ is matrix multiply in python

# P is 4 by 4 and p_tilde is 8 by 4 by 1, so this is not a standard matrix multiply!
# It is using a feature called broadcasting. It will effectively multiple each
# of the 4 by 1 homogeneous point representations by the 4 by 4 matrix to
# result in a 8 by 4 by 1 output.
```



In the experiment, the crucial change occurred in the code where we defined the rotation matrix R. Which was originally set to produce a 30-degree rotation around the X-axis, we modified it to represent a 90-degree rotation around the Z-axis.

The line R_new = rotation_euler(0, 0, np.pi/2) is where this change was implemented. This new rotation matrix was then incorporated into the existing transformation matrix P, which in turn influenced how the cube's 3D coordinates were mapped onto the 2D image plane. The rest of the code remained unchanged but operated on this newly transformed set of camera coordinates.

Tweaking just this one aspect, altered the entire 'viewpoint' of the camera, leading to a significantly different 2D projection of the cube. A vivid example of how minor changes in a system's parameters can lead to substantial differences in output. In practical applications like robotics project where the machine has to navigate a room it needs to know not just what it's looking at, but also from which angle it's viewing objects, understanding how these parameters affect the output is critical.

In summary, the cube's altered appearance on the 2D plane can be directly traced back to our modification of the rotation matrix in the code. It's an insightful exercise that shows how a camera's orientation impacts what it 'sees', highlighting the complex relationship between geometry and image formation.