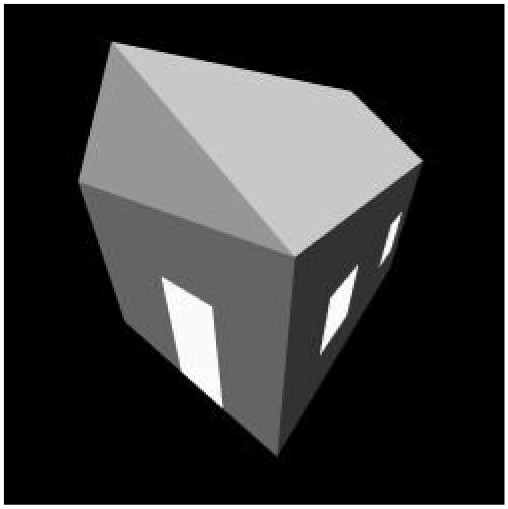
Task 1

Load the images and displaying them

Here are the two images that will be used for task1

```
house_left = imread("../left.jpg");
cameraman = imread("cameraman.tif");
figure;
imshow(house_left);
```



```
figure;
imshow(cameraman);
```

First we generate their histograms

```
house_hist = imhist(house_left);
camera_hist = imhist(cameraman);
```

Then generate their thresholds, first the house:

```
[t_house,gm_house] = otsu_thresh(house_hist);
fprintf("House threshold = %d, goodness = %f",t_house,gm_house);
```

```
House threshold = 4, goodness = 0.855831
```

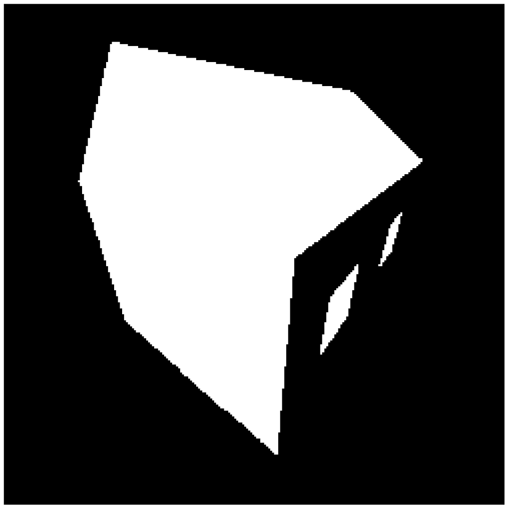
Then do the same for the camera man:

```
[t_camera,gm_camera] = otsu_thresh(camera_hist);
fprintf("Cameraman threshold = %d, goodness = %f",t_camera,gm_camera);
```

```
Cameraman threshold = 82, goodness = 0.846090
```

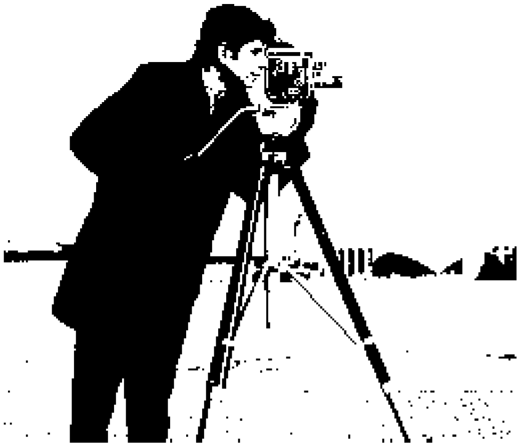
Display the thresholded house

```
figure;
imshow(house_left>t);
```



Display the thresholded cameraman

```
figure;
imshow(cameraman>t);
```



Not sure what to comment on here, the otsu thresh function produces the expected result, it runs in O(n) time which is quite good.

Here is the code used for otsu thresh

```
function [T, GM] = otsu_thresh(h)
    h_orig = h;
```

```
% Inteclass variance maximization
h = h';
% Get the sum of the histogram
MN = sum(h);

% Normalize the histogram
h = (h ./ MN);
% Compute the cumsums
p1 = @(k) sum(h(1:k));
p2 = @(k) sum(h(k + 1:end));

% Compute average intensities
m1 = @(k) sum(h(1:k) .* ((1:k) - 1)) / p1(k);
m2 = @(k) sum(h(k + 1:end) .* ((k + 1:numel(h))) - 1) / p2(k);

% Average intensity up to K
m = @(k) sum(h(1:k) .* ((1:k) - 1));

% Cumulative image intensity
mg = sum(h(1:end) .* ((1:numel(h)) - 1));

% Global variance squared
varg = sum(((((1:numel(h))) - mg).^2) .* h(1:end));

% Variance between classes squared
varb = @(k) ((mg * p1(k) - m(k))^2) / (p1(k) * (1 - p1(k)));

% p1(k) * ((m1(k) - mg)^2) + p2(k) * ((m2(k) - mg)^2);

% Select arbitrary initial k, this might be wrong
k = floor(numel(h) ./ 2);

vars = [];
% Get the best K
for i = 1:numel(h)
    vars = [vars; varb(i)];
end

best = -inf;
best_indecies = 0;

for i = 1:numel(h)

    if vars(i) > best
        % New best value found
        best = vars(i);
        % Only one valid index of this value so far
        best_indecies = i;
    end

    if vars(i) == best
        % append new valid index
        best_indecies = [best_indecies; i];
    end

end
% Handle multiple best indecies
```

```
        index = sum(best_indecies) / numel(best_indecies);
        % Goodness of the k choise
        GM = best / varg;
        % Best thersholding point
        T = h_orig(index);

    end
```

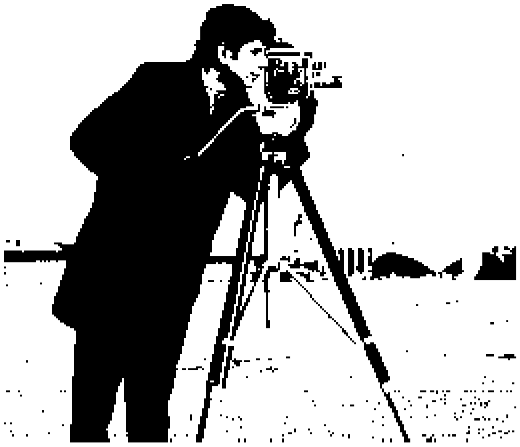
Comparing to the matlab function

Calling the function according to the exmaple provided in the graythresh function

```
threshold = graythresh(cameraman);
BW = imbinarize(cameraman,level);
fprintf("Matlabs cameraman threshold = %d",floor(threshold*255));
```

```
Matlabs cameraman threshold = 88
```

```
figure, imshow(BW)
```



The matlab "otsuthresh" function is a lot more efficient than my implementation. It seems to use bucket sort idea's, and is verry heavely inspired by the original paper.