Image Segmentation via Thresholding

In [1]:
```
import numpy as np
import math
import cv2
import matplotlib.pyplot as plt
```

Algorithm Basic Global Thresholding

In [4]:
```
from IPython.display import Image
Image("midean.png")
```

Out[4]:

1. **Input**:
   - Grayscale image (a 2D array of pixel intensities).
   - Initial guess for the threshold value, T_0.
   - Tolerance value to determine convergence.
2. **Initialization**:
   - Set T to the initial guess value T_0.
3. **Iterative Thresholding**:
   - **Repeat until convergence**:
     a. **Partition the Image**:
     - Divide the image into two sets based on the current threshold T:
       - Foreground (F): Pixels with intensities greater than T.
       - Background (B): Pixels with intensities less than or equal to T.

     b. **Calculate Mean Intensities**:
     - Compute the mean intensities of pixels in F and B:
       - Calculate the average intensity of pixels in F (denoted as μ_F).
       - Calculate the average intensity of pixels in B (denoted as μ_B).

     c. **Update Threshold**:
     - Compute a new threshold value T_{new} using:
       $T_{\text{new}} = \frac{\mu_F + \mu_B}{2}$

     d. **Check Convergence**:
     - Calculate the absolute difference between T_{new} and T:
       - If the absolute difference $|T_{\text{new}} - T|$ is less than the tolerance value, stop iterating.

     e. **Update Threshold**:
     - Update T to T_{new}.
4. **Output**:
   - Use the final threshold T to binarize the image:
     - Set pixel intensity to 0 (representing background) if the pixel intensity is less than or equal to T.
     - Set pixel intensity to 255 (representing foreground) if the pixel intensity is greater than T.

Apply Basic Global Thresholding

In [2]:
```
def basic_global_thresholding(img,deltheshold):
    initial_mean = np.mean(img)
    while(True):
        pre_mean = initial_mean
        background = []
        foreground = []
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                if(img[i][j] > initial_mean):
                    foreground.append(img[i][j])
                else:
                    background.append(img[i][j])
        mean_back = np.mean(background)
        mean_fore = np.mean(foreground)
        initial_mean = (mean_back + mean_fore)/2
        if(abs(initial_mean - pre_mean) <= deltheshold):
            break
    return math.ceil(initial_mean)
```

Algorithm Otsu Thresholding

In [6]:
```
from IPython.display import Image
Image("otsu.png")
```

Out[6]:

1. **Compute Histogram**:
   - Calculate the histogram of the input grayscale image. The histogram represents the frequency of each pixel intensity level.
2. **Normalize Histogram**:
   - Normalize the histogram so that each bin value represents the probability density function of the pixel intensities.
3. **Compute Cumulative Sum**:
   - Compute the cumulative sum $\text{cumsum}(i)$ of the normalized histogram up to intensity level i.
4. **Compute Cumulative Mean**:
   - Compute the cumulative mean $\text{cum\_mean}(i)$ of the pixel intensities up to intensity level i:
     $\text{cum\_mean}(i) = \sum_{k=0}^{i} k \times \text{hist}(k)$
5. **Global Mean**:
   - Calculate the global mean intensity μ of the entire image:
     $\mu = \sum_{i=0}^{L-1} i \times \text{hist}(i)$
     where L is the number of intensity levels (typically 256 for an 8-bit grayscale image).
6. **Compute Between-Class Variance**:
   - For each possible threshold t (from 0 to $L-1$):
     a. Compute the weight $w_0(t)$ of the background (pixels with intensity < t):
     $w_0(t) = \text{cumsum}(t)$
     b. Compute the weight $w_1(t)$ of the foreground (pixels with intensity $\geq t$):
     $w_1(t) = 1 - w_0(t)$
     c. Compute the mean intensity $\mu_0(t)$ of the background:
     $\mu_0(t) = \frac{\text{cum\_mean}(t)}{w_0(t)}$
     d. Compute the mean intensity $\mu_1(t)$ of the foreground:
     $\mu_1(t) = \frac{\mu - \text{cum\_mean}(t)}{w_1(t)}$
     e. Compute the between-class variance $\sigma_b^2(t)$:
     $\sigma_b^2(t) = w_0(t) \times w_1(t) \times (\mu_0(t) - \mu_1(t))^2$
7. **Find Optimal Threshold**:
   - The optimal threshold T is the one that maximizes the between-class variance $\sigma_b^2(t)$:
     $T = \arg\max_t \sigma_b^2(t)$
8. **Threshold the Image**:
   - Use the optimal threshold T to binarize the image:
     - Set pixel intensity to 0 (background) if $\text{intensity} < T$.
     - Set pixel intensity to 255 (foreground) if $\text{intensity} \geq T$.

Apply Otsu Thresholding

In [3]:

```
def otsu_thresholding(image):
    histogram, _ = np.histogram(image, bins=256, range=(0, 256))

    # Normalize histogram
    histogram = histogram / float(image.size)

    max_variance = 0
    threshold = 0

    for i in range(1, 256):
        w0 = np.sum(histogram[:i])
        w1 = np.sum(histogram[i:])

        if w0 == 0:
            mu0 = 0
        else:
            mu0 = np.sum(np.arange(i) * histogram[:i]) / w0

        if w1 == 0:
```

```
            mu1 = 0
        else:
            mu1 = np.sum(np.arange(i, 256) * histogram[i:]) / w1

        variance = w0 * w1 * (mu0 - mu1) ** 2

        if variance > max_variance:
            max_variance = variance
            threshold = i

    return threshold
```

In [4]:

```
def segmented_image(img,threshold):
    bin_image = np.zeros_like(img)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if(threshold<img[i][j]):
                bin_image[i][j] = 1

    return bin_image
```

In [5]:

```
img = cv2.imread('CoverImages/lena.tiff',0)
plt.imshow(img, cmap='gray')
plt.title("Real Image")
plt.xticks([])
plt.yticks([])
plt.show()
```

In [6]:

```
threshold_value = otsu_thresholding(img)
threshold_value
```

Out[6]: 118

In [7]:
```
seg = segmented_image(img,threshold_value)
```

In [8]:
```
plt.imshow(seg, cmap='gray')
plt.title("Apply Otsu Thresholding")
plt.xticks([])
plt.yticks([])
plt.show()
```

In [9]:
```
threshold_value = basic_global_thresholding(img,0.01)
seg1 = segmented_image(img,threshold_value)
```

In [10]:
```
threshold_value
```

Out[10]: 118

In [11]:
```
plt.imshow(seg1, cmap='gray')
plt.title("Apply Basic Global Thresholding")
plt.xticks([])
plt.yticks([])
plt.show()
```

In [ ]: