Akshay Vijay Khanna
#unityID : akhanna3

ECE 558 - Digital Imaging System

Project - 1: Describes how aspects of the affine 3D geometry of a scene can be computed from a single perspective image with some prior knowledge.

- **Image acquisition:**
  - Step 1: Take an image example with a single perspective and three planes visible. The three planes are XY, XZ, YZ.

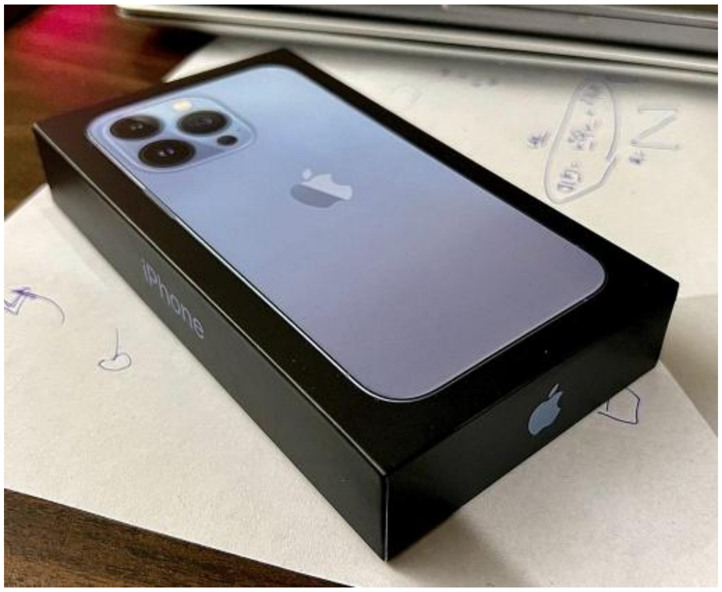


Figure 1

  - Step 2: Define the coordinates of the corner of the selected image and draw the contours. The points and contours are as described:

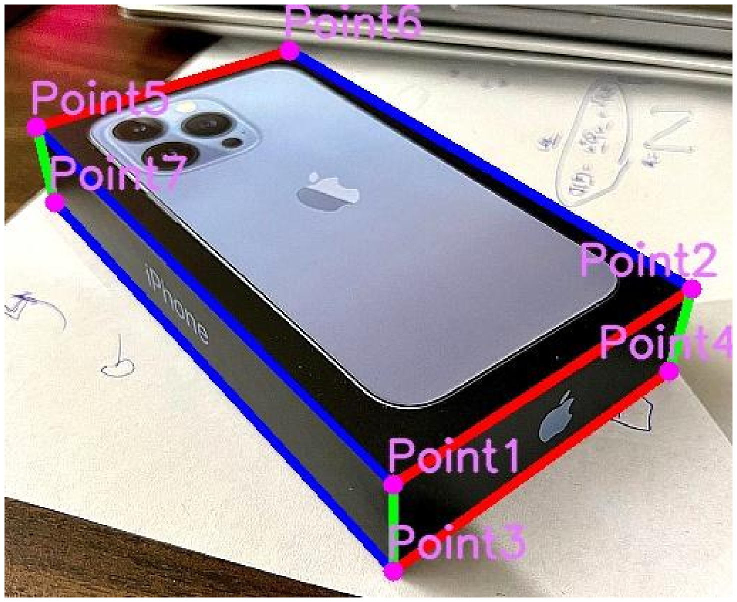


Figure 2

Akshay Vijay Khanna
#unityID : akhanna3

```
import cv2
import numpy as np
from scipy import linalg
# loading the image sample and resizing as per requirement
testImage = cv2.imread(r'D:\\NCSU\\NCSU Courses\\DIS - 558\\DIS
Project1\\ReferenceImage\\Sample_1.jpg')
baseImage = cv2.resize(testImage,None, fx=0.5,fy=0.5)
Rows, Columns, Temporary = baseImage.shape
# image sharpening filter
sharpenedFilter = np.array([[-0.1,-0.1,-0.1],
                           [-0.1,1.9,-0.1],
                           [-0.1,-0.1,-0.1]])
sharpenedImage = cv2.filter2D(baseImage, -1, sharpenedFilter)
# specifying coordinates
x1e1 = y1e1 = z2e1 = Point1 = [273,338,1]
z1e1 = y3e1 = x1e2 = Point2 = [483,200,1]
z2e2 = x2e1 = y2e1 = Point3 = [273,399,1]
z1e2 = x2e2 = Point4 = [467,258,1]
y1e2 = z3e1 = x3e1 = Point5 = [22,86,1]
x3e2 = y3e2 = Point6 = [200,32,1]
y2e2 = z3e2 = Point7 = [35,139,1]
# drawing X lines # red # 0,0,255 #
cv2.line(sharpenedImage,(x1e1[0],x1e1[1]),(x1e2[0],x1e2[1]),(0,0,255),5)
cv2.line(sharpenedImage,(x2e1[0],x2e1[1]),(x2e2[0],x2e2[1]),(0,0,255),5)
cv2.line(sharpenedImage,(x3e1[0],x3e1[1]),(x3e2[0],x3e2[1]),(0,0,255),5)
# drawing Y lines # blue # 255,0,0 #
cv2.line(sharpenedImage,(y1e1[0],y1e1[1]),(y1e2[0],y1e2[1]),(255,0,0),5)
cv2.line(sharpenedImage,(y2e1[0],y2e1[1]),(y2e2[0],y2e2[1]),(255,0,0),5)
cv2.line(sharpenedImage,(y3e1[0],y3e1[1]),(y3e2[0],y3e2[1]),(255,0,0),5)
# drawing Z lines # green # 0,255,0 #
cv2.line(sharpenedImage,(z1e1[0],z1e1[1]),(z1e2[0],z1e2[1]),(0,255,0),5)
cv2.line(sharpenedImage,(z2e1[0],z2e1[1]),(z2e2[0],z2e2[1]),(0,255,0),5)
cv2.line(sharpenedImage,(z3e1[0],z3e1[1]),(z3e2[0],z3e2[1]),(0,255,0),5)
# Corner defined in # 255,118,233 #
cv2.circle(sharpenedImage,(Point1[0],Point1[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point1', (Point1[0] -5, Point1[1] -10),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2, lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point2[0],Point2[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point2', (Point2[0] -80, Point2[1] -10),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2, lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point3[0],Point3[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point3', (Point3[0] -5, Point3[1] -10),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2, lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point4[0],Point4[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point4', (Point4[0] -50, Point4[1] -10),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2, lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point5[0],Point5[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point5', (Point5[0] -5, Point5[1] -10),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2, lineType=cv2.LINE_AA)
```

```
cv2.circle(sharpenedImage,(Point6[0],Point6[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point6', (Point6[0] -5, Point6[1] -10),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2, lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point7[0],Point7[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point7', (Point7[0] -5, Point7[1] -10),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2, lineType=cv2.LINE_AA)
```

- **Computing vanishing points:**
  - Step 1: Select 2 lines in plane XZ (line X1 and line X2), plane YZ (line Y1 and line Y2) and plane XY (line Z1 and line Z2).
  - Step 2: Cross product of (line X1 and line X2), (line Y1 and line Y2), and (line Z1 and line Z2) respectively.

```
# Specifying lines and finding vanishing points
lineX1 = aX1,bX1,cX1 = np.cross(Point3,Point4)
lineX2 = aX2,bX2,cX2 = np.cross(Point1,Point2)
lineY1 = aY1,bY1,cY1 = np.cross(Point3,Point7)
lineY2 = aY2,bY2,cY2 = np.cross(Point1,Point5)
lineZ1 = aZ1, bZ1, cZ1 = np.cross(Point3,Point1)
lineZ2 = aZ2, bZ2, cZ2 = np.cross(Point4,Point2)
vanishingPtX = np.cross(lineX1, lineX2)
vanishingPtY = np.cross(lineY1, lineY2)
vanishingPtZ = np.cross(lineZ1, lineZ2)
```

  - Step 3: Scaling the vanishing points and defining reference points for all axes and world origin.

```
## ---> Scaling vanishing points that makes the last coordinate is 1
vanishingPtX = vanishingPtX/(vanishingPtX[2])
vanishingPtX = np.array(vanishingPtX)
vanishingPtY = vanishingPtY/(vanishingPtY[2])
vanishingPtY = np.array(vanishingPtY)
vanishingPtZ = vanishingPtZ/(vanishingPtZ[2])
vanishingPtZ = np.array(vanishingPtZ)
## ---> Reference selection
refWorldOrigin = Point3  #worldOrigin
refPointX = Point4     #xAxis
refPointZ = Point1     #yAxis
refPointY = Point7     #zAxis
refWorldOrigin = np.array([refWorldOrigin])
refPointX = np.array([refPointX])
refPointZ = np.array([refPointZ])
refPointY = np.array([refPointY])
```

  - Step 4: Finding the alpha(scaling factor) for X,Y,Z and divide the first row of individual alpha with individual length.

```
## ---> Find Length of the lines
lengthX = np.sqrt(np.square(refPointX[0] - refWorldOrigin[0]) +
np.square(refPointX[1] - refWorldOrigin[1]))
lengthY = np.sqrt(np.square(refPointY[0] - refWorldOrigin[0]) +
np.square(refPointY[1] - refWorldOrigin[1]))
lengthZ = np.sqrt(np.square(refPointZ[0] - refWorldOrigin[0]) +
np.square(refPointZ[1] - refWorldOrigin[1]))
```

```
alphaX,residual,rank,s = linalg.lstsq( (vanishingPtX-refPointX).T , (refPointX
- refWorldOrigin).T )
alphaX = alphaX[0][0]/lengthX
alphaY,residual,rank,s = linalg.lstsq( (vanishingPtY-refPointY).T , (refPointY
- refWorldOrigin).T )
alphaY = alphaY[0][0]/lengthY
alphaZ,residual,rank,s = linalg.lstsq( (vanishingPtZ-refPointZ).T , (refPointZ
- refWorldOrigin).T )
alphaZ = alphaZ[0][0]/lengthZ
```

- **Computing the projection matrix and homograph matrix:**
  - Step 1: Defining the projection matrix.

```
## ---> Projection Matrix definition
projectionX = alphaX*vanishingPtX
projectionY = alphaY*vanishingPtY
projectionZ = alphaZ*vanishingPtZ
projectionMatrix = np.empty([3,4])
projectionMatrix[:,0] = projectionX
projectionMatrix[:,1] = projectionY
projectionMatrix[:,2] = projectionZ
projectionMatrix[:,3] = refWorldOrigin
```

  - Step 2: Defining the homography matrix from the projection matrix.

```
## ---> Homography matrix for XY plane
homographicXY=np.zeros((3,3))
homographicXY[:,0] = projectionX
homographicXY[:,1] = projectionY
homographicXY[:,2] = refWorldOrigin
## ---> Homography matrix for ZY plane
homographicYZ=np.zeros((3,3))
homographicYZ[:,0] = projectionZ
homographicYZ[:,1] = projectionY
homographicYZ[:,2] = refWorldOrigin
## ---> Homography matrix for XZ plane
homographicZX=np.empty((3,3))
homographicZX[:,0] = projectionX
homographicZX[:,1] = projectionZ
homographicZX[:,2] = refWorldOrigin
```

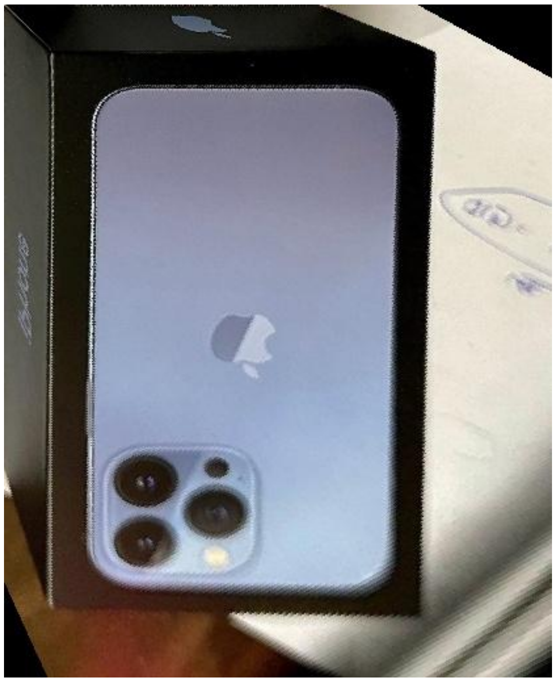
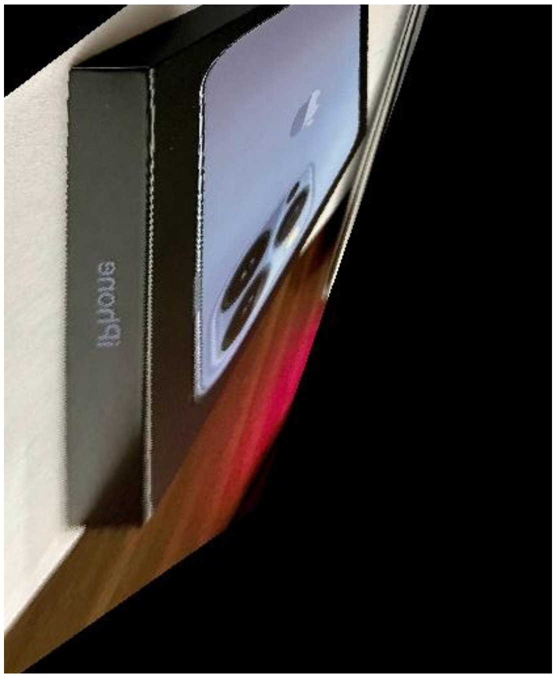
- Computing the texture maps for XY, YZ, and XZ planes respectively:
  - Step 1: Defining the Projection matrix.

```
## ---> Output adjustmen
homographicYZ[0,2] = homographicYZ[0,2]+50 #moving the image matrix towards
the left
homographicYZ[1,2] = homographicYZ[1,2]+100  #moving the image matrix
downwards
homographicZX[0,2] = homographicZX[0,2]-20 #moving the image matrix towards
the left
homographicZX[1,2] = homographicZX[1,2]+100  #moving the image matrix
downwards
```

```
## ---> Warpping the perspective with respect to homography matrix an dimage
size as per the original image
outputXY =
cv2.warpPerspective(baseImage,(homographicXY),(Rows,Columns),flags=cv2.WARP_IN
VERSE_MAP)
outputYZ =
cv2.warpPerspective(baseImage,(homographicYZ),(Rows,Columns),flags=cv2.WARP_IN
VERSE_MAP)
outputXZ =
cv2.warpPerspective(baseImage,(homographicZX),(Rows,Columns),flags=cv2.WARP_IN
VERSE_MAP)
```

- Step 2: Computing image cropping using Get Perspective Transform (it can also be done using Numpy cropping).

```
## ---> Cropping function
perspectiveTransformXZ =
cv2.getPerspectiveTransform(perspectiveCoordsXZ,planeXZ)
perspectiveTransformYZ =
cv2.getPerspectiveTransform(perspectiveCoordsYZ,planeYZ)
perspectiveTransformXY =
cv2.getPerspectiveTransform(perspectiveCoordsXY,planeXY)
croppedOutputXZ =
cv2.warpPerspective(baseImage,perspectiveTransformXZ,(widthXZ,heightXZ),flags=
cv2.WARP_INVERSE_MAP)
croppedOutputYZ =
cv2.warpPerspective(baseImage,perspectiveTransformYZ,(widthYZ,heightYZ),flags=
cv2.WARP_INVERSE_MAP)
croppedOutputXY =
cv2.warpPerspective(baseImage,perspectiveTransformXY,(widthXY,heightXY),flags=
cv2.WARP_INVERSE_MAP)
```

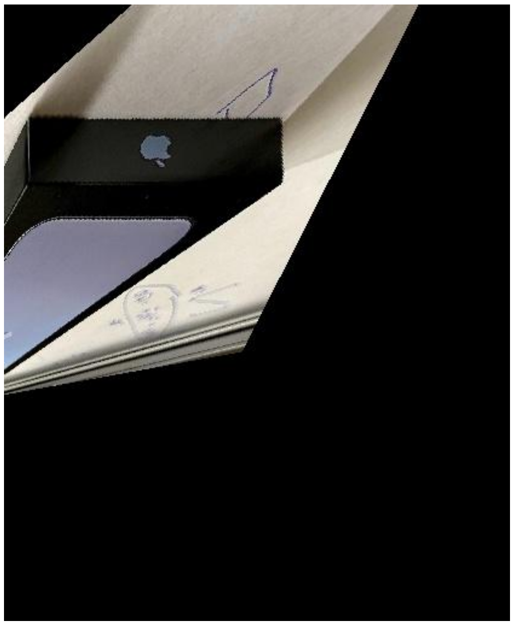


Figure 3:XY, 4:YZ, 5:XZ

- Step 3: Saving and displaying outputs

```
# Saving all the output images
filename = 'contours.jpg'
cv2.imwrite(filename, sharpenedImage)
filename1 = 'planeXY.jpg'
