Image Reproduction using Buttons

Linus Mossberg

Linköpings Universitet

Abstract

This report presents a method of reproducing images with mosaics of circle-packed buttons. An image-driven circle packing method is developed to determine the placement and size of buttons. The circle packing method utilizes image segmentation and an iterative maximum inscribed circle placement method to distribute buttons to perceptually similar image regions. A color matching function is then developed which utilizes both a mean color descriptor and a dominant color descriptor to find buttons that matches the resulting circular regions in the reference image. Color correction methods for both individual buttons and reproduced images are presented to improve the reproduction quality. A method of reducing the number of unique buttons used in a reproduction to the most important ones using K-means is finally presented. The full reference quality metrics S-CIELAB and SSIM are used to evaluate the results.

1 INTRODUCTION

Commonly used elements in image reproduction include dots of ink with a few primary colors placed at different density and equally sized squares with a weighted combination of a few primary colors placed on a regular grid. This report aims to present a method of reproducing images with sewing buttons of different sizes instead.

The method can be broken down to the following steps:

- Button database creation
- Circle packing
- Color matching
- Color correction
- Optimizations

The report will present and motivate each of these steps. The method is implemented as a MATLAB program that takes an RGB reference image and several settings as input and returns the image reproduced with buttons.

2 METHOD

The following sections describes each step in the image reproduction method, as well as some results to motivate the selected methods.

2.1 DATABASE

The first step was to create a database with images buttons. The size and button variety of this database is important since this determines the effective color gamut that can be reproduced. The database was created by first finding images of buttons on websites such as Flickr with licenses that permits sharing and adoption. The images were then cropped and split into individual square images for each button such that the buttons precisely fit and touches all four borders of the image. It was then possible to programmatically generate circular masks for the alpha channels of each image that masks out the background from the buttons. This finally resulted in a total of 1630 buttons. The color gamut of this database is shown in Figure 1.

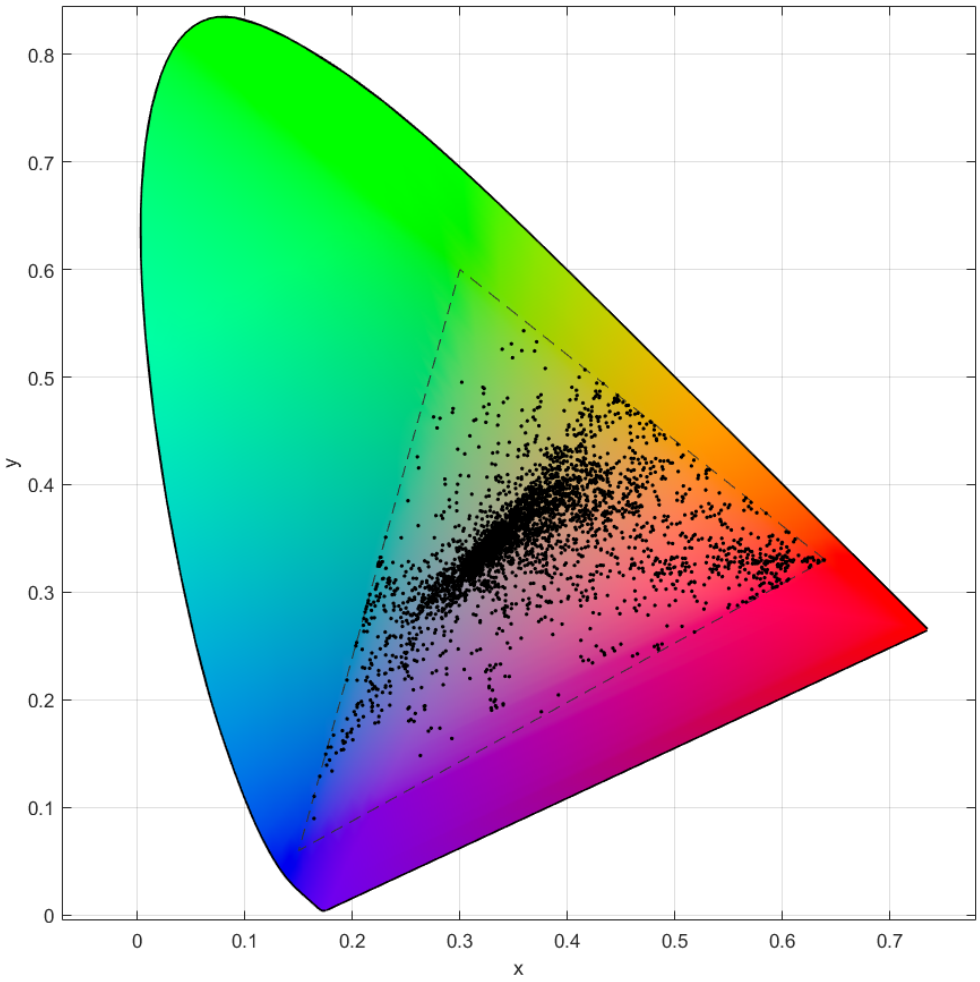

Figure 1 - Database color gamut with chromaticity diagram and sRGB gamut. The dots represent the most perceptually dominant colors of the buttons. At most 3 dots per button are plotted.

2.2 Image-Driven Circle Packing

The size and position of buttons in the reproduced image is determined by an image-driven circle packing method. The following sections details each step in this algorithm.

2.2.1 Image Segmentation

The first step in the circle packing algorithm is to segment the image into perceptually self-similar regions. The circles are then packed within each region individually, which preserves perceptually distinct borders and ensures that each button only need to reproduce a perceptually small range of colors.

A simple way of performing image segmentation is to use *K-means clustering* on color coordinates of pixels in an image. This is performed as following:

1. Place K centroids in the color space using some initial placement method.
2. For each centroid:
    a. Find the color coordinates that are closer to this centroid than to any other centroid.
    b. Calculate the mean coordinate of these color coordinates.
    c. Move the centroid to this mean coordinate.
3. Repeat from 2 until the centroids no longer move.

The color coordinates closest to each centroid then belongs to a common cluster [1], and the corresponding pixels constitutes a segmented image region. If this is done in the *CIELAB* color space, then the clusters will correspond to perceptually self-similar colors [2] which yields perceptually self-similar segmented image regions. An example of this is shown in Figure 2.

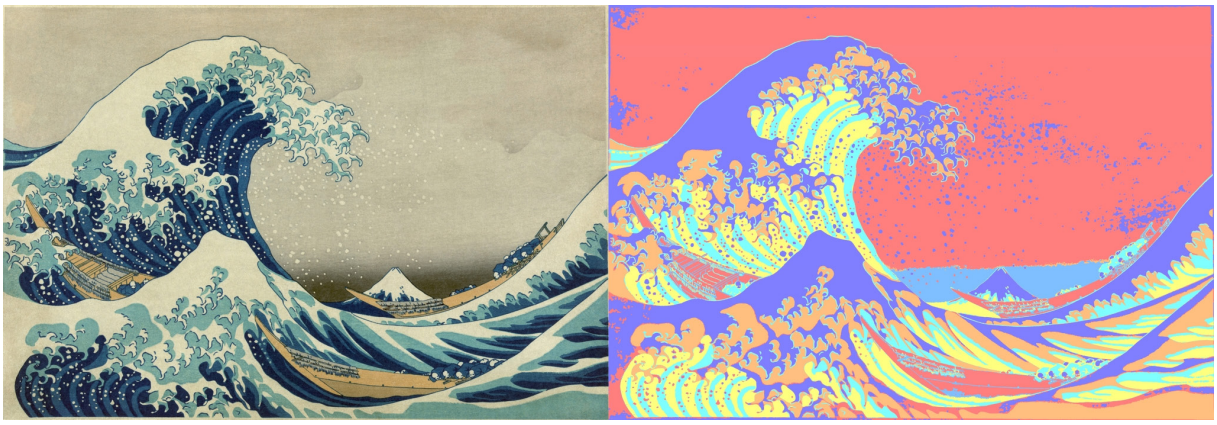

Figure 2 - Result of K-means image segmentation for K = 6.

The chosen K-value greatly influence the resulting reproduction. Large values result in a finer segmentation, which in turn result in more and smaller packed circles in the next steps. This can increase detail, but it can also result in an unnecessary number of buttons and clutter. Small values can on the other hand result in a rough segmentation that may remove detail.

The best results are typically achieved when the image segmentation corresponds to the most perceptually distinct regions in the image that a human might pick out. It is therefore

best to pick this value manually. A method of estimating this value has however been implemented by using the *Calinski-Harabasz criterion* [3] to evaluate the optimal number of clusters of the pixel color coordinates in the CIELAB color space. This is used when no K-value is specified, and it often matches up well with the value a human might pick.

2.2.1.1 Filtering

One problem with this image segmentation method is that it does not consider the spatial relationships between the pixels in the segmented image. This can result in thin, disjointed and noisy sections in the segmented image. If the minimum circle allowed during the circle placement stage can't fit in a section, then no circle can be placed at that location. This can result in regions with sparsely placed circles, especially in locations with dense detail in the reference image.

To solve this and improve the circle placement distribution, a filtering method was developed that takes the minimum circle radius r_{min} into account. The method works as following:

1. Generate a circular structure element with radius r_{min}.
2. For each image segment S in the segmented image:
    a. Perform morphological erosion on S with the structuring element. This leaves all pixels that can be used as center positions for a circle of radius r_{min} without extending the circle outside of the original region S.
    b. Perform morphological dilation on the result with the structuring element. This expands the region to include all pixels that a circle of radius r_{min} can cover. Call the result S', this is the new region for this image segment.
    c. Find the remaining pixels $U = S \,\&\, !\, S'$. These pixels correspond to unusable regions that are impossible to cover with a circle of radius r_{min}.
    d. Distribute each connected pixel region in U to the neighbouring image segment that has the most neighbouring pixels.

A visualization of this is shown in Figure 3. The unusable pixels U in step 2.c is shown as gray pixels in the rightmost image.

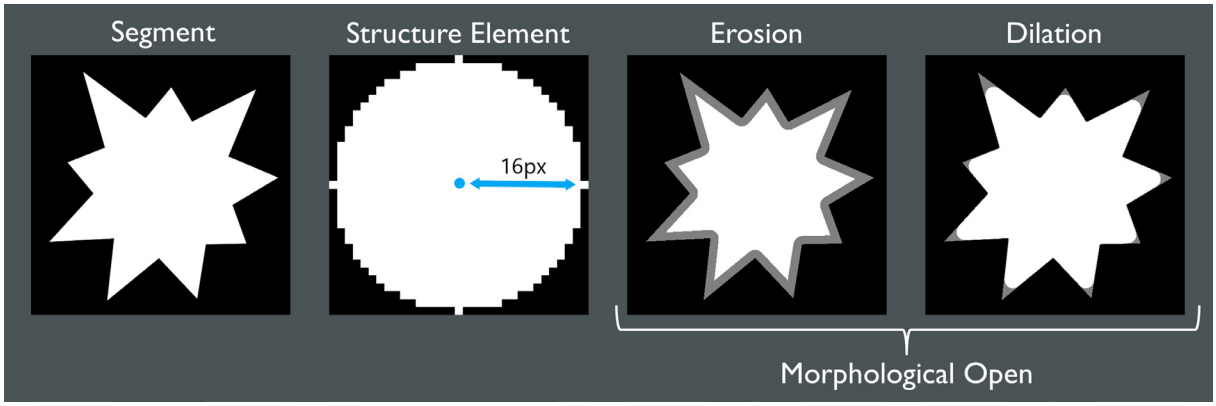

Figure 3 - Segmented image filtering steps

The result of this filtering on the previous segmented image is shown in Figure 4.

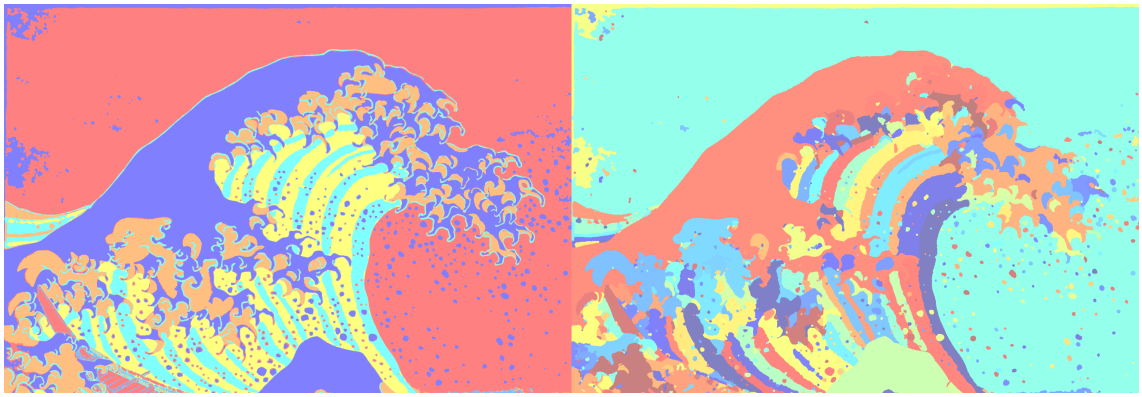

Figure 4 - Segmented image before and after filtering. The labelling is changed to connected pixel segments before the filtering for optimization purposes, which changes the visualized segment colors in the after image.

Much of the detail has been removed, but these are details that would be impossible to reproduce given the specified minimum circle radius r_{min}. This filtering results in a better circle distribution that improves the final image reproduction.

2.2.2 Iterative Maximum Inscribed Circle Placement

The circle placement algorithm is then performed on each image segment individually. The algorithm iteratively places circles within each image segment using the following steps:

1. Find the maximum inscribed circle that can fit in the image segment.
2. If the circle radius is smaller than r_{min}, stop the processing of this segment.
3. Otherwise, save the coordinate, radius and color information of this circle to a list and remove the circle area from the image segment. Repeat from 1.

The maximum inscribed circle is found by performing a distance transform on the binary image of the image region. This transform returns a distance map where each pixel value corresponds to the Euclidean distance to the nearest edge in the region, i.e. the nearest zero-valued pixel [4]. The maximum value in this map corresponds to the radius of the maximum inscribed circle, and the corresponding pixel coordinate is the center coordinate of this circle. This process is shown in Figure 5.

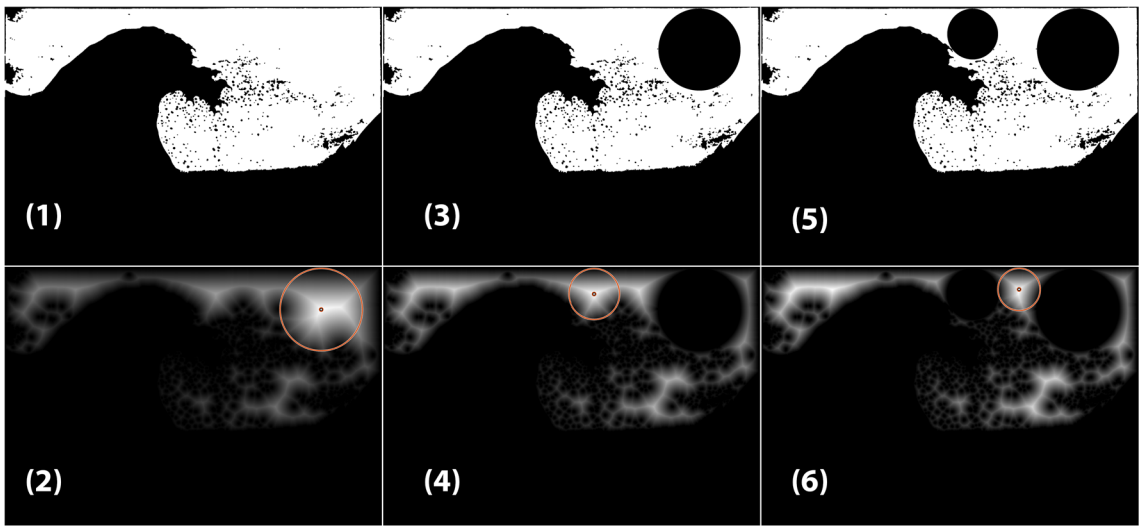

Figure 5 - Circle placement algorithm. The number corresponds to the process order. The top images show the active image region at different stages, and the bottom images show the resulting distance maps with the maximum inscribed circle.

Once the algorithm has completed the circle placement process for an image region, there will be several pixels remaining in the region that hasn't been covered by a circle. Each of the connected regions of these remaining pixels are then distributed to the neighbouring image segment that has the most neighbouring pixels, provided that this image segment hasn't been processed yet.

The result once this has been performed on each region is a list of circles. These circles have been drawn for the above example in Figure 6.

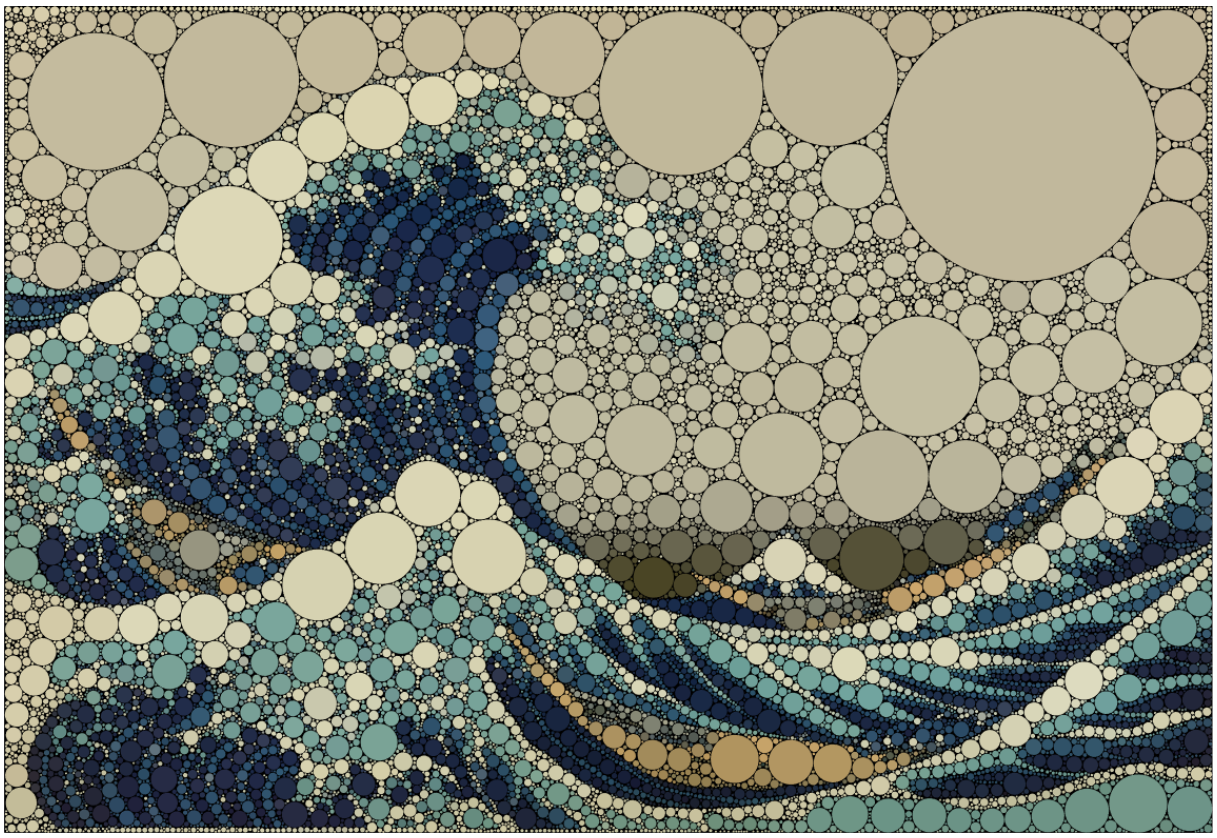

Figure 6 - The result of the circle packing stage. The color of each circle is the mean color of the area that they cover in the reference image.

These circles will be replaced by color matched buttons from the database in the next steps.

2.3 Color Matching

A color matching function has been implemented that attempts to find the button in the database that best reproduces a region in the reference image. The color matching function works by first defining a color descriptor $CD(X)$ that describes the color contents of a collection of colors X, and then by defining a difference function $D(A, B)$ that defines the difference between two color descriptions $A = CD(X_A)$ and $B = CD(X_B)$. In this case, X corresponds to either the pixel colors of a button or the pixel colors of a region in the reference image.

Color descriptions are first calculated and stored for each button in the database. The color matching function then matches a button to a region in the reference image by first calculating the color description of the region, and then calculating the color difference with

each button using the corresponding difference function. The region is finally matched with the button that had the smallest difference.

2.3.1 Mean Colors

A simple color descriptor of a collection of colors $X = \{x_1, \dots, x_M\}$ is their mean color:

$$CD(X) = \frac{1}{M}\sum_{i=1}^{M} x_i$$

The difference between two mean colors A and B can then simply be defined as the distance between them in a color space:

$$D(A, B) = \|B - A\|$$

This difference metric is made to correspond well with the perceptual difference by first converting the collection of colors to the CIELAB color space.

2.3.2 Dominant Colors

Instead of describing a collection of colors with a single color, it can be useful to describe it as a distribution of several of the most dominant colors in the collection instead. For example, instead of describing the flag of Ukraine as beige, it may be more informative to describe it as 50% blue and 50% yellow.

Perceptually dominant colors of a collection of colors are typically clustered together in different regions of the CIELAB color space. It is therefore possible to find these clusters by using K-means clustering on the color coordinates in the CIELAB color space. The dominant colors are then the centroids (mean colors) of the clusters, and the relative number of colors in each cluster are the corresponding dominance percentages. A dominant color description A then consists of K dominant colors $C_1, \dots C_K$ paired with their dominance percentages $P_1, \dots P_K$:

$$A = \{ \langle C_1, P_1 \rangle, \dots, \langle C_K, P_K \rangle \}$$

where:

$$\sum_{d=1}^{K} P_d = 100\%$$

An example of this for a few buttons compared to their mean colors is shown in Figure 7.

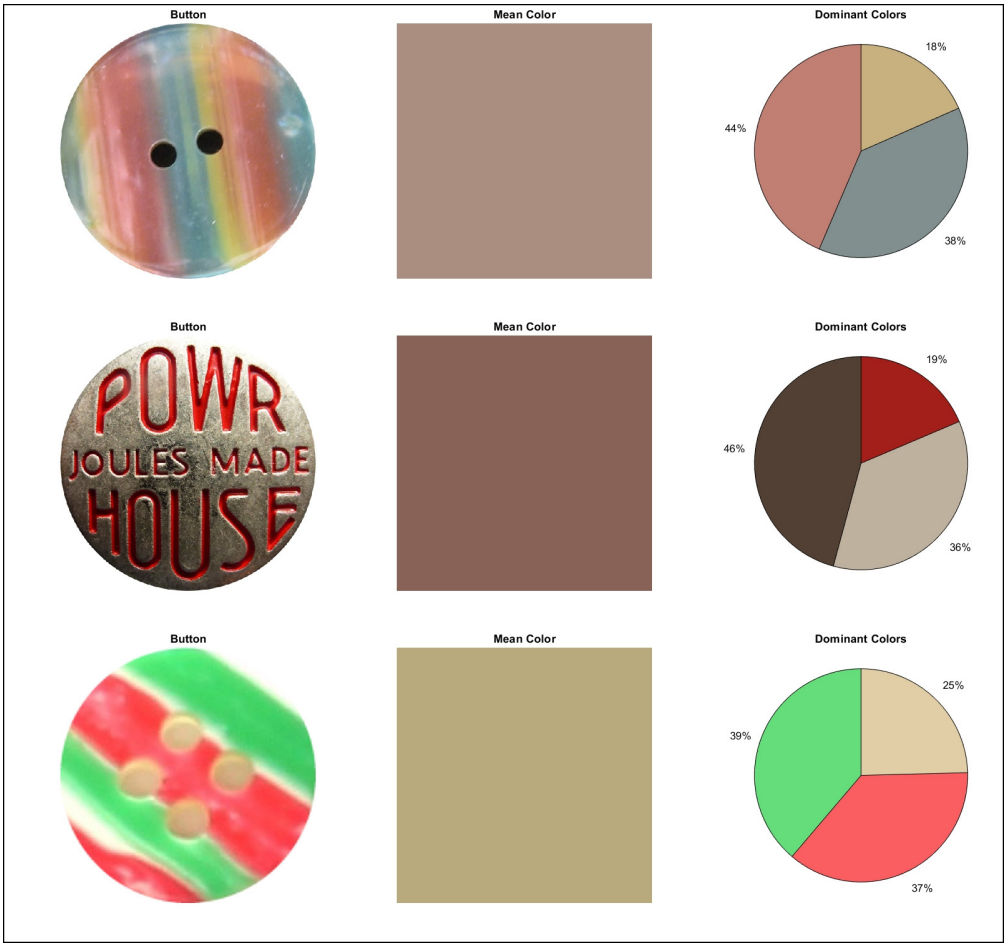

Figure 7 - Comparison between mean color and three dominant colors for a few buttons.

As seen here, the three dominant colors of a button is much more descriptive than the mean color (one dominant color) since buttons tends to have multiple perceptually distinct colors.

2.3.2.1 Dominant Colors Difference

Comparing the difference of two sets of dominant colors is a more complex problem than comparing the difference of individual mean colors. The paper *Extraction of Perceptually Important Colors and Similarity Measurement for Image Matching, Retrieval, and Analysis* [5] presents a general solution that, given unlimited time and memory, results in what can be considered to be the correct solution, i.e. the distance between the dominant color distributions of the two sets in a color space.

This method works by first quantizing the dominant color distributions of the compared sets A and B into N discrete nodes each, where each node represents a color from the set. An illustration of this for two sets of dominant colors A and B for $N = 4$ is shown in Figure 8.

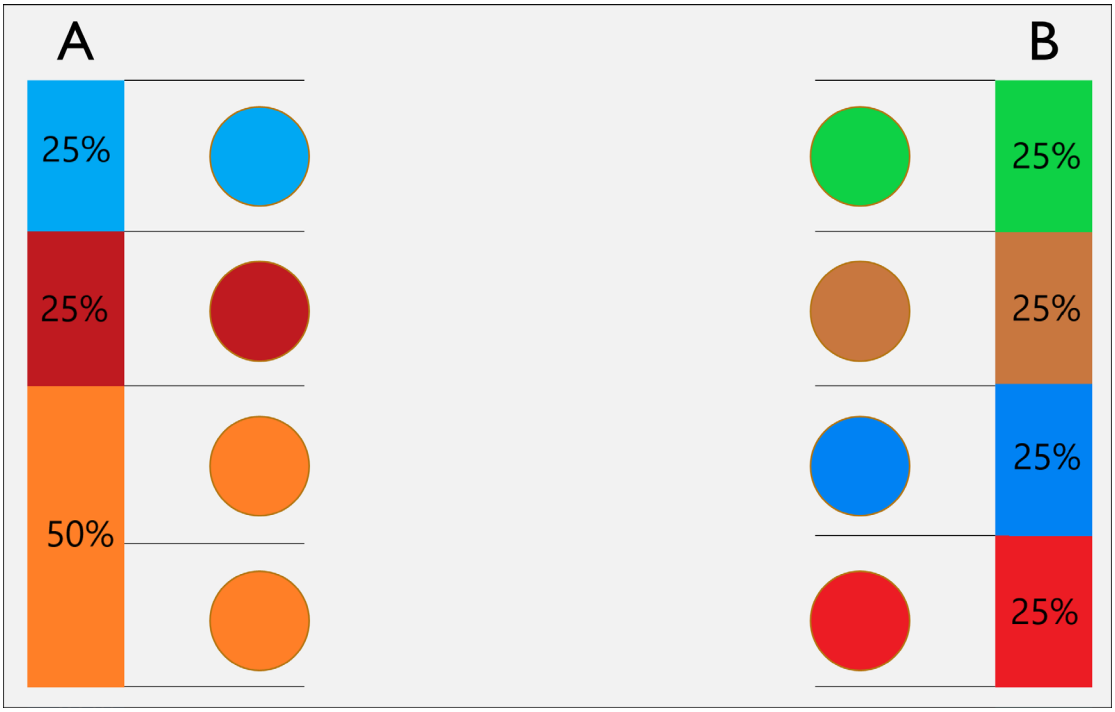

Figure 8 - Dominant color distributions quantized to nodes

In this case, the dominant color distributions can be evenly quantized to four nodes. It is generally not possible to evenly allocate a discrete number of nodes across a real distribution of dominant colors, and more nodes generally results in a better representation of the dominant color distribution. To perform the node allocation optimally, the initial node counts n_d that should be allocated to each dominant color with index $d \in 1, \dots, K$ is found using:

$$n_d = \lfloor N \cdot P_d \rfloor$$

where P_d is the dominance percentage of dominant color d. This leaves a total number of remaining nodes R given by:

$$R = N - \sum_{d=1}^{K} n_d$$

The remainder r_d from the floor conversion for each dominant color d is then found using:

$$r_d = N \cdot P_d - n_d$$

The node counts of the subset S that consists of the R dominant color indices that had the largest remainder r_d is then incremented by one. This results in the final node counts allocated to each dominant color:

$$n'_d = n_d + |d \cap S|$$

Which results in a total of N allocated nodes. This method is commonly referred to as the *Largest Remainder Method* [6].

Each node in set A is then connected with an edge to each node in set B to form a bipartite graph. Each edge is assigned a weight which is defined to be the CIELAB color distance between the colors that the nodes represents. The weight of the edge that connects node i in set A and node j in set B is therefore:

$$w(i,j) = \Delta E_{ab}(C_{A,i}, C_{B,j})$$

where $C_{A,i}$ and $C_{B,j}$ are the colors that node i in set A and node j in set B represents respectively. An illustration of this with the previous example is shown in Figure 9.

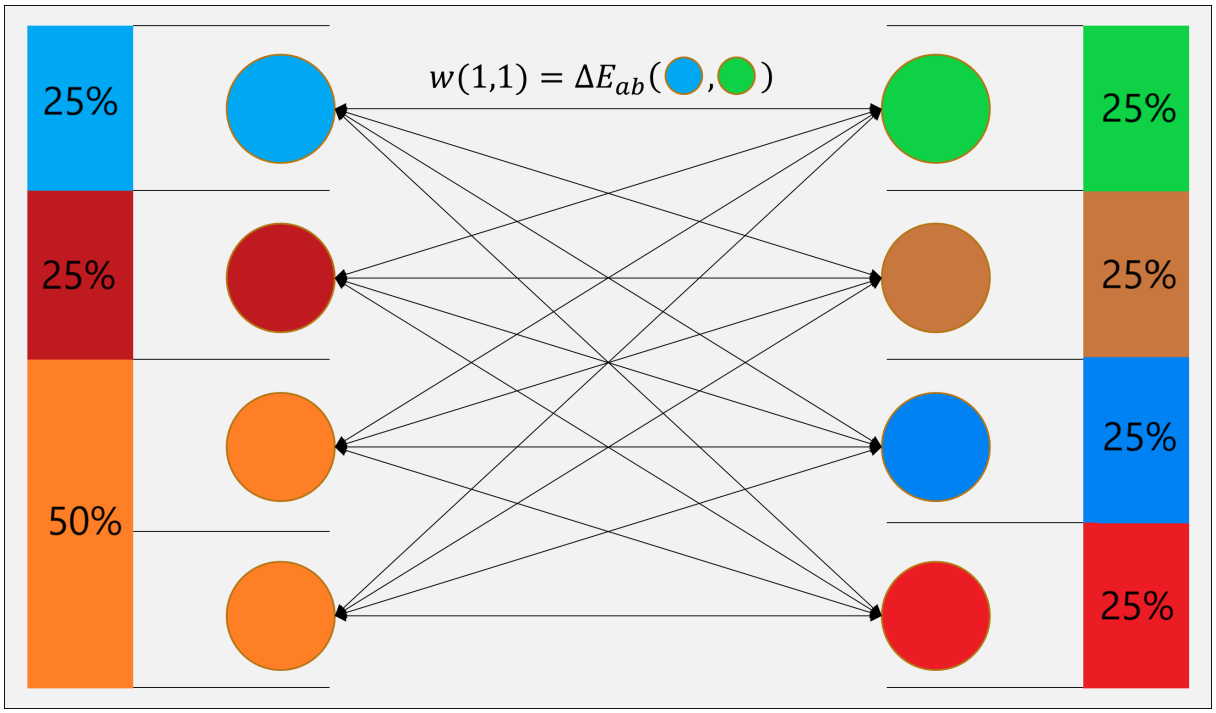

Figure 9 - Weighted bipartite graph

The problem is now to find the optimal node pairings that minimizes the sum of the resulting edge weights, and the resulting difference between the dominant colors A and B is defined to be the average weight of these edges. This problem is commonly referred to as *The Assignment Problem* and it can be solved using *The Hungarian Algorithm* [7] for example. The built-in MATLAB function *matchpairs* was used in practice, however. An illustration of how this might look for the previous example is shown in Figure 10.

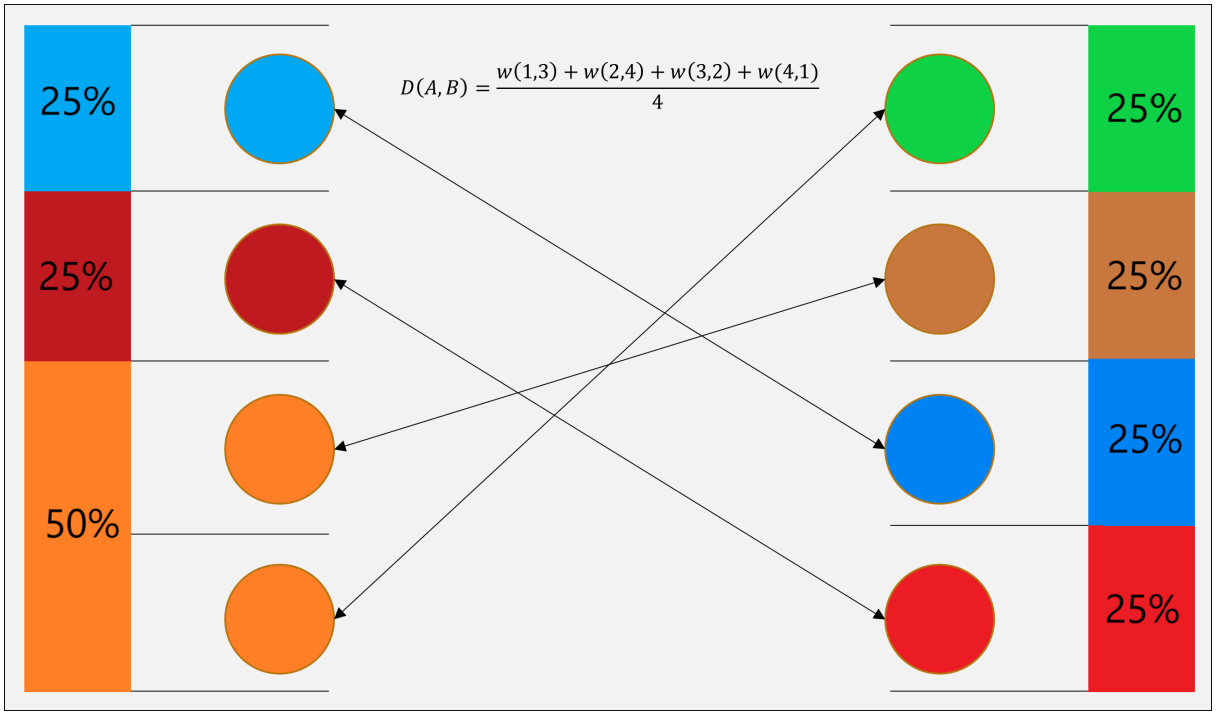

Figure 10 - Optimal mapping

This difference metric is a type of average distance between the sets of dominant colors in the CIELAB color space, and as such it corresponds well to the perceptual difference between the two sets. This difference metric can now be used in the same way as the mean color difference metric to find the button in the database that best matches a region in the reference image.

2.3.2.1.1 Speed and Accuracy Trade-Off

Because a type of uniform quantization is used to convert the dominant color distributions to the node representation with N discrete nodes, the mean squared error of the node representation should be [8]:

$$\frac{(100\%/N)^2}{12} = \frac{2500}{3N^2}\%$$

The time complexity of algorithms that solve the assignment problem are typically in the order of $O(N^3)$ however [7], which means that a trade-off between speed and accuracy must be made. 30 nodes give a mean squared error of about 1%, and this number of nodes often results in the same button match as 1000 nodes. The program therefore generally uses 30 nodes for button matching. It is however possible to use variable numbers of nodes depending on for example the resulting size of the matched button in the reproduced image.

2.3.3 Difference Metric Comparison

Figure 12 and Figure 13 shows the result of using mean and dominant color matching to reproduce the reference image shown Figure 11. The minimum circle radius used in the circle packing stage has been set higher than normal to try and make the buttons more visible in this small format. This reduces the overall reproduction quality, however.

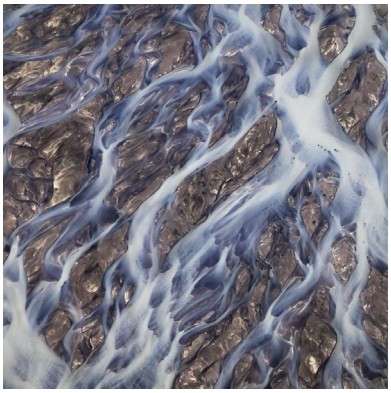

Figure 11 – Reference Image

Figure 12 – Left: Mean Color Matching. Right: Dominant Color Matching

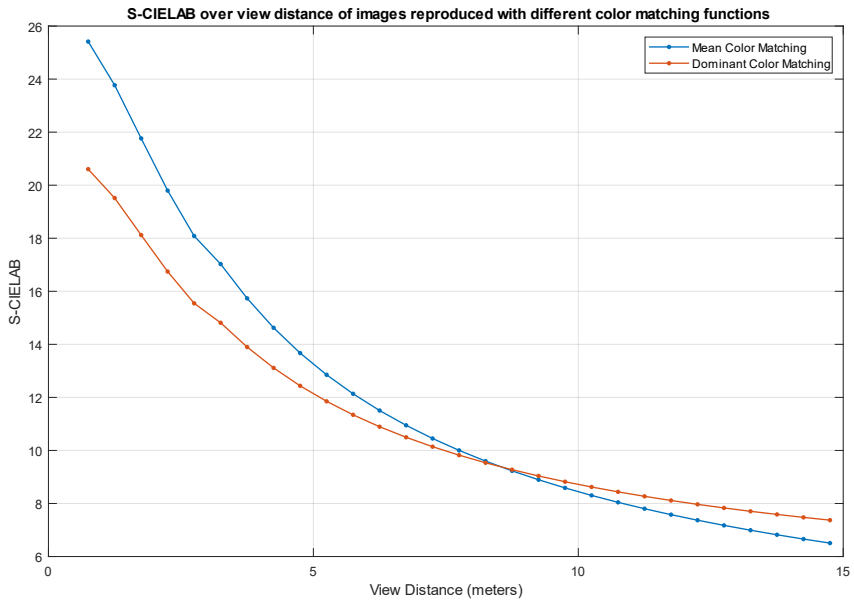

Figure 13 - Mean S-CIELAB of images reproduced with different color matching methods. Viewed as if image is 1 meter wide (4096x4096, 104ppi) from different view distances.

It is difficult to convey the perceptual difference in Figure 12 because of the downsampling required to fit the images in this document, which makes the buttons approach their mean color. The subjective quality difference in full format is very apparent however and probably even more so than the S-CIELAB metric. The *Structural Similarity Index* is 0.2733 for the image reproduced with mean color matching and 0.3517 for the image reproduced with dominant color matching, which is a 29% improvement.

Interestingly, mean color matching overtakes dominant color matching when viewing from a distance further away than about 9 meters according to the S-CIELAB metric. This is because the low-pass filter applied to model the human visual system causes the buttons to approach their mean color with increasing view distance. The mean color matching method only picks buttons based on the closest mean, which results in a better reproduction when a button is filtered down to its mean color.

This effect is utilized in the program to improve both performance and reproduction quality. Due to the quadratic increase in buttons with decreasing circle radius, most buttons are small in a reproduced image. Most of the area is however made up of a small number of large buttons, since the area increases quadratically with increasing radius.

As a result, if dominant color matching is used exclusively for buttons that will be larger than a specified minimum radius in the reproduced image, then mean color matching is used on most buttons while dominant color matching is used on most of the area. Mean color matching is much faster than dominant color matching, and as a result this method is almost as fast as only using mean color matching for all buttons. The reproduction quality can also be improved since dominant color matching is better at reproducing perceptually large regions while mean color matching is better at reproducing perceptually small regions as seen in Figure 13.

The minimum radius should be chosen depending on view distance. If the angular resolution of the human eye is assumed to be $0.02°$ (50 samples per degree), then the minimum radius in pixels could be set to:

$$\left\lfloor \frac{d \cdot ppi \cdot \tan\left(\frac{\pi}{180}\right)}{2 \cdot 50} \right\rfloor$$

where d is the view distance in inches and ppi is the pixels per inch of the reproduced image. This would make the minimum radius so small that only one sample would be perceived for the entire button surface. This results in a minimum radius of 2 pixels for a view distance of 3 meters and 104 pixels per inch. This is often excessive given that buttons often are radially symmetric and uses repeating patterns, which causes all perceived samples of a button to converge to a similar color sooner. The minimum radius used is therefore often 16 pixels for the previous example, which corresponds to about 7.5 perceived samples across the diameter of a button, or about 44 perceived samples for the entire button surface.

