**ADVANCED ALGORITHMS**

**Final Project**

**Image to Cartoon using MATLAB**

**By**

**Rishi Reddy (rc81)**

**Professor**

**Dr. Zhong-Hui Duan**

**Academic Integrity Pledge**

Program: - Image to Cartoon

Programmer: - Sukesh Reddy, Shashank Reddy, Vandana Rao and Rishi Reddy

Academic Integrity Pledge

* All source code and documentation used in my program is either my original work, or was derived, by me, from the source code published in the text book for this course or presented in class. Any source code in this project that is derived from code published elsewhere is documented as such so the original author receives due credit.
* I have not used source code obtained from another student, or any other unauthorized source, either modified or unmodified. I have not helped another student write their program by providing a printed or electronic copy of my solution.
* I have not discussed coding details about this project with anyone other than my instructor. I understand that I may discuss the concepts of this program with other students, and that another student may help me debug my program. However, the responsibility to write each program belongs solely to the program's author.
* I have violated neither the spirit nor letter of these restrictions.

Sukesh Reddy

Shashank Reddy

Vandana Rao

Rishi Reddy 4/28/2014

Signature of Programmer Date

**Introduction:**

Filtering is perhaps the most fundamental operation of image processing and computer vision. In the broadest sense of the term “filtering,” the value of the filtered image at a given location is a function of the values of the input image in a small neighborhood of the same location

The goal of this project is to cartoonify an image allowing users to apply the cartoon algorithm to images of their choice. The algorithm is designed to provide artistically and comically appealing results on as wide a range of pictures as possible, although it is conceded that not all inputs will yield equally satisfying results. The process to obtain cartoon effect is divided into two branches- one for detecting and bold the edges, and one for smoothing and quantizing the colors in the image. At the end, the resulting images are combined to achieve the effect. We have used both bilateral filtering and median filtering for smoothing images while preserving edges and also for removal of salt and pepper noise from the images.

The other important aspect of the cartoon effect is that of blockish color regions. In this branch of the algorithm, the colors are repeatedly smoothed to create homogenous color regions. The colors in these regions are then quantized at a lower quantization.

**Description and Implementation:**

The Process to produce the cartoon effect is to smooth and quantize the colors of the image. Color quantization reduces the number of colors used in an image; this is important for displaying images on devices that support a limited number of colors and for efficiently compressing certain kinds of images. The basic implementation is divided into following steps:

**Quantize Colors**

**Median Filter**

**Bilateral Filtering**

**Bilateral Filter:**

Bilateral filtering smooths images while preserving edges, by means of a nonlinear combination of nearby image values. The method is noniterative, local, and simple. It combines gray levels or colors based on both their geometric closeness and their photometric similarity, and prefers near values to distant values in both domain and range. In contrast with filters that operate on the three bands of a color image separately, a bilateral filter can enforce the perceptual metric underlying the CIE-Lab color space, and smooth colors and preserve edges in a way that is tuned to human perception. Also, in contrast with standard filtering, bilateral filtering produces no phantomcolors along edges in color images, and reduces phantom colors where they appear in the original image.

This filter is the key element in the color image processing chain, as it homogenizes color regions while preserving edges, even over multiple iterations. The bilateral filter works similarly to a Gaussian filter in that it assigns to each pixel a weighted sum of the pixel values in the neighborhood. However, the difference is that the weights are further adjusted depending on how different the pixel values are. In this way, a pixel that is close in color to the centroid pixel will have a higher weight than a pixel at the same distance with a more distinct color. This extra step in the weight calculation is important because it means that sharp changes in color (edges) can be preserved, unlike with a simple Gaussian blur. However, the bilateral filter run- time is dependent on the kernel size, and testing showed that running more iteration with a smaller kernel yielded results that were more aesthetically pleasing, and faster than those yielded by applying a bilateral filter with a large kernel size with less iterations. Once the filtering is complete, the image is restored to its original size using linear interpolation to fill-in the missing pixels.

