HACETTEPE UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

Image Processing

Assignment 1

Tunahan Pınar - 21727652

PART 1

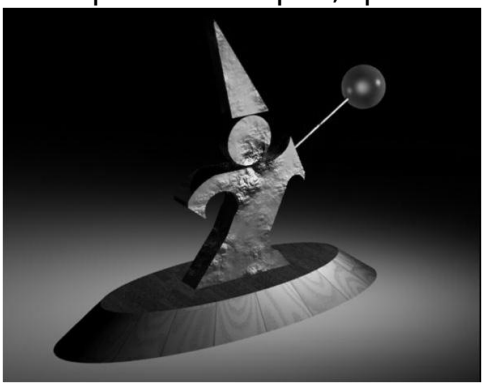
Quantization is an image compression technique that compresses a range of values in each pixel of the image. In this way, we can obtain smaller size images. This technique, which is used in many areas such as saving storage and image processing, also has some disadvantages. As we mentioned above, we encounter many losses depending on the method we use because we compress some values.

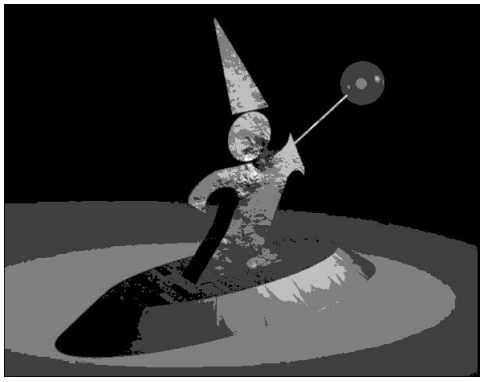
In image processing, quantization is a technique that couse loss while compressing values of image. When the number of values in a given image is reduced, the image becomes more compressible so reducing the number of colors required to represent a digital image makes it possible to reduce its file size.

In the algorithm we examined, we performed quantization using the following formula;

*np.round(source / (255 / q)) * (255 / q)*

When we look at this formula, we see that we quantify our pixel range between 0-255. At the end, image will have 'q+1' different values. That's why we lost a lot of value as shown in below original image and examples when q=1, q = 2 and q=4 respectively.





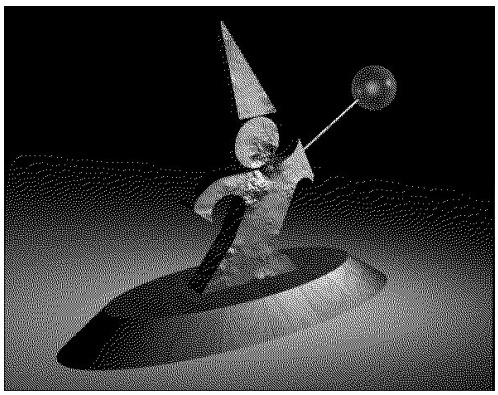
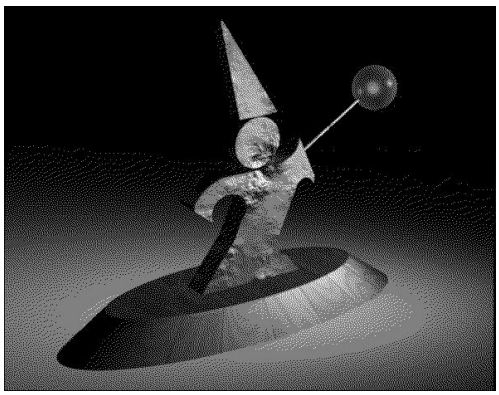
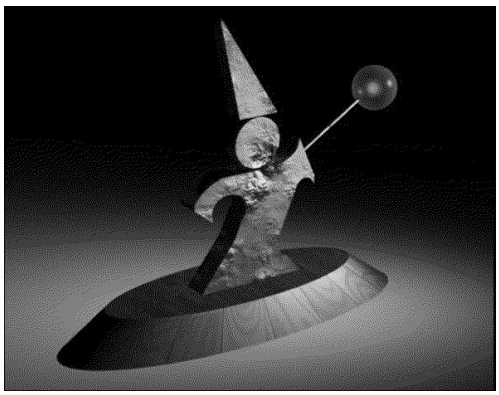
When we look at the pictures we can easily see the when q increase, more color and details added. But that is not enough most of the case. So can do dithering for that purpose.

