Image Processing Lab2

Name: 徐竣霆

Student ID: 110060012

Proj03-01: Image Enhancement Using Intensity Transformations

Result

Original

Original Image

logTransform

Log Transform

powerlawTransform(r = 0.2)

powerlawTransform(r = 0.4)

powerlawTransform(r = 0.5)

powerlawTransform(r = 1)

powerlawTransform(r = 2)

Power Law Transform, r = 2

powerlawTransform(r = 3)

Power Law Transform, r = 3

powerlawTransform(r = 5)

Implementation

logTransform

```
function output = logTransform(input, c)
    output = c * log(1 + input);
    output = output / max(output(:));
end
```

Based on ch3(p6). Divided by maxvalue will automatically normalize the range to $[0, 1]$.

powerlawTransform

```
function output = powerlawTransform(input, c, r)
    output = c * input .^ r;
    output = output / max(output(:));
end
```

Based on ch3(p8). Divided by maxvalue will automatically normalize the range to $[0, 1]$.

Discussion

I actually don't know whether c should stay 1 all the time, since all of the cases I have seen in slides is 1. And since we have to normalize the range of intensity to $[0, 1]$ under all scenarios, it seems c is useless(I'm not sure whether I have misunderstood something). As for the root causes for the differences, logTransform tends to brighten darker regions(due to the characteristic of log function). On the otherhand, for powerlawTransform, if $\gamma < 1$, it maps a narrow range of dark values into a wider range; if $\gamma > 1$, it maps a narrow range of bright values into a wider range.

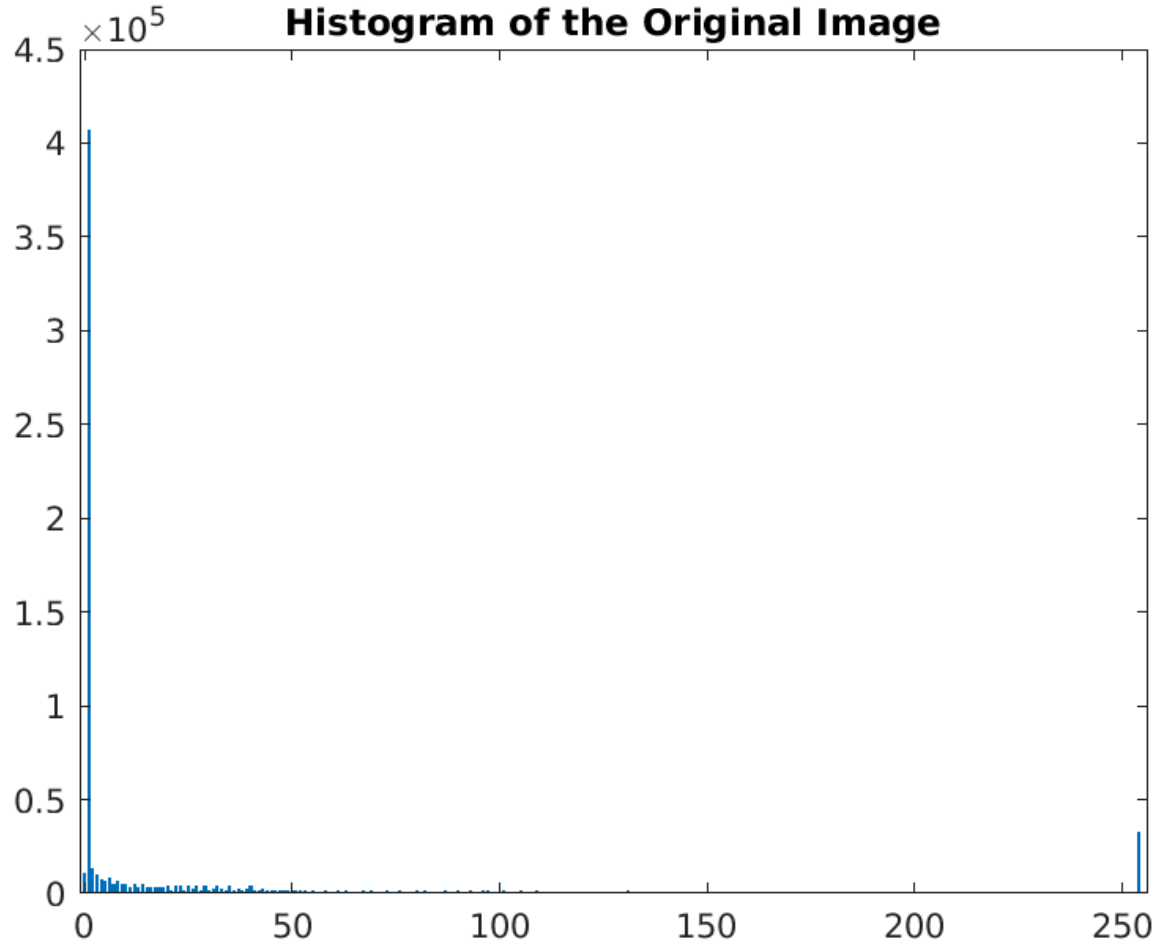
Proj03-02: Histogram Equalization

Result

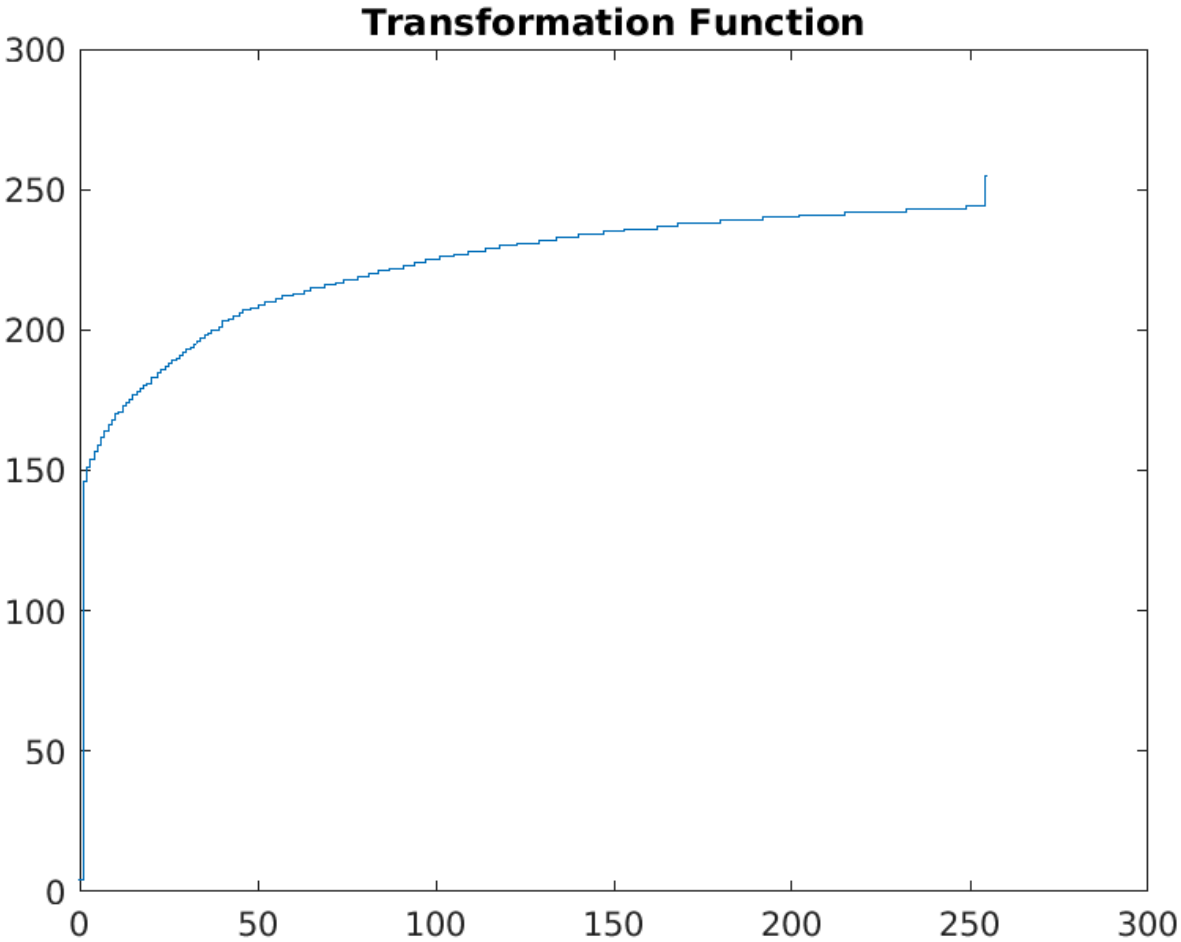
Original

Original Image

Histogram of Original

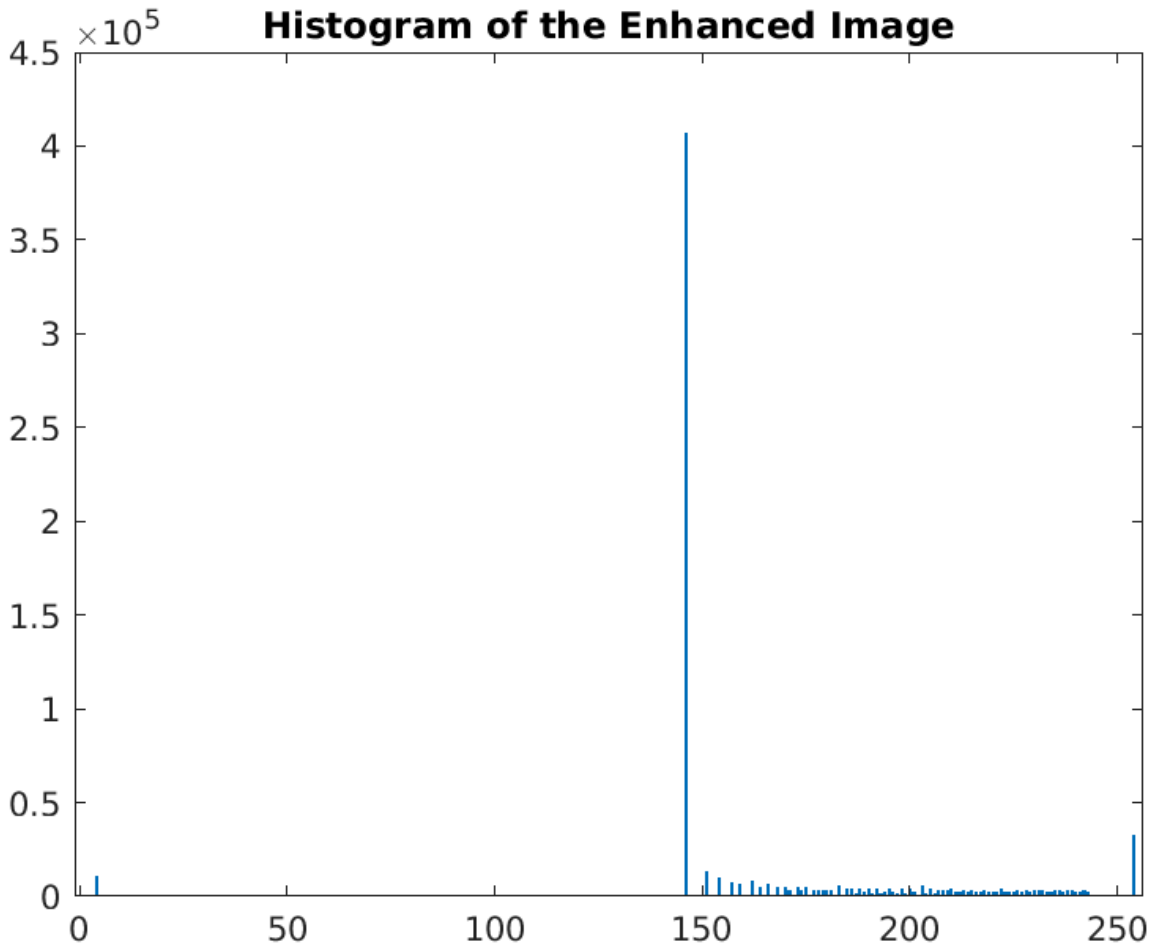


Transformation Function



Enhanced

Histogram of Enhanced



Implementation

imageHist

```
function histVector = imageHist(input)
    histVector = single(zeros(1, 256));
    for i = 1:size(input, 1)
        for j = 1:size(input, 2)
            histVector(input(i, j) + 1) = histVector(input(i, j) + 1) + 1;
        end
    end
end
```

histEqualization

```
function [output, T] = histEqualization(input)
    % Get the histogram of the input image
    histVector = imageHist(input);

    % Get the cumulative distribution function of the histogram
    cdf = cumsum(histVector); % cumsum is a great function

    % Normalize the cdf
    cdf = cdf / cdf(end);

    % Get the transformation function
    T = round(255 * cdf);
    T = uint8(T);

    % Apply the transformation function to the input image
    output = T(input + 1);
end
```

cumsum is a great function, which helps me to compute the prefix sum array.
Since we know $M \times N$ equals to the total number of pixels, i.e the last item in prefix sum array, divide by it and scale by maximum intensity. And then the mapping is quite straightforward.

Discussion

The Histogram Equalization maps a very narrow interval of dark pixels into the upper half of the gray scale. In this picutre, more than half of total region is dark and it's trivial(let's call it background). Such measure makes noise more conspicuous, which is what we do not want. However, it also makes the originally implicit part(left part of the spine) become more explicit, which is exactly what we want.

Proj03-03 ~ 03-04: Spatial Filtering, Enhancement Using the Laplacian

---

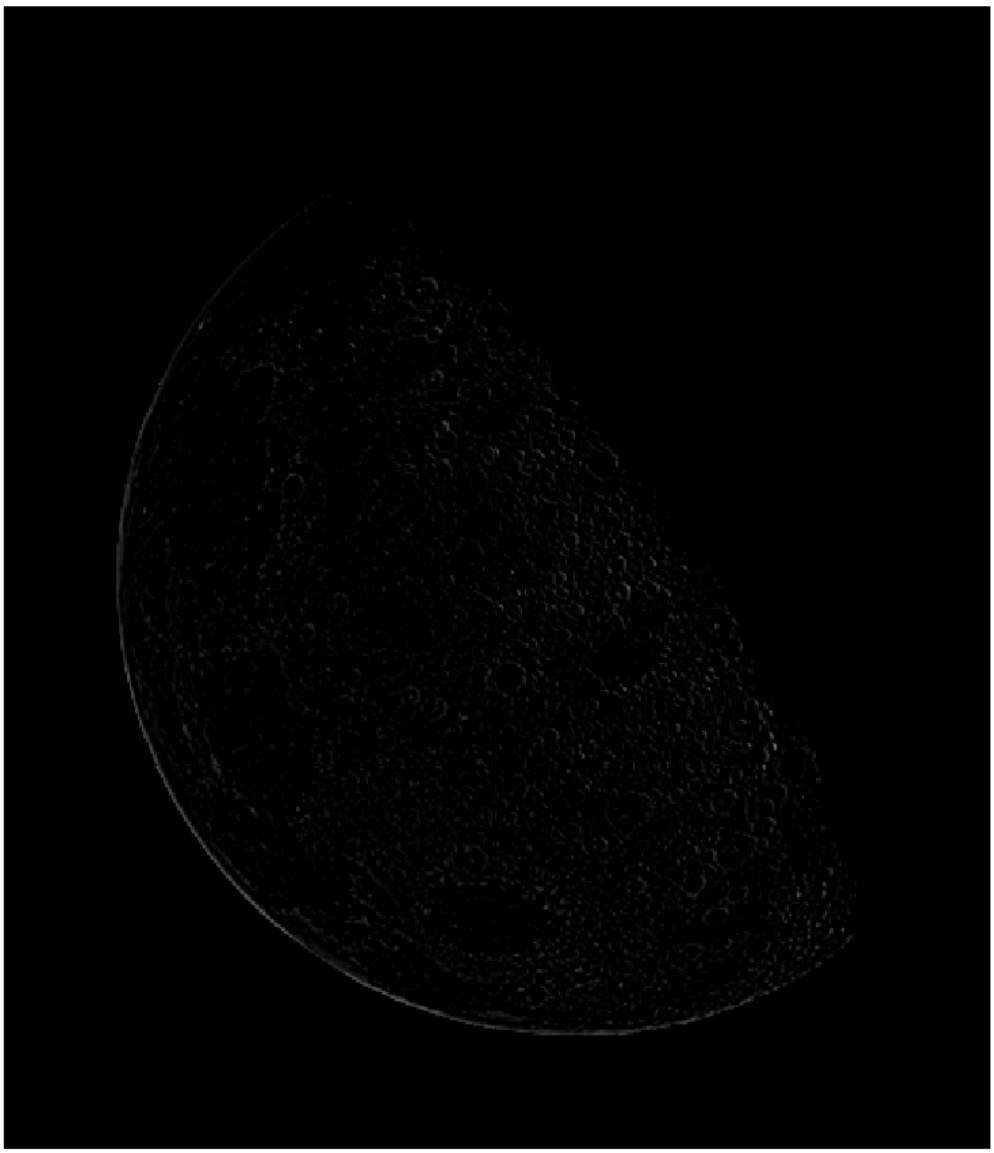
Result

Original

(a)

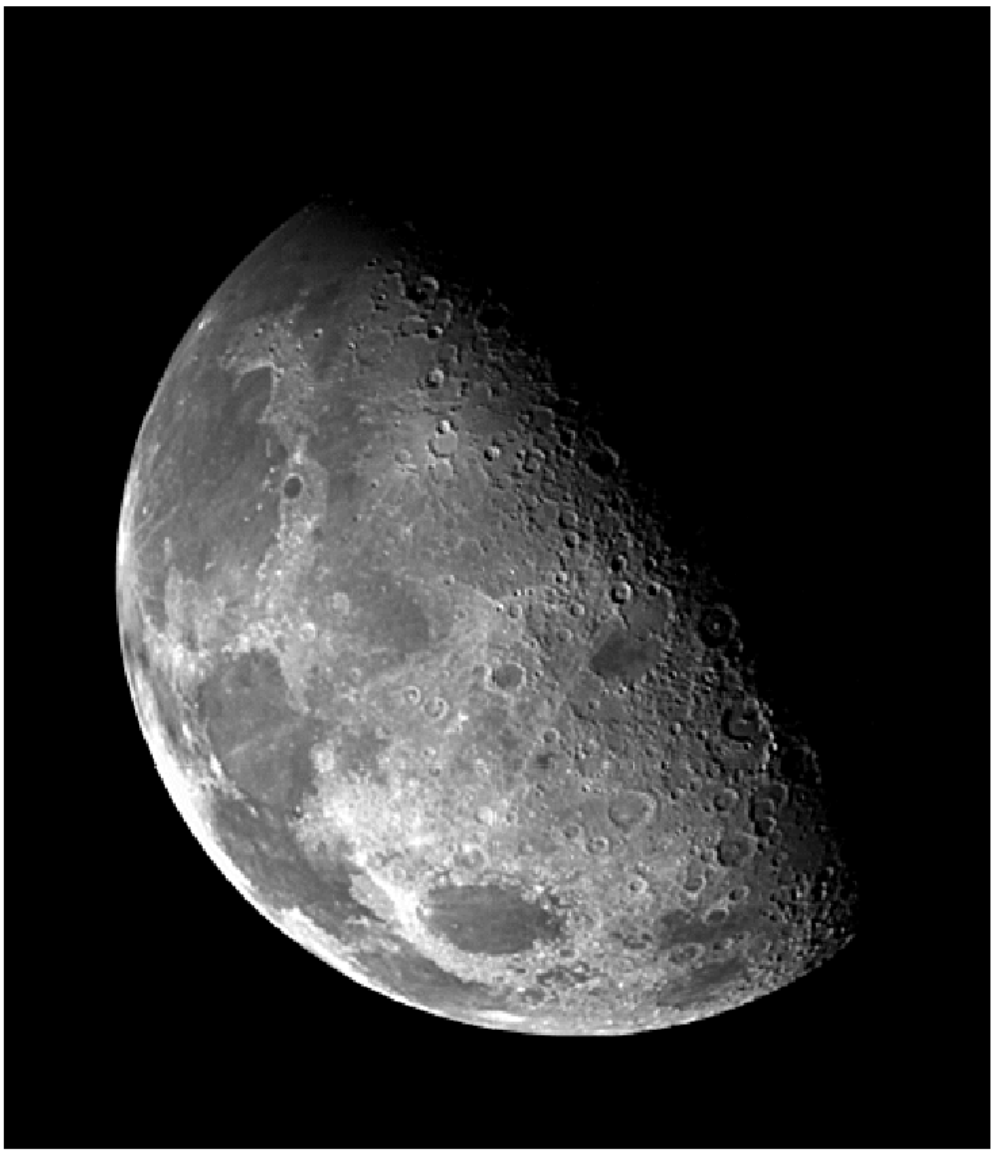
scaledLaplacian

(b)



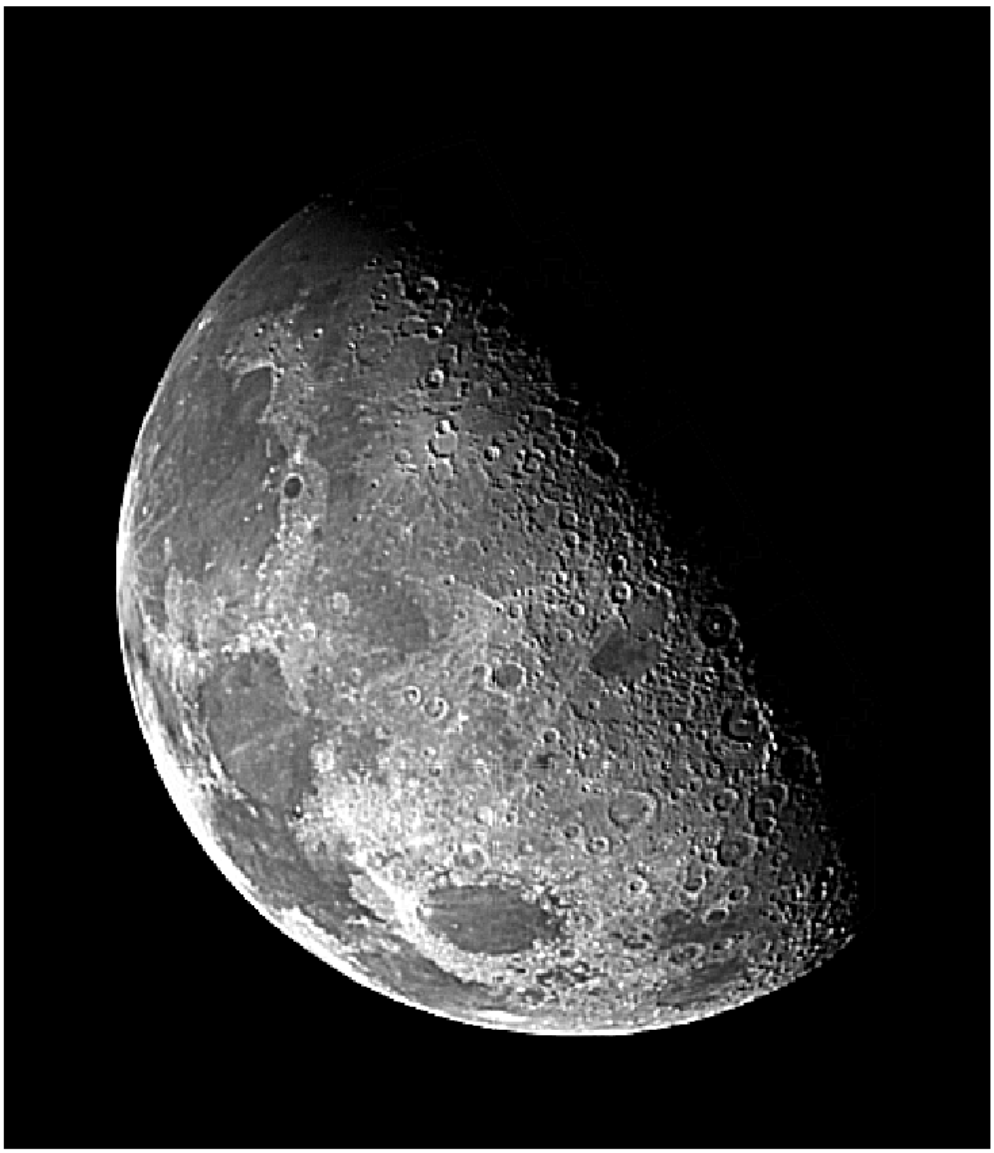
Enhanced Image

(c)



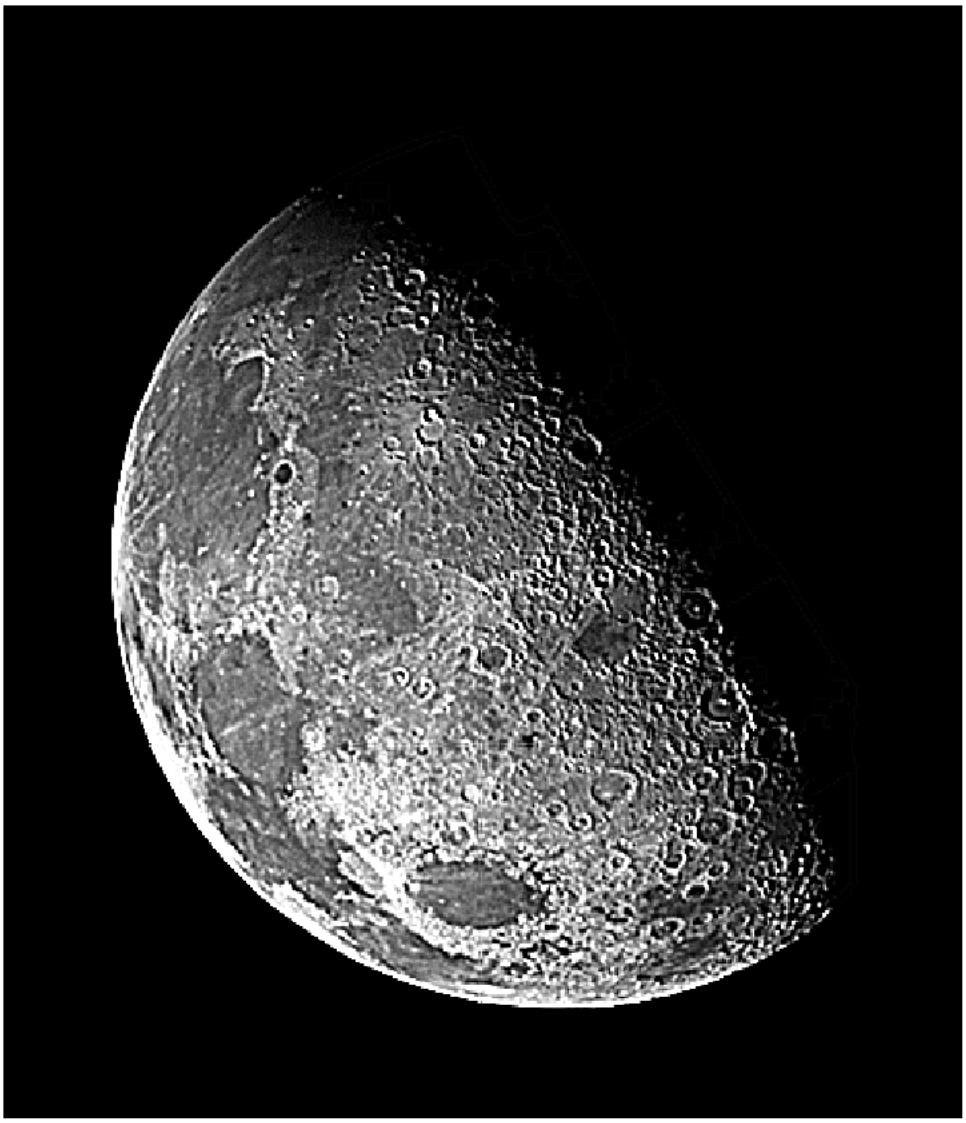
Enhanced Image Variant 1(-8)

(d)



Enhanced Image Variant 2(5 x 5)

(e)

