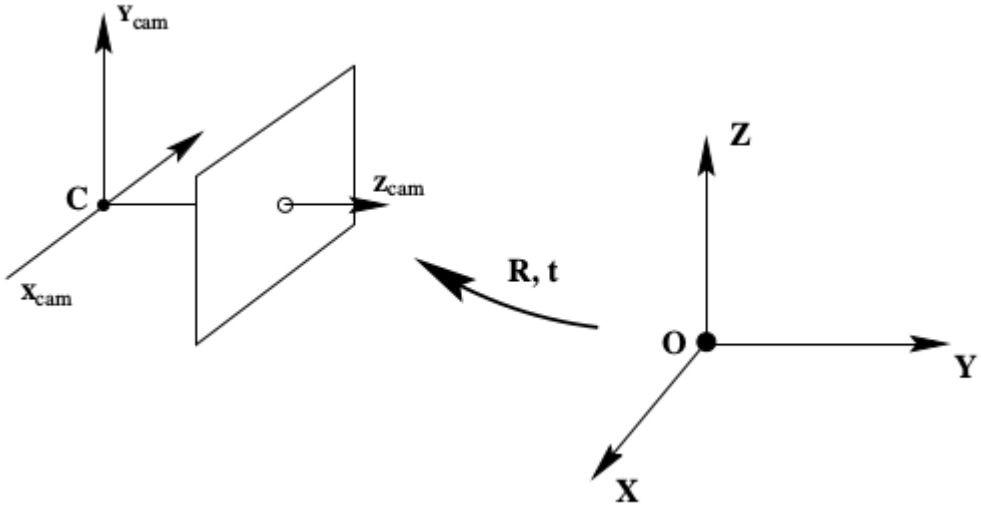
Xiangyi Cheng (xxc283)

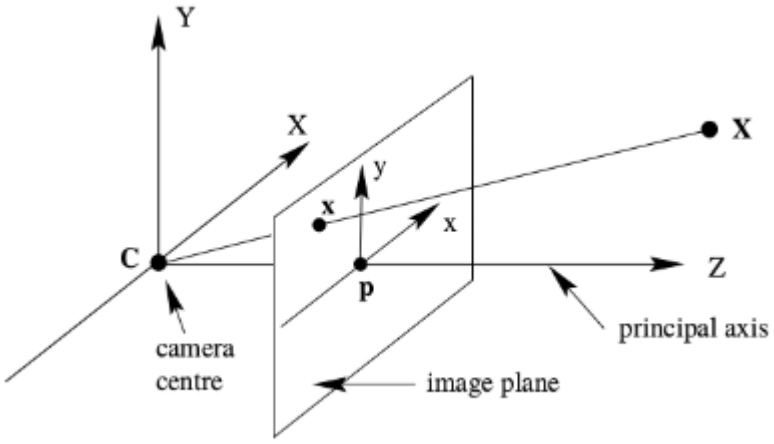
Concept and Background

The aim of this exercise is to build a 3D virtual world with a set of points and set two cameras in different locations. The cameras should be defined by position, target, up, focal length, film width, film height. The numbers of horizontal and vertical pixel of the images taken by the cameras are specified as width and height.

The camera is a mapping between the 3D world and a 2D image. Projection is how the camera convert the 3D object into 2D image. There are several camera models, such as finite cameras, the projective camera, cameras at infinity, etc.. The most specialized and simplest model among them is called the basic pinhole camera. Before deeping into the projection concept, we have to understand another two terms called the world coordinate frame and the camera coordinate frame. The world coordinate frame is the frame we define the location of the points and the cameras while the camera coordinate frame whose center is the camera position which we will use to compute the projection plane corner points as shown below. The camera coordinate frame could be calculated based on the camera parameters.



After obtaining the coordiantes, the image below illustrates the projection in terms of a ponhole camera which belongs to finite camera category. C is the camera center and the distance between C and P is the focal length. X represents the object in 3D world while x is projected location in the image. Based on the known focal width, focal lengt and the distance from the target position to the real position of the objects, we are able to calculate the projective plane by applying the similar triangle rule.



Below is an example shows how to build a set of points in a 3D virtual world and the projection by two cameras set in different position and angles.

Implementation and Exploration

The code is developed in python. Import several useful libraries which will be used later at the beginning.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
```

A set of points are defined by x,y,z location in various colors. Plot these defined points using mpl_toolkits.mplot3d library in 3D world. The plot is returned to be used in the latter application.

In [2]:
```
def define_3Dpoints():

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

    # define the location of the positions
    x=[-0.5, 0, 0.5]
    y=[-0.5, 0, 0.5]
    z=[-0.5, 0, 0.5]
    X, Z, Y = np.meshgrid(x, y, z)
    dimension=X.shape[0]*Y.shape[0]*Z.shape[0]

    # define the colors of the points
    colors=np.zeros((dimension,3))
    for i in range (0, dimension):
        colors[i,:]=np.random.rand(3)

    # 3D plot
    ax.scatter(X,Y,Z,marker='.',c=colors)

    ax.set_xlabel('X label')
    ax.set_ylabel('y label')
    ax.set_zlabel('z label')
    ax.set_xlim([-3,4])
    ax.set_ylim([-3,3])
    ax.set_zlim([-2,3])

    return ax
```

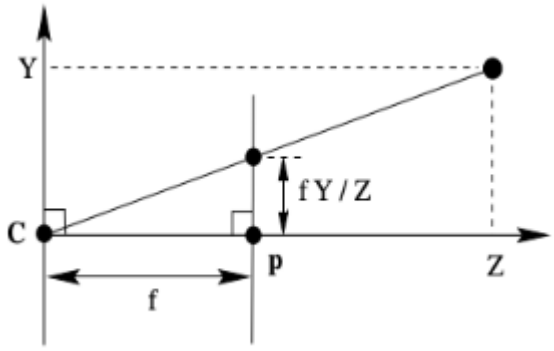
The position of each camera is specified by six parameters, x,y,z and three angles. Becides these, we also have to know the focal length, focal width, target position, film width and film height to compute the projective plane. To recover the 2D image, the number of horizontal and vertical pixels are also critical. There is also a vector called up which is to compute the coordinates of the camera.

In [5]:
```
def plot_camera(ax,camera):
    cam1_pos,cam2_pos,target,up,focal_length,film_height,
    film_width,width,height= camera_specification()
    if camera==1:
        cam_pos=cam1_pos
        color='r'
    elif camera==2:def camera_specification():
    # position parameters
    r= 5
    alpha1= np.pi/6
    beta= np.pi/6

    # camera 1 parameters
    cam1_pos= [r*np.cos(beta)*np.cos(alpha1),
               r*np.cos(beta)*np.sin(alpha1), r*np.sin(beta)]
    target= np.array([0,0,0])
    up= np.array([0,0,1])
    focal_length= 0.06
    film_width= 0.035
    film_height= 0.035
    width= 256
    height= 256

    # camera 2 parameters, others are the same as the camera 1
    alpha2= np.pi/3
    cam2_pos= [r*np.cos(beta)*np.cos(alpha2),
               r*np.cos(beta)*np.sin(alpha2), r*np.sin(beta)]
    return cam1_pos,cam2_pos,target,up,focal_length,
           film_height,film_width,width,height
```

The camera coordiante frame is defined below. Basically, the z is the vector pointing from the camera to the target. Use cross product to calculate the y axis which is perpendicular to the z axis of the camera coordinate and the z axis of the world coordinate which is pointing up. With y and z axis, we are able to get the x axis which are perpendicular to y and z axis by applying the cross product to them. Normalization is applied to to simply the vector by only remaining the direction of the vector.

In [8]:
```
def camera_view_unit(target,cam_pos,up):
    zcam= target-cam_pos
    xcam= np.cross(zcam,up)
    ycam= np.cross(zcam,xcam)

    # normalization
    if np.linalg.norm(xcam)!=0:def plot_camera(ax,camera):
    cam1_pos,cam2_pos,target,up,focal_length,film_height,
    film_width,width,height= camera_specification()
    if camera==1:
        cam_pos=cam1_pos
        color='r'
    elif camera==2:
        xcam= xcam/np.linalg.norm(xcam)

    if np.linalg.norm(ycam)!=0:
        ycam= ycam/np.linalg.norm(ycam)

    if np.linalg.norm(zcam)!=0:
        zcam= zcam/np.linalg.norm(zcam)

    return xcam,ycam,zcam
```

With defined camera, we are able to calculate its projective plane by obtaining the four corners coordinates. The approach of calculating the points is to use similar triangle rule as shown below.




```
x= 0.5* film width* d/ focal length

y= 0.5* film height*d/ focal length

p1 = cam_pos + x* xcam + y* ycam + d* zcam

p2 = cam_pos + x* xcam - y* ycam + d* zcam

p3 = cam_pos - x* xcam - y* ycam + d* zcam

p4 = cam_pos - x* xcam + y* ycam + d* zcam
```

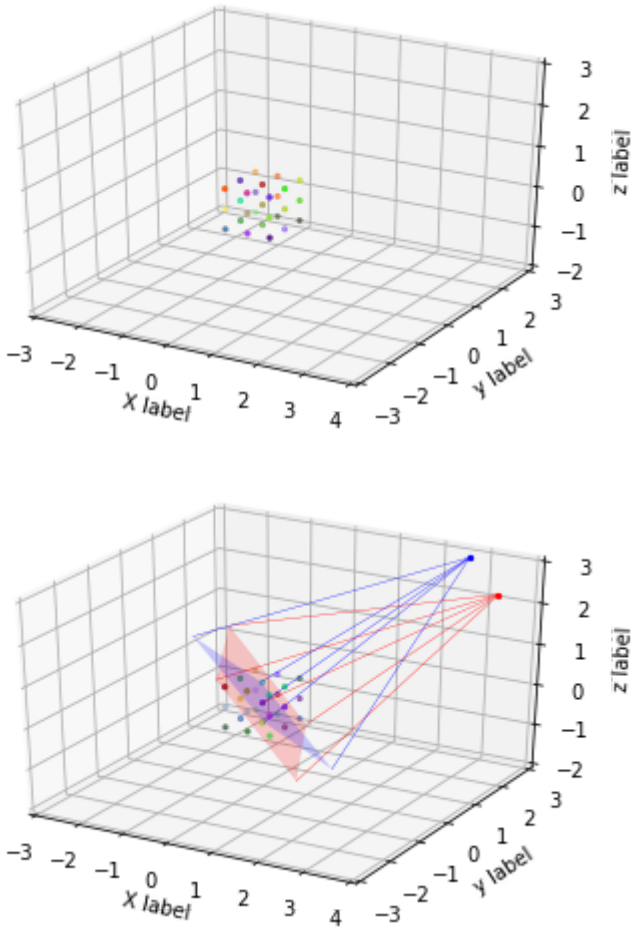
p1, p2, p3, p4 are the coordinates of the four corners for the projective plane where x and y are the offset from the camera position. To calculate the x and y, similar triangular rule was applied where d is the distance between the camera to the target, in this case, point C to Z. Focal length

is shown as f in the image above. The distance beween point Y and C is what we are trying to compute as x and y.

In [19]:
```
def plot_camera(ax,camera):
    cam1_pos,cam2_pos,target,up,focal_length,film_height,
    film_width,width,height= camera_specification()
    if camera==1:
        cam_pos=cam1_pos
        color='r'
    elif camera==2:
        cam_pos=cam2_pos
        color='b'
    else:
        print 'the camera is not defined.'

    ax.scatter(cam_pos[0],cam_pos[1],cam_pos[2],marker='.',c=color)

    xcam,ycam,zcam= camera_view_unit(target,cam_pos,up)

    # four corners on the plane through focal points
    d= np.linalg.norm(target-cam_pos)

    x= 0.5* film_width*d/focal_length
    y= 0.5* film_height*d/focal_length

    p1 = cam_pos + x* xcam + y* ycam + d* zcam
    p2 = cam_pos + x* xcam - y* ycam + d* zcam
    p3 = cam_pos - x* xcam - y* ycam + d* zcam
    p4 = cam_pos - x* xcam + y* ycam + d* zcam

    # connect these 4 points
    plotx=[p1[0],p2[0],p3[0],p4[0],p1[0]]
    ploty=[p1[1],p2[1],p3[1],p4[1],p1[1]]
    plotz=[p1[2],p2[2],p3[2],p4[2],p1[2]]

    # plot the projective plane
    verts=[zip(plotx,ploty,plotz)]
    face=Poly3DCollection(verts,alpha=0.2,facecolors=color,
                          linewidth=0.5)

    ax.add_collection3d(face)
    alpha=0.2
    if camera==1:
        face.set_facecolor((1,0,0,alpha))
    else:
        face.set_facecolor((0,0,1,alpha))

    # connect plane corner points to the camera
    ax.plot([cam_pos[0],target[0]],[cam_pos[1],target[1]],
            [cam_pos[2],target[2]],c=color,linewidth=0.3)

    ax.plot([cam_pos[0],p1[0]],[cam_pos[1],p1[1]],
            [cam_pos[2],p1[2]],c=color,linewidth=0.3)
    ax.plot([cam_pos[0],p2[0]],[cam_pos[1],p2[1]],
            [cam_pos[2],p2[2]],c=color,linewidth=0.3)
    ax.plot([cam_pos[0],p3[0]],[cam_pos[1],p3[1]],
            [cam_pos[2],p3[2]],c=color,linewidth=0.3)
    ax.plot([cam_pos[0],p4[0]],[cam_pos[1],p4[1]],
            [cam_pos[2],p4[2]],c=color,linewidth=0.3)
```

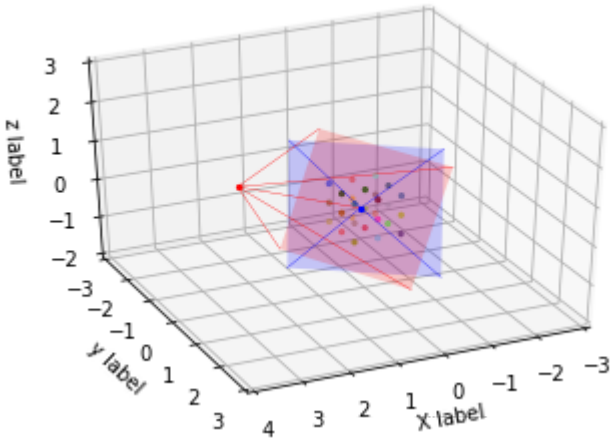
Plot the virtual world and the two set cameras with projective plane. Camera 1 is in red while camera 2 is in blue color.

In [15]:
```
ax=define_3Dpoints()
plt.show()
ax=define_3Dpoints()
plot_camera(ax,1)
plot_camera(ax,2)
plt.show()
```

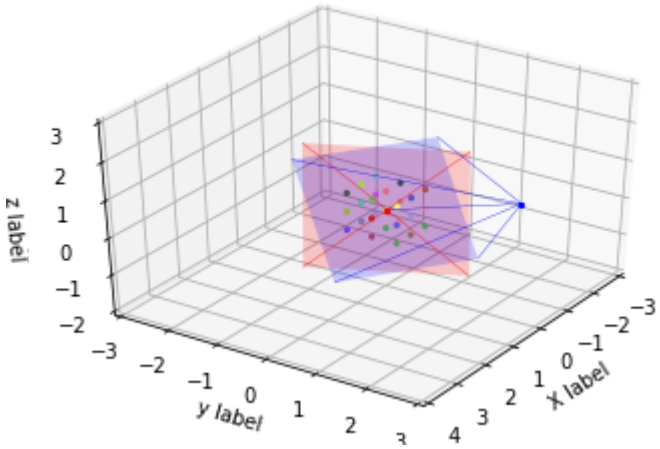


View the projection in different angles.

In [11]:
```
ax=define_3Dpoints()
plot_camera(ax,1)
plot_camera(ax,2)
ax.view_init(elev=35,azim=65)
plt.show()
```



In [12]:
```
ax=define_3Dpoints()
plot_camera(ax,1)
plot_camera(ax,2)
ax.view_init(elev=37.5,azim=34)
plt.show()
```



Conclusion and Discussion

To conclude, this exercise is to build a 3D virtual world with a set of points. Then two cameras in different locations were added into the world. The projective plane of each camera was calculated based on the parameters of the camera. Specically, the plane was set by four corner points computed by camera coordinate frame and similar triangle rule.

The exploration of this exercise is that instead of using MATLAB, I did the work all in python. Since there are several convenient build-in function which are relevant to camera setting in MATLAB, I cannot use it directly in python. Therefore, I had to figure out some ways to display the work in python reasonably.

Reference

Richard Hartley, Andrew Zisserman, "Multiple View Geometry in Computer View", second edition, 2004