PIXELATED ABSTRACTION

BY TIMOTHY GERSTNER

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Computer Science

Written under the direction of

Andrew Nealen

and approved by

New Brunswick, New Jersey

May, 2013

ABSTRACT OF THE THESIS

Pixelated Abstraction

by Timothy Gerstner

Thesis Director: Andrew Nealen

Pixel art remains a contemporary art form and a common rendering technique in digital games and media. However, the manual creation of pixel art is often time consuming and requires a degree of skill that is not easily obtained by novices of the art. Few, if any, methods exist to automatically generate pixel art. Naive downsampling techniques such as nearest neighbor and cubic downsampling do not adequately preserve features or maintain a vibrant palette. In this thesis we present our work on automatically and semi-automatically converting high resolution images into an output that approximates the manual results of pixel artists. This is a multi-step, iterative algorithm that simultaneously solves for a palette and a mapping of segments of the input image to pixels in the output. We provide a set of controls that give the user flexible influence on the output and the ability to work anywhere between a purely automated and purely manual process. We present the automated and semi-automated results of our algorithm and compare them to the results generated using naive downsampling techniques and the manual results produced by expert pixel artists. Through a formal user study and interviews with expert pixel artists, we demonstrate that our results offer an improvement over the naive methods.

Acknowledgements

This thesis is based on my work in the published pieces “Pixelated Image Abstraction” [Gerstner et al., 2012] and “Pixelated Image Abstraction with Integrated User Constraints” [Gerstner et al., 2013]. This research is supported in part by the Sloan Foundation, the NSF (CAREER Award CCF-06-43268 and grants IIS-09-16129, IIS-10-48948, IIS-11-17257, CMMI-11-29917, IIS-09-16845, DGE-05-49115), and generous gifts from Adobe, Autodesk, Intel, mental images, Microsoft, NVIDIA, Side Effects Software, and the Walt Disney Company. The following copyrighted images are used with a Creative Commons license or by permission: Figure 1.1(a) by Jason Baker, Figure 2.2(right) by Celia Marsh, Figure 2.1(left, middle) by Alice Bartlett, Figure 5.1 by Louis Vest, Figure 5.5 (giraffe) by Paul Adams, and Figure 5.5 (pagoda) by William Warby. The pixel art in Figure 5.7 is copyright Adam Saltsman (top) and Ted Martens (bottom).

I’d like to thank my coauthors, colleagues, and everyone who helped me along the way. I’d especially like to thank my advisor, Andrew Nealen, for giving me such a unique opportunity in my life, being an amazing mentor, and always encouraging me to try harder and experience new things.

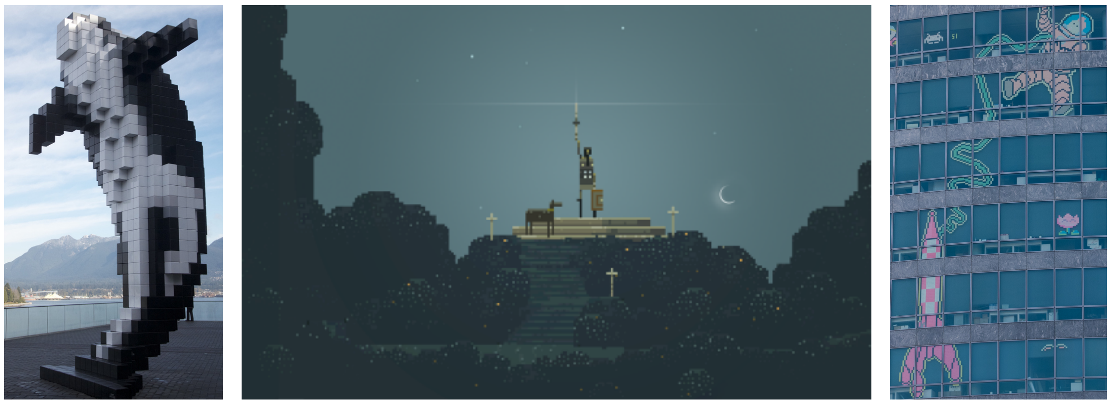
I also could not have done this without the help of my family, my friends, and Jess. Thank you for always supporting me and keeping me sane.

