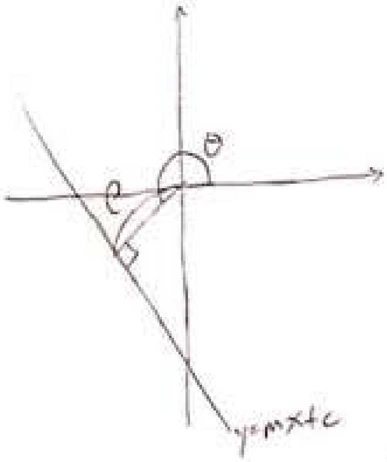
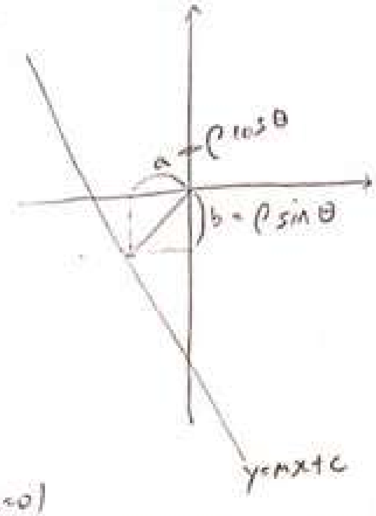
1.

1) If you draw the perpendicular line to a line $y=mx+c$ from the origin in 2 dimensional rectangular coordinates, you can define the angle between the line and the x-axis in counter-clockwise as θ, and the distance between the origin and $y=mx+c$ as ρ.





(c=0)

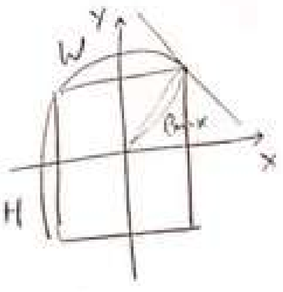
If $y=mx+c$ passes the origin, define the angle between $y=mx+c$ and x-axis in counter-clockwise as θ, and $\rho=0$. Then the gradient of the perpendicular line is $\frac{\rho/\sin\theta}{\rho\cos\theta}$, so $\frac{\sin\theta}{\cos\theta} \times m = -1$, therefore $m = -\frac{\cos\theta}{\sin\theta}$, and $c = \frac{\rho}{\sin\theta}$.

Therefore, we can say that every point on $y=mx+c$ can be denoted as $\rho = x\cos\theta + y\sin\theta$.

and on θ-ρ plane, each point on $y=mx+c$ denotes a sinusoid.

Using $\cos(x-\alpha) = \cos x\cos\alpha + \sin x\sin\alpha$, $x\cos\theta + y\sin\theta = R\cos(x-\alpha) = \rho$ where $\tan\alpha = \frac{y}{x}$ and $R=\sqrt{x^2+y^2}$. Therefore, amplitude of the sinusoid is determined by $\sqrt{x^2+y^2}$ and phase of the sinusoid is determined by $\tan\alpha = \frac{y}{x}$.

2) Assuming that you take center pixel as the origin, θ varies from 0 to 2π and ρ_{max} equals half of the diagonal, which is $\sqrt{W^2+H^2} \times \frac{1}{2}$



$0 \le \theta \le 2\pi$

$0 \le \rho \le \frac{1}{2}\sqrt{W^2+H^2}$

2.

1) Lambertian BRDF $: f(\theta_i, \phi_i; \theta_r, \phi_r) = \dfrac{L(\theta_r, \phi_r)}{E(\theta_i, \phi_i)} = \dfrac{\rho_d}{\pi}$

$$L(\theta_r, \phi_r) = \frac{\rho_d}{\pi}\frac{d\phi_i}{dA} = \frac{\rho_d}{\pi}\frac{d\phi_i}{dw}\cos\theta_i = \frac{\rho_d}{\pi} I\cos\theta_i = \frac{\rho_d}{\pi} I\vec{n}\cdot\vec{s}$$

$$E(\theta_i, \phi_i) = \frac{d\phi_i}{dA} \qquad dw = \frac{dA'}{R^2} = \frac{dA\cos\theta_i}{R^2}$$

By conservation of energy,

$$0 \le \int_{\text{hemisphere}} L(\theta_r, \phi_r) = \int_{\text{hemisphere}} \frac{\rho_d}{\pi} I\cos\theta_i\, dw \le I$$

$$\frac{\rho_d}{\pi}\int_{\text{hemisphere}} \cos\theta_i\, dw \le 1 \qquad w = \frac{A}{r^2} \rightarrow dw = \frac{dA}{r^2} = \sin\theta\, d\theta\, d\varphi$$

$$\frac{\rho_d}{\pi}\int_0^{2\pi}\int_0^{\frac{\pi}{2}} \cos\theta\sin\theta\, d\theta\, d\varphi \le 1$$

$$\frac{\rho_d}{\pi}\int_0^{2\pi}\int_0^{\frac{\pi}{2}} \frac{1}{2}\sin 2\theta\, d\theta\, d\varphi \le 1$$

$$\frac{\rho_d}{2\pi}\int_0^{2\pi}\left[-\frac{1}{2}\cos 2\theta\right]_0^{\frac{\pi}{2}} d\varphi \le 1$$

$$\frac{\rho_d}{2\pi}\int_0^{2\pi} 1\, d\varphi \le 1$$

$\therefore \rho_d \le 1$

$\therefore 0 \le \rho_d \le 1$

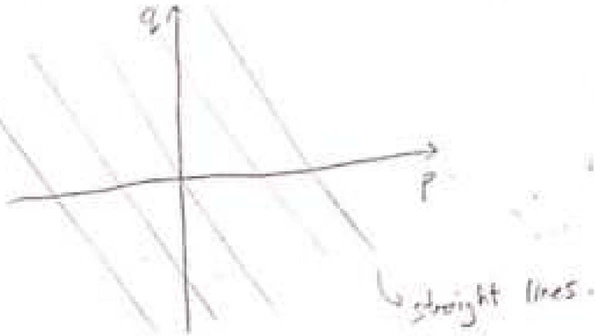
2) Since power coming from a foreshortened area is the same. in this case, because brightness of the surface increases with increasing angle between viewer and surface, we can say that $\text{BRDF} = \dfrac{L(\theta_i, \phi_i)}{E(\theta_i, \phi_i)}$ is proportional to $\dfrac{1}{\cos\theta_r}$ ($\text{BRDF} \propto \dfrac{1}{\cos\theta_r}$). Therefore putting the constant part as 1, we can say that image intensity $I = \dfrac{\cos\theta_i}{\cos\theta_r}$. In surface normal, vector of viewer is $(0,0,1)$ and vector of surface normal is $(p,q,1)$. Therefore

$$\cos\theta_r = \frac{0\cdot p + 0\cdot q + 1\cdot 1}{\sqrt{0^2+0^2+1^2}\sqrt{p^2+q^2+1}} = \frac{1}{\sqrt{p^2+q^2+1}}$$

$$\therefore R(p,q) = \frac{\cos\theta_i}{\cos\theta_r} = \frac{n\cdot s}{\cos\theta_r} = \frac{pp_s + qq_s + 1}{\cancel{\sqrt{p^2+q^2+1}}\sqrt{p_s^2+q_s^2+1}} \cdot \cancel{\sqrt{p^2+q^2+1}} = \frac{pp_s + qq_s + 1}{\sqrt{p_s^2+q_s^2+1}}$$

$R(p,q) = c$

$pp_s + qq_s + 1 = c\sqrt{p_s^2 + q_s^2 + 1}$



3.

$x_i = PX_i$

With p_i being ith row of P, Put p as $p = \begin{pmatrix} p_1^T \\ p_2^T \\ p_3^T \end{pmatrix}$

$p_1^T X_i = w u_i$
$p_2^T X_i = w v_i$
$p_3^T X_i = w$

$\Rightarrow$
$X_i^T p_1 - w u_i = 0$
$X_i^T p_2 - w v_i = 0$
$X_i^T p_3 - w = 0$

$$\underbrace{\begin{pmatrix} X_i^T & 0 & 0 & -u_i \\ 0 & X_i^T & 0 & -v_i \\ 0 & 0 & X_i^T & -1 \end{pmatrix}}_{= A_i\ (3\times 13)} \underbrace{\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ w \end{pmatrix}}_{= p\ (13\times 1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\underbrace{\begin{pmatrix} X_1^T & 0 & 0 & -u_1 & 0 & \cdots \\ 0 & X_1^T & 0 & -v_1 & 0 & \cdots \\ 0 & 0 & X_1^T & -1 & 0 & \\ X_2^T & 0 & 0 & 0 & -u_2 & \\ 0 & X_2^T & 0 & 0 & -v_2 & \\ 0 & 0 & X_2^T & 0 & -1 & \\ \vdots & \vdots & & & & \end{pmatrix}}_{= A} \underbrace{\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ w_1 \\ w_2 \\ \vdots \end{pmatrix}}_{= p} = 0$$

$\min_{\|p\|=1} \|Ap\|$ $\quad A \underset{\text{svd}}{=} U\Sigma V^T$

$\|Ap\| = \|U\Sigma V^T p\| = p^T V \Sigma^T \underbrace{U^T U}_{=I} \Sigma V^T p$
$= \|\Sigma V^T p\|$

$\|V^T p\| = p^T V V^T p = \|p\|$

put $V^T p = y$ Then

$A: m\times n \quad U: m\times m \quad \Sigma: m\times n \quad V: n\times n \quad p: n\times 1$

$\min_{\|p\|=1} \|Ap\| = \min_{\|y\|=1} \|\Sigma y\|$

By SVD, $\Sigma = \begin{pmatrix} \sigma_1 & & & 0 \\ & \sigma_2 & & \\ & & \ddots & \sigma_s \\ 0 & & & \end{pmatrix}$ where $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_s$ are eigenvalues of A.

Therefore to minimize $\|\Sigma y\|$ and $\|y\| = 1$, $y = (0, 0, \cdots 1)^T$

$p = Vy$ Therefore, p is the last column of V corresponding to smallest eigenvalue of A.

4. Hough Transform for Line Detection

2) non maximal suppression in edge detection

In this part, I divided the 2pi into four sections.(0~pi/4, pi/4~pi/2, 2/pi~3pi/4, 3pi/4~pi) This is possible because we look at both direction of gradient direction and its opposite at the same time. I calculated tan(Io) using edge orientation value Io, in order to interpolate between the two pixels the direction is pointing to. Then as I got two interpolated values, I compared them with the pixel edge magnitude Im of the original edge. If any interpolated value is bigger than the edge magnitude of the original, I suppressed the original edge magnitude by putting its value as zero. Otherwise if the edge magnitude is the biggest, which means that it is local maximum, I preserved its original value.

```python
# non maximal suppression
Im_nms = np.zeros(Im.shape)
Im_h, Im_w = Im.shape
for i in range(1, Im_h-1):
    for j in range(1, Im_w-1):
        if (0 <= Io[i, j] < (np.pi / 4) or -np.pi <= Io[i, j] < -(np.pi * 3 / 4)):
            a = np.abs(np.tan(Io[i, j]))
            p = a * Im[i-1, j+1] + (1-a) * Im[i, j+1]
            r = a * Im[i+1, j-1] + (1-a) * Im[i, j-1]
        elif ((np.pi / 4) <= Io[i, j] < (np.pi / 2) or -(np.pi * 3 / 4) <= Io[i, j] < -(np.pi / 2)):
            a = np.abs(1/np.tan(Io[i, j]))
            p = a * Im[i-1, j+1] + (1-a) * Im[i-1, j]
            r = a * Im[i+1, j-1] + (1-a) * Im[i+1, j]
        elif ((np.pi / 2) <= Io[i, j] < (np.pi * 3 / 4) or -(np.pi / 2) <= Io[i, j] < -(np.pi / 4)):
            a = np.abs(1/np.tan(Io[i, j]))
            p = a * Im[i-1, j-1] + (1-a) * Im[i-1, j]
            r = a * Im[i+1, j+1] + (1-a) * Im[i+1, j]
        elif ((np.pi * 3 / 4) <= Io[i, j] <= np.pi or -(np.pi / 4) <= Io[i, j] < 0):
            a = np.abs(np.tan(Io[i, j]))
            p = a * Im[i-1, j-1] + (1-a) * Im[i, j-1]
            r = a * Im[i+1, j+1] + (1-a) * Im[i, j+1]

        if Im[i, j] > p and Im[i, j] > r:
            Im_nms[i, j] = Im[i, j]
        else:
            Im_nms[i, j] = 0
```

Example 4.2.1. Implemented code of non maximum suppression

Example 4.2.2. Before non maximum suppression

Example 4.2.3. After non maximum suppression

3) Hough transform

Example 4.3.1. Im after applying the threshold (threshold = 0.03)

Example 4.3.2. H(rhoRes=1, thetaRes=pi/180)

4) HoughLines

```python
# loop through number of peaks to identify
indicies = []
lRho = []
lTheta = []
H1 = np.copy(H)
for i in range(nLines):
    idx = np.argmax(H1) # find argmax in flattened array
    H1_idx = np.unravel_index(idx, H1.shape) # remap to shape of H
    indicies.append(H1_idx)

    # surpess indicies in neighborhood
    idx_y, idx_x = H1_idx # first separate x, y indexes from argmax(H)
    lRho.append(idx_y)
    lTheta.append(idx_x)
    # if idx_x is too close to the edges choose appropriate values
    if idx_x - (nhood_size_x // 2) < 0:
        min_x = 0
    else:
        min_x = idx_x - (nhood_size_x // 2)
    if (idx_x + (nhood_size_x // 2) + 1) > H.shape[1]:
        max_x = H.shape[1]
    else:
        max_x = idx_x + (nhood_size_x // 2) + 1

    # if idx_y is too close to the edges choose appropriate values
    if idx_y - (nhood_size_y / 2) < 0:
        min_y = 0
    else:
        min_y = idx_y - (nhood_size_y // 2)
    if (idx_y + (nhood_size_y // 2) + 1) > H.shape[0]:
        max_y = H.shape[0]
    else:
        max_y = idx_y + (nhood_size_y // 2) + 1

    # bound each index by the neighborhood size and set all values to 0
    for x in range(min_x, max_x):
        for y in range(min_y, max_y):
            # remove neighborhoods in H1
            H1[y, x] = 0

            # highlight peaks in original H
            if (x == min_x or x == (max_x - 1)):
                H[y, x] = 255
            if (y == min_y or y == (max_y - 1)):
                H[y, x] = 255
```

Example 4.4.1. Implemented code of non maximum suppression

Source :

http://fourier.eng.hmc.edu/e161/dipum/houghpeaks.m

https://gist.github.com/ri-sh/45cb32dd5c1485e273ab81468e531f09

Example 4.4.2 Plotted lines on the original image

rhoRes and thetaRes is needed when deciding the size of the neighborhood(nhood_size) which is a area surrounding the local maximum on Hough space. I suppressed values smaller than the local maximum in the neighborhood.

5) HoughLineSegments

Example 4.5.1. Plotted line segments on the original image