Programming Assingment 2:

Due on April 11 @ 4:00pm

CS 685/485 – Computer Vision

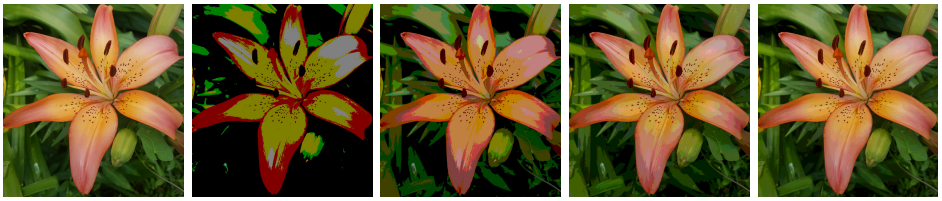
Contents

Problem 1

[60 pts] In this problem you will investigate the use of color spaces and their application in color compression. You will realize a by-product of color compression usually referred to as toon-shading or cell-shading. Toon shading is a technique in computer graphics that utilizes a quantized color space in order to deliver a shading technique that represents cartoon-like colors.

[30 Pts] Toon Shading

Figure 1 shows how toon-shaded images will differ from their original versions. Figure 1(a) shows the original image file. Toon-shaded images with 2, 4, 8, and 16 distinct colors based on the original color image are shown in Figure 1(b)-Figure 1(e). In this part of the project, you will implement an algorithm to perform toon shading based on an input image.



(a) (b) (c) (d) (e)

Figure 1: Toon Shading Representation of an image. (a) The Original Image. (b)-(e) Toon shaded image with 2, 4, 8, and 16 color-levels.

Toon Shading Parameters

Your algorithm will take in one input image and one parameter as the shading levels as shown in Algorithm 2. This parameter indicates the number of color levels in each color channel in the resultant toon shaded image. For example a parameter value of 2 will result in the toon shaded image to be comprised of 2 distinct values for each color channels (Figure 1(b)), while, a parameter value of 16 will result in 16 distinct values in each color channel (Figure 1(e)).

Algorithm 1: Toon Shading Algorithm

Data: I : Input-Image, $F_l = 256$: Input-Channel-Max, L : Toon-Levels.
Result: O : Toon-Shaded-Image.
begin
  $I \leftarrow$ Read Input Image File.
  $L \leftarrow$ Set Toon Levels.
  $O \leftarrow$ Quantize-Channels(I, F_l, L).
  Show-Image(I)
  Show-Image(O)
  $O \rightarrow$ Save Output Image File.

Quantize-Channels(O, I, F_l ,L) `/* Quantization of Colors */`
begin
  for *each pixel I(i,j)* **do**
    for *each channel* I_c*(i,j)* **do**
      O_c(i,j)$\leftarrow$ Quantized I_c(i,j) by L from F_l original levels.

Your Task is to implement this algorithm in the appropriate color space. Note that you will quantize the color channels by integer division. In essence, you will have to take each original input color value ranging

Problem 1 continued on next page...

from 0 to 255, and quantize it to the targeted number of color levels. For example, if you are using 2 as number of color levels, all values between 0 to 127 will be represented as 0, and all values between 128 to 255 will be represented as 128.

[30 Pts] Color Compression

As you noticed toon-shading acts as a color compression technique. For example, if we utilize 8 toon-levels, we can potentially reduce the number of colors from $256 \times 256 \times 256$ or 168 million colors to 512 colors. This is a significant saving in storage space. However, directly reducing the number of colors also impacts the visual quality of the image (see Figure 1). Preserving the quality of the image while reducing its size is possible in other color spaces. Figure 2 shows how color compression could result in preserving the details from original images, even after significant compression levels, if an appropriate color space is used. Figure 2(a) shows the original image file while the compressed images with 2, 4, 8, and 16 distinct colors based on the original color image are shown in Figure 2(b)-Figure 2(e). In this part of the project, you will implement an algorithm to perform color compression based on an input image.

(a) (b) (c) (d) (e)

Figure 2: Color Compression of an image. (a) The Original Image. (b)-(e) Color-Compressed image with 2, 4, 8, and 16 color-levels.

Compression Parameters

Your algorithm will take in one input image and one parameter as the color-compression value, as shown in Algorithm 2. This parameter indicates the number of color levels in the resultant compressed image. For example a parameter value of 2 will result in the image to be comprised of 2 distinct colors (Figure 2(b)), while a parameter value of 16 will result in 16 distinct colors (Figure 2(e)).

Algorithm 2: Color Compression Algorithm

Data: I : Input-Image, $F_l = 256$: Input-Channel-Max, L : Color-Levels.
Result: O : Color-Compressed-Image.
begin
  $I \leftarrow$ Read Input Image File.
  $L \leftarrow$ Set Color Levels.
  $O \leftarrow$ Convert-Color-Space-and-Quantize(I, F_l, L).
  Show-Image(I)
  Show-Image(O)
  $O \rightarrow$ Save Output Image File.

Convert-Color-Space-and-Quantize(O, I, F_l ,L) `/* Color Conversion and Compression */`
begin
  tmp ←Convert-Color-Channel(I)
  for *each pixel tmp(i,j)* **do**
    for *appropriate color channel* tmp_c*(i,j)* **do**
      O_c(i,j)← Quantized tmp_c(i,j) by L from F_l original levels.

Your Task is to implement this algorithm in the appropriate color space. Note that you will quantize the appropriate color channel(s) by integer division. In essence, you will have to take each original input color value ranging from 0 to 255, and quantize it to the targeted number of color levels. For example, if you are using 2 as number of color levels, all values between 0 to 127 will be represented as 0, and all values between 128 to 255 will be represented as 128.