Table of Contents

Chapter 1

Introduction

1.1 Motivation



(a) (b) (c)

Figure 1.1: Examples of Pixel art in modern day. (a) The "Digital Orca" statue by Douglas Coupland, outside the Vancouver Convention Center. (b) A scene from the digital game Superbrothers: Swords & Sworcery EP. (c) A view of a multi-floored building where the employees used post-it notes to create pieces of pixel art.

Pixel art is a contemporary art style and a significant part of our culture. The popularity of modern games such as the award winning Superbrothers: Sword & Sworcery EP, shown in Figure 1.1(a), demonstrates that the appeal of pixel art transcends its origin as a solution to hardware limitations. In recent years, many art communities have recognized the significance of pixel art as a rendering style in games. The 2012 "The Art of Video Games" exhibit at the Smithsonian American Art museum not only heavily featured classic pixel art games such as Pacman, The Legend of Zelda, and Space Invaders, it also featured the 2011 pixel art game Minecraft, which has sold over 20 million copies [Nunneley, 2013]. Pixel art was also featured by the Museum of Modern Art in the 2011 "Talk To Me" exhibit, and is currently the most common rendering style in their permanent video game artwork collection [Antonelli, 2013].

The impact of pixel art is much broader than digital games. Companies such as Coca-Cola, Honda, Adobe, and Sony use pixel art in their advertisements [Vermehr et al., 2012], and in 2012 the Emmy nominated TV show "Community" featured an entire episode rendered as pixel art. The art style is a part of our physical live's as well. The "Digital Orca"" by Douglas Coupland, shown in Figure 1.1(a), is a popular sight at the Vancouver Convention Center. France was recently struck by a "Post-it War" [1], where people used Post-It notes to create pixel art on their windows, competing with their neighbors across workplaces, small businesses, and homes (see Figure 1.1(c) for an example).

What makes pixel art both compelling and difficult are the limitations imposed on the medium. The defining characteristic of pixel art is that each individual pixel is important to the whole, and pieces are generally composed with as few pixels and colors as possible. The task of a pixel artist is to carefully choose the set of colors and their placement in the image that best depicts the subject, and is generally completed by artists pixel-by-pixel, with many iterations. This can take a significant amount of time and requires a degree of skill that is not easily acquired by novices of the art. However, few, if any methods exist to automatically or semi-automatically create effective pixel art, which limits the amount of people the art style is accessible to.

There are many methods to automatically downsample images, two of which are shown in Figure 1.2, but at such small resolutions they do not accurately capture the subject and often look blurred, noisy, and distorted. Automated and semi-automated methods have been proposed for other popular art forms, such as line drawing [DeCarlo et al., 2003, Judd et al., 2007] and painting [Gooch et al., 2002]. Methods such as those proposed by DeCarlo and Santella [DeCarlo and Santella, 2002] and Winnemöller et al. [Winnemöller et al., 2006] not only abstract images, but do so while retaining salient features. A similar method for pixel art creation would benefit the work process of existing artists and open the art style to a larger audience.

