Image reproduction using rectangular beads
TNM097

Johnny Elmér
jonel107@student.liu.se

March 9, 2022

Abstract

This report describes a method for creating reproductions of given input images using beads of a rectangular shape, resulting in a checkerboard-esque pattern at close viewing. The method is divided into four main parts: Creating colour palettes (both non-specialised and specialised), creating a reproduction using only colours, masking the reproduction to get the wanted bead-like effect and finally measuring the quality of the reproduction. The resulting reproductions as well as the calculated quality measurements (ΔE and S-CIELAB) are presented and show a lot of potential, but it is clear that there are several possible improvements that could be made.

Contents

1

Introduction

1.1 Background

Reproductions of images can be made in a myriad of different ways and for equally as many reasons. When printing an image an original is being reproduced using dots of ink, most often with the use of half toning, where the small dots will be overlooked from a distance, creating a clear image on the paper. Other times images are reproduced for the sake of entertainment, for example when images are reproduced into pixel art to be used in colour by numbers or cross stitch patterns.

1.2 Aim

This report aims to mix both these concepts and give a method to how an image can be reproduced using rectangular shaped beads (created through binary masking) which, from afar, are not clearly visible. The method presented is split up into four main parts: Creating colour palettes, creating a reproduction using only colours, masking the reproduction to get the wanted bead-like effect and finally measuring the quality of the reproduction. All parts of the method are described and their choices motivated in the following chapter.

1.3 Program used

The program used for the project is MATLAB R2021b together with the Image Processing Toolbox as well as S-CIELAB [1].

2

Method

This chapter describes the method used for the project, such as what type of input images were used, how the colour palettes were created, how the binary masking was done and how the quality measurements were calculated.

2.1 Input images

The input images, as seen in Figure 2.1, were all chosen to have the same height as they did width, this to simplify the quality measurements needing to be calculated later in the project. The images were cropped to 2000x2000 pixels in size.

(a) (b) (c)

Figure 2.1: Names of the input images: (a) "Dino" (b) "Swing" (c) "Tree Gap"

2.2 Colour palettes

This section explains the process of creating the colour palettes used for the reproductions in the project.

2.2.1 Non-specialised colour palettes

The first palette, SpecRgb100, was created by picking out 100 specific colours (using their RGB values) to get as much of an even spread over the colour space as possible, saving each colour in a row of a .mat-file. The second palette, RanRgb100, was made by using a random colour generator online to generate 100 colours, each colour being saved in a row of a .mat-file. Care was taken to not have any repeating colours. The only two colours deliberately chosen were the black and white (0,0,0 and 255,255,255 respectively). The colours were then sorted from lowest to highest value in the red channel. Both palettes can be seen in Figure 2.2.

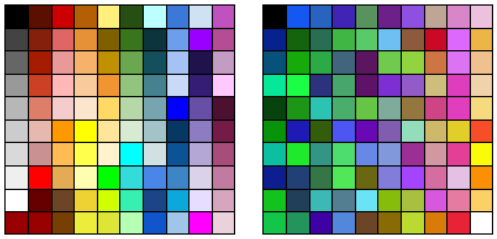


Figure 2.2: The two main colour palettes used in the project, SpecRgb100 to the left and RanRgb100 to the right.

Optimised palettes were then created by taking the two previously made palettes and removing redundant colours. These optimised palettes only contained approximately fifty or twenty of the original colours and were created in two different ways. The evenly distributed palette was optimised by only taking every other or every fifth colour into a new .mat-file (SpecRgb50 and SpecRgb20 respectively), this was done since the colours were already evenly distributed. For RanRgb100, where the colours were only randomly chosen, the process was different. Using a nested for-loop, the ΔE of each colour in the palette was calculated in relation to all the other colours. If the ΔE was smaller than a chosen threshold value, the colour was not saved into the new palette. The threshold value was adjusted manually until the number of colours in the new palette was as close to fifty or twenty as possible, because of this both RanRgb51 and RanRgb21 contain one more colour than its evenly distributed counterpart. The resulting colour palettes mentioned in this section can be seen in Figure 2.3 and 2.4.

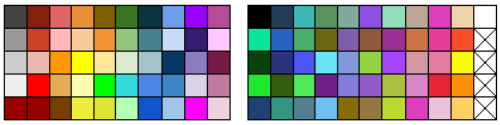


Figure 2.3: Two of the optimised colour palettes, SpecRgb50 to the left and RanRgb51 to the right.

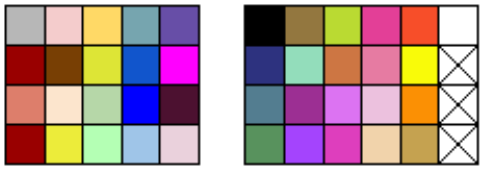
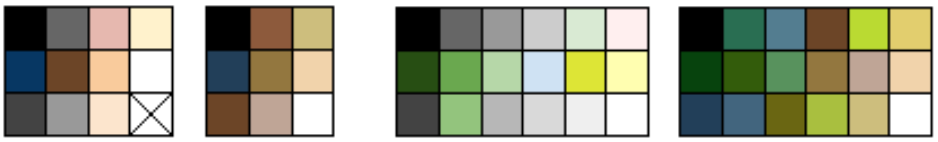


Figure 2.4: The last two of the non-specialised colour palettes, SpecRgb20 to the left and RanRgb21 to the right.

2.2.2 Specialised colour palettes

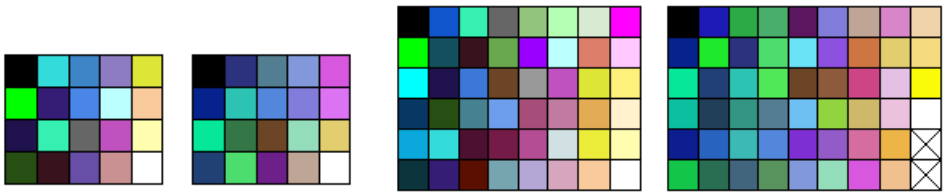
Other than the colour palettes mentioned in 2.2.1 several specialised colour palettes were created. Similarly to the optimised colour palettes these take the main colour palettes (SpecRgb100 and RanRgb100) and create smaller palettes, but instead of comparing the colours to other colours in the palette during the optimisation the focus is to use the input image as a reference. To get a list of colours that was smaller than the original 100, MATLAB's *rgb2ind* function is used, given the input image and the amount of colours (N) wanted as input. The resulting colourmap is then traversed with a nested for-loop, comparing each colours ΔE value to the colours in palette, the colour with the smallest ΔE value being saved into a separate matrix (similar to what is described in more detail in the next section). The matrix containing the best matching colours is then traversed again to remove any duplicate colours and then saved as a .mat file.

