Pixel Art Generator

Image Processing Project

Arno De Haseleer
Etro
VUB
Brussels, Belgium
arno.ghislain.de.haseleer@vub.be

Rogier De Nys
Etro
VUB
Brussels, Belgium
rogier.de.nys@vub.be

Warre De Winne
Etro
VUB
Brussels, Belgium
warre.annie.j.de.winne@vub.be

Sam Mousavi
Etro
VUB
Brussels, Belgium
sam.mousavi@vub.be

***Abstract*—This paper discusses the implementation of a pixel art generator in MATLAB using the algorithm constructed in (Gerstner et al., 2013, p. 3). The pixel art generator converts a normal image to a pixelated image of a user defined size. It constructs a color palette based on the maximum allowed amount of colors and the colors of the original image. Clean borders of the resulting image are obtained using a clustering algorithm.**

I. Introduction

This paper discusses the implementation of this algorithm constructed in MATLAB. The first part discusses how the algorithm itself is constructed and how it is implemented in MATLAB. The second section discusses the added GUI and user controls. Section IV then discusses the obtained results from the generator. Finally a conclusions are made and discussed in Section V. Note that any explanation and reasoning regarding the construction of algorithm itself can be found in (Gerstner et al., 2013, p. 3).

II. Algorithm

This section explains the algorithm in four steps: *initialisation*, *superpixel refinement*, *palette refinement* and *palette saturation*. The algorithm is iterative, which means that the *superpixel refinement* and the *palette refinement* steps are repeated until the temperature T reaches a value below 1.

A short overview of the used terms:

- *palette*: limited collection of colors used to color the pixelated image. The palette size, the maximum amount of colors allowed in the output image, is defined by the user.
- *superpixel*: partition of the input image which has a center relative to the input image and color. The color is, for this implementation, one of the palette colors. The amount of superpixels is again defined by the user.
- temperature T: a parameter used in the palette refinement step that determines the current color clustering variance. The name has been chosen in analogy with the physical process of annealing. This value starts out high, and is progressively lowered.

Note that the algorithm uses the CIELAB color space.

A. Initialisation

The first part of the algorithm consists of preparing the input image for the algorithm. The initialisation consists of three steps: initialising the superpixels, initialising and assigning the palette and calculating the initial temperature T.

The first step assigns each pixel of the input image to a superpixel. The amount of superpixels is equal to the amount of pixels of the pixelated image. The centers of these superpixels are initialised in a grid across the input image. Each pixel of the input image is then assigned to the superpixel that is the closest to it in euclidean distance. This results in a rectangular superpixel grid, where each input pixel has been assigned to the resulting output pixel in its place.

The second step initialises the palette with only one color. This color is equal to the mean of the pixels colors of the input image. Each superpixel gets assigned this color. $P(c_k)$ is the probability that color c_k is assigned to any superpixel, this is equal to one since only one color c_0 is present. Note that the actual palette starts with two colors that are the same. These two colors are the subclusters of the first palette color.

Finally, the temperature T is set to $1.1T_c$, where T_c, the critical temperature, is calculated as twice the variance along the major principal axis of the set in the LAB space. This is computed in MATLAB by first calculating the variance of the input image times the highest value obtained from the `pca()` function for each color component in the LAB space. The norm of those three variances is then calculated to get T_c.

B. Superpixel refinement

Before the superpixel refinement takes place, the colors in the palette are disturbed by a disturbance factor $e_{perturbation}$ if the palette has not reached its maximum amount of colors. If it has, the disturbance factor will be equal to $e_{perturbation}/100$.

The superpixel refinement algorithm is a modified version of SLIC (Simple Linear Iterative Clustering). This algorithm tries to minimize a certain norm, which in our case represents the distance in the combined color and (x, y)-space between a pixel p_i of the input image and a superpixel p_s. This norm is given by:

$$d(p_i, p_s) = d_c(p_i, p_s) + m\sqrt{\frac{N}{M}}d_p(p_i, p_s) \qquad (1)$$

d_c is the distance in the color space between the input pixel p_i and the superpixel p_s. This is calculated by taking the norm of the difference between the input pixel color and superpixel color in the LAB space. Note that the superpixel color used here is the palette color that is assigned to that superpixel. d_p is the distance in the (x, y)-space between the input pixel and the center of the superpixel. N is the amount of superpixels, M is the amount of pixels of the input image and m is a weight that controls the importance of the distance versus the color when minimising (1). In this case $m = 45$ is chosen due to the use of the palette colors. A high m gives more importance to the distance and less to the color.

After the assigning of the input pixels to superpixels, Laplacian smoothing of the centers of the superpixels is performed. This smoothing improves the correspondence between the superpixel and output pixel neighbourhoods. It moves the center of each superpixel 40% relative to the distance between the center of the superpixel and its 4-connected neighbors (see (2)).

$$SP_c = 0.6\, SP_c + 0.4\, \text{mean}(SP_{c,\,neighbors}) \tag{2}$$

With SP_c the center of the current superpixel and $\text{mean}(SP_{c,\,neighbors})$ the mean of the positions of the centers of the 4-connected neighbors of the current superpixel. Note that the superpixels at the edge of the image are not moved since they only have two or three connected neighbors.

Finally, the average color of each superpixel is calculated by taking the mean of the colors of the input pixels assigned to that superpixel. These average colors are then bilaterally filtered to avoid problems with color gradients (Gerstner et al., 2013, p. 5). This filtering is implemented in MATLAB by using the `imbilatfilt()` function. The colors that result from this filtering are used for the next step of the algorithm.

C. Palette refinement

This part is based on MCDA (mass constrained deterministic annealing). The palette refinement consists of three steps: associate superpixels to colors in the palette, refine the palette and then expand the palette. The expansion of the palette is only performed when the palette has converged for the current temperature T. After expansion, the temperature is lowered.

The superpixels are associated with colors in the palette according to

$$P\left(c_k \mid p_s\right) = \frac{P\left(c_k\right) e^{-\frac{\|m'_s - c_k\|}{T}}}{\sum_{j=1}^{K} P\left(c_j\right) e^{-\frac{\|m'_s - c_j\|}{T}}} \tag{3}$$

This equation calculates the probability that a superpixel p_s is associated with a color c_k in the palette. m'_s is the mean color of the superpixels after bilateral filtering (obtained in the previous step). $P(c_k)$ is the probability that color c_k is assigned to any superpixel and is given in (4). T is the current temperature.

$$P\left(c_k\right) = \sum_{s=1}^{N} P\left(c_k \mid p_s\right) P\left(p_s\right) \tag{4}$$

$P(p_s)$ has a uniform distribution here and is equal to $\frac{1}{N}$, with N the number of superpixels. Note that for every iteration, first $P(c_k \mid p_s)$ and then $P(c_k)$ is calculated for every superpixel and palette color. After these probabilities have been calculated, the palette color that maximizes $P(c_k \mid p_s)$ is assigned to each superpixel.

The next step is refining the palette. This is done by recalculating the palette colors c_k using (5).

$$c_k = \frac{\sum_{s=1}^{N} m'_s P\left(c_k \mid p_s\right) P\left(p_s\right)}{P\left(c_k\right)} \tag{5}$$

This equation recalculates the palette colors based on the weighted average of all the superpixel mean colors, using the probability of the association with that color.

The expansion step occurs when the palette has converged for the current temperature T. This means that the norm of the palette colors did not change much compared to the previous iteration. This maximum change is set by the $e_{palette}$ parameter. If the norm is smaller than $e_{palette}$, then the palette has converged. If the palette has indeed converged, the temperature is lowered by a factor of 0.7.

Then it is checked whether the palette can expand. If the palette has reached its maximum amount of colors chosen by the user, no palette expansion takes place. If it has not reached that maximum amount, the possibility of splitting a color of the palette in two is checked. This is done by checking if the two subclusters of any palette color have diverged enough. This divergence is calculated by taking the norm of the two subcluster colors. If this value is above a certain value, $e_{cluster}$, then the palette color is split into two palette colors. After the split, a slight perturbation to the split colors is added such that they are distinctly different.

D. Palette saturation

After the temperature T has reached a value smaller than one, the iterating is stopped. As a final step, the palette colors are saturated. This is done by multiplying the a and b channels of the LAB color space with a saturation factor equal to 1.1.

III. GUI and user controls

A simple GUI was created using the MATLAB *App Designer* tool and can be seen in Fig. 1. It contains a button that opens the file explorer of the operating system where the user can select the file for conversion. Then the user inputs the size of the wanted pixelated image into "output height" and "output width". The palette size is also chosen. Note that for these inputs, error messages occur when they do not have the correct value. The converter will also not run if these values are not correct, or no input image is selected.

IV. Experimental results

Converted images and their original images are shown in Figs. 2 & 3.

It is clear that there are some artefacts from the pixel art conversion. One can conclude that the pixel art converter does not run like it is supposed to. This performance, however,

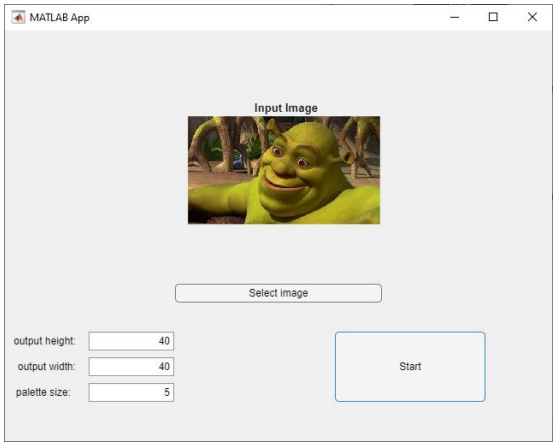

Fig. 1. GUI of the pixel art converter

Fig. 2. Pixel art with size 40x40 and 6 colors in the palette

Fig. 3. Pixel art with size 40x80 and 6 colors in the palette

is most likely caused by badly tuned parameters like $e_{cluster}$, $e_{palette}$ and $e_{perturbation}$. It is quite difficult to find parameter values that work great for each image. There also was not enough time to evaluate a large set of images for different parameter values. This time deficiency is mainly caused by the fact that the program runs very slow.

The program speed is mainly determined by the input image size. It further decreases for larger output images and for larger palette sizes. There are however ways to decrease the run time. One could make it run faster by downsampling the input image. This reduces the pixel count of the input image and thus speeds up the program. But these lower resolution input images can cause bad input pixel to superpixel mappings since not many input pixels are available for each superpixel.

V. Conclusion

This paper presented an implementation of a pixel art generator in MATLAB based on the algorithm proposed by Gerstner et al. in their 2013 paper. The generator consists of initialisation, superpixel refinement, palette refinement, and palette saturation steps. A user-friendly GUI allows input selection for the image, output size, and palette size.

The experimental results showed potential, although some artifacts were observed in the generated pixelated images. Fine-tuning the algorithm's parameters, such as clustering and convergence thresholds, is necessary for improved results. Additionally, the program's performance is slow, especially for larger inputs. Optimization techniques like image downsampling could enhance efficiency.

In addition to the implemented algorithm, the original paper describes ways to allow the user to control certain aspects of the algorithm, like setting importance maps, controlling the superpixels and palette colors,... These have not been implemented, although the GUI could allow for more manual control of the process.

In summary, further refinement and parameter tuning are needed to improve the pixel art generator's quality. Optimization strategies, such as image decimation, can address performance issues. Enhancing the generator's effectiveness and speed will lead to better pixelated images and a more satisfying user experience.

References

[1] Gerstner, T., DeCarlo, D., Alexa, M., Finkelstein, A., Gingold, Y., Nealen, A. (2013). Pixelated image abstraction with integrated user constraints. Computers & Graphics, 37(5), 333–347.

VI. Appendix

This section gives a short overview of what each student did for the project.

A. Arno De Haseleer

- Implemented 4.2 Laplacian smoothing of the pixelsin MATLAB
- Created the GUI

B. *Rogier De Nys*

- Debugged 4.1 (Superpixel initialisation)
- Debugged 4.2 (Superpixel refinement & Laplacian smoothing)
- Various optimizations (mainly 4.1 & 4.2), decreasing run time
- Expanded the report

C. *Warre De Winne*

- Implemented 4.1 Initialisation in MATLAB (with bugs)
- Implemented 4.2 Superpixel refinement (not Laplacian smoothing) in MATLAB (with bugs)
- Implemented part of 4.3 Palette refinement (with bugs)
- Implemented 4.4 Palette saturation
- Debugged Normalisation of c_k (4.3)
- Debugged Divergence in subclusters (4.4)
- Debugged palette coloring (4.4)
- Wrote *Abstract*, *Introduction*, *Algorithm*, part of *GUI* and part of *Experimental results*

D. *Sam Mousavi*

- Debugged 4.1 Initialisation in MATLAB (with bugs)
- Debugged 4.3 Palette refinement
- Debugged palette coloring (4.4)
- Wrote *Conclusion*