NC STATE UNIVERSITY

Digital Imaging System Project 1

Implementation of Single view metrology

Aditya Rasam
200153631

Table of Contents

I. Introduction:

This project aims at building a 3D model of an object using a 2D image of the object. The procedure and the algorithm follow the Single View metrology paper by Criminisi.

Figure 1: 3-point perspective Image of a BOX

II. Image Acquisition and Annotation:

The input image is required to be 3-point perspective and accordingly the image was taken following the 3-point perspective image guide as mentioned in the project description document. This image was annotated to indicate the lines along X,Y and Z that acts as an evidence of being a 3D-perspective image. The annotation was done using python script.

Figure 2: Annotated Image

III. Computing the Vanishing points:

Parallel lines in real world intersect at the image plane. The point at which these lines meet in each X,Y and Z direction respectively can be estimated by extending the lines manually in these directions. Multiple horizontal vanishing points constitutes horizon or vanishing line. The vanishing line that results from lines in Z direction is in direction normal to the ground or reference plane. In this project, we computed the vanishing points using the method suggested by Prof. Robert. T Collins in project description document.

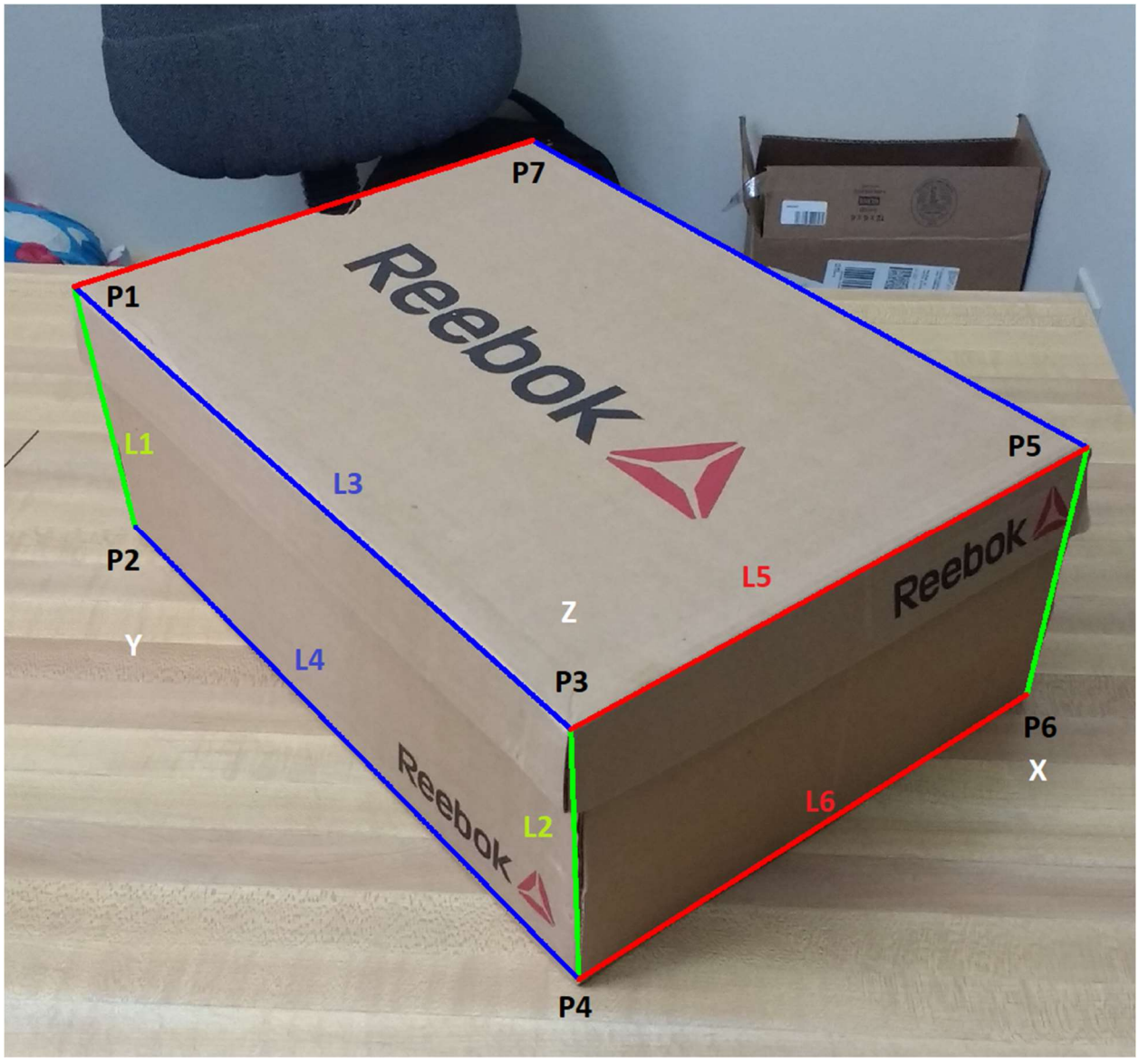


Figure 3: Vanishing Point Estimation

Procedure: In this project, the endpoints for the lines on the box along X, Y and Z direction are to be given as arguments to the function that creates the 3D model. So, in all there are 7 points as shown in the image with X,Y and Z orientation. These points are converted to homogeneous co-ordinate within the function.

1. Each line is estimated by taking the cross product of its end-point.
    a. E.g. line 'L3' is estimated by taking cross product of vectors formed by its end-points P1 and P3
2. Now the vanishing point is estimated from the lines parallel in the required direction. This is done by taking the cross product of the two lines.
    a. E.g. vanishing point in x direction is calculated by taking cross product of line L5 and L6.

Thus vanishing points in X,Y and Z direction are computed using two lines in each direction. Following this the projection and homograph matrix are computed

IV. Computing the projection matrix and homograph matrix

As computed by the code, the reference distance from the selected reference point is given in the following image. Based on the computed vanishing points, the projection matrix can be computed as follows.

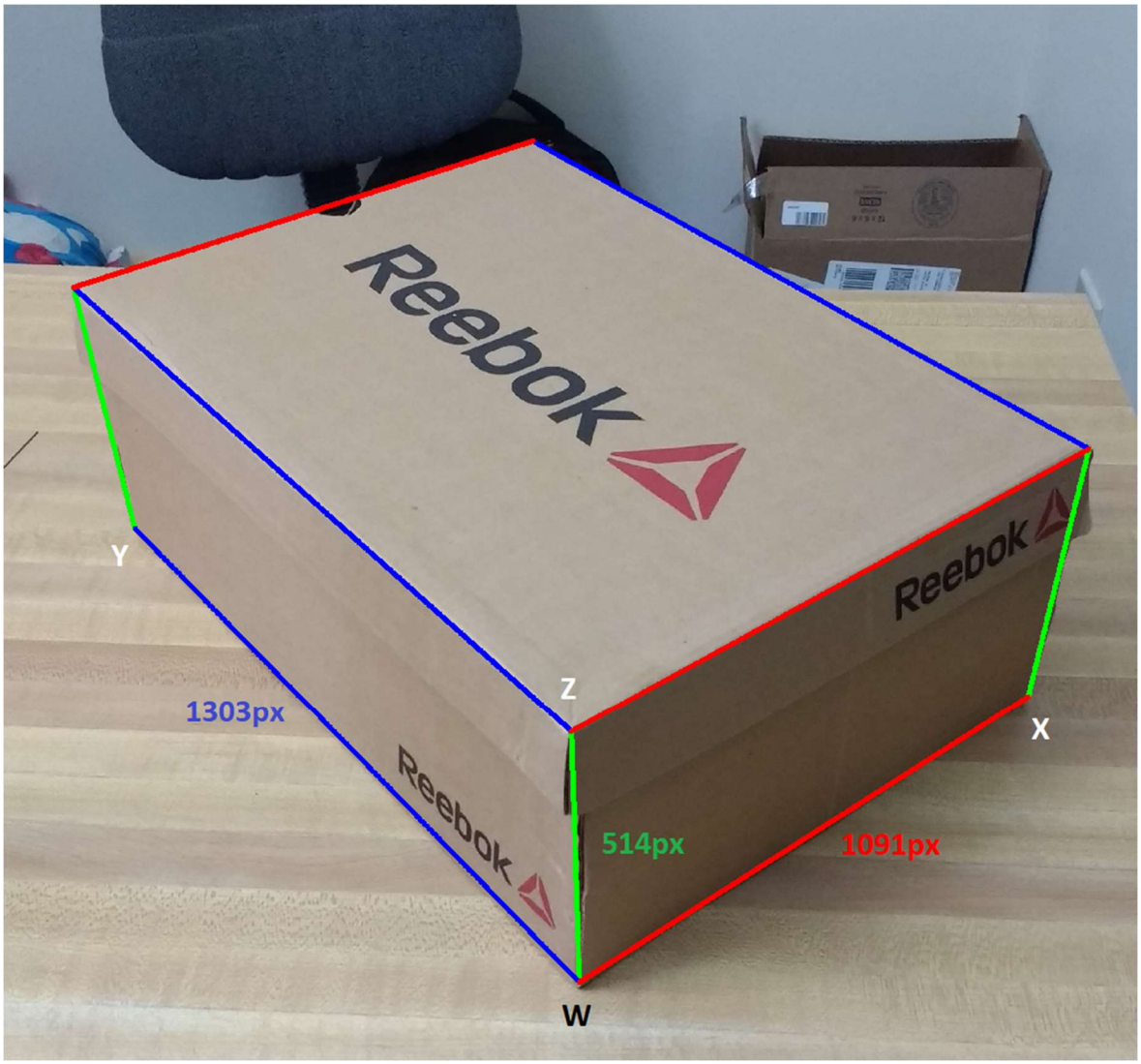


Figure 4: Reference distance

Computation of Projection Matrix:

The Projection matrix is given as

$$P = [p1\ p2\ p3\ p4]$$

Where,

Each column = Product of the vanishing point in that direction and the scaling factor α

$$p1 = \alpha_x * V_x \quad p2 = \alpha_y * V_y \quad p3 = \alpha_z * V_z \text{ and } p4 = \text{world origin}$$

Thus, the first three columns of projection matrix are known only to a certain scale and the fourth column is the world origin in the image co-ordinate. Hence, to find the projection matrix we need to know the vanishing points, scaling factors and the world origin in image co-ordinates.

Calculation of scaling factor:

To calculate the scaling factor I fixed the world co-ordinate and reference points in X,Y and Z directions. Also, their distances towards the origin was calculated. Finally, the scaling factor was calculated by solving the following equation in python

$$\alpha_x * \text{ref\_distanceX} * [Vx - X\_ref] = [X\_ref - Wo]$$

where

ref_distanceX is a scalar distance of reference point in X direction from world origin.

X_ref is the vector location of reference point in X direction

Vx is the vanishing point in X direction

Wo is the world origin in image co-ordinate

Similarly scaling factors α_y and α_z are calculated.

$$\alpha_y * \text{ref\_distanceY} * [Vy - Y\_ref] = [Y\_ref - Wo]$$

$$\alpha_z * \text{ref\_distanceZ} * [Vz - Z\_ref] = [Z\_ref - Wo]$$

This ends the calculation of projection matrix.

Computation of Homograph Matrix:

The homograph matrix is calculated from the projection matrix.

Hxy is the 1st,2nd and 4th column of Projection matrix.

Hyz is the 2nd,3rd and 4th column of Projection matrix.

Hxz is the 1st,3rd and 4th column of Projection matrix.

V. Computing the texture maps for XY, YZ and XZ planes respectively

Using H_{xy}, H_{yz} and H_{xz} matrices, perspective transformation of the image was carried out to result in texture maps for XY, YZ and XZ as shown in the figure.

Texture maps

Figure 7: XY texture map

Figure 6: YZ texture map

Figure 5:XZ texture map

Cropped Images

Figure 9:XY Cropped

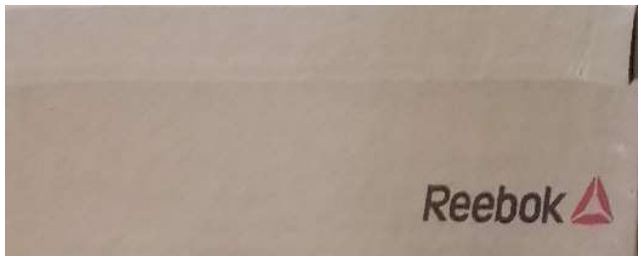


Figure 8:YZ Cropped

Figure 10: XZ cropped

VII. Reconstruction of 3D model:

To reconstruct a 3D model, the obtained texture maps and the co-ordinates of the points in the respective plane was passed to a VRML script. This file was imported within a 3D reconstruction software like 'View 3D scene' to result in a 3D model as shown below.

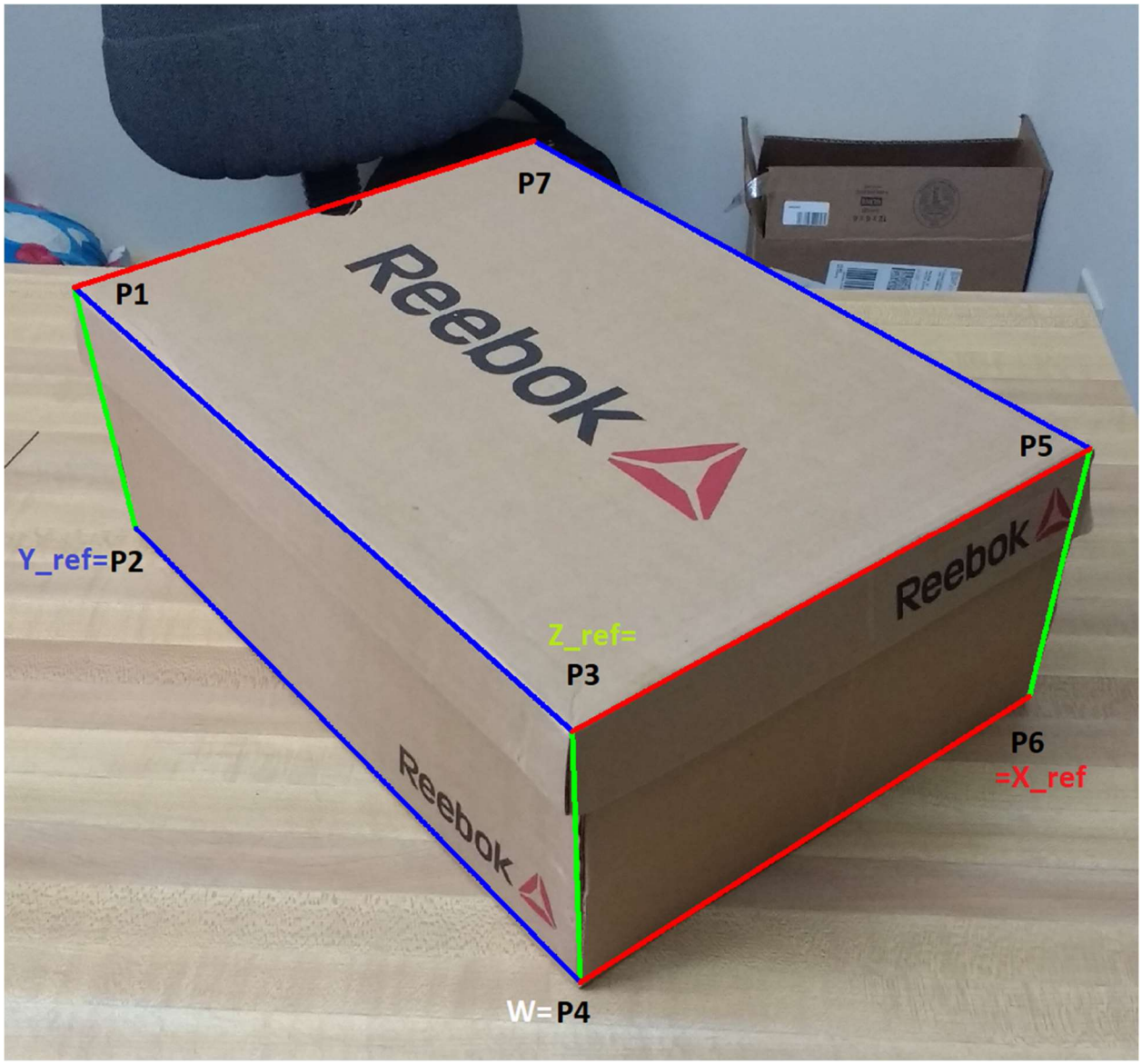
Figure 12: 3D model

VIII. Python Script for Single View Geometry.

Function Description:

SingleViewGeom(P1, P2, P3, P4, P5, P6, P7, X_ref, Y_ref, Z_ref, W): The function creates the texture maps using a 3-point perspective image.

The location of the points (in the argument) on the box is given in the image.



The arguments for the coded function needs to be passed in following manner and data type.

Input	Data Type	Description	Example
Image	String	Name of Image file with format	'testbox0.jpg'
P1	Tuple	Pixel Co-ordinate 1	(143,580) px
P2	Tuple	Pixel Co-ordinate 2	(269,1074) px
P3	Tuple	Pixel Co-ordinate 3	(1165,1491) px
P4	Tuple	Pixel Co-ordinate 4	(1181,2005) px
P5	Tuple	Pixel Co-ordinate 5	(2227,911) px
P6	Tuple	Pixel Co-ordinate 6	(2102,1420) px
P7	Tuple	Pixel Co-ordinate 7	(1086,280) px
X_ref	Tuple	Reference co-ordinate in X direction	P6
Y_ref	Tuple	Reference co-ordinate in Y direction	P2
Z_ref	Tuple	Reference co-ordinate in Z direction	P3
W	Tuple	World origin	P4

Instructions to run the script.

1. The user is required to manually enter the arguments mentioned above.
2. Then the user needs to run the script once.
3. Finally, the user should call the function to display the output images for XY,YZ and XZ texture maps.

IX. Code:

Python Script

```
def SingleViewGeom(P1, P2, P3, P4, P5, P6, P7, X_ref, Y_ref, Z_ref, W):

    import cv2

    import numpy as np

    import numpy.linalg as lin

    from scipy.spatial import distance


    #from matplotlib import pyplot



    #p =  'C:\Users\adity\Documents\NCSU\Proj1\testbox.jpeg'#'Desktop\\testbox.jpeg'
```

```
img = cv2.imread('textbox0.jpg')
S = img.shape
print('s', S)

P1= np.array(P1)
P2= np.array(P2)
P3= np.array(P3)
P4= np.array(P4)
P5= np.array(P5)
P6= np.array(P6)
P7 = np.array(P7)

w=1

#Line 1 P1 & P2
temp = [P1[0],P1[1],w]
e1_1 = np.array(temp)
temp = [P2[0],P2[1],w]
e2_1 = np.array(temp)
temp  = np.cross(e1_1, e2_1)
l1 = np.array(temp)


#Line 2 P1 & P2
temp = [P3[0],P3[1],w]
e1_2 = np.array(temp )
temp = [P4[0],P4[1],w]
e2_2 = np.array(temp)
```

```
temp  = np.cross(e1_2, e2_2)
l2 = np.array(temp)

V1 =  np.array(np.cross(l1,l2))

#Line 3 P1 & P3
temp = [P1[0],P1[1],w]
e1_3 = np.array(temp )
temp = [P3[0],P3[1],w]
e2_3 = np.array(temp)

temp  = np.cross(e1_3, e2_3)
l3 = np.array(temp)


#Line 4 P2 & P4
temp = [P2[0],P2[1],w]
e1_4 = np.array(temp )
temp = [P4[0],P4[1],w]
e2_4 = np.array(temp)


temp  = np.cross(e1_4, e2_4)
l4 = np.array(temp)

V2 =  np.array(np.cross(l3,l4))


 #Line 5 P3 & P5
```

```
temp = [P3[0],P3[1],w]
e1_5 = np.array(temp )
temp = [P5[0],P5[1],w]
e2_5 = np.array(temp)

temp  = np.cross(e1_5, e2_5)
l5 = np.array(temp)

#Line 6 P4 & P6
temp = [P4[0],P4[1],w]
e1_6 = np.array(temp )
temp = [P6[0],P6[1],w]
e2_6 = np.array(temp)

temp  = np.cross(e1_6, e2_6)
l6 = np.array(temp)
V3 = np.array(np.cross(l5,l6))

tV1 = V1[:]/V1[2]

tV2 = V2[:]/V2[2]
tV3 = V3[:]/V3[2]

#print('V1', V1)
#print('V2', V2)
#print('V3', V3)

#Vanishing points
Vy = np.array([tV2]).T
```

```
Vx = np.array([tV3]).T
Vz = np.array([tV1]).T

# print('tV1', Vx)
# print('tV2', Vy)
# print('tV3', Vz)

#World Origin in image-cords
WO = np.array(W)
WO = np.append([WO],[1])
WO = np.array([WO]).T
#Reference axis-cords in im-cords
ref_x = np.array(X_ref)#[ 197 , 317 , 1 ]
ref_x = np.append([ref_x],[1])
x_ref =  np.array([ref_x]).T

ref_y = np.array(Y_ref)#[ 442 , 317 , 1 ]
ref_y = np.append([ref_y],[1])
y_ref =  np.array([ref_y]).T


ref_z = np.array(Z_ref)#[ 319 , 227 , 1 ] #556  [ 778 , 400 , 1 ]
ref_z = np.append([ref_z],[1])
z_ref =  np.array([ref_z]).T

ref_x_dis = distance.euclidean(x_ref,WO)
ref_y_dis = distance.euclidean(y_ref,WO)
ref_z_dis = distance.euclidean(z_ref,WO)
```

```
print('X',ref_x_dis)

print('Y',ref_y_dis)

print('Z',ref_z_dis)


#%% Scaling factors of the projection matrix

temp = np.array((x_ref - WO))

tempx,resid,rank,s = np.linalg.lstsq((Vx-x_ref),temp)

a_x = (tempx )  / ref_x_dis  #%( A \ B ==> left division )


temp = np.array((y_ref - WO))

tempy,resid,rank,s = np.linalg.lstsq((Vy-y_ref),temp)

a_y = (tempy )  / ref_y_dis  #%( A \ B ==> left division )


temp = np.array((z_ref - WO))

tempz,resid,rank,s = np.linalg.lstsq((Vz-z_ref),temp)

a_z = (tempz )  / ref_z_dis  #%( A \ B ==> left division )


p1 = Vx*a_x

p2 = Vy*a_y

p3 = Vz*a_z

p4 = np.array(WO)


P = np.concatenate((p1, p2, p3, p4), axis =1)


Hxy = np.concatenate((p1, p2, p4), axis =1)

Hyz = np.concatenate((p2, p3, p4), axis =1)

Hzx = np.concatenate((p1, p3, p4), axis =1)


warp = cv2.warpPerspective(img, Hxy , (S[0], S[1]), flags = cv2.WARP_INVERSE_MAP)
```

```
warp1 = cv2.warpPerspective(img, Hyz , (S[0], S[1]), flags = cv2.WARP_INVERSE_MAP)
warp2 = cv2.warpPerspective(img, Hzx , (S[0], S[1]), flags = cv2.WARP_INVERSE_MAP)

img_ann=cv2.line(img ,(P1[0],P1[1]),(P2[0],P2[1]), (0,255,0), 10)
img_ann=cv2.line(img_ann ,(P3[0],P3[1]),(P4[0],P4[1]), (0,255,0), 10)
img_ann=cv2.line(img_ann ,(P5[0],P5[1]),(P6[0],P6[1]), (0,255,0), 10)

img_ann=cv2.line(img_ann ,(P1[0],P1[1]),(P3[0],P3[1]), (255,0,0), 10)
img_ann=cv2.line(img_ann ,(P2[0],P2[1]),(P4[0],P4[1]), (255,0,0), 10)
img_ann=cv2.line(img_ann ,(P7[0],P7[1]),(P5[0],P5[1]), (255,0,0), 10)

img_ann=cv2.line(img_ann ,(P6[0],P6[1]),(P4[0],P4[1]), (0,0,255), 10)
img_ann=cv2.line(img_ann ,(P3[0],P3[1]),(P5[0],P5[1]), (0,0,255), 10)
img_ann=cv2.line(img_ann ,(P1[0],P1[1]),(P7[0],P7[1]), (0,0,255), 10)

cv2.imshow('Ann', img_ann)
cv2.imshow('XY', warp)
cv2.imshow('YZ', warp1)
cv2.imshow('ZX', warp2)
cv2.waitKey(0)
cv2.destroyAllWindows()

return
```

VRML script

```
#VRML V2.0 utf8
Collision {
```

```
    collide FALSE
    children [
 Shape {
      appearance Appearance {
            texture ImageTexture {
                  url "XYC1.png"
            }
      }
      geometry IndexedFaceSet {
            coord Coordinate {
                  point [
                        0.000000 0.000000 514.000000,
                        1091.000000 0.000000 514.000000,
                        1091.000000 1303.000000 514.000000,
                        0.000000 1303.000000 514.000000,
                  ]
            }
            coordIndex [
                  0, 1, 2, 3, -1,
            ]
            texCoord TextureCoordinate {
                  point [
                        -0.000000 0.000000,
                        1.000000 0.000000,
                        1.000000 1.000000,
                        -0.000000 1.000000,
                  ]
            }
            texCoordIndex [
                  0, 1, 2, 3, -1,
            ]
            solid FALSE
      }
}
 Shape {
      appearance Appearance {
            texture ImageTexture {
                  url "ZXC1.png"
            }
      }
      geometry IndexedFaceSet {
            coord Coordinate {
                  point [
                        0.000000 0.000000 0.000000,
                        1091.000000 0.000000 0.000000,
                        1091.000000 0.000000 514.000000,
                        0.000000 0.000000 514.000000,
                  ]
            }
            coordIndex [
                  0, 1, 2, 3, -1,
            ]
```

```
            texCoord TextureCoordinate {
                  point [
                        -0.000000 0.000000,
                        1.000000 0.000000,
                        1.000000 1.000000,
                        -0.000000 1.000000,
                  ]
            }
            texCoordIndex [
                  0, 1, 2, 3, -1,
            ]
            solid FALSE
      }
}
 Shape {
      appearance Appearance {
            texture ImageTexture {
                  url "YZC1.png"
            }
      }
      geometry IndexedFaceSet {
            coord Coordinate {
                  point [
                        0.000000 1303.000000 0.000000,
                        0.000000 0.000000 0.000000,
                        0.000000 0.000000 514.000000,
                        0.000000 1303.000000 514.000000,
                  ]
            }
            coordIndex [
                  0, 1, 2, 3, -1,
            ]
            texCoord TextureCoordinate {
                  point [
                        -0.000000 0.000000,
                        1.000000 0.000000,
                        1.000000 1.000000,
                        -0.000000 1.000000,
                  ]
            }
            texCoordIndex [
                  0, 1, 2, 3, -1,
            ]
            solid FALSE
      }
}
     ]
}
```