Because the input images contain large fields of similar colours and the palettes (see Figure 2.2) contain only a small amount of colours there will be a lot of repetition. Furthermore, since some of the input images have a fairly low variation in colour, the resulting palettes will not always contain as many colours as the N value suggests. The resulting palettes are therefore for the most part rather small, as seen in Figure 2.5 and 2.6. This process was done for three different images, which gave a total of six specialised colour palettes. These images are shown in 3.



(a) (b)

Figure 2.5: (a) Dinosaur Spec11 & Ran9 (b) Tree Gap Spec18 & Ran18



(a) (b)

Figure 2.6: (a) Swing Spec20 & Ran20 (b) Swing Spec48 & Ran52

2.3 Reproduction using only colours

The chosen input image is first read into MATLAB using *imread*, then the wanted palette (as a .mat file) is loaded in and put into a variable called *palette*. Each value in *palette* is then divided by 255 to get it between 0 and 1 before converting the values into CIELAB values using *rgb2lab*. The CIELAB values are saved into a separate variable called *paletteLab*, since *palette* is needed at a later stage, it is not altered in any way.

The height and width was set to 200x200 and 20x20 for the ”reproduction” and for the bead respectively. The size of the beads were chosen in such a way that the shape of beads in the final reproduction image would only be clear upon closer inspection. Using the values given a matrix of ones (*reproduction*) is created using the *ones* function with the height set as the height of the ”reproduction” multiplied with the height of the bead, (the same being done for the width), as well as three channels for the RGB values. The input image is then resized to be the wanted size (200x200) using *imresize* and then converted into CIELAB using *rgb2lab*, the result is saved into a variable called *imLab*.

Using a nested for-loop each pixel of *imLab* is traversed and compared with each colour in *paletteLab* using ΔE and saved into a vector called *difference*. The index of the smallest value in *difference* is found using *find* and sent together with *palette*, the bead height and bead width into a function called *makeBeadBox* which creates a matrix with the height and width given and the colour at the index given in *palette*. The matrix given is saved in a variable called *beadBox* and then placed into *reproduction*.

2.4 Masking

After filling *reproduction* with colours from *palette* the image is masked using a binary image, this section explains the process of creating and applying it on the image. (The bead and mask are created inside a function called *createBinaryMask*.)

The first step in creating the mask is to create the actual bead shape. First a 2D matrix called *bead* is created using MATLAB's *zeros* function together with the given height and width of the bead. A for-loop is then used which goes from one to half the bead height. In the first iteration the middle uppermost and lowermost pixel/pixels are set to one (depending on if even or uneven number of pixels in the bead's width), as the square shape is created from the bottom and top to the middle at the same time. For each iteration the indexes of the matrix is either increased or decreased so that fewer or more pixels are being converted into ones at each row. The final result can be seen in Figure 2.7.

Figure 2.7: The resulting bead.

When the bead is created a 2D binary image is created using MATLAB's *zeros* function with the height and width of *reproduction*. A nested for-loop is then used to, similarly to how the *beadBox* was put into *reproduction*, fill the binary image with beads. The finished binary mask is then applied onto the *reproduction* using $binaryMask. * reproduction + \ binaryMask * B$; where B dictates the intensity of the background. For images which are mainly on the darker side $B = 0.1$ was chosen to make the background near black and for lighter images $B = 0.8$ was chosen to make the background near white in colour. B is chosen manually to make the image look closer to the original from a subjective standpoint but it was considered to instead chose B by looking at the input image and calculating it's luminance, but this process was not implemented because of time constraints.

2.5 Quality measurements

This section explains the process of calculating the different objective quality measurements.

2.5.1 ΔE

The masked image, which in the code was called *finalReproduction*, was sent to a function called *checkQuality* together with the original image. To calculate the ΔE both images need to be the same size and in the CIELAB colour space, which leads to the original image having to be scaled using MATLAB's *imresize* using the bicubic method. The reproduction cannot be scaled down as it risks destroying the rectangular shapes of the beads, since the scaling is around a factor of two in each direction, scaling the original image should not impact the comparison too much. After scaling both images are converted from RGB to CIELAB values using MATLAB's *rgb2lab*, allowing for the ΔE to be calculated. The maximum and mean differences are then returned, the mean being the more interesting one.

2.5.2 S-CIELAB

In the same function as ΔE (*checkQuality*), full-reference S-CIELAB [1] is also calculated. The *finalReproduction* and the original image are converted into XYZ values (the original image being scaled up before doing so). The function used is the *scielab* function written by Xuemei Zhang in the mid to late nineties. The function takes five inputs, the samples per degree wanted, the original image in XYZ values, the reproduction in XYZ values, the white point and 'xyz'. The white point used is that of a $D65$ illuminant ($D65 = [95.05, 100, 108.9]$) as is standard and the samples per degree is calculated using $sampPerDeg = ppi * distance * tan(\pi/180)$; where ppi is the number of samples per inch, $distance$ is the viewing distance in inches. For this calculation ppi was set to 300 and $distance$ was set to 40 inches, taking into account that the viewers distance from the screen. The mean value of the return value of the *scielab*-function was then calculated using MATLAB's *mean*, which was then returned together with the max and mean values of the ΔE.

2.6 Saving the results

The resulting image is saved using MATLAB's *imwrite*-function and values given by the quality measurements written down and recorded in tables. This process is repeated for each image and for each palette used with said image.

3

Results

In this chapter the final results of the reproduction are shown as well as the resulting quality measurements. To avoid this section becoming far too long only the results of reproducing the Dinosaur image is shown. The results from the Swing and Tree Gap image using non-specialised and specialised palettes can be seen in Appendix A and B respectively.

