Part 3: Multi-Scale Edge Detection

```
images = ["../images/CARTOON.jpg", "../images/flowergray.jpg", "../
images/kitty.jpg", ...
    "../images/polarcities.jpg", "../images/text.jpg" ];

filter = [0.25,0.25; 0.25,0.25];
secondDerivative = [-0.125, -0.125, -0.125; -0.125, 1, -0.125; -0.125,
 -0.125, -0.125];

threshold = 5;

for image = images
    % Load image
    img = double(imread(image));
    [height_G, width_G] = size(img);
    side = log2(height_G);

    % Generate the second order derivative images
    temp = imfilter(img, secondDerivative, 'replicate', 'same');
    temp(temp>0) = 1;
    temp(temp<=0) = 0;

    % Plot image
    figure;
    subplot(3, 3, 1);
    imshow(temp);
    title('Original image')

    for i= 1:side
        img = imfilter(img, filter, 'replicate', 'same');
        % Half size
        [height, width] = size(img);
        img = img(1:2:height, 1:2:width);

        secondDer = imfilter(img,
 secondDerivative, 'replicate', 'same');
        secondDer = imresize(secondDer, 2^i, 'bilinear');

        % Detect the zero crossing in the segmented image.
        segment = secondDer;
        segment(secondDer>0) = 1;
        segment(secondDer<=0) = 0;

        % Get all neighbors
        cross = zeros(size(segment));
        for j = 1: height_G
            for k = 1:width_G
                neighbors = getNeighbor(segment, j, k);
                if (~all(neighbors == segment(j, k)))
                    cross(j, k) = 1;
                end
```

```
            end
        end

        localVariance = lclvr(secondDer, cross);

        edgeImage = ones(size(localVariance));
        edgeImage(localVariance > threshold) = 0;

        % Plot nth time result
        subplot(3, 3, 1 + i);
        imshow(edgeImage);
        title(sprintf('%d times filtering', 2^i));
    end
end

function localVariance = lclvr(image, cross)
    [height, width] = size(image);
    localVariance = zeros([height width]);
    for i = 1: height
        for j = 1: width
            if (cross(i, j) == 0)
                continue
            end
            % Compute neighbor's local variance
            neighbors = getNeighbor(image, i, j);
            localVariance(i, j) = var(neighbors(:));
        end
    end
end

function neighbors = getNeighbor(image, i, j)
    index = 0;
    [height, width] = size(image);
    neighbors = ones(1, 9) * image(i, j);
    for x = -1:1
        for y = -1:1
            index = index + 1;
            % Only for inbound elements
            if (inbound(i, j, x, y, width, height) == 0)
               continue
            end
            neighbors(index) = image(i + x, j + y);
        end
    end
end

function isInBound = inbound(i, j, x, y, width, height)
    isInBound = (i + x <= height && j + y <= width && i + x > 0 && j +
 y > 0);
end
```

Original image

2 times filtering

4 times filtering

8 times filtering

16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering

Original image

2 times filtering

4 times filtering

8 times filtering

16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering

Original image

2 times filtering

4 times filtering

8 times filtering

16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering

Original image

2 times filtering

4 times filtering

8 times filtering

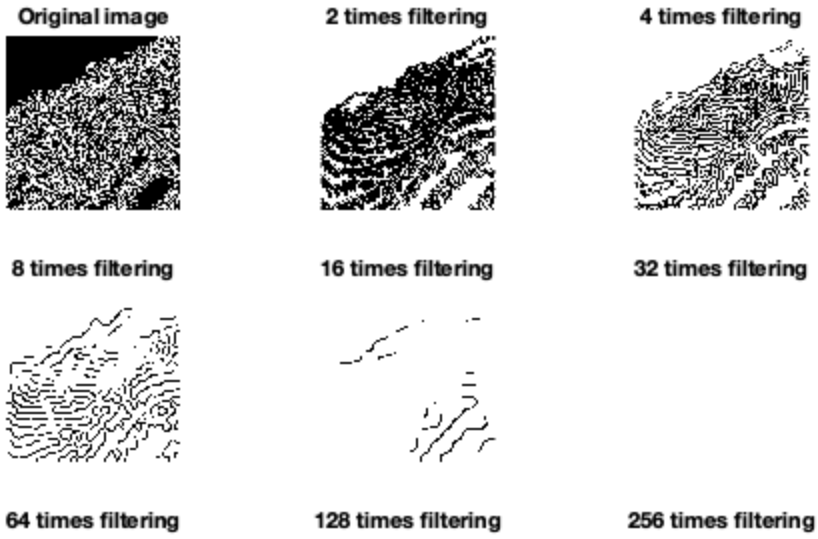
16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering



Published with MATLAB® R2019b