- **How is the given method achieves to prevent quantization error? Explain with examples.**

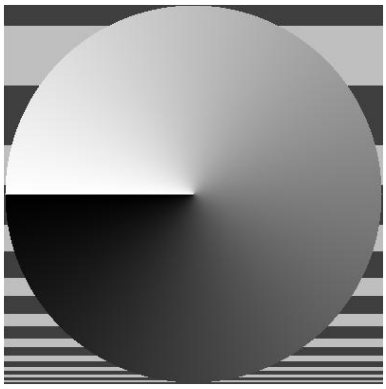
Our technique which is named ‘FloydSteinberg’ is can reduce the effect that quantization did. FloydSteinberg allowed us to capture many details by calculating the error difference between the original image and the quantized image and spreading this value over the entire image from right to left and from top to bottom. Code example is in below picture;

```
def floyd_steinberg(source, q):
    i = 255
    for y in range(1, source.shape[0] - 1):
        for x in range(1, source.shape[1] - 1):
            oldpixel = source[y][x]
            newpixel = np.round(source[y][x] / (i / q)) * (i / q)
            source[y][x] = newpixel

            quant_error = oldpixel - newpixel

            source[y, x + 1] = source[y, x + 1] + quant_error * 7 / 16
            source[y + 1, x] = source[y + 1, x] + quant_error * 5 / 16
            source[y + 1, x - 1] = source[y + 1, x - 1] + quant_error * 3 / 16
            source[y + 1, x + 1] = source[y + 1, x + 1] + quant_error * 1 / 16

    return source
```

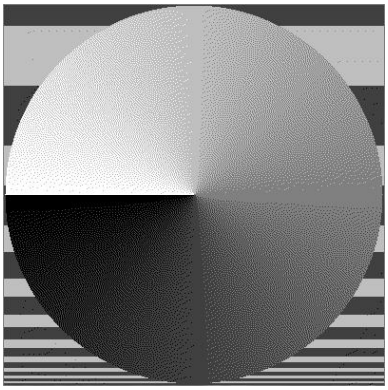
In this way, we have obtained more detailed and more detailed photographs with color transitions. We can see below the dithered versions of examples above when q=1, q = 2, and q=4 respectively.

As seen in the examples, our technique made many details visible. I will also show a simpler example below to show how the color transitions differ and improve.


Original image


quantized q=4


dithered q=4

When q=4, we have 5 different color in quantized image. So it is really different from the original images because it is very smooth when compare with quantized. So we can perform FloydSteinberg and result shown above. It is very smooth when compare with quantized image. It is easily shown when look at the color shift between black and almost white on the circle in the image.

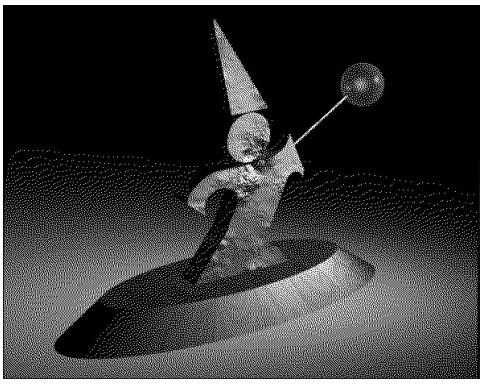
• What is the behavior of the algorithm for different q parameters?

The algorithms works very good on the images that provided in the assignment. It has huge improvement on the lower q value especially in q=1. I think it works like that because of the color values we have on the lower q values. For example when q=1, we have only 2 color on the quantized and there is almost no detail.

In low q values, dithering cause too many visible and seperate dot on the image but on the high q values for example q=4, we have very smooth image when compare with q=2 and q=1. It think, it happens because lower q values has bigger error values and due to this, algorithm can not spread over the whole picture accurately.

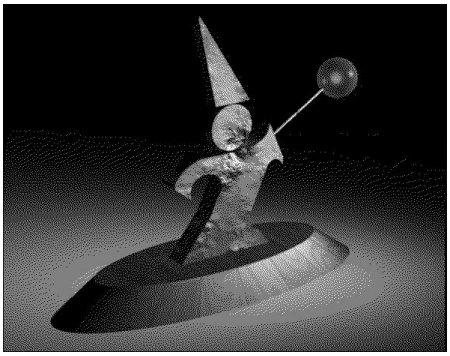
• Compare the quantized image and dithered image for different q parameters

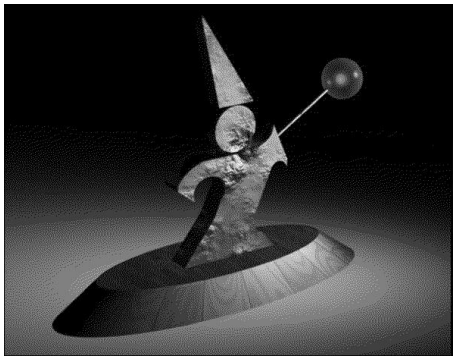
FloydSteinberg is works very nice in above examples. When q=1 it increase the color range and the details very much when comparing the quantized one.



In q=1 the algorithm working very good on where color shifting is more sharp. We can see that on the picture when look at the color transitions that shift towards black from gray as you go up and look to the base parts of the object. I think it happen because we have only 2 different color on quantized image.

Also we can see that details and color shifts are more become more visible on q=2 and q=4.



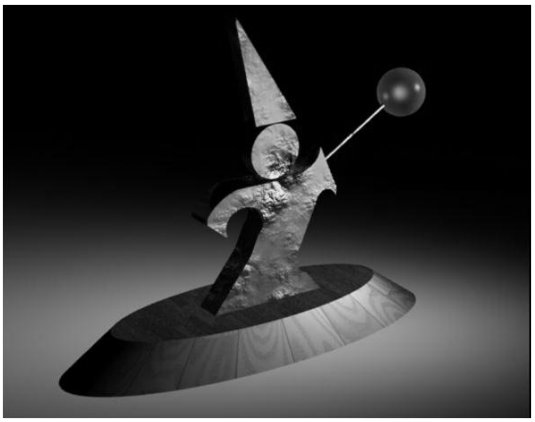


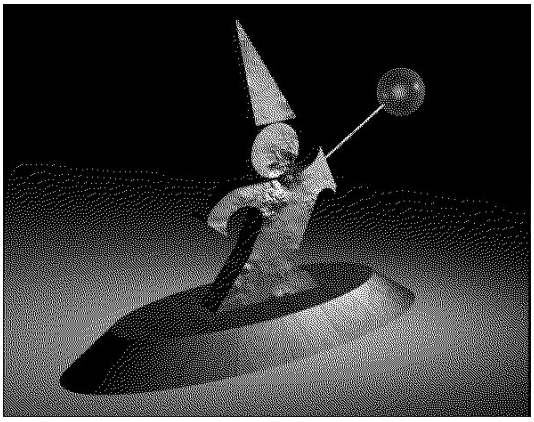
In q=1, there is too much detail become visible because in quantized q=2 has almost no detail. But when q increase, there is really good improvements on the images. There is less visible dots on the picture due to more colors on the quantized images. Also, details on the object become more visible and more accurate.

• What are the disadvantages of Floyd-Steinberg dithering algorithm? Explain with examples.

Most significant disadvantages of the this algorithm I see is sometimes there are too much non-uniform dots on the picture. This issue increase when q going to lower. Also, there is no clear pattern of dots is formed in places of same gray areas. Both can be seen below picture.

When we look at the grayness of the picture in the middle, is it almost same in the original and it going to black very smoothly. But our dithered image is non-uniform dot patterns in these areas.





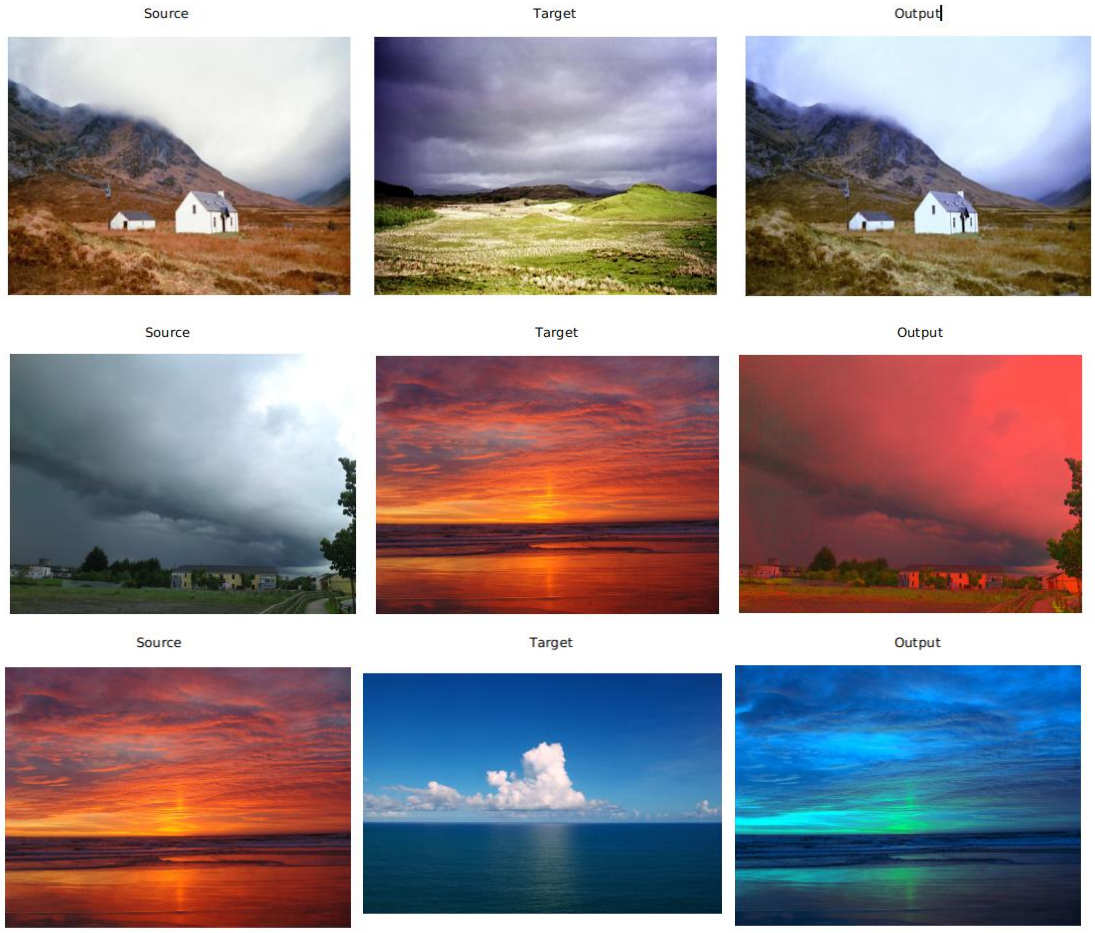
PART 2

• Why does the algorithm change color space from RGB to Lab?

LAB color space is designed designed to approximate human vision and the L component matches with the human perception of lightness. So, it represent the luminosity. It is more useful for estimating small differences in colors and it has bigger color gamut than the RGB.

A and B component are the color components and they are separated from lightness so we can adjust the lightness without effecting the colors unlike the RBG. So that, it minimizes correlation between channels. Also, it prevent from undesirable cross-channel artifacts while transferring colors between images.

• Show the results of your implementation for several images



Source Target Output

• What are the disadvantages of the given color transfer algorithm? Show some failure results of the algorithm.

This algorithm actually work pretty well in most of the case. But, I think, when try to this algorithm, we should choose the pictures carefully with good reasons. If pictures doesn't chosen correctly, there is meaningless outputs comes out. Also, it reduce the good details on the image sometimes. For example in below;

Source Target Output

When we look at the above transformation, there is lots of problem, I think. First of all, the image become a very bright. Due to this, there are many missing details, especially in the leaves on the ground. Secondly, the colors on the images unpredictable and meaningless. This can be notice on the gray and gray with yellow tones areas.

Most of the colors transform to very bright pink-blue tones. I think, it shouldn’t be happen because target ground has green and gray tones and source has orange and gray tones.

```
# mean and std
s_means, s_stds = mean_std(LAB_s)
t_means, t_stds = mean_std(LAB_t)

# Subtract the mean of source image from the source image
l_sub_mean = LAB_s[:, :, 0] - s_means[0]
a_sub_mean = LAB_s[:, :, 1] - s_means[1]
b_sub_mean = LAB_s[:, :, 2] - s_means[2]

# Scale the data points and add target's mean
l_scale = ((t_stds[0] / s_stds[0]) * l_sub_mean) + t_means[0]
a_scale = ((t_stds[1] / s_stds[1]) * a_sub_mean) + t_means[1]
b_scale = ((t_stds[2] / s_stds[2]) * b_sub_mean) + t_means[2]
```

When we look at the code, we can see that the transformation relies on the global color statistics so the sky on the target must have too much effect on the other side of the image.

As shown in the code picture, all the means and stds are coming from channels and channels includes all the image values which belong to specific channels. So that, in our example, the brightness has too much effect in some areas. Maybe this effect can be reduced by clustering the image by splitting it into different regions. In this way, in our example, sky colors and brightness don’t affect the too much to the ground.

References

[1] R.W. Floyd, L. Steinberg, An adaptive algorithm for spatial grey scale. Proceedings of the Society of Information Display 17, 75–77 (1976)

[2] [2] Erik Reinhard, Michael Ashikhmin, Bruce Gooch and Peter Shirley, ’Color Transfer between Images’, IEEE CG&A special issue on Applied Perception, Vol 21, No 5, pp 34-41, September - October 2001

[3] https://en.wikipedia.org/wiki/Quantization_(image_processing)

[4] https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering