Mini-Project 1: Hough Transform

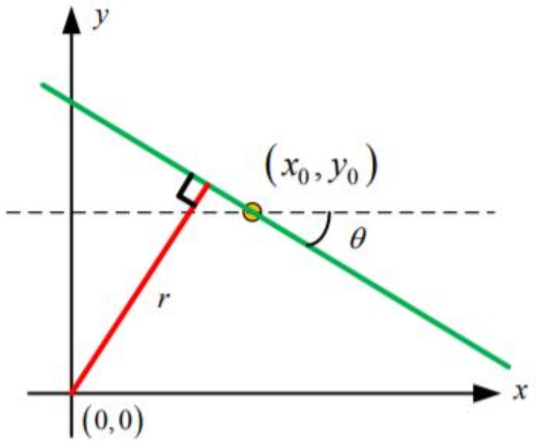
Objective:

The objective of the mini-project was to identify the shape and orientations of various solid polygons by detecting lines using the Hough Transform method.

Code explanation:

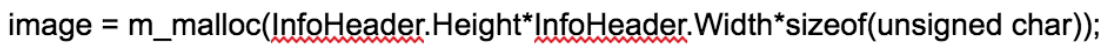
In order to determine the locations of the lines in the polygon we used the Hough Transform method. This method involves transforming a point in the Cartesian coordinate system to a r - q coordinate system. A line is drawn through the point, and the angle of the line is varied incrementally from $0°$ - $180°$ in $15°$ increments. The radius determined the shortest distance of the line to the origin. The locations of r and theta are shown in Figure 0a.

Figure 0a: Locations of r and theta used for Hough transform

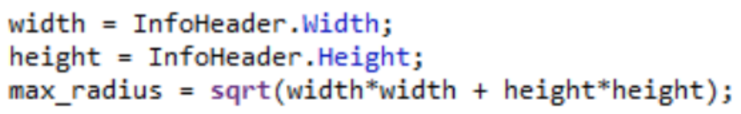


Source: 113DA CCLE

The bmp image was first stored in a 1-dimensional array named image[]. The total area of the image determined the size of this array.



```
image = m_malloc(InfoHeader.Height*InfoHeader.Width*sizeof(unsigned char));
```



```
width = InfoHeader.Width;
height = InfoHeader.Height;
max_radius = sqrt(width*width + height*height);
```

Using the InfoHeader.Width and InfoHeader.Height functions provided by the bmp.h file we can determine the size of the image. This is helpful when dynamically determining the shape of an image.

Now since all the bitmap pixels that represent the lines have a value of less than 10, they were all stored as 1's in the image array. Whereas all other pixel values were given a value of 0. This was done through the following code provided to us:

```
for(i = 0; i< InfoHeader.Height; i++)
        for(j = 0;j < InfoHeader.Width; j++)
                if(bitmap[(i*InfoHeader.Width+j)*3] < 10)
                        image[i*InfoHeader.Width+j] = 1;
                else
                        image[i*InfoHeader.Width+j] = 0;
```

Since all the lines are represented by 1's, we must traverse through the 1-D array image[] and determine the values of r at each q. This is done using the following code:

