This code is developed under Ubuntu 18.04, using Python3 with Eclipse.

To run this code, make sure that the txt files are in a directory above where the Main.py is.

This code also requires the following Python3 packages:
        numpy ,tkinter, cv

This code has been only tested under ubuntu18.04 in terminal. To run this code, type "Python3 Main.py", GUI interface will take user input.

Type the file name including .txt

Type the coordinate of the view point, separate with comma. For example 20,20,20

Type the pattern board information, the first 3 will be the point on plane, the following 6 are the two directions. For example 0,0,0,0,0,1,1,1,0 means [0,0,0] is on the plane, [0,0,1] is one of the direction, [1,1,0] is the other direction

Type pattern width and spacing, separate with comma. for example 15, 12 means 15 for width, 12 for spacing

First calculate the normal directions of vertices on the board by taking average values of the vector cross product around the vertex.

Find the incident ray by subtracting **e** from the vertex location p_{ij} , store in a 2D array.

Calculate the reflection ray direction using equation $v_r = v_i - 2(\hat{n} \cdot v_i)\hat{n}$

Using point p_{ij} with v_r , calculate intersection between the reflect ray and pattern board, this will give u = $\frac{(b \times -v_r) \cdot (p_{ij} - p_0)}{-v_r \cdot (a \times b)}$ and v = $\frac{(-v_r \times a) \cdot (p_{ij} - p_0)}{-v_r \cdot (a \times b)}$, u will be the distance from the pattern edge, using u we can calculate if the intersection in the the black region or white region. This information will be used to render the WRL file.

```
#!/bin/bash
'''
Created on Oct 8, 2018

@author: grant
'''
from tkinter import *
import tkinter.simpledialog as db
import tkinter.messagebox as mb
import numpy as np
from cv2.cv2 import subtract, norm


def convertVertex(floatArray):
    l = len(floatArray)
    stop = 0
    index = 2
    toReturn = []
    while (stop != 1):
        tempArray = [floatArray[index],floatArray[index + 1],floatArray[index + 2]]
        tempArray = np.array(tempArray)
        toReturn.append(tempArray)
        index += 3
        if (index >= l):
            stop = 1

    return toReturn

def averageVectors(vectorArray):
    l = len(vectorArray)
    toReturn = vectorArray[0]
    for i in range(l - 1):
        toReturn = np.add(toReturn, vectorArray[i + 1])

    toReturn /= l
    toReturn /= np.linalg.norm(toReturn)
    return toReturn

def calculateNormal(vertex2DArray):
    dim = len(vertex2DArray)
    toReturn = [[0 for _ in range(dim)] for _ in range (dim)]

    for i in range(dim):
        for j in range(dim):

            # Corner cases
            if (i == 0 and j == 0):
                c0 = vertex2DArray[i][j]
                cw = vertex2DArray[i + 1][j]
                cd = vertex2DArray[i][j + 1]

                vw = np.subtract(cw, c0)
                vd = np.subtract(cd, c0)
                norm = np.cross(vd,vw)
                norm = averageVectors([norm])
                toReturn[i][j] = norm

            elif(i == 0 and j == dim - 1):
                c0 = vertex2DArray[i][j]
                cw = vertex2DArray[i + 1][j]
                ca = vertex2DArray[i][j - 1]
```

```
            vw = np.subtract(cw, c0)
            va = np.subtract(ca, c0)
            norm = np.cross(vw,va)
            norm = averageVectors([norm])
            toReturn[i][j] = norm

        elif(i == dim - 1 and j == 0):
            c0 = vertex2DArray[i][j]
            cs = vertex2DArray[i - 1][j]
            cd = vertex2DArray[i][j + 1]

            vs = np.subtract(cs, c0)
            vd = np.subtract(cd, c0)
            norm = np.cross(vs,vd)
            norm = averageVectors([norm])
            toReturn[i][j] = norm

        elif(i == dim - 1 and j == dim - 1):
            c0 = vertex2DArray[i][j]
            cs = vertex2DArray[i - 1][j]
            ca = vertex2DArray[i][j - 1]

            vs = np.subtract(cs, c0)
            va = np.subtract(ca, c0)
            norm = np.cross(va,vs)
            norm = averageVectors([norm])
            toReturn[i][j] = norm

        elif(i == 0):
            c0 = vertex2DArray[i][j]
            ca = vertex2DArray[i][j - 1]
            cd = vertex2DArray[i][j + 1]
            cw = vertex2DArray[i + 1][j]

