1 Problem Statement

In this assignment we are trying to convert an input photograph and process it to output an image which looks like a painterly rendered or non-photorealistic image of the scene in the original photograph.

2 Painting Philosophy

In this section we would consider how does a photograph differ from an image in terms of various digital image properties.

The most obvious thing is the restriction of color palette. A photograph, depending on the device may have thousands of different colour shades, whereas, in a painting we expect to see far less number of colors.

The second thing that we would consider is the lack of noise in a painting. A painting created by strokes is not expected to have

3 Solution Approach

3.1 Pre-Processing

As discussed in the section [painting philosophy] a painterly rendered image is expected to negligible Salt and Pepper Noise and it is expected to be more uniform. Hence, before applying operations on the input image we apply median filter to remove noise. We have used the MATLAB function medfilt2 for this purpose. The critical parameter in applying median filter is the m-by-n neighbourhood/window for applying the filter. A large value may induce blurring in the image owing to loss of color information.

Another reason of applying median filter was to give an initial slightly flat look to the image as the painterly rendered image is not expected to have significant color gradient except at the edges.

The subjective optimal choice of the filter window varied from image to image. Application of large window on images with large amount of detail distorted images with more details by blurring.

3.2 Color Quantization

A painter’s palette is expected to have far less color than a photograph taken using a device supporting large number of colors. So an important part of creating a painterly rendered image is to restrict the number of colors in the image and still retaining the essential image details.

Restricting the palette (Color Quantization) may lead to certain artifacts such as contouring in the resultant image. Dithering can be used to represent the image using the reduced color palette and still give the visual effect of having more details there by reducing the effect of contouring.

3.2.1 Median Cut and Dithering

In this assignment we have used Median Cut algorithm to perform color quantization. We have worked in the RGB space. The RGB color cube is populated with the frequency data for each tuple (r,g,b).

For generating a quantization resulting in the restriction of the number of colors to K values we iteratively divide the color cube according to the following algorithm.
We first shrink the color cuboid to tightly fit the non-zero population values in the cube. Then we iterative choose the box with the longest edge and divide it into two parts at the median of the longest edge. We do this repeatedly K-1 times to get K color boxes.

The algorithm works on the assumption that dividing a color cuboid along the longest dimension at the median point divides it into two cuboids of approximately equal number of points.

In the end we find the mapping of the old (r,g,b) values to new (r,g,b) values using the nearest neighbour algorithm. Floyd-Steinberg dithering is combined in this process of mapping pixel to new (r,g,b) value by propogating the error caused by the quantization along the image.

### 3.2.2 Segmentation

We use the Quad Tree based segmentation, where in we first segment the image based on dissimilarity and then merge regions based on similarity. We do this segmentation in the gray scale image. Regions are split based on the difference between maximum and minimum intensity exceeding a threshold, and are merged based on the mean and standard deviations of the regions being merged. On obtaining segments, we color them using the average colors in the regions.

### 3.3 Edge Enhancement

We do a canny edge detection, curve fitting, and differential thickening of edges as part of edge enhancement. Following are brief details about each.

#### 3.3.1 Edge Detection

We use canny edge detector on the intensity image of one fourth the original size, obtained using the impyramid function in matlab. We do this to obtain better more linked edges and also to speed up the following steps.

#### 3.3.2 Curve Fitting

1. Detection of edge end points - We do a mask operation, to detect edge end points.
2. Delinking crossing edges - We design another mask to delink crossing edges. This is essential for the next curve fitting step.
3. Edge traversal to obtain ordering of points (as distance from one of the end points)
4. Parametric curve fitting using distance from end point as a parameter. We use csaps curve fit in matlab to fit a smooth curve through the points obtained.

#### 3.3.3 Differential Thickening

Drawing of the edge, using the fit curve in original scale. Differential thickening by painting extra pixels perpendicular to the gradient at each pixel.

### 3.4 Post Processing

Once we apply dithering to remove contouring caused by color quantization the image seems dotted. This is not expected from a painting painted using smooth strokes. Hence we apply a low pass filter wiener2 (MATLAB). Wiener filter is an adaptive noise removal filter and it causes far less blurring that a median filter.

### 4 Results