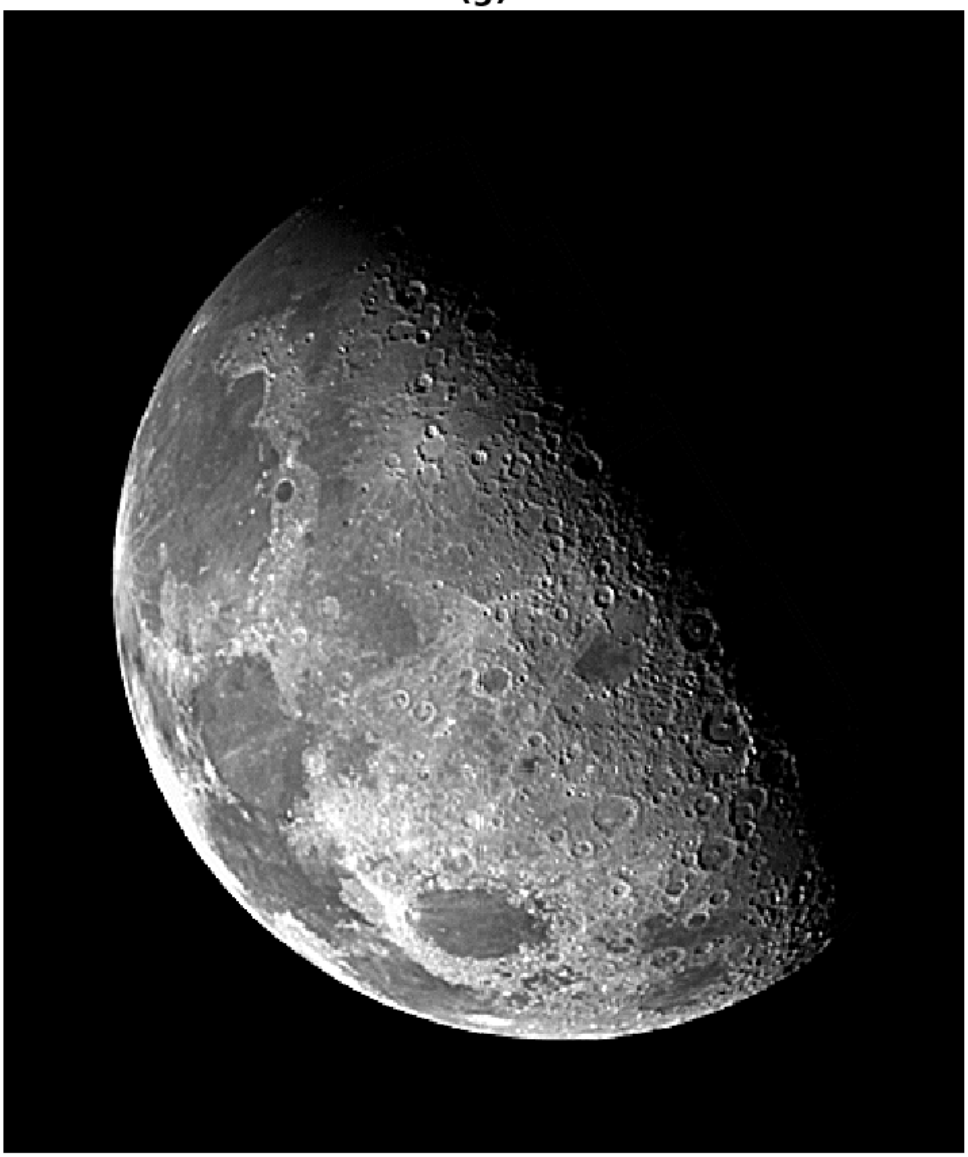


Enhanced Image Variant 3(scale = -0.5)

(f)

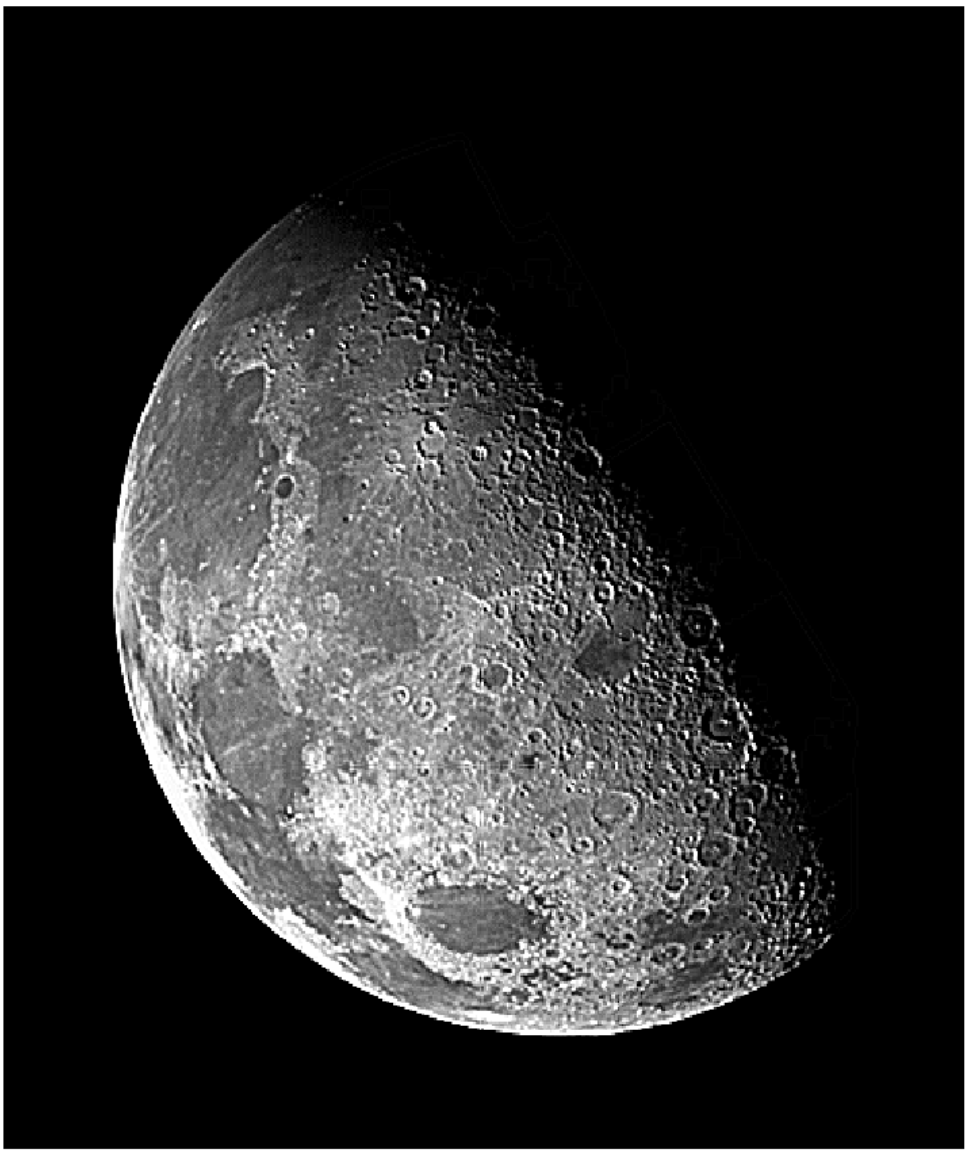
Enhanced Image Variant 4(scale = -2)

(g)



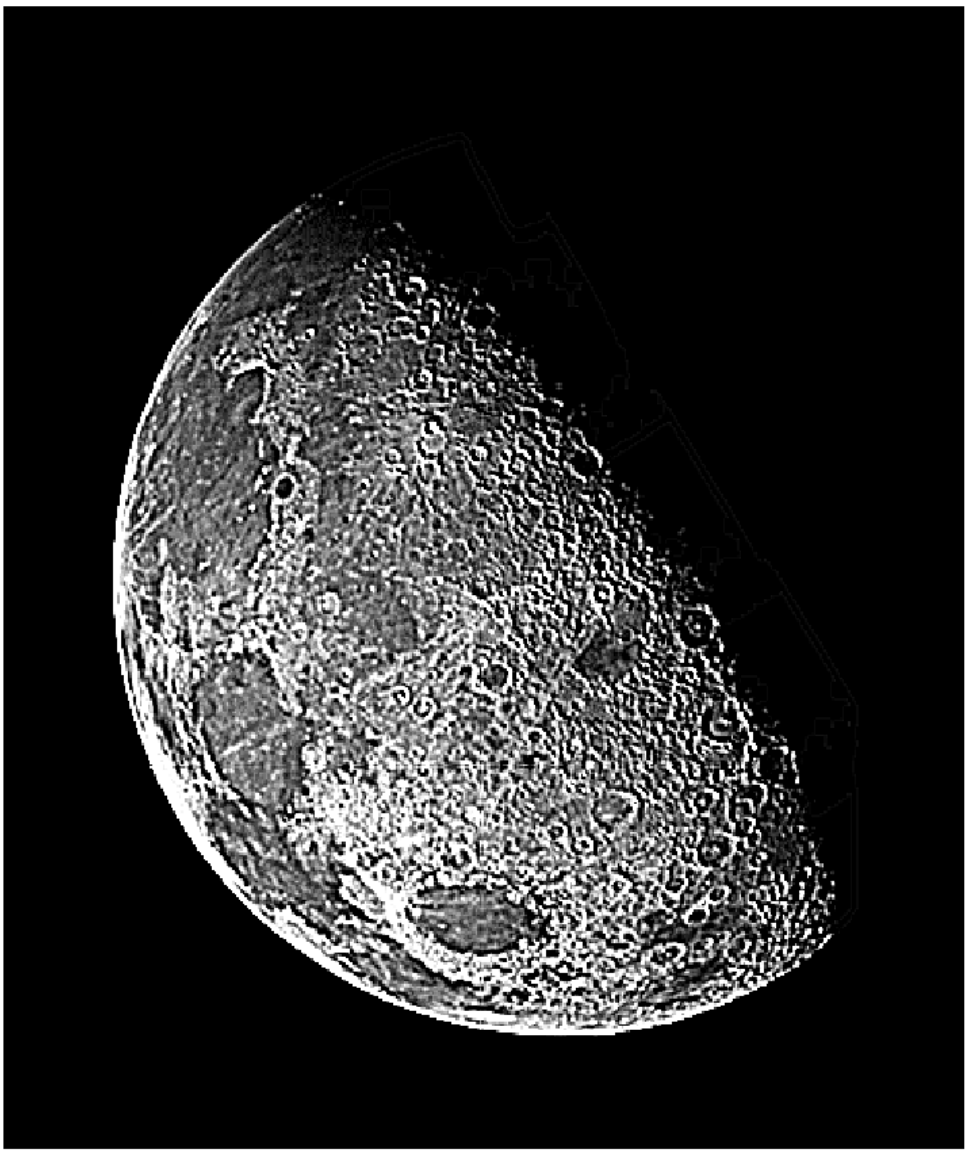
Enhanced Image Variant 5(scale = -4)

(h)



Enhanced Image Variant 6(5 x 5, scale = -2)

(i)



Enhanced Image Variant 7(5 x 5, scale = -4)

(j)

Implementation

spatialFiltering

```
function output = spatialFiltering(input, mask)
    % Get the size of the input image
    [M, N] = size(input);

    % Get the size of the mask
    [m, n] = size(mask);

    % Calculate the padding sizes
    paddingX = floor(m / 2);
    paddingY = floor(n / 2);

    % padding with zeros
    paddedInput = padarray(input, [paddingX, paddingY], 0, 'both');

    % Initialize the output image
    output = zeros(M, N, 'single');

    % Perform spatial filtering
    for i = 1:M
        for j = 1:N
            % Extract the region of interest
            roi = paddedInput(i:i + m - 1, j:j + n - 1);

            % Perform convolution
            output(i, j) = sum(roi(:) .* mask(:));
        end
    end

end
```

Since the mask size usally odd numbers, the corresponding padding will be floor(n / 2). After deciding the padding policy, implement the convolution.

laplacianFiltering

```
function [output, scaledLaplacian] = laplacianFiltering(input, laplacianMask, s
    scaledLaplacian = scale * spatialFiltering(input, laplacianMask);
    output = input + scaledLaplacian;
end
```

Discussion

Since the borders of this image are all dark(value = 0), using no matter "pad with zeros", "mirroring", "circular padding" will leads to the same result. Also, we can see that larger mask size leads to more enhancement. Similarly, the smaller(negative number) scale leads to larger enhancement.