**Steps followed to perform Bilateral Filtering:**

* Select the appropriate filter here it is (function B = bfilter2(A,w,sigma)where A represents double precision matrix of size N\*M\*3 with normalized values in closed interval[0,1] and W represents the Half size gausian bilateral filter window and sigma represents the standard deviation of bilateral filter
* Apply gray scale or color bilateral filtering. In our case we are using the color bilateral filtering through the function

function B = bfltColor(A,w,sigma\_d,sigma\_r)

* Convert RGB or Input Image to CIE-LAB Color Space. (L\*a\*b where

L\* represents brightness and has range from 0(black) to 100(white),

a\* represents degree of redness-greenish whose range is -100 to 100 where positive value corresponds to redness and negative to greenish.

b\* represents degree of yellowish bluish whose range is same that of A where positive value corresponds of yellowish and negative to bluish.

* Creates a 2D grid of w\*w like clusters.

([X, Y] = meshgrid(xgv,ygv) replicates the grid vectors xgv and ygv to produce a full grid. This grid is represented by the output coordinate arrays X and Y. The output coordinate arrays X and Y contain copies of the grid vectors xgv and ygv respectively. The sizes of the output arrays are determined by the length of the grid vectors. For grid vectors xgv and ygv of length M and N respectively, X and Y will have N rows and M columns.)

* Calculating the Euclidean distance of a pixel form center of the window in the mesh grid.

G = exp(-(X.^2+Y.^2)/(2\*sigma\_d^2));

(In this case, the origin is always the center pixel, so \sqrt((X.^2+Y.^2)) will represent the Euclidean distance of a pixel from the center of the window, where X and Y are pixel coordinates with respect to the center pixel.

* Using maximum luminance the range variance values are rescaled.
* Applying bilateral filter to the CIE LAB COLORSPACE image after all the parameter values are set and the procedure is as follows:

-dim takes the size of A matrix

-Now the dim matrix values are set to zeros using zeros function.

-Computing the i values till dim(1) and j values till dim(2) we get the local region that is I=A(iMin:iMax,jMin:jMax,: )

**Quantize Colors:**

Color Quantization is a process that reduces the number of distinct colors used in an image, usually with the intention that the new image should be as visually similar as possible to the original image. Color quantization is critical for displaying images with many colors on devices that can only display a limited number of colors, usually due to memory limitations, and enables efficient compression of certain types of images.

**Steps followed to perform Quantization:** This function uses the bilateral filter to abstract an image following the method outlined in:

* C = Cartoon(A) modifies the color image A to have a cartoon like appearance.[A must be a double precision matrix of size NxMx3 with normalized values in the closed interval [0,1].]
* Set the parameters for bilateral filter i.e A=N\*M\*3 and W=5 and Sigma[3 0.1]
* Set parameters for Image Abstraction and Gradient Functions such as max\_gradient,sharpness levels,quant levels,min\_edge\_strength values.(Note:If max\_gradient value is high the edges are hardly noticeable)
* Convert RGB or Bilateral Image to CIE-LAB Color Space. (L\*a\*b where

L\* represents brightness and has range from 0(black) to 100(white),

a\* represents degree of redness-greenish whose range is -100 to 100 where positive value corresponds to redness and negative to greenish.

b\* represents degree of yellowish bluish whose range is same that of A where positive value corresponds of yellowish and negative to bluish.

* Determine Gradient magnitude of Luminance

[GX,GY] = gradient(B(: , : ,1)/100)

Build The Gradient Matrix:This matrix will contain gradient magnitude for each pixel of the image on which you are detecting edges

G = sqrt(GX . ^ 2+ GY. ^ 2)

Normalize the matrix: To get the pixel values within closed interval [0.1] we normalize the matrix.

G = G/Max\_gradient

* We create simple edge map using gradient magnitudes. ( E = G)
* Per Pixel “sharpening” parameters are determined.
* Apply soft luminance quantization.
* Now convert the image back to RGB image.
* Reducing the size of E matrix to 1\*1\*3 matrix

E= G

E=1-E

**Median Filter:**

Before any further processing, a median filter is applied in order to reduce any salt and pepper noise that may be in the image. At this stage, the median filter kernel is say for example 7\*7 with the centroid as the center element in the matrix. The RGB pixel values at the centroid are set to be the median of the 49 RGB pixel values in the neighborhood. In this way, any extreme specks are smoothed over. The median filter is small enough to preserve edges in the image, which is important. Too large a kernel size would result in washed out edges.

**Steps followed to perform median filtering:**

* B = MEDFILT3(A,[M N P]) performs median filtering of the 3D array A. Each output pixel contains the median value in the M-by-N-by-P neighborhood around the corresponding pixel in the input array.
* We used 3\*3 windows size to perform the median filter on the image.

**Roles of Team Members:**

**Cartoon to Image:** Vandana Rao and Rishi Reddy

**Bilateral filtering:** Vandana Rao and Rishi Reddy

**Median filtering:** Vandana Rao and Rishi Reddy

**Test Cases:**

**Case1:**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Figure 1:-** This above figure illustrates the image to cartoon. The colors are slightly more varied to begin with, and the lighting is non-uniform .The green color present in the input image is quantized to a different color in the cartoon image.

**Case2:**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Figure 2:-** The dolphin image converted to cartoon. We notice that the sky has been turned into a single block of color and the edges of dolphin, splash parts of the image are highlighted. This is ideally the effect that this algorithm attempts to deliver

**Case 3:**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Figure 3:** Image of a person getting converted to cartoon. The algorithm didn’t give satisfactory result. This is because the algorithm is not that good in handling portrait images in a natural way. As face casts many shadows so it’s difficult to detect the edges, as we haven’t performed edge detection which analyses every pixel in an image in relation to the neighboring pixels to sharpen the image.

**Problems Faced:**

As we are new to Matlab we have learnt some commands on images with help command provided by Matlab. Finding the process to create images to cartoon, searching for whether there are any good functions to convert the images was bit challenging. We have not implemented the Edge detection for an image. If we have implemented that we could get more cartooned effect for the image.

**Analysis:**

* With color images, an additional complication arises from the fact that between any two colors there are other, often rather different colors. For instance, between blue and red there are various shades of pink and purple. Thus, disturbing color bands may be produced when smoothing across color edges. The smoothed image does not just look blurred; it also exhibits odd-looking, colored auras around objects.
* Specific goal of our project is to produce the cartoon effect which is divided into two branches- one for detecting and bold the edges, and one for smoothing and quantizing the colors in the image. Output images are evaluated on how well they meet these criteria. The above figures contain examples of desired outputs.
* In the example in figure 1, the colors are slightly more varied to begin with, and the lighting is non-uniform .The green color present in the input image is quantized to a different color in the cartoon image.
* In Figure 2, notice that the sky has been turned into a single block of color and the edges of dolphin, splash parts of the image are highlighted. This is ideally the effect that this algorithm attempts to deliver.
* Finally, Figure 3 shows an example of where the algorithm didn’t give satisfactory result. This is because the algorithm **is not that good in handling portrait images** in a natural way. As face casts many shadows so it’s difficult to detect the edges, as we haven’t performed edge detection which analyses every pixel in an image in relation to the neighboring pixels to sharpen the image.
* From the above all examples we can see that from RGBcolor images only Blue color is clearly and neatly visible.
* All shadows and edges are preserved, but most of the shading is gone, and no “new” colors are introduced by filtering

**Conclusion:**

Thus we have implemented image to cartoon with help of matlab and have done some analysis to find which kind of images do work well for image to cartoon conversion and have tested this for various images.

**References:**

1. <http://www.mathworks.com/help/matlab/ref/zeros.html#btov2gk-2>
2. <http://www.mathworks.com/help/matlab/ref/set.html>
3. <http://www.mathworks.com/matlabcentral/fileexchange/12191-bilateral-filtering>
4. <http://www.mathworks.com/help/fixedpoint/ref/quantize.html>
5. <http://www.mathworks.com/help/images/ref/medfilt2.html>