[1]http://postitwar.com/

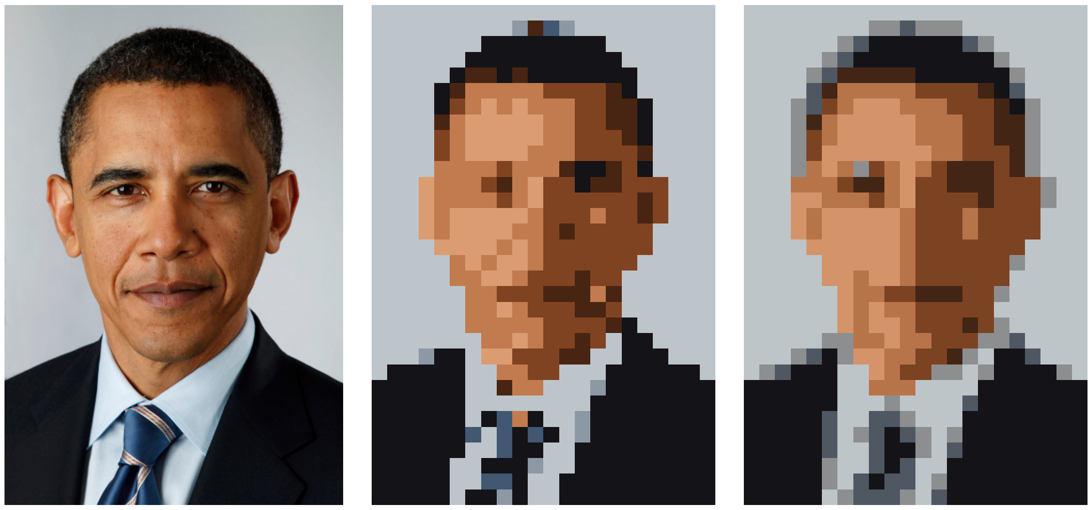


Figure 1.2: Naive downsampling methods do not sufficiently capture the original subject (left) in the style of pixel art. These results were created using nearest neighbor (middle) and cubic (right) downsampling techniques in conjunction with median cut color quantization [Heckbert, 1982]. In this example, the naive methods produce results that are blurry, distorted, and missing important features.

1.2 Summary of Contribution

In this thesis we propose a method to generate pixel art from a high resolution input image. We begin by presenting an entirely automated version of the method, as originally published in our paper "Pixelated Image Abstraction" [Gerstner et al., 2012]. This is an iterative algorithm, and each iteration is a multi-step process that simultaneously segments the original image and solves for a limited sized palette. To segment the image, we utilize a modified version of a segmentation algorithm proposed by Achanta et al. [Achanta et al., 2010] and map each pixel in the output to a segment of the input image. To find the palette, and its mapping to pixels in the output, we use an adaptation of deterministic annealing [Rose, 1998]. In our algorithm we make these two steps interdependent. The final solution is an optimization of both steps with respect to each other and the original spatial and palette sizes defined by the user.

We next propose a semi-automated version of the method, as described in the extended version of our paper, "Pixelated Image Abstraction with Integrated User Constraints" [Gerstner et al., 2013]. We extend the automated algorithm with a set of controls that allow the user to work anywhere between the manual process of an artist and the automated process of our algorithm. The controls incorporate the user into the iterative algorithm and give them the ability to introduce more advanced techniques typical to pixel art and leverage their high level understanding of the scene as necessary, while leaving as much of the computational work to the algorithm as desired.

The goal of our proposed method is to assist existing artists in the medium by reducing the amount of manual labor required to create a piece from an original source, as well as make the art style accessible to a more general audience. Applications of this work include creating pixel art and low-resolution images for digital games, applications, websites, and other digital and physical media where pixelated and abstract representations are used.

1.3 Overview of Thesis

The rest of the thesis is structed as follows: in Chapter 2 we give an overview of pixel art, including its history and fundamental characteristics. In Chapter 3 we review previous research on pixel art and related topics. Our core automated algorithm is presented in Chapter 4, the results of which are demonstrated in Chapter 5. Chapter 6 extends the algorithm with the addition of user controls, which allows for the semi-automated creation of pixel art. Chapter 7 demonstrates results using the extended algorithm. Finally, in Chapter 8 we present our conclusions and propose potential paths for future work.