The best result for the Dinosaur image (objectively speaking) came from the use of palette RanRgb21 (according to S-CIELAB), with DinoRanRgb9 being a close second. The subjective opinion of the author of this report agrees with the objective results to a degree, but he does not see a clear raise in quality from DinoRanRgb9 to RanRgb21 as the quality measurements would suggest.

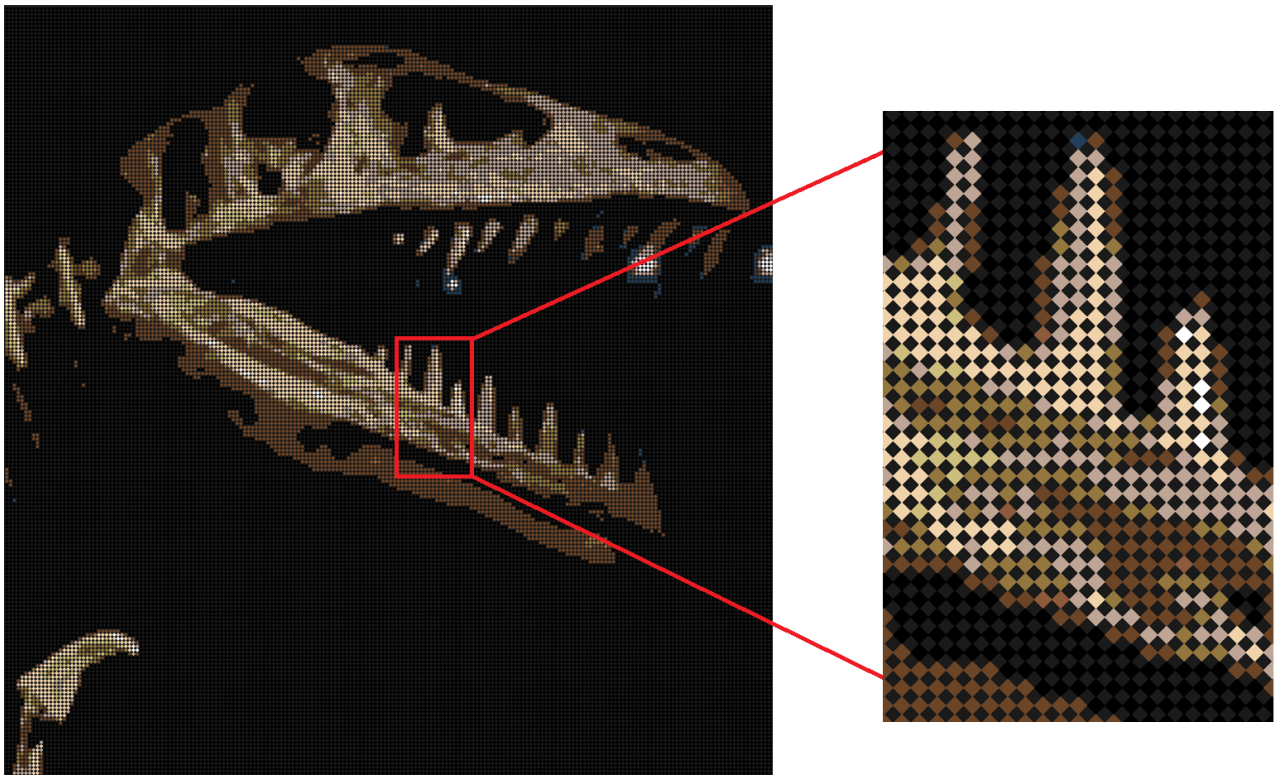
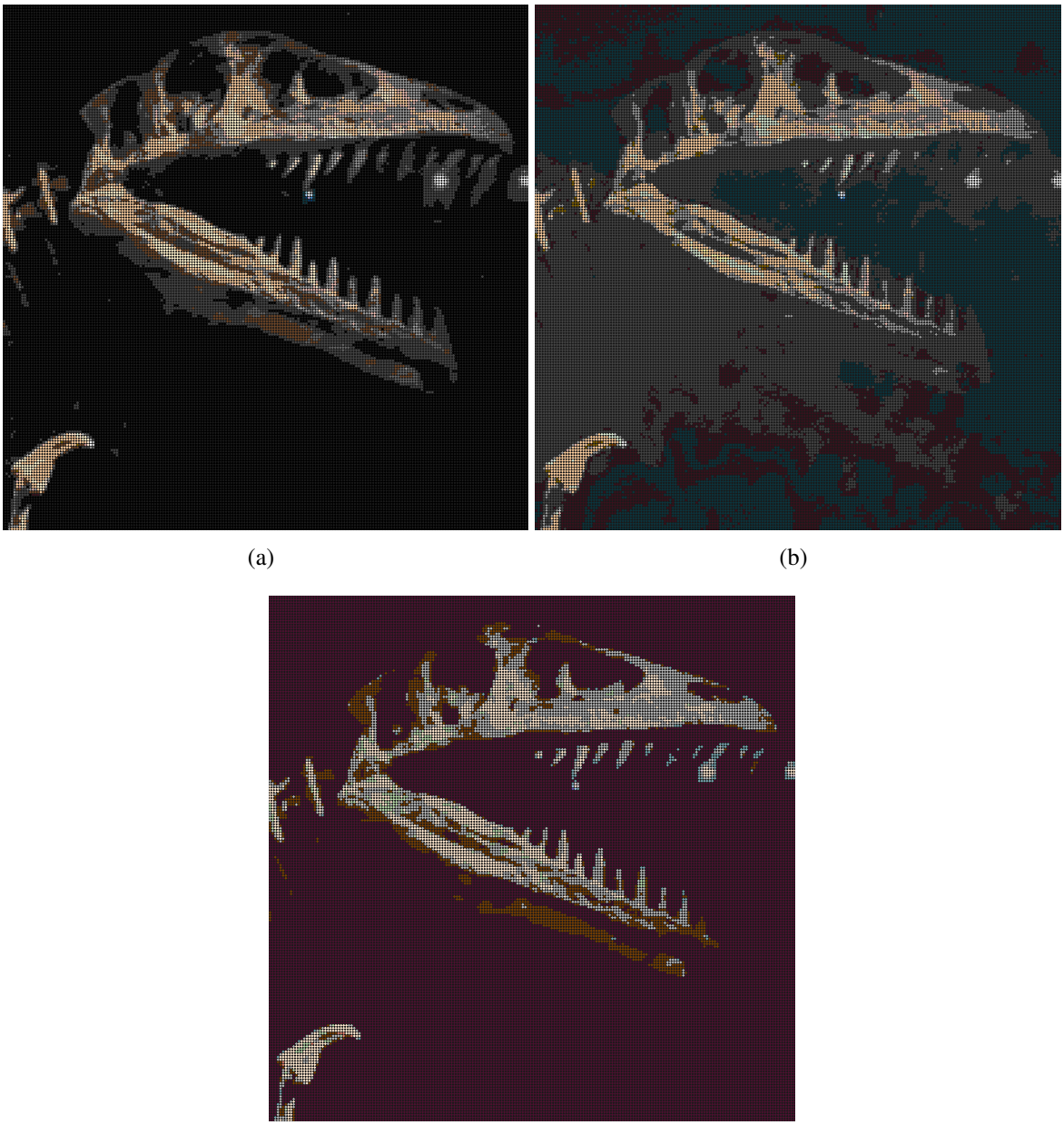


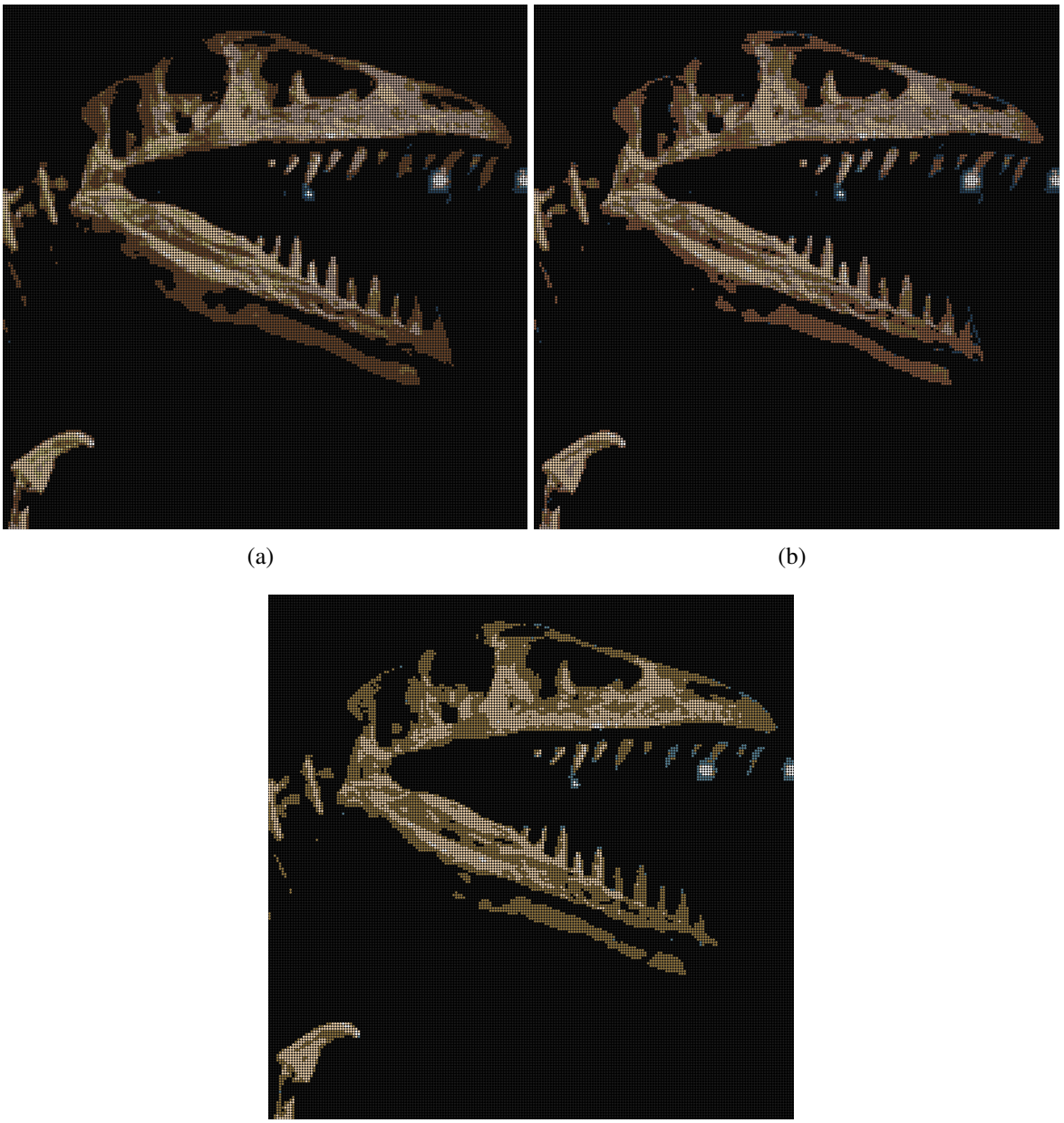
Figure 3.1: Zoomed in image of the final reproduction to show the beads clearly.



(a) (b)

(c)

Figure 3.2: Non-specialised non-random palettes: (a) Spec100 (b) Spec50 (c) Spec20



(a) (b)

(c)

Figure 3.3: Non-specialised random palettes: (a) Ran100 (b) Ran51 (c) Ran21

(a)

(b)

Figure 3.4: Specialised palettes: (a) Spec11 (b) Ran9

Palette \Quality	Mean $\triangle$E	S-CIELAB
SpecRgb100	12.0489	0.3562
RanRgb100	12.0952	0.3420
SpecRgb50	17.1098	0.4852
RanRgb51	12.3143	0.3505
SpecRgb20	24.0234	0.6451
RanRgb21	12.5514	0.3323

Table 3.1: Quality measurement results using non-specialised colour palettes.

Palette \Quality	Mean $\triangle$E	S-CIELAB
Dino_SpecRgb11	12.0503	0.3556
Dino_RanRgb9	12.0968	0.3419

Table 3.2: Quality measurement results using specialised colour palettes.

4

Discussion

This chapter describes some of the practical problems with the method and how they could possibly be improved. Some issues relate to the speed of the calculations while others relate more to the choice of colour palettes or sizing.

4.1 Practical problems

There are a few practical problems that are important to note with this method. One of these is that the input images need to be cropped to have the same height as they have width for the sake of the quality measurements actually being able to work, which means that a lot of images are impossible to use unless only the visual aspect is wanted.

When it comes to the topic of size, two other problems arise: The first problem is the size of the beads, the smaller the size of the beads the higher the risk of their shape being ”messed up”, giving the results a completely different look. The second problem is the size of the palettes, the larger the palette used the longer the code takes to process, this is mostly because of how the colour picking is done when creating the first reproduction.

The final problem arises when doing the quality measurements. To be able to accurately measure the difference in quality the reference image (the input image) has to be the same size as that of the reproduction. Because of the rectangular shape of the reproductions beads, resizing it could lead to the shape changing in an unwanted way, because of this the reference image is instead scaled up using a bicubic *imresize*. Resizing the original image is in no way ideal, but is allowed because it is only done by the scale of two, however this could possibly be avoided by simply changing the shape of the bead.

4.2 Possible improvements

Some possible improvements which could help with some of the issues mentioned in the previous section are as follows: The bead could be saved as an image and immediately be put into the reproduction instead of doing it in a separate step through binary masking, which could possibly help making the run time faster. The shape of the bead could also be changed from a rectangular one to

a rounded one, which would negate the issue of resizing the original, since the reproduction's beads have less risk of being warped when sized down.

The way in which the SpecRgb50 and SpecRgb20 are done currently, the first colour of SpecRgb100 is skipped, and therefore there is no black in the resulting palettes. The problem with this is most obvious when looking at the Dinosaur image, as the background becomes rather strange looking (see Figure 3.2 b and 3.2 c). If instead the first colour was saved this issue could have been avoided, but it is hard to say if this would affect images on the lighter side (like Tree Gap) without testing it.

Another aspect connected to the colours in the project is how the images are chosen. In this method the background colour of the reproduction, which is decided by the value of B in the binary masking step, is not taken into account when choosing the colour during the creation of the reproduction. This means that the resulting image can become up to 50% lighter or darker than the original, depending on what background intensity is chosen. This of course affects the final result of the quality measurements and is therefore not optimal. An improvement would be to take this aspect into account when choosing the matching colours, compensating for the background by either picking lighter or darker colours rather than the closest match.

Some other interesting possible improvements, which could lead to further issues unless compensated, are as follows: A larger original palette to get larger optimised palettes for images which are fairly monotone like the Dinosaur or Tree Gap image, as well as a function that loaded the wanted colour palette without having to clutter up the main as much. (As of now the main function is rather cluttered with palettes being loaded in or not by simply commenting out those not wanted. To see the code in all its cluttered glory, please visit the following link: `https://github.com/JohnnyElmer/TNM097-project`)

5

Conclusion

The binary mask works well for getting the wanted effect in the reproduction but the choice of rectangular beads brings with it problems that could have been avoided with a different shape, more specifically a round one.

The specialised palettes also function well and create subjectively and objectively good results, which in turn leads to less colours having to be used during comparisons. However, the differences in S-CIELAB values and differences in the subjective comparisons are in some cases not all that large, and because of this the extra step of creating specialised palettes may be unnecessary if it is not crucial to speed up the process as much as possible.

The method in general is a good base to work off of but could benefit greatly from being worked with and expanded further, both with the possible improvements described in this report and in ways the author of this report has not personally thought of. But for now, the results are satisfactory and rather enjoyable to look at.

Bibliography

[1] Xuemei Zhang, D.A. Silverstein, J.E. Farrell, and B.A. Wandell. Color image quality metric s-cielab and its application on halftone texture visibility. In *Proceedings IEEE COMPCON 97. Digest of Papers*, pages 44–48, 1997.

Appendix A

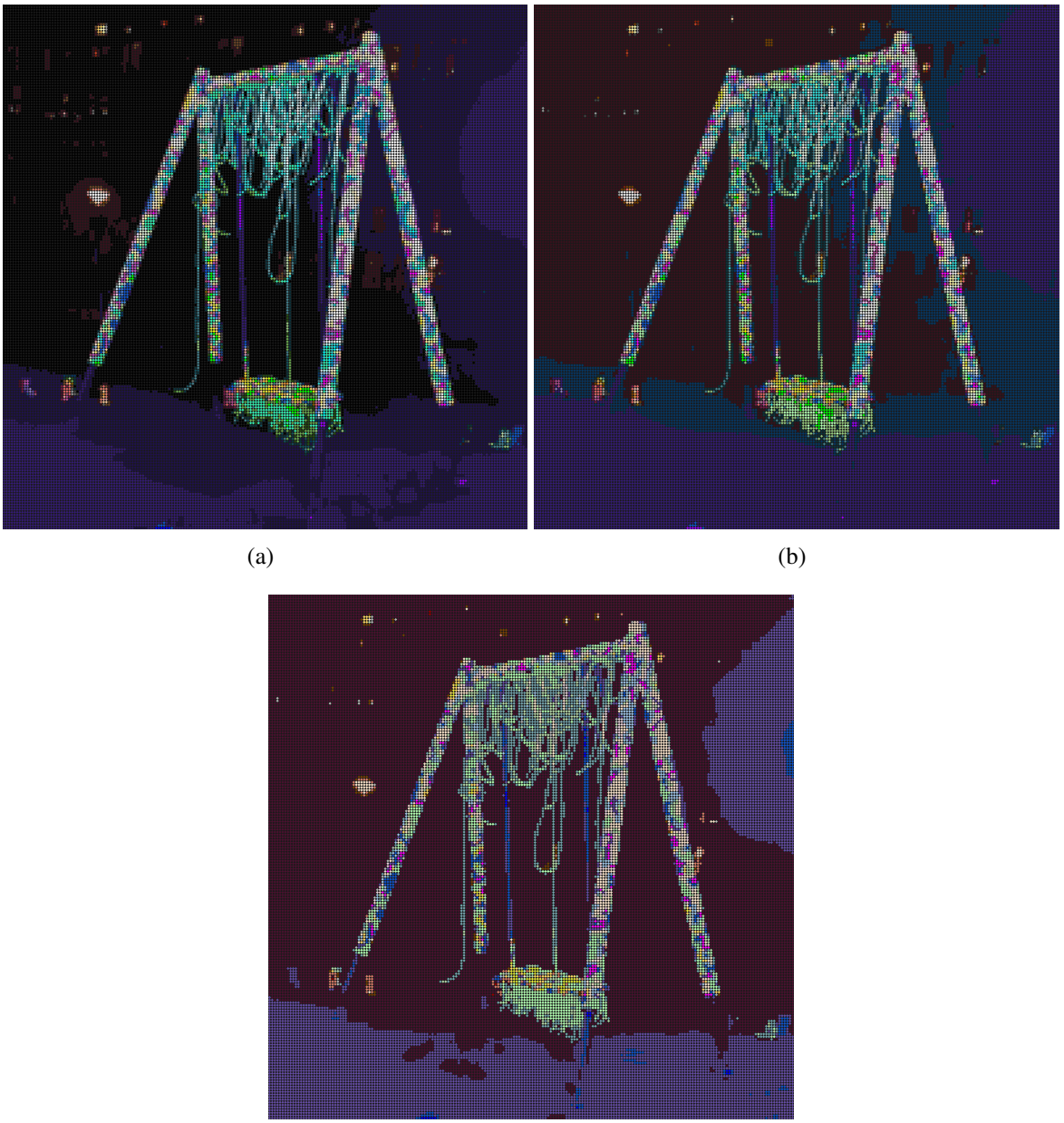
Non-Specialised Colour palette results

Palette \Quality	Mean $\triangle$E	S-CIELAB
SpecRgb100	26.0055	0.7686
RanRgb100	28.1288	0.8735
SpecRgb50	29.0871	0.8446
RanRgb51	28.6154	0.8559
SpecRgb20	34.3108	1.0875
RanRgb21	29.1121	0.8982

Table A.1: Quality measurement results for the Swing image using non-specialised palettes.

Palette \Quality	Mean $\triangle$E	S-CIELAB
SpecRgb100	29.0850	1.9779
RanRgb100	30.3461	2.0431
SpecRgb50	31.5967	2.1517
RanRgb51	30.5597	2.0481
SpecRgb20	37.7965	2.5730
RanRgb21	32.6228	2.1850

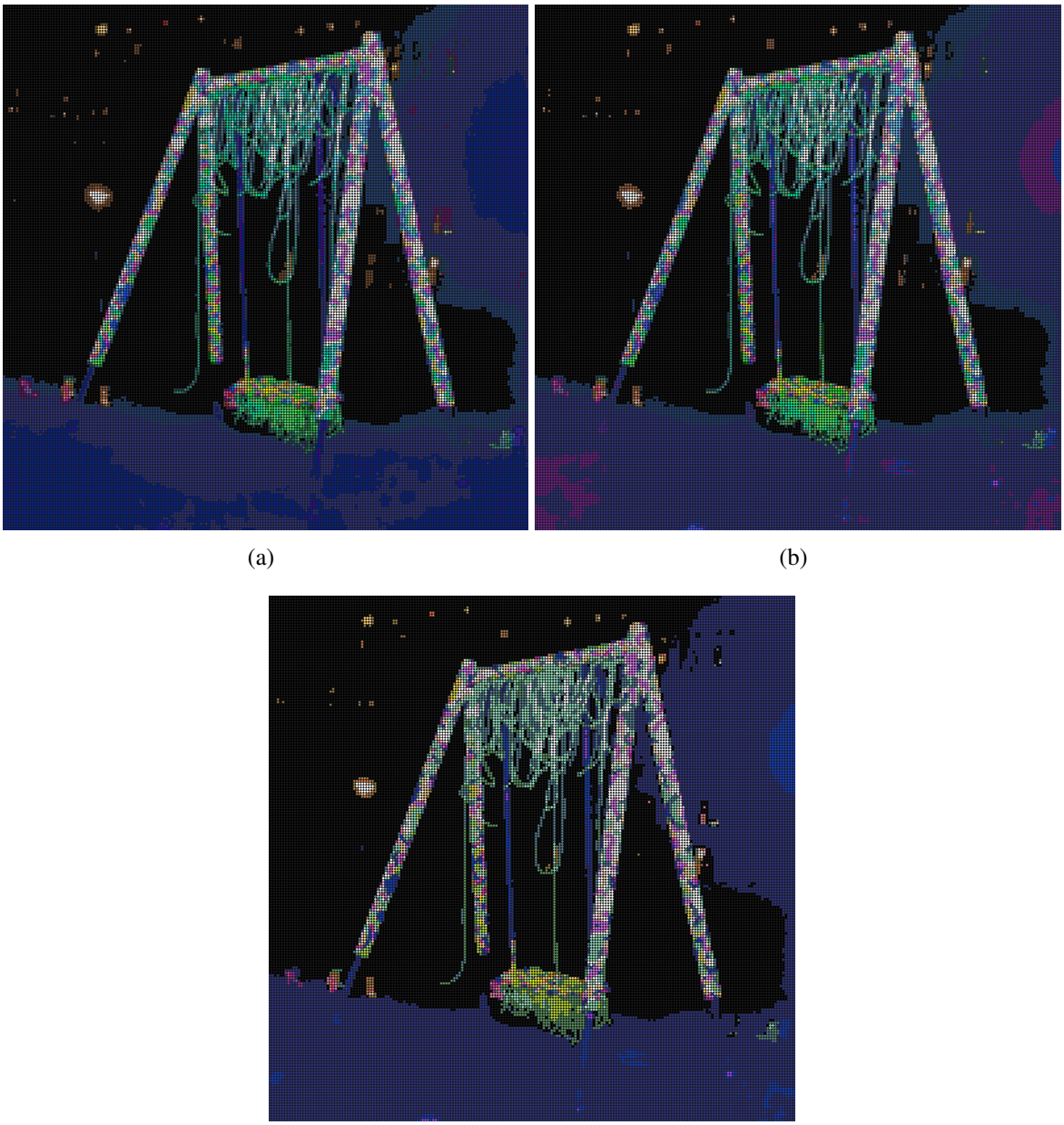
Table A.2: Quality measurement results for the Tree Gap image using non-specialised palettes.



(a) (b)

(c)

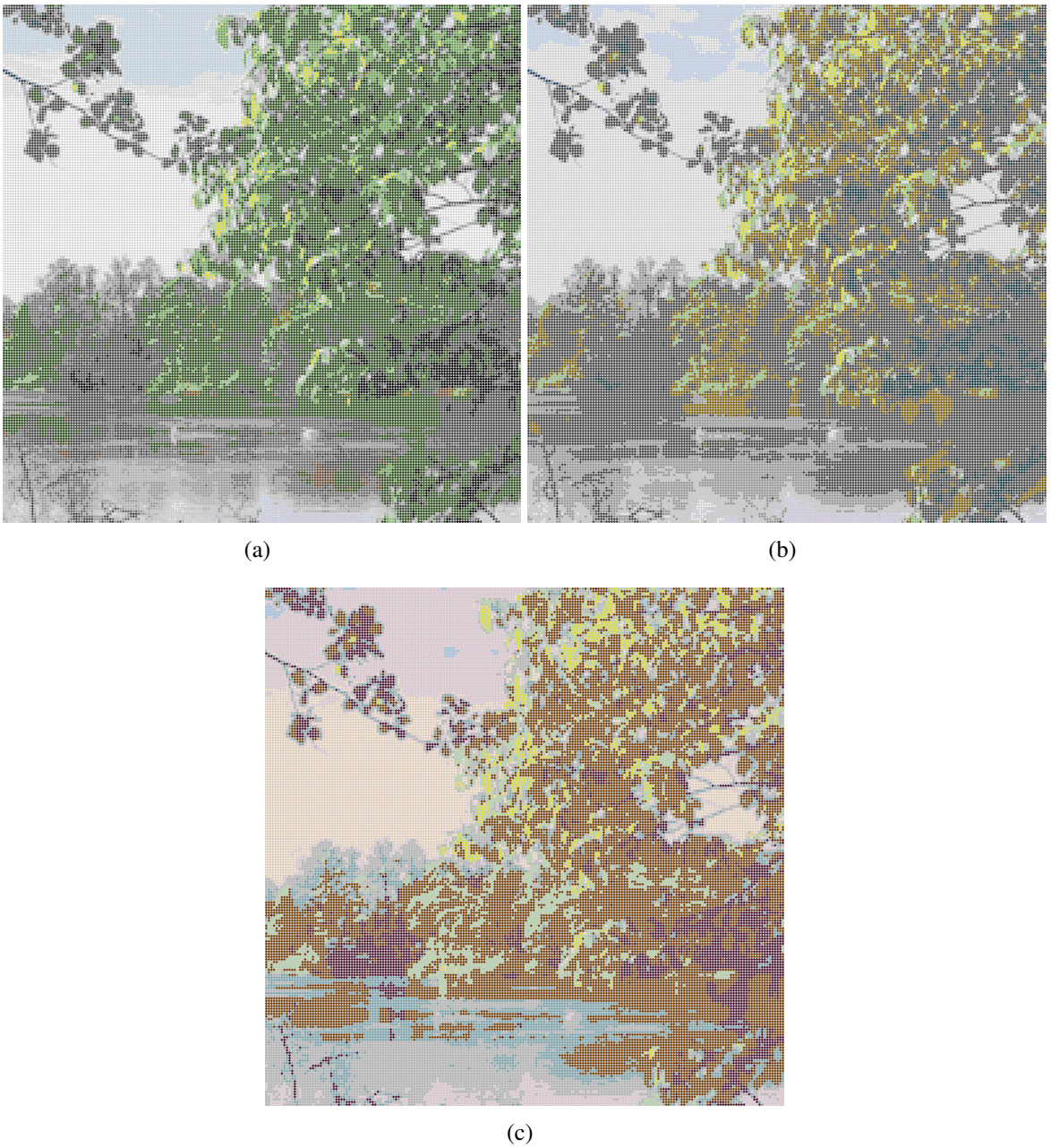
Figure A.1: Swing Non-Specialised non-random: (a) Spec100 (b) Spec50 (c) Spec20



(a) (b)

(c)

Figure A.2: Swing Non-Specialised random: (a) Ran100 (b) Ran51 (c) Ran21



(a) (b)

(c)

Figure A.3: Tree Gap Non-Specialised non-random: (a) Spec100 (b) Spec50 (c) Spec20

(a) (b)

(c)

Figure A.4: Tree Gap Non-Specialised random: (a) Ran100 (b) Ran51 (c) Ran21

Appendix B

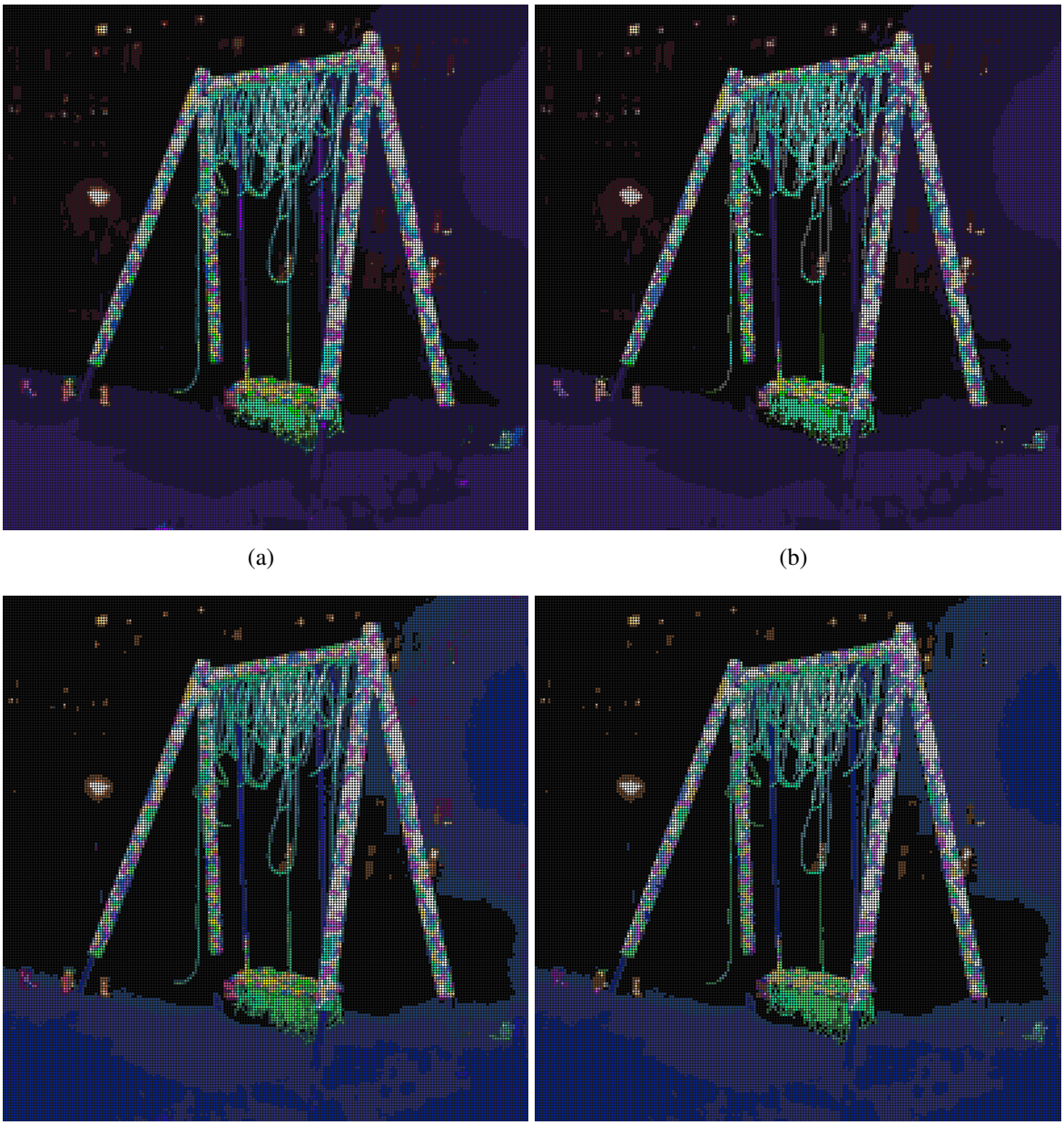
Specialised Colour palette results

Palette \Quality	Mean $\triangle$E	S-CIELAB
Swing_SpecRgb48	26.0287	0.7680
Swing_RanRgb52	28.1570	0.8730
Swing_SpecRgb20	26.2310	0.7735
Swing_RanRgb20	28.3882	0.8881

Table B.1: Quality measurement results for the Swing image using specialised palettes.

Palette \Quality	Mean $\triangle$E	S-CIELAB
TreeGap_SpecRgb18	29.0939	1.9806
TreeGap_RanRgb18	30.3482	2.0502

Table B.2: Quality measurement results for the Tree Gap image using specialised palettes.

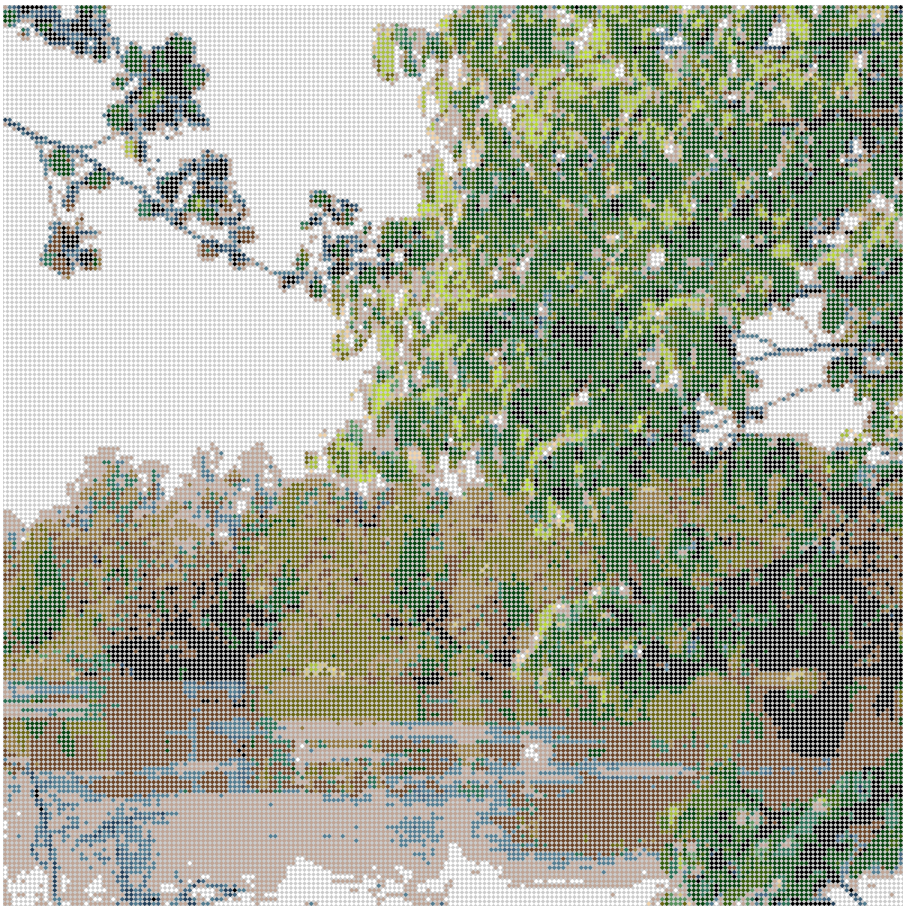


(a) (b)

(c) (d)

Figure B.1: Swing Specialised: (a) Spec48 (b) Spec20 (a) Ran52 (b) Ran20

(a)



(b)

Figure B.2: Tree Gap Specialised: (a) Spec18 (b) Ran18