Deliverables

a) An electronic (on Canvas) report describing your results. The report must include the following:

Toon Shading :

- Your approach in toon shading, broken down in steps.
- Explain which color space you used.
- Explain the equation(s) used to quantize the color channels.
- Your results for toon shading.

Color Compression :

- Your approach in color compression, broken down in steps.
- Discuss which color space used, and why.
- Discuss which color channels were used for compression, and why.
- Your results for color compression.

Comparison :

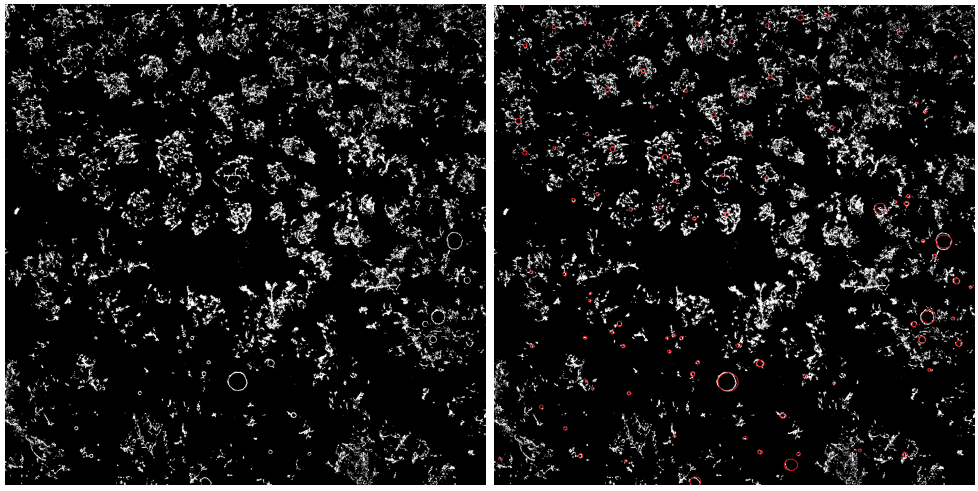
- A side-by-side comparison of the results of toon shading and color compression results.
- A discussion of these results and the comparisons in terms of quality and size.

b) One ZIP file containing the following:

- The source code files in C/C++. Make sure the source code is fully documented.
- A README file with instructions on how to compile and run the program and any parameters that need to be set.

Problem 2

[40 pts] As discussed in class, the Hough Transform is an elegant technique to detect representatives of any structure for which a parametric or descriptive model can be derived. In this problem you will implement a Hough Transform technique to detect tree trunks in an image generated from a slice of point cloud data acquired by scanning a forest using an advanced LiDAR system.



(a) (b)

Figure 3: Hough Transform application in detecting trees in a scan of a forest site from LiDAR data. (a) The Original Image. (b) Image with tree trunks detected. Tree trunks are shows with red circles.

Figure 3(a) shows a slice of the point cloud data acquired from LiDAR scans of a typical forest in the Sierra Nevada range. Detected trees are shown as the red circles using Hough Transform in Figure 3(b). Your task is to write a suitable algorithm based on Hough Transform to detect circles in the original image (Figure 3(a)).

[40 Pts] Hough Transform Circle Detection

The OpenCV library has a Hough Transform circle detector implementation. You may find details about this module from `https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html`.

In essence, the function `HoughCircles( src_gray,circles,CV_HOUGH_GRADIENT,1,min_dist,Canny_th,Ctr_th,min_r,max_r)` takes a gray scale input image `src_gray`, and generates a list of detections in the `circles` array represented as a 2D vector. Each element of `Circles` contains three values, the `(x,y)` location, and `r` radius of each detected circle.

`min-dist` is the minimum distance between detected circles, `Canny_th` is the upper threshold of Cannny edge detector, `Ctr_th`is the threshold for center detection, and `min_r`, and `max_r` are the minimum and maximum radii of the detected circles.

In order to draw a circle on an image `src`, you may utilize the following function in OpenCV:

```
circle( src, center, radius, Scalar(0,255,0), 3, 8, 0 );
```

This function uses a 2D vector called `center` containing the `(x,y)` location of the center and the `radius` value for the radius of the circle. The other parameters represent the color, thickness, and aliasing properties of the drawn circle.

Problem 2 continued on next page...

Algorithm 3: Hough Tree Detection Algorithm

Data: I : Input-Image.
Result: O : Output-Image, $Circles$: Vector of Circles (x,y,r).
begin
  $I \leftarrow$ Read Input Image File.
  $(O \leftarrow, Circles)$ Detect-Circles(I).
  Show-Image(I)
  Show-Image(O)
  $O \rightarrow$ Save Output Image File.
  $Circles \rightarrow$ Save Circles (x,y,r) array into a .xlsx file.

Implementation

Your task is to implement a suitable tree detection algorithm that utilizes the Hough Transform as a circle detector. Note that if you simply use the `HoughCircles` function directly you will detect lots of overlapping circles.

Deliverables

a) An electronic (on Canvas) report describing your results. It must include the following items:

- Your approach, broken down in steps.
- The results of your program. These results will include the image of the LiDAR data with the circles superimposed in a Red or Yellow color. See Figure 3(b).
- A discussion of the time, detection results, and application of your program.
- for all figures provide captions with a brief description.

b) One ZIP file containing the following:

- the source code files in C/C++. Make sure the source code is fully documented.
- A README file with instructions on how to compile and run the program and any parameters that need to be set.