Chapter 2

Pixel Art

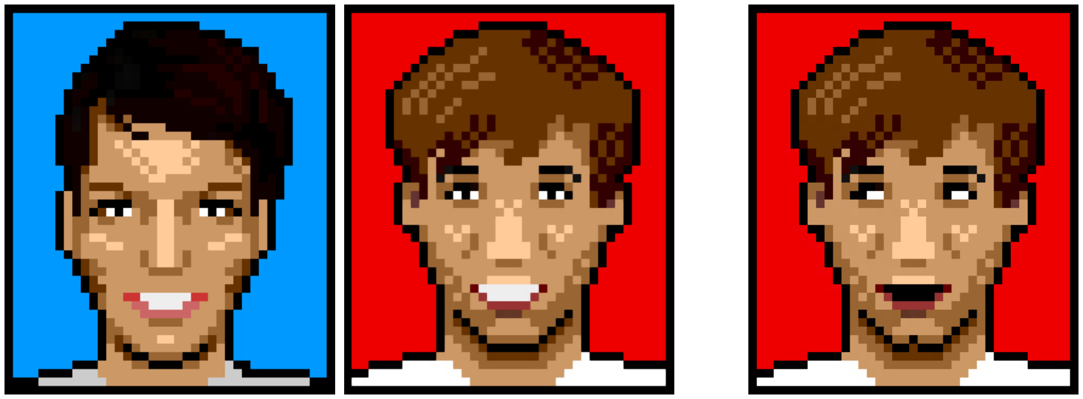


Figure 2.1: (left and middle) Two examples of pixel art, “Alice Blue” and “Kyle Red” by Alice Bartlett. Notice that although the facial features are no longer proportionally accurate and are represented abstractly by only a few pixels, the subjects are still easily recognizable and distinguishable. (right) An example of how important pixel placement and color can be. Even though only a single color and 6 pixels were swapped, the end result conveys a significantly different subject.

Pixel art is an art style where the pieces are composed of individual units, known as pixels, that are selected from a limited palette and placed in a regular grid, as shown in Figure 2.1. What differentiates pixel art from rasterized digital images is that each pixel is significant to the perception of the image. The modern form of pixel art originated as a style of rendering objects, figures, and scenes on early graphical displays. However, the core concepts of the art style are a much deeper part of human culture, and have been used by artists for thousands of years. The traditional art style of mosaics, which involves combining single colored tiles into an image, has been used by artists since as early as 3000 BC. Figure 2.2(left) shows an example of a Roman Mosaic, made in 100AD, which, while not entirely on a regular grid, exhibits many of the same properties as pixel art. Nearly as old is the art of cross stitching Figure 2.2(right), a form of embroidery where X shaped stitches of a single color are placed in a regular grid on a cloth to form an image. The palette of cross stitching is limited to the colors of thread available to the artist.

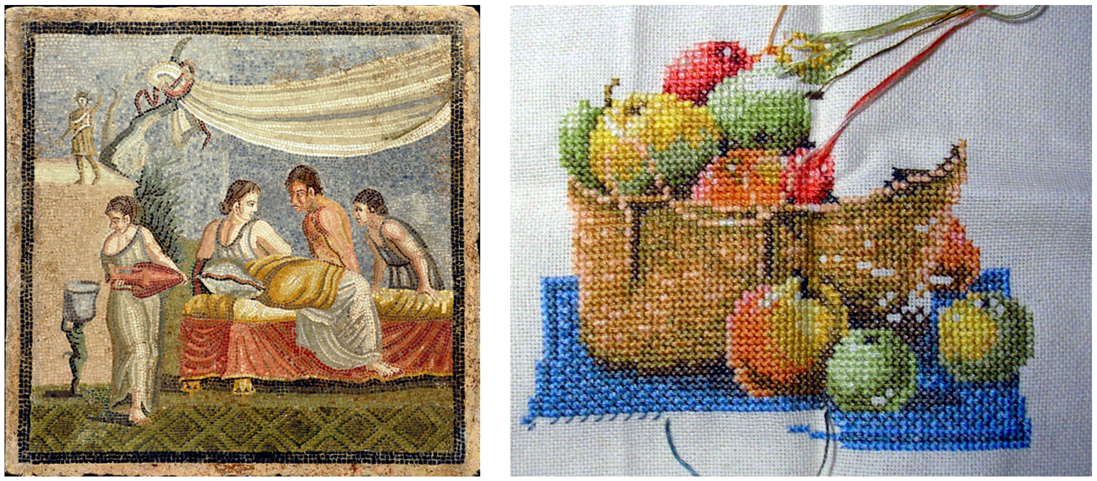
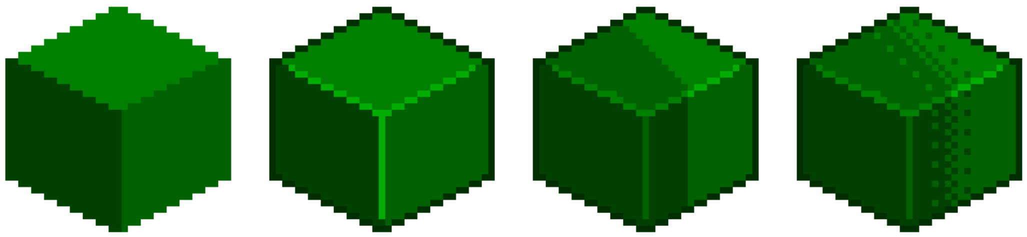


Figure 2.2: Pixel art shares many characteristics and techniques of traditional art styles that have been a part of our culture for thousands of years. (left) A roman mosaic created sometime around 100AD. (right). A piece of embroidery created using regularly spaced X shaped stitches in a method known as cross-stitching.

At the most basic level, there are two tasks required to created a piece of pixel art: (1) choosing a palette of colors and (2) determining where in the output image these colors should be placed. In this thesis we will focus on the creation of pixel art from a source image. In this case, the choices palette and color must be such that the resulting image retains a likeness to the original subject. What makes these tasks difficult is the constraints placed on both the palette and output image size. Historically, these constraints were imposed by the hardware limitations of the device, but with current displays capabilities these limitations are no longer a concern. These days, the palette and image sizes are self imposed by the artists to retain the distinctness of each pixel in the final piece.

Pixel art is essentially a form of abstraction. Artists carefully choose which features of a subject are important to the piece, and how to represent the features using only a few pixels. For example, in Figure 2.1 the mouth is represented by only the teeth and bottom lip, and the eyes are represented using only 3 pixels, but the features are still easily understood. Similarly, the artist is able to convey shape and texture in the hair using no more than three colors and the selective placement of highlights and shadows. It is also common for artists to make features no longer proportional and fit naturally to the pixel grid, which research has shown contributes to the perception of the image [Marr and Hildreth, 1980].

As a result of the spatial and palette sizes, altering a single pixel or color can drastically change the resulting image, which is demonstrated in Figure 2.1(right). In this example, changing a single color in the mouth gives the appearance of a gasp rather than a smile, moving a single pixel in a line gives the impression of a cleft chin, and shifting the "pupil" of each eye one pixel to the left causes the subject to appear to be looking to the side. In terms of the physical image, these are very small changes, but the perceived subject looks significantly different than the original.



(a) Isometric (b) Edge Highlighting (c) Simple Shading (d) Dithering

Figure 2.3: Examples of different styles and techniques used in pixel art. (a) A cube drawn using a isometric, or "3/4" perspective, where horizontally straight lines are rendered using a diagonal 2:1 pixel pattern. (b) An example of the same cube drawn using edge highlighting. (c) The cube in a shadow drawn using a simple color difference. (d) The same shadow draw using a dithering technique. Notice how the shadow appears to be more of a penumbra, but the technique also gives the cube the appearance of a texture.

Within the form of pixel art, there are many different styles and techniques, the combination of which can create significantly different pieces. Many styles developed as a result of their application. For example, unlike Figure 2.1, the cube in Figure 2.3(a) was created using an isometric, or "3/4 perspective", which is a popular style used in many 2.5D games. However, which choice of technique also comes down to the artist's preference. For example, many artists use edge highlighting (Figure 2.3(b)), a technique where the edges are rendered using brighter and darker hues than the surface, which can help emphasize shape boundaries and indicate lighting direction. When shading objects, some artists prefer to use a simple boundary as in Figure 2.3(c), while others prefer to use a pattern technique known as dithering, shown in Figure 2.3(d). However, each method can come with its own drawbacks. For instance, dithering can often add the unwanted perception of a texture to an object, and is therefore used sparingly for cases such as skin. These techniques are also dependent on the resolution, and often become less practical to use as the resolution decreases and the image becomes more abstract.

The decision of which techniques and styles to use is dependent on both the application and the artist's preference, and to make the choice with an automated algorithm would require a deep understanding of the scene and the artist's goals. Therefore, in order to create a more general and automated algorithm, the method we introduce in Chapter 4 focuses on the two primary components of pixel art creation: choosing a palette of colors and using this palette to map regions of the input image to the output image. In Chapter 6, we will explain how the automated algorithm can be extended by user controls to leverage the user's understanding of the scene and incorporate more advanced techniques into the output.

Chapter 3

Related Work

While, to the author's knowledge, there is little known work on automatically or semi-automatically producing pixel art, there is still a great deal of similarity between our problem and the fields of color quantization, image segmentation and image abstraction.