            va = np.subtract(ca, c0)
            vw = np.subtract(cw, c0)
            vd = np.subtract(cd, c0)

            n1 = np.cross(vw,va)
            n2 = np.cross(vd,vw)
            norm = averageVectors([n1,n2])
            toReturn[i][j] = norm

        elif(i == dim - 1):
            c0 = vertex2DArray[i][j]
            ca = vertex2DArray[i][j - 1]
            cd = vertex2DArray[i][j + 1]
            cs = vertex2DArray[i - 1][j]

            va = np.subtract(ca, c0)
            vs = np.subtract(cs, c0)
            vd = np.subtract(cd, c0)

            n1 = np.cross(va,vs)
            n2 = np.cross(vs,vd)
            norm = averageVectors([n1,n2])
            toReturn[i][j] = norm

        elif(j % dim == 0):
            c0 = vertex2DArray[i][j]
            cw = vertex2DArray[i + 1][j]
            cs = vertex2DArray[i - 1][j]
```

```
                cd = vertex2DArray[i][j + 1]

                vw = np.subtract(cw, c0)
                vs = np.subtract(cs, c0)
                vd = np.subtract(cd, c0)

                n1 = np.cross(vs,vd)
                n2 = np.cross(vd,vw)
                norm = averageVectors([n1,n2])
                toReturn[i][j] = norm

            elif((j + 1) % dim == 0):
                c0 = vertex2DArray[i][j]
                cw = vertex2DArray[i + 1][j]
                cs = vertex2DArray[i - 1][j]
                ca = vertex2DArray[i][j - 1]

                vw = np.subtract(cw, c0)
                vs = np.subtract(cs, c0)
                va = np.subtract(ca, c0)

                n1 = np.cross(vw,va)
                n2 = np.cross(va,vs)
                norm = averageVectors([n1,n2])
                toReturn[i][j] = norm

            else:
                c0 = vertex2DArray[i][j]
                cw = vertex2DArray[i + 1][j]
                ca = vertex2DArray[i][j - 1]
                cs = vertex2DArray[i - 1][j]
                cd = vertex2DArray[i][j + 1]

                vw = np.subtract(cw, c0)
                va = np.subtract(ca, c0)
                vs = np.subtract(cs, c0)
                vd = np.subtract(cd, c0)

                n1 = np.cross(vw, va)
                n2 = np.cross(va, vs)
                n3 = np.cross(vs, vd)
                n4 = np.cross(vd, vw)
                norm = averageVectors([n1,n2,n3,n4])
                toReturn[i][j] = norm
    return toReturn

def calculateIncidentArray(viewPoint, vertices):
    dim = len(vertices)
    toReturn = [[0 for _ in range(dim)] for _ in range (dim)]

    for i in range(dim):
        for j in range(dim):
            currVertex = vertices[i][j]
            incidentRay = np.subtract(currVertex, viewPoint)
            toReturn[i][j] = incidentRay


    return toReturn

def calculateReflection(incidentRays, normalDirections):

    dim = len(incidentRays)
```

```
    toReturn = [[0 for _ in range(dim)] for _ in range (dim)]

    for i in range(dim):
        for j in range(dim):
            projection = np.dot(incidentRays[i][j], normalDirections[i][j])
            projection = np.multiply(normalDirections[i][j], projection * 2)
            #reflection = np.add(incidentRays[i][j], projection)
            reflection = np.subtract(incidentRays[i][j], projection)
            toReturn[i][j] = reflection

    return toReturn

def calculateIntersection(p01, p02, la, p0, lab):
    toReturn = []

    nlab = np.multiply(-1, lab)
    temp = np.cross(p02, nlab )
    temp2 = np.subtract(la, p0)
    num = np.dot(temp,temp2)
    temp = np.cross(p01, p02)
    denom = np.dot(nlab, temp)
    toReturn.append(num/denom)

    temp1 = np.cross(nlab, p01)
    temp2 = np.subtract(la, p0)
    num = np.dot(temp1, temp2)
    temp = np.cross(p01, p02)
    denom = np.dot(nlab, temp)
    toReturn.append(num / denom)


    return toReturn

def convertTo2DArray(dataPoints, dim):
    toReturn = [[0 for _ in range(dim)] for _ in range (dim)]
    direction = 0 # 0 means x increase first

    c0 = dataPoints[0]
    c1 = dataPoints[1]


    c01 = np.subtract(c1, c0)


    if (c01.item(1) <= 0.0001):
        direction = 0
    else:
        direction = 1
    counter = 0

    if (direction == 0):
        for i in range(dim):
            for j in range(dim):
                toReturn[i][j] =dataPoints[counter]
                counter += 1

    elif(direction == 1):
        for i in range(dim):
            for j in range(dim):
                toReturn[j][i] = dataPoints[counter]
                counter += 1
```

```
    return toReturn

def calculateColor(intersection, width, space):
    length = intersection[0]
    combined = width + space
    rem = length % combined

    if (rem < width):
        return 0
    elif (rem >= width):
        return 1

def generatePointsString(vertex2DArray):
    toReturn = "point["
    dim = len(vertex2DArray)

    for i in range(dim):
        for j in range(dim):
            currString = np.array2string(vertex2DArray[i][j],
formatter={'float_kind':lambda x: "%.5f" % x})
            currString = currString[1:-1]
            currString += ", "
            toReturn += currString

    toReturn = toReturn[:-2]
    toReturn += " ]"

    return toReturn

def generateCoordIndex(size):
    toReturn = "coordIndex[ "
    for i in range (size - 1):
        for j in range (size - 1):
            current = ""
            yCord = i * size
            current += str(yCord + j) + ", " + str(yCord + j + 1) + ", "
            nextYcord = (i + 1) * size
            current += str(nextYcord + j + 1) + ", " + str(nextYcord + j) + ", -1, "
            toReturn += current

    toReturn = toReturn[:-2]
    toReturn += "]"
    return toReturn

def generateColorIndex(colorInfo, size):
    toReturn = "colorIndex[ "

    for i in range(size - 1):
        for j in range(size - 1):
            current = str(colorInfo[i][j]) + ", " + str(colorInfo[i][j + 1]) + ", "
            current += str(colorInfo[i + 1][j + 1]) + ", " + str(colorInfo[i + 1][j])
+ ", " + "-1, "
            toReturn += current

    toReturn = toReturn[:-2]
    toReturn += " ]"
    return toReturn

if __name__ == '__main__':
    pass
```

```
root = Tk()
w = Label(root, text = "Computer Aided Design HW 6")
w.pack()
mb.showinfo("Enter Instructions", "Please enter numbers, \
when multiple numbers needed, separate with comma")
fileName = db.askstring("File Name", "File Name")
viewPoint = db.askstring("View Point", "view point")
patternPlane = db.askstring("Zebra Plane","A Zebra Pattern Plane, in order of p0 a
and b")
otherZebra = db.askstring("Zebra Plane", "line thickness and line spacing")


fileData = open("../" + fileName).read().split()
fileData = list(map(float,fileData))
length = fileData[0]
width = fileData[1]
vertexArray = convertVertex(fileData)
vertex2DArray = convertTo2DArray(vertexArray, int(width))
normal2DArray = calculateNormal(vertex2DArray)



viewPoint = list(map(float, viewPoint.split(",")))
viewPoint = np.array(viewPoint)
incidentRay2DArray = calculateIncidentArray(viewPoint, vertex2DArray)
reflectRay2DArray = calculateReflection(incidentRay2DArray, normal2DArray)
#print(viewPointArray)
patternPlane = list(map(float, patternPlane.split(",")))
p0 = np.array(patternPlane[:3])
p02 = np.array(patternPlane[3:6])
p01 = np.array(patternPlane[6:9])



#print(p0)
#print(a)
#print(b)
otherZebra = list(map(float, otherZebra.split(",")))
thickness = otherZebra[0]
spacing = otherZebra[1]
dim = int(width)
colorData = [[0 for _ in range(dim)] for _ in range (dim)]
# calculate color
for i in range(int(width)):
    for j in range(int(width)):
        la = vertex2DArray[i][j]
        lab = reflectRay2DArray[i][j]
        intersect = calculateIntersection(p01, p02, la, p0, lab)
        currColor = calculateColor(intersect, thickness, spacing)
        colorData[i][j] = currColor


pointsString = generatePointsString(vertex2DArray)
coordIndexString = generateCoordIndex(dim)
colorIndexArray = generateColorIndex(colorData, dim)
filename = fileName[:-3] + "wrl"
f = open(filename, "w+")
toPutInFile = "#VRML V2.0 utf8\n#Colored IndexedFaceSet Example \n#by Qiaojie Zheng
\n"
toPutInFile +=   "Shape{ \n"
toPutInFile +=  "    geometry IndexedFaceSet{ \n"
```

```
toPutInFile +=  "        coord Coordinate{ \n"
toPutInFile +=  "            " + pointsString + "\n"
toPutInFile +=  "        }\n"
toPutInFile +=  "        " + coordIndexString + "\n"
toPutInFile +=  "        color Color{\n"
toPutInFile +=  "            color[0 0 0, 1 1 1] \n"
toPutInFile +=  "        }\n"
toPutInFile +=  "        " + colorIndexArray + "\n"
toPutInFile +=  "        solid FALSE"
toPutInFile +=  "    }\n }"
f.write(toPutInFile)
f.close()
```