1 Overview

In the lab we implemented the paper [Diamant, 2005], where an innovative top-down image segmentation is introduced. Unlike traditional top-down approaches, where user interaction is required, this approach uses coarse downsampled image segmentation as cues for image segmentation. Realizing this is a good chance to deploy the clustering segmentation approach to the coarse image, we decided to implement this framework.

2 Structure

seg.m
This is the main script for the lab.

1. Take in an image (image_name in line 5).
2. Converts it to grayscale and runs image pyramid (shrink(...)).
3. Performs segmentation on the coarse image (spectral_clustering(...) or simple_segment(...)).
4. For each level down in the pyramid, perform expand(...) on the image, which goes form the previous level segmentation to the next level.

- output_image = shrink(image, scale)
  The function shrinks the image. Given an image and the scale, the image is divided into small windows of scale * scale. Each window becomes a pixel of the next level image by averaging. The resulted image is scaled down by scale in both height and width.
• \( \text{labels} = \text{spectral\_clustering}(\text{image}, k, r, \sigma_1, \sigma_2) \)
  This function performs spectral clustering where \( k \) is the number of cluster in k-means++ and \( r \) is the number of eigen value used. Two sigmas are for the weights of intensity and distance. The output of the function is a matrix of the same size as image, where each entry is an integer between 1 to \( k \), specifying which cluster (segment) that pixel belongs to.

• \( \text{labels} = \text{simple\_segment}(\text{image}, \text{threshold}) \)
  This function performs a simple segmentation on image by breadth-first search. Starting at any unlabeled pixel, BFS is started on the pixel. Enqueue any pixel to the neighbor of the front pixel in the queue as long as it is within an intensity difference of \( \text{threshold} \). The output is in the similar format to \text{spectral\_clustering}(...).

• \( \text{labels} = \text{do\_kmeanspp}(\text{image}, k) \)
  This function performs a segmentation using k-means++. It makes use of only intensity information, disregarding the distance.

• \( \text{new\_labels} = \text{cleanup\_labels}(\text{labels}) \)
  This function cleans up the labels. Some segmentation methods, like kmeans or spectral clustering, the segments are not necessary connected. This function breaks such segments into two.

• \([\text{segimg}, \text{pos\_labels}] = \text{compute\_avg\_intensity}(\text{labels}, \text{image})\)
  This function computes a segmented image for display. \text{segimg} is an image where each pixel is colored in the average color of its segment. \text{pos\_labels} is \( k \) cell, where \( k \) is the number of segments, where each cell stores the list of the positions of the pixels in the segment.

\((\text{segimg}, \text{labels}) = \text{expand}(\text{segimg\_prev}, \text{labels\_prev}, \text{image\_prev}, ... \text{image}, \text{scale}, \text{free\_pix\_thres}, \text{new\_seed\_thres}, \text{grow\_seg\_thres}, \text{min\_size\_thres})\)
This is the function for expanding in the image pyramid. Given segmented image (generated with \text{compute\_avg\_intensity}(...)), labels and the image of the previous level. It goes through the following pipeline.

1. Free the pixels on the segment boundaries for relabeling later by \text{free\_pixels}(...).

2. Grow the segments with \text{grow\_segments}(...). The segments are trying to regain pixels in a BFS manner. New segments would be generated if a pixel is much too difference to the segment that is trying to recruit it, in which case \text{grow\_seed}(...) is called to grow the new segment.

3. Free the pixels that belong to segments that are smaller than \text{min\_size\_thres}.

4. Grow the segments again with \text{grow\_segments}(...) without generating new segments.

• \([\text{free\_pix}, \text{labels}] = \text{free\_pixels}(\text{image}, \text{image\_prev}, \text{labels\_prev}, \text{segimg\_prev}, \text{scale}, \text{free\_pix\_thres})\)
  The function frees the pixel on segment boundary, and the pixels who is
too different (difference larger than free_pix_thres) from its pivot (the average intensity of the four pixels in the scale * scale window). The output free_pix is a Boolean matrix where each entry is \{0,1\} denoting whether the pixel is freed. labels is the same segmentation labels as labels_prev other than the freed pixels being labeled as 0.

- **new_labels = grow_segments(labels, free_pix, image, grow_seg_thres, new_seed_thres, allow_seed)**
  This function grows the segment, given the labels (freed pixels are marked as 0). It starts in BFS manner, where all the labeled pixels with freed neighbors are enqueued at first. Then while processing the queue, enqueue the unlabeled pixels that are within an intensity difference grow_seg_thres. If an unlabeled pixel falls out of the intensity range, grow_seed(...) is called to grow a new segment.

- **labels = grow_seed(x, y, labels, image, free_pix, new_seed_thres)**
  This function grow a new segment, given the seed position \((x, y)\) and the labels. The new segment grows in the same manner in BFS. It takes all freed pixels as possible candidates (not only the ones unlabeled).

### 3 Summary

We implemented the whole framework and produced nice result. It helps students understand the concept of image pyramid, which is widely used in computer vision. In the lab questions, the students are asked to implement spectral clustering and one of BFS or kmeans clustering. All the Matlab codes are implemented by our group, except the kmeans++ algorithm, which comes from the moodle site.

### References