2.4 Color Corrections

The following sections describes the implemented color correction methods and the expected improvements these can have on the reproduction quality.

2.4.1 Button Color Correction

As seen in Figure 1, the chrominance gamut that can be reproduced with the database is limited, which is true for the luminance part as well. To try and correct for this, a button color correction method has been implemented. This method matches the mean color A of each button with the mean color B of the image region that the button reproduces in the CIELAB color space by adding a color factor F to each pixel in the corresponding button. This color factor is simply defined by:

$$F = B - A$$

and it represents the mean color difference vector between the collections of colors in the CIELAB color space. Once this color vector is added to each pixel in a button, the button correctly reproduces the mean color of the region in the reference image. It is also possible to only correct for certain color components such as luminance by setting the a^* and b^* components to 0 in F. The program corrects for all components however, because the distribution of buttons is still large enough to where only correcting for one color component typically results in barely noticeable improvements. The result of this can be seen in Figure 14 c). The reference image used in this test turned out to be relatively well-reproducible with uncorrected buttons, but the difference can be much greater for other reference images.

2.4.2 Image Color Correction

The same correction can also be applied to the entire reproduced image, i.e. the mean color of the button corrected image from the previous step can be matched with the mean color of the reference image in the CIELAB color space by adding a constant factor to each pixel.

The image means are not matched by only correcting the individual buttons because there are pixels in the reproduced image with no button coverage. These pixels are set to the background color, black, and they constitute about 5-10% of pixels for images with very densely packed buttons (resolutions in order of $4096x4096$ with $r_{min} = 4px$). This is unavoidable because of the geometrical properties of circle packing. The result of this correction can be seen in Figure 14 d), and the difference is mostly that the image becomes slightly brighter to compensate for the black background pixels.

2.4.3 Color Correction Comparisons

A comparison between these corrections using the mean S-CIELAB quality metric is shown in Figure 15, and the corresponding images are shown in Figure 14.

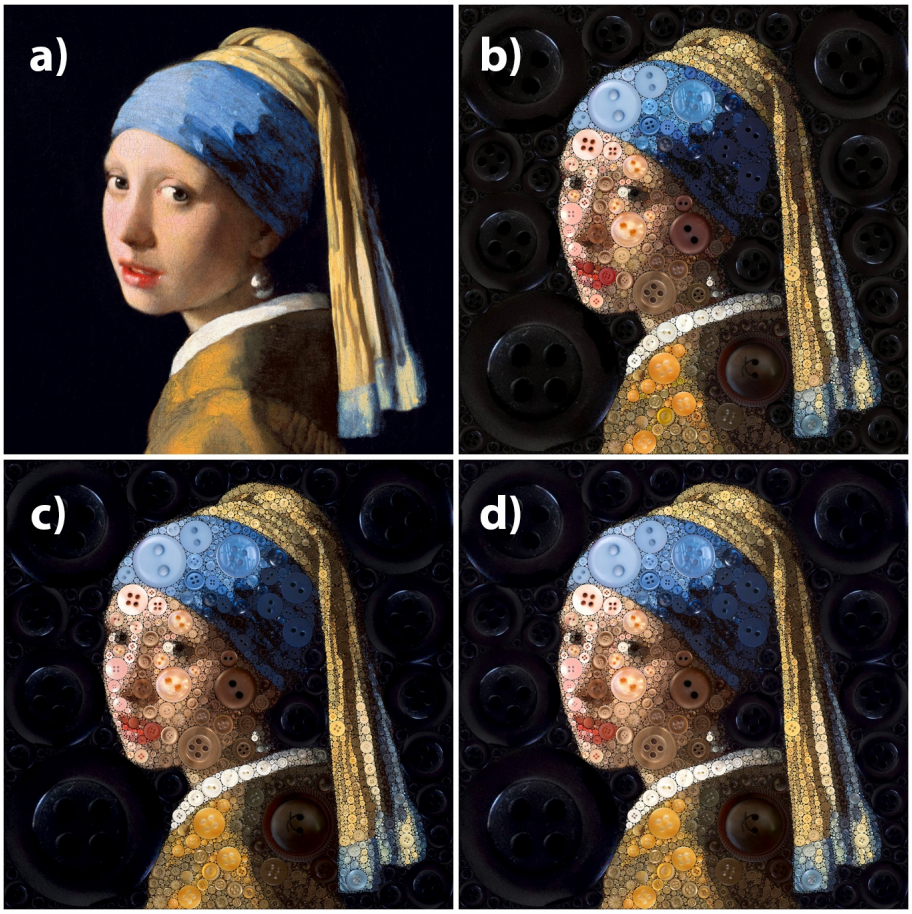

Figure 14 - a) Reference image. b) Uncorrected. c) Color corrected buttons. d) Color corrected buttons and image.

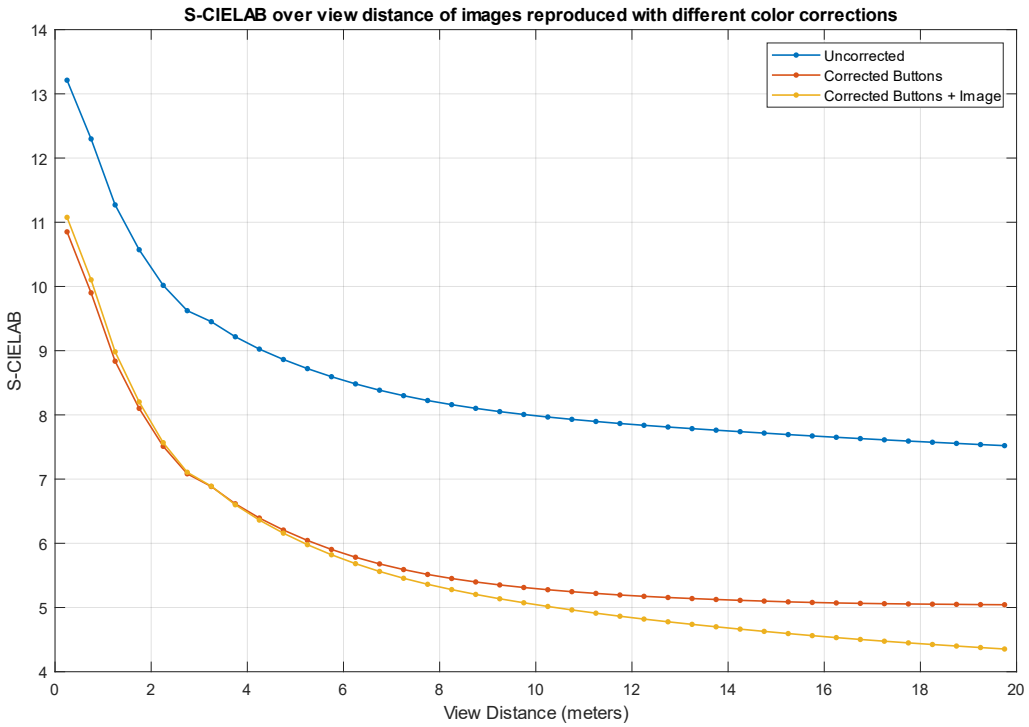

Figure 15 - Mean S-CIELAB of images reproduced with different color corrections. Viewed as if image is 88 centimeters wide (3600x3600, 104ppi) from different view distances

As seen here, the button correction results in an improved reproduction compared to the uncorrected image, and this improvement becomes greater with increasing viewing distance. This is expected since the perceived color samples approaches the mean of image regions with increasing viewing distance because of the low-pass filter used to model the human visual system.

The image with only button correction is slightly better than the image that also corrects the image mean for distances up to about 3 meters. This is probably because the black background pixels influence fewer perceived color samples when viewing up close. The image with corrected image mean becomes increasingly better for view distances above 3 meters however. This is probably because the background pixels influence more of the perceived samples, which causes the perceived samples of the image without image mean correction to be darker than the perceived samples of the reference image.

2.5 Optimizations

If the output of this program were to be translated to the real world, then it may be useful to reduce both the number of unique buttons and the number of total buttons used in the reproduction. The following sections describes such methods and how these influences the reproduction quality.

2.5.1 Unique Button Reduction

A method to reduce the number of unique buttons that the program can pick from when reproducing an image has been implemented. This algorithm attempts to pick the most important buttons needed to reproduce the image, i.e. the ones that minimize the reproduction error. This is done by performing K-means clustering on the reference image in the CIELAB color-space to create K different perceptually self-similar clusters of pixels. For each of these clusters, three dominant colors are generated and the button with the most similar dominant colors are selected from the database. This results in a new reduced button database consisting of K buttons which is used to reproduce the image as usual. The result of using different amounts of unique buttons when reproducing an image is shown in Figure 16 and Figure 17.

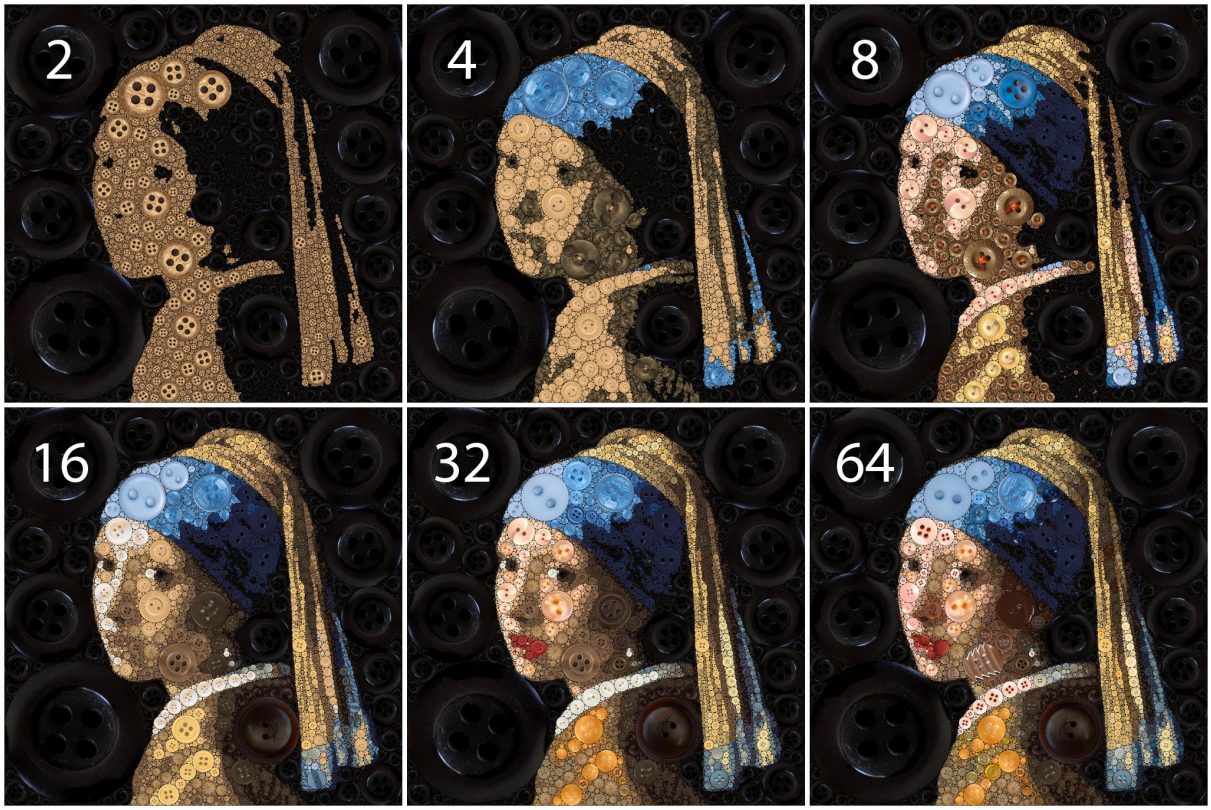

Figure 16 - Images reproduced with different amounts of unique buttons

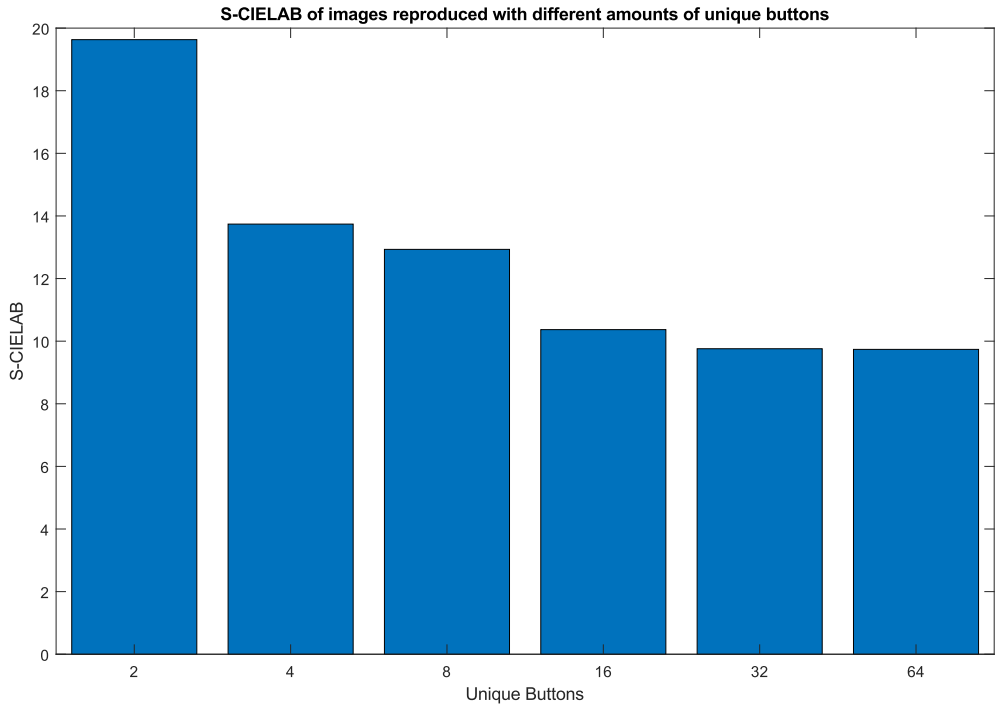

Figure 17 - Mean S-CIELAB of images reproduced with different amounts of unique buttons. Viewed as if image is 88 centimeters wide (3600x3600, 104ppi) from a distance of 3 meters.

As seen here, the results quickly improve with more unique buttons in the beginning, but the improvement for each doubling then quickly diminishes. This shows that about 32 unique

buttons might be enough in this case. Using the entire database results in an improvement of ~ 0.5 units however, and the diminishing improvement is probably partially caused by the fact that it becomes increasingly difficult to find good solutions using K-means with increasing K-values.

2.5.1.1 Button Variety Increase

Methods of increasing the variety of buttons have also been implemented, which has the opposite effect of the previous method. This is done by using a button history buffer of a specified size that keeps track of the buttons that were last used during the mosaic construction. The program then attempts to find the most similar button which is not contained in the button history when matching the next button from the database. This results in suboptimal matches for the benefit of increasing the button variety, which breaks up patterns of the same button. The program only picks a suboptimal button if it is within a specified matching threshold compared to the best match, however.

2.5.2 Total Button Reduction

It is also possible to control the total number of buttons used to reproduce an image by changing the minimum circle radius r_{min} used in the circle packing stage. The result of using different amounts of total buttons when reproducing an image is shown in Figure 18 and Figure 19.

Figure 18 - Images reproduced with different amounts of total buttons. Achieved by setting minimum circle radius to 39, 32, 25, 18, 11 and 4 pixels.

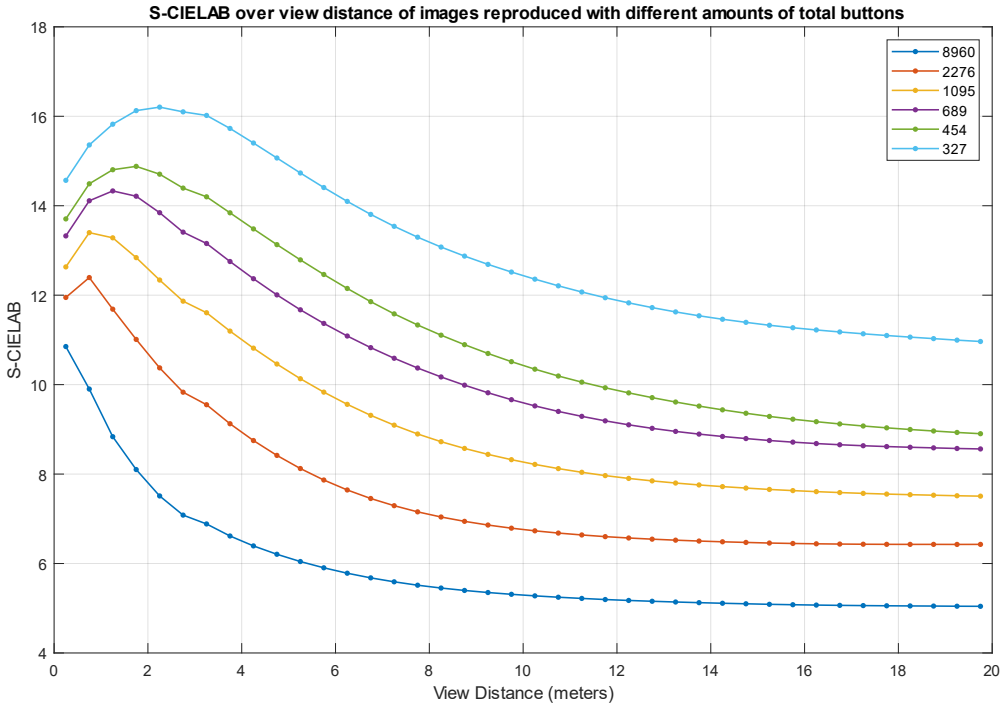

Figure 19 - Mean S-CIELAB of images reproduced with different amounts of total buttons. Viewed as if image is 88 centimeters wide (3600x3600, 104ppi) from different view distances. Achieved by setting minimum circle radius to 39, 32, 25, 18, 11 and 4 pixels.

The number of circles increases quadratically with decreasing minimum circle radius, while the reproduction quality increases more linearly, judged both visually and by looking at the mean S-CIELAB quality metric in Figure 19 for different view distances. It is therefore possible to use this method to greatly reduce the number of total buttons used in the reproduction without losing much quality. This is similar to reducing the resolution in normal images with downsampling.

3 Results and Discussion

The source code and more results can be found on github:

https://github.com/linusmossberg/button-mosaic

Figure 20-22 show some final reproduced results of famous paintings. The mean S-CIELAB value corresponds to a view distance of 3 meters with 104 PPI. The images are 4096 pixels wide, which corresponds to 1 meter with 104 PPI. The time corresponds to the time it took to reproduce the image on a system with an old Intel i7-3770k processor.

The goal when reproducing these images were to make them look as good as possible. No optimizations have therefore been used.

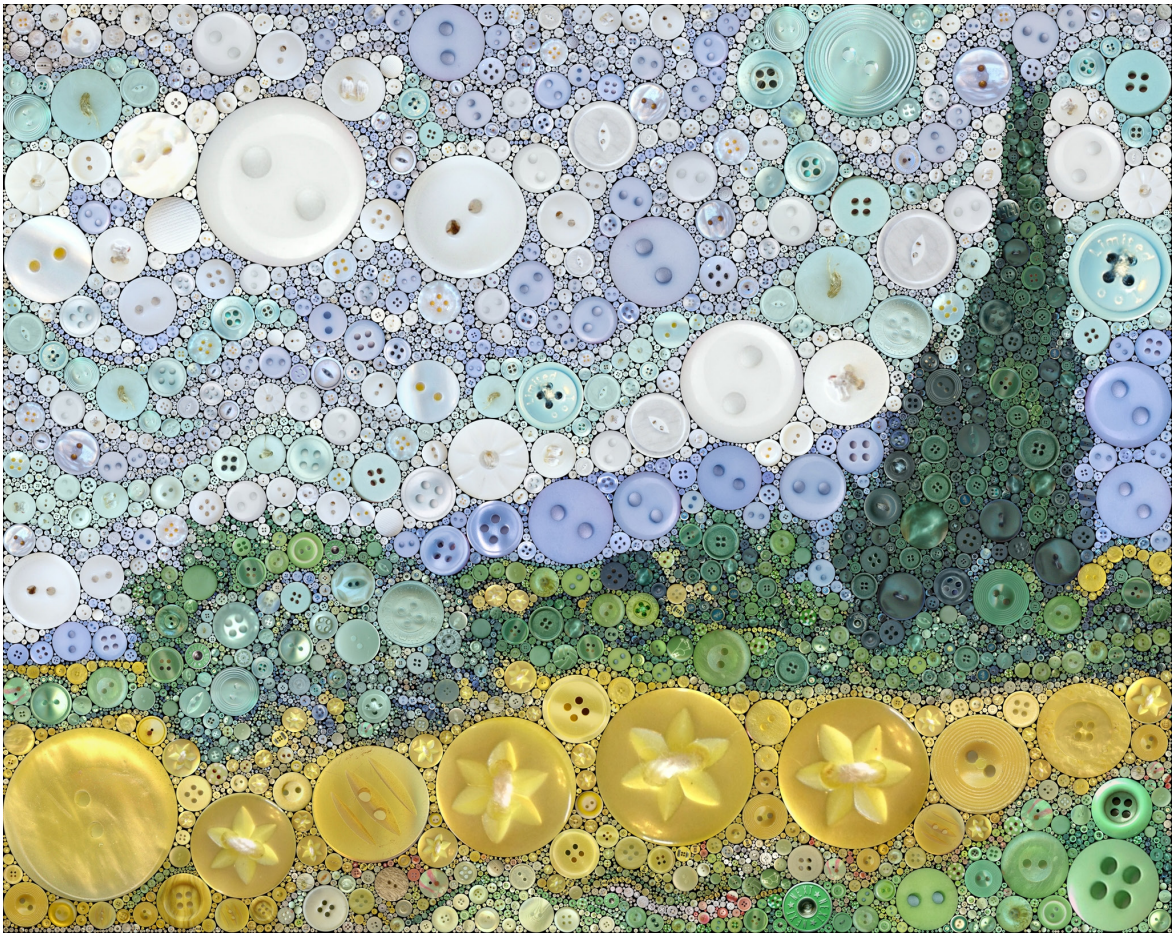

Figure 20 - Wheat Field with Cypresses | Mean S-CIELAB: 8.6595 | SSIM: 0.5213 | Buttons: 12 655 | Time: 46 min

Figure 21 - The Great Wave off Kanagawa | Mean S-CIELAB: 8.2225 | SSIM: 0.5339 | Buttons: 15 808 | Time: 57 min

Figure 22 - The Scream (1910 version) | Mean S-CIELAB: 9.2890 | SSIM: 0.5520 | Buttons: 27 510 | Time: 101 min

3.1 Future Work

There are several things that I have not had the time to experiment with and optimizations that could be made.

It may for example be both faster and more accurate to perform the maximum inscribed circle placement using a polygonal representation of the image segments, and then utilizing Voronoi diagrams to find maximum inscribed circles. This is probably not as suitable for

MATLAB however, but implementing the program in a faster and more open system programming language like C++ is another thing I'd like to do.

An interesting extension that I would like to try is to predefine the physical size of the buttons, rather than scaling the buttons to whatever fits best. This would be simple to implement using the current circle packing method, and it would make the results more realistic and translatable to the real world. Packing objects of arbitrary shape could also be possible by using the major axis of objects. The objects would have to be color matched and selected during the packing stage for this to work, however.

4 REFERENCES

[1] T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition., Springer, 2009.

[2] I. C. Consortium, "Image technology colour management — Architecture, profile format, and data structure," 2004.

[3] T. Calinski and J. Harabasz, "A dendrite method for cluster analysis," Communications in Statistics, 1974.

[4] C. Maurer, R. Qi and V. Raghavan, "A Linear Time Algorithm for Computing Exact Euclidean Distance Transforms of Binary Images in Arbitrary Dimensions,," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003.

[5] A. Mojsilovic, J. Hu and E. Soljanin, "Extraction of Perceptually Important Colors and Similarity Measurement for Image Matching, Retrieval, and Analysis," 2002.

[6] "Wikipedia," 9 Mars 2020. [Online]. Available: https://en.wikipedia.org/wiki/Largest_remainder_method. [Accessed 21 Mars 2020].

[7] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," 1987.

[8] K. Sayood, Introduction to Data Compression, Morgan Kaufmann Publishers Inc, 2005.