cv2.imwrite(filename1, outputXY)
filename2 = 'planeYZ.jpg'
cv2.imwrite(filename2, outputYZ)
filename3 = 'planeXZ.jpg'
cv2.imwrite(filename3, outputXZ)
filename4 = 'croppedOutputXZ.jpg'
cv2.imwrite(filename4, croppedOutputXZ)
filename5 = 'croppedOutputYZ.jpg'
cv2.imwrite(filename5, croppedOutputYZ)
filename = 'croppedOutputXY.jpg'
cv2.imwrite(filename, croppedOutputXY)
cv2.imshow('imageSample1', sharpenedImage)
cv2.imshow('imageSampleXZ', outputXZ)
cv2.imshow('imageSampleYZ', outputYZ)
cv2.imshow('imageSampleXY', outputXY)
cv2.waitKey(0)
cv2.imshow('imageSampleXZ', croppedOutputXZ)
cv2.imshow('imageSampleYZ', croppedOutputYZ)
cv2.imshow('XYplane.jpg', croppedOutputXY)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

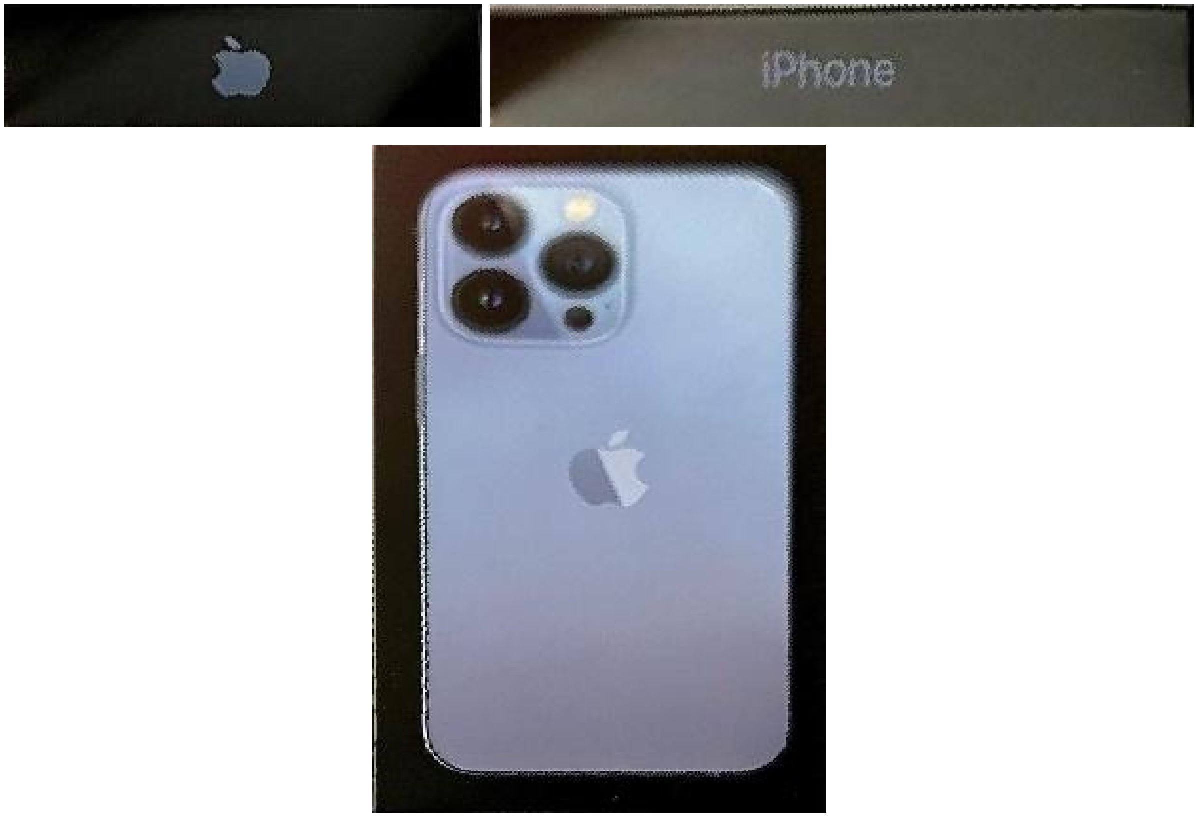


Figure 5:XZ, 6:YZ, 7:XY

- **Visualizing the reconstructed 3D model:**
  - Step 1: Used blender for 3D projection using cropped planes.

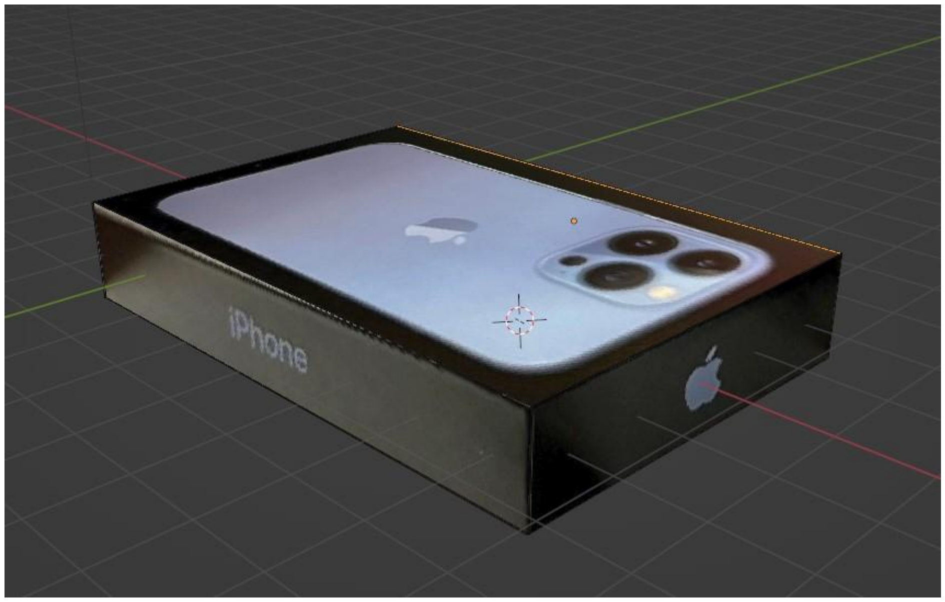


Figure 8

Akshay Vijay Khanna
#unityID : akhanna3

Code:

```
# importing all the libraries required
import cv2
import numpy as np
from scipy import linalg

# loading the image sample and resizing as per requirement
testImage = cv2.imread(r'D:\\NCSU\\NCSU Courses\\DIS - 558\\DIS
            Project1\\ReferenceImage\\Sample_1.jpg')
baseImage = cv2.resize(testImage,None, fx=0.5,fy=0.5)
Rows, Columns, Temporary = baseImage.shape

# image sharpening filter
sharpenedFilter = np.array([[-0.1,-0.1,-0.1],
                          [-0.1,1.9,-0.1],
                          [-0.1,-0.1,-0.1]])
sharpenedImage = cv2.filter2D(baseImage, -1, sharpenedFilter)

# specifying coordinates
x1e1 = y1e1 = z2e1 = Point1 = [273,338,1]
z1e1 = y3e1 = x1e2 = Point2 = [483,200,1]
z2e2 = x2e1 = y2e1 = Point3 = [273,399,1]
z1e2 = x2e2 = Point4 = [467,258,1]
y1e2 = z3e1 = x3e1 = Point5 = [22,86,1]
x3e2 = y3e2 = Point6 = [200,32,1]
y2e2 = z3e2 = Point7 = [35,139,1]

# drawing X lines # red # 0,0,255 #
cv2.line(sharpenedImage,(x1e1[0],x1e1[1]),(x1e2[0],x1e2[1]),(0,0,255),5)
cv2.line(sharpenedImage,(x2e1[0],x2e1[1]),(x2e2[0],x2e2[1]),(0,0,255),5)
cv2.line(sharpenedImage,(x3e1[0],x3e1[1]),(x3e2[0],x3e2[1]),(0,0,255),5)

# drawing Y lines # blue # 255,0,0 #
cv2.line(sharpenedImage,(y1e1[0],y1e1[1]),(y1e2[0],y1e2[1]),(255,0,0),5)
cv2.line(sharpenedImage,(y2e1[0],y2e1[1]),(y2e2[0],y2e2[1]),(255,0,0),5)
cv2.line(sharpenedImage,(y3e1[0],y3e1[1]),(y3e2[0],y3e2[1]),(255,0,0),5)

# drawing Z lines # green # 0,255,0 #
cv2.line(sharpenedImage,(z1e1[0],z1e1[1]),(z1e2[0],z1e2[1]),(0,255,0),5)
cv2.line(sharpenedImage,(z2e1[0],z2e1[1]),(z2e2[0],z2e2[1]),(0,255,0),5)
cv2.line(sharpenedImage,(z3e1[0],z3e1[1]),(z3e2[0],z3e2[1]),(0,255,0),5)

# Corner defined in # 255,118,233 #
cv2.circle(sharpenedImage,(Point1[0],Point1[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point1', (Point1[0] -5, Point1[1] -10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2,
            lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point2[0],Point2[1]), 7, (255,0,255), cv2.FILLED)
```

```
cv2.putText(sharpenedImage, 'Point2', (Point2[0] -80, Point2[1] -10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2,
            lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point3[0],Point3[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point3', (Point3[0] -5, Point3[1] -10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2,
            lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point4[0],Point4[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point4', (Point4[0] -50, Point4[1] -10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2,
            lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point5[0],Point5[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point5', (Point5[0] -5, Point5[1] -10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2,
            lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point6[0],Point6[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point6', (Point6[0] -5, Point6[1] -10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2,
            lineType=cv2.LINE_AA)
cv2.circle(sharpenedImage,(Point7[0],Point7[1]), 7, (255,0,255), cv2.FILLED)
cv2.putText(sharpenedImage, 'Point7', (Point7[0] -5, Point7[1] -10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255,118,233), 2,
            lineType=cv2.LINE_AA)

# Specifying lines and finding vanishing points
lineX1 = aX1,bX1,cX1 = np.cross(Point3,Point4)
lineX2 = aX2,bX2,cX2 = np.cross(Point1,Point2)
lineY1 = aY1,bY1,cY1 = np.cross(Point3,Point7)
lineY2 = aY2,bY2,cY2 = np.cross(Point1,Point5)
lineZ1 = aZ1, bZ1, cZ1 = np.cross(Point3,Point1)
lineZ2 = aZ2, bZ2, cZ2 = np.cross(Point4,Point2)
vanishingPtX = np.cross(lineX1, lineX2)
vanishingPtY = np.cross(lineY1, lineY2)
vanishingPtZ = np.cross(lineZ1, lineZ2)

# Homography Matrix Formation #

## ---> Reference selection
refWorldOrigin = Point3  #worldOrigin
refPointX = Point4     #xAxis
refPointZ = Point1     #yAxis
refPointY = Point7     #zAxis

## ---> Scaling vanishing points that makes the last coordinate is 1

vanishingPtX = vanishingPtX/(vanishingPtX[2])
vanishingPtX = np.array(vanishingPtX)
vanishingPtY = vanishingPtY/(vanishingPtY[2])
vanishingPtY = np.array(vanishingPtY)
vanishingPtZ = vanishingPtZ/(vanishingPtZ[2])
vanishingPtZ = np.array(vanishingPtZ)
```

```
## ---> Find Length of the lines

lengthX = np.sqrt(np.square(refPointX[0] - refWorldOrigin[0]) +
            np.square(refPointX[1] - refWorldOrigin[1]))
lengthY = np.sqrt(np.square(refPointY[0] - refWorldOrigin[0]) +
            np.square(refPointY[1] - refWorldOrigin[1]))
lengthZ = np.sqrt(np.square(refPointZ[0] - refWorldOrigin[0]) +
            np.square(refPointZ[1] - refWorldOrigin[1]))

## ---> Finding aplha with source as vanishing points and destinations as
            reference points

refWorldOrigin = np.array([refWorldOrigin])
refPointX = np.array([refPointX])
refPointZ = np.array([refPointZ])
refPointY = np.array([refPointY])

alphaX,residual,rank,s = linalg.lstsq( (vanishingPtX-refPointX).T , (refPointX
            - refWorldOrigin).T )
alphaX = alphaX[0][0]/lengthX
alphaY,residual,rank,s = linalg.lstsq( (vanishingPtY-refPointY).T , (refPointY
            - refWorldOrigin).T )
alphaY = alphaY[0][0]/lengthY
alphaZ,residual,rank,s = linalg.lstsq( (vanishingPtZ-refPointZ).T , (refPointZ
            - refWorldOrigin).T )
alphaZ = alphaZ[0][0]/lengthZ

## ---> Projection Matrix definition

projectionX = alphaX*vanishingPtX
projectionY = alphaY*vanishingPtY
projectionZ = alphaZ*vanishingPtZ
projectionMatrix = np.empty([3,4])
projectionMatrix[:,0] = projectionX
projectionMatrix[:,1] = projectionY
projectionMatrix[:,2] = projectionZ
projectionMatrix[:,3] = refWorldOrigin


## ---> Homography matrix for XY plane

homographicXY=np.zeros((3,3))
homographicXY[:,0] = projectionX
homographicXY[:,1] = projectionY
homographicXY[:,2] = refWorldOrigin

## ---> Homography matrix for ZY plane

homographicYZ=np.zeros((3,3))
homographicYZ[:,0] = projectionZ
homographicYZ[:,1] = projectionY
homographicYZ[:,2] = refWorldOrigin
```

```
## ---> Homography matrix for XZ plane

homographicZX=np.empty((3,3))
homographicZX[:,0] = projectionX
homographicZX[:,1] = projectionZ
homographicZX[:,2] = refWorldOrigin

## ---> Output adjustment

homographicYZ[0,2] = homographicYZ[0,2]+50 #moving the image matrix towards
            the left
homographicYZ[1,2] = homographicYZ[1,2]+100  #moving the image matrix
            downwards
homographicZX[0,2] = homographicZX[0,2]-20 #moving the image matrix towards
            the left
homographicZX[1,2] = homographicZX[1,2]+100  #moving the image matrix
            downwards

## ---> Warpping the perspective with respect to homography matrix an dimage
            size as per the original image

outputXY =
            cv2.warpPerspective(baseImage,(homographicXY),(Rows,Columns),flags
            =cv2.WARP_INVERSE_MAP)
outputYZ =
            cv2.warpPerspective(baseImage,(homographicYZ),(Rows,Columns),flags
            =cv2.WARP_INVERSE_MAP)
outputXZ =
            cv2.warpPerspective(baseImage,(homographicZX),(Rows,Columns),flags
            =cv2.WARP_INVERSE_MAP)

# Defining perspective transform for cropping perspective planes

widthXZ, heightXZ = lengthX, lengthZ
widthYZ, heightYZ = lengthY, lengthZ
widthXY, heightXY = lengthX, lengthY

widthXZ = int(widthXZ)
heightXZ = int(heightXZ)
widthYZ = int(widthYZ)
heightYZ = int(heightYZ)
widthXY = int(widthXY)
heightXY = int(heightXY)

planeXY = np.float32([[Point5[0],Point5[1]],[Point6[0],Point6[1]],
            [Point2[0],Point2[1]],[Point1[0],Point1[1]]])
planeYZ =
            np.float32([[Point5[0],Point5[1]],[Point1[0],Point1[1]],[Point3[0]
            ,Point3[1]], [Point7[0],Point7[1]]])
```

```
planeXZ = np.float32([[Point2[0],Point2[1]],
            [Point4[0],Point4[1]],[Point3[0],Point3[1]],[Point1[0],Point1[1]]]
            )

perspectiveCoordsXZ =
            np.float32([[0,0],[0,heightXZ],[widthXZ,heightXZ],[widthXZ,0]])
perspectiveCoordsYZ = np.float32([[0,0],[widthYZ,0], [widthYZ,heightYZ],
            [0,heightYZ]])
perspectiveCoordsXY = np.float32([[0,0],[widthXY,0], [widthXY,heightXY],
            [0,heightXY]])

## ---> Cropping function

perspectiveTransformXZ =
            cv2.getPerspectiveTransform(perspectiveCoordsXZ,planeXZ)
perspectiveTransformYZ =
            cv2.getPerspectiveTransform(perspectiveCoordsYZ,planeYZ)
perspectiveTransformXY =
            cv2.getPerspectiveTransform(perspectiveCoordsXY,planeXY)

croppedOutputXZ =
            cv2.warpPerspective(baseImage,perspectiveTransformXZ,(widthXZ,heig
            htXZ),flags=cv2.WARP_INVERSE_MAP)
croppedOutputYZ =
            cv2.warpPerspective(baseImage,perspectiveTransformYZ,(widthYZ,heig
            htYZ),flags=cv2.WARP_INVERSE_MAP)
croppedOutputXY =
            cv2.warpPerspective(baseImage,perspectiveTransformXY,(widthXY,heig
            htXY),flags=cv2.WARP_INVERSE_MAP)

# Saving all the output images

filename = 'contours.jpg'
cv2.imwrite(filename, sharpenedImage)
filename1 = 'planeXY.jpg'
cv2.imwrite(filename1, outputXY)
filename2 = 'planeYZ.jpg'
cv2.imwrite(filename2, outputYZ)
filename3 = 'planeXZ.jpg'
cv2.imwrite(filename3, outputXZ)
filename4 = 'croppedOutputXZ.jpg'
cv2.imwrite(filename4, croppedOutputXZ)
filename5 = 'croppedOutputYZ.jpg'
cv2.imwrite(filename5, croppedOutputYZ)
filename = 'croppedOutputXY.jpg'
cv2.imwrite(filename, croppedOutputXY)

cv2.imshow('imageSample1', sharpenedImage)
cv2.imshow('imageSampleXZ', outputXZ)
cv2.imshow('imageSampleYZ', outputYZ)
cv2.imshow('imageSampleXY', outputXY)
cv2.waitKey(0)
```

```
cv2.imshow('imageSampleXZ', croppedOutputXZ)
cv2.imshow('imageSampleYZ', croppedOutputYZ)
cv2.imshow('XYplane.jpg', croppedOutputXY)
cv2.waitKey(0)
cv2.destroyAllWindows()
```