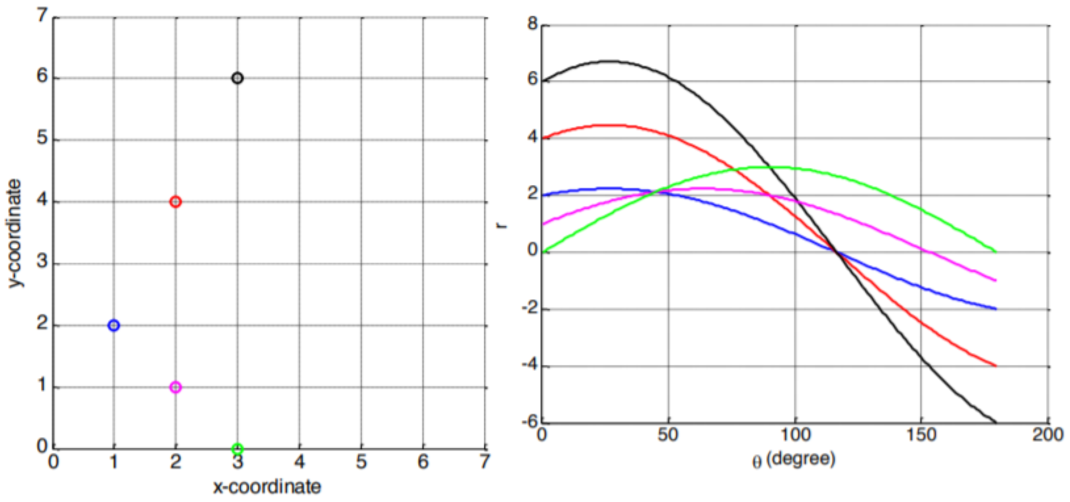
```
        for(m=0; m<height*width; m++)
        {

                    if(image[m] == 1)
                    {
                            x = m%width; // 100 is number of columns
                            y = floor(m/width); //100 is number of columns
                            for(theta = 0; theta<13; theta++){
                                    radius = y*sin((thetas[theta]/180.0)*pi) +
x*cos((thetas[theta]/180.0)*pi);
                                    int_radius = (int)radius + max_radius;
                                    votes[theta][int_radius]=votes[theta][int_radius]+1; // add
vote
                            }
                    }
            }
```

The x and y variables are used because we are given a 1-D array called "image" and we need to map them to Cartesian coordinates corresponding to where each point exists in the image. The 1's and 0's are mapped to columns and rows so the maximum occurrences of the 1's could be determined shown by the if statement.

Once the location of the 1 in the array is determined, we iterate through thetas for that point and then calculate the radius for each of the angles. The values for each (theta, r) are stored in a new 2-D array called votes[theta][int_radius]. This 2-D array holds the values of all the locations where 1 is found and increments each cell whenever a corresponding theta and radius value is determined. Every point for which (theta, r) are the same lie on the same line. Therefore, if many "votes" exist for a given (theta, r), that means that it is likely that a line exists there. In other words, we can determine the peak values of the intersection of the lines shown by the sample plot below where (theta, r) matches for multiple points.

Figure 0b: Cartesian points to Hough Transform

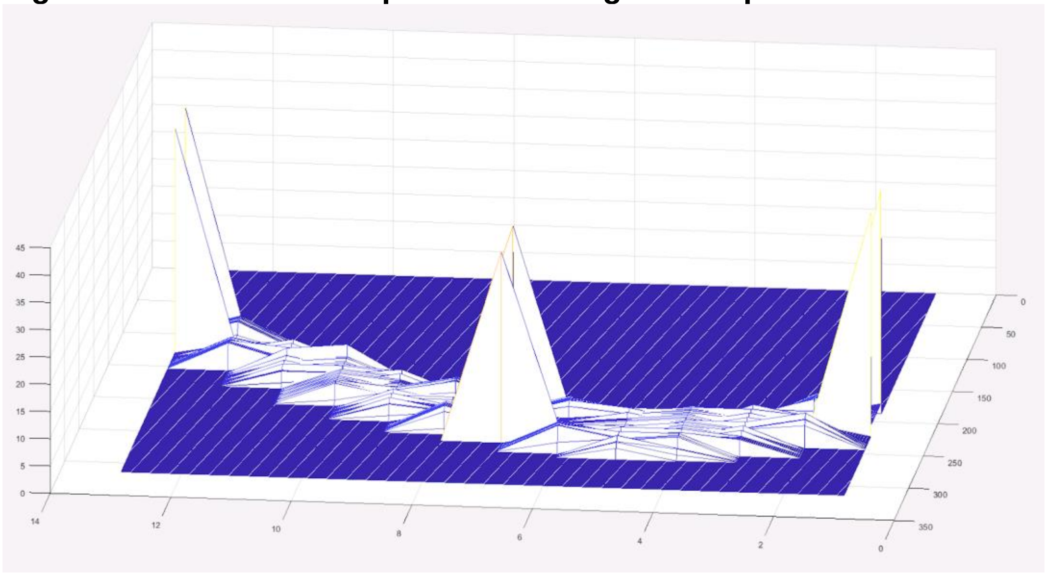


Source: 113DA CCLE

This allows us to count the number of ‘votes’ in the 2-D array and determine the lines in the polygon.

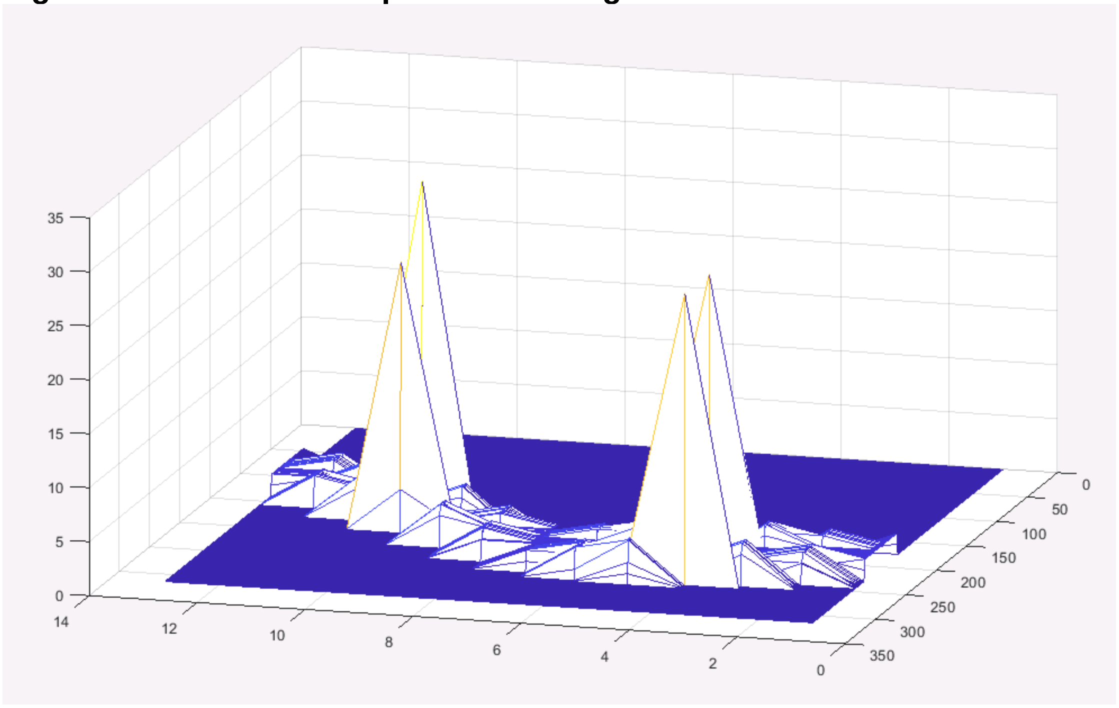
Next we use a mesh plot on Matlab to obtain a better visual of the peaks or ‘votes’. The “votes” array from the previous section was used to create the plot. The horizontal axis is theta ranging from 0 to 180, the vertical axis is radius, and the height of the peaks is the number of “votes” for any given (theta, r). A larger peak means that there are more votes, so we can determine that a longer line lies at a larger peak than at a smaller peak. The following shows the various Hough Transform 3D plot for different figures:

Figure 1: MATLAB Mesh plot for an image of a square



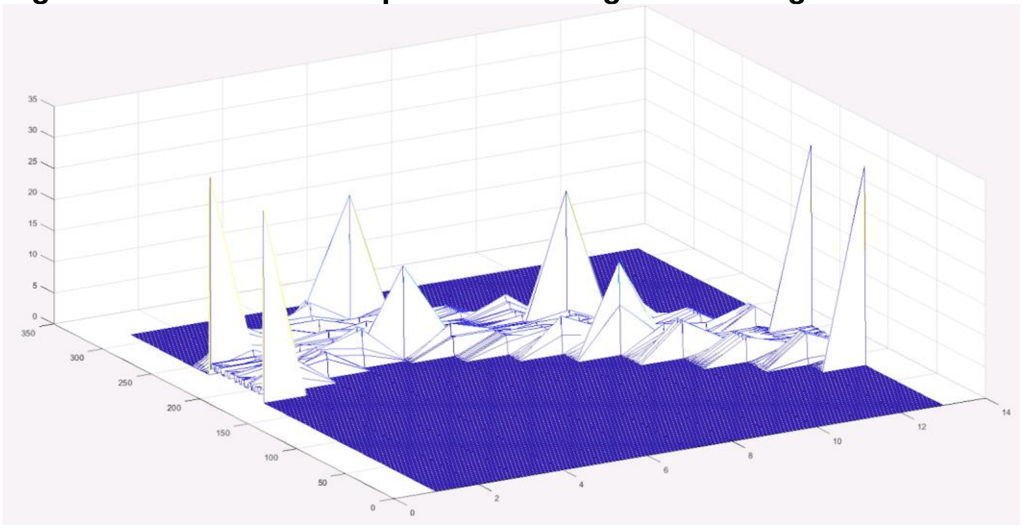
In the image of the square, we see four half-peaks at 0 and 180, which tells us that the image has two horizontal lines. There are two full peaks at 90 degrees, which tells us there are two vertical lines. The peak heights are uniform, which tell us the lines are all the same length.

Figure 2: MATLAB Mesh plot for an image of a diamond



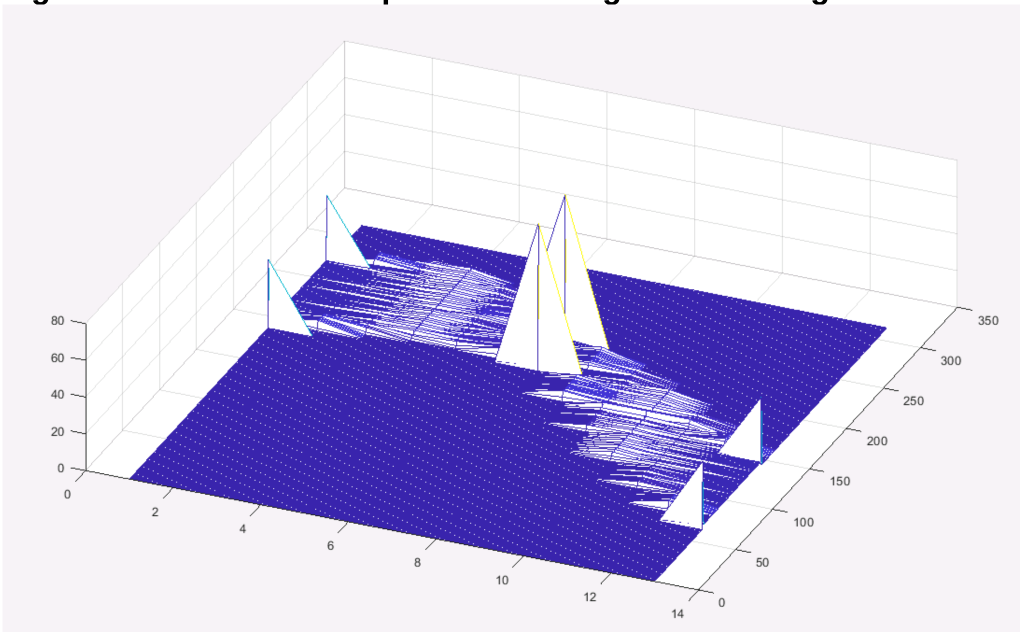
This mesh plot has the same characteristics as the square, except there are two peaks at 45 degrees and two peaks at 135 degrees, which indicate diagonal lines that make up a square tilted at 45 degrees.

Figure 3: MATLAB Mesh plot for an image of a hexagon



We can tell that this plot is of a hexagon because there are four half-peaks (two) plus four peaks, so six lines at equal increments of angles.

Figure 4: MATLAB Mesh plot for an image of a rectangle



We were given an unknown image and asked to determine its shape and orientation given this mesh plot. We could tell it was a rectangle oriented at zero degrees because of the two horizontal lines (half-peaks at 0 and 180 degrees) and two vertical lines (two full peaks at 90 degrees). We knew that the vertical lines were longer than the horizontal lines because the peaks at 90 degrees are much larger.