For example, one aspect of creating pixel art is to reduce the number of colors used in the output, while still remaining faithful to the original image, which is essentially color quantization. Color quantization is a classical problem that has been studied since the 1980's, when it was useful for indexed color displays. Works of the time made use of methods such as octrees [Gervautz and Purgathofer, 1990], median-cut [Heckbert, 1982], binary trees [Orchard and Bouman, 1991], and dynamic programming [Wu, 1992] to cluster the pixels of the input image into a reduced palette. However, these methods were generally used to produce an output at the same resolution as the input. As such, these color quantization methods do not account for a much lower spatial resolution in the output, which can change feature emphasis and image statistics.

Also applicable to the problem of color quantization are more general clustering methods, such as k-means clustering [MacQueen, 1967]. Of particular interest is a method known as deterministic annealing (DA) [Rose, 1998], which is a method that uses a probabilistic assignment while clustering. Similar to k-means, it uses a fixed number of clusters, but it is independent of initialization. Also, different from simulated annealing [Kirkpatrick et al., 1983], it does not randomly search the solution space and will converge to the same result every time. In Chapter 4 we will show how we use an adapted version of DA for color palette optimization.

It is important to note that we are not the first to consider using DA for image processing purposes. Puzicha et al. [Puzicha et al., 2000] proposed a method that reduces the palette of an image and applies half-toning using a model of human visual perception. However, like the previously mentioned color quantization techniques, their method emphasizes a solution that optimizes color reduction, while our method instead emphasizes color and spatial reduction in parallel. Furthermore, at low spatial resolutions dithering can introduce the perception of texture into the output.

Another field related to our problem is the field of image segmentation, which has been extensively studied in the context of computer vision applications. Solutions to the image segmentation problem include graph-cut techniques such as the method proposed by Shi and Malik [Shi and Malik, 1997] and superpixel-based methods QuickShift [Vedaldi and Soatto, 2008], Turbopixels [Levinshtein et al., 2009], and SLIC [Achanta et al., 2010]. In particular, SLIC (Simple Linear Iterative Clustering) produces regularly sized and spaced regions. This, coupled with the method's relatively low computational overhead and very few input parameters, makes it an appropriate starting point for components of our problem. In Chapter 4 we will show how we modify SLIC and incorporate it into our solution to map regions of the input image to each pixel in the output.

More recently, there have been several pieces of research specifically on pixel art. Published concurrently with our first paper [Gerstner et al., 2012], Inglis and Kaplan [Inglis and Kaplan, 2012] proposed a method to convert vector line art into pixel art. At low resolutions, a single misplaced pixel can drastically change the perception of a curve. Their method removes these disruptive artifacts from the output by shifting each path's end points and local extrema to pixel centers and by enforcing that a monotonic curve whose slope is monotonic is represented by monotonic pixel spans.

Kopf and Lischinski [Kopf and Lischinski, 2011] proposed a method that extracts vector art representations from pixel art. This problem is almost the inverse of the one presented in this paper. However, while their solution focuses on interpolating unknown information, converting an image to pixel art requires compressing known information. In Chapter 5 we analyze how well our method serves as an inverse to theirs, in comparison to naive methods.

Chapter 4
Automated Method

In this chapter, we describe our method for automatically transforming a high resolution input image to a low resolution, limited palette output in the style of pixel art. Our method solves for two components simultaneously. The first component is a limited palette of colors used in the input that best represent the original image. The second is a mapping of regions of the input image to pixels in the output image. The goal of this mapping is to minimize distortion and retain the important features of the original image. Our method is an iterative solution that creates an interdependency between these two components and solves for them simultaneously.

4.1 Background

Our method builds upon two existing clustering algorithms, SLIC and DA, which we give a brief overview of in the following two sections. We encourage the reader to be familiar with both methods before implementing our proposed algorithm.

4.1.1 Simple Linear Iterative Clustering (SLIC)

Part of our problem is to segment the original image into regions, which we then map to pixels in the output. Achanta et al. [Achanta et al., 2010] proposed a method that iteratively segments an image into regions called “superpixels”. These superpixels are clusters of individual pixels in the image, and are represented by a single position and color. The algorithm is analogous to k-means clustering [MacQueen, 1967] in a five dimensional space (two positional and three color).

The algorithm has two main steps. In the first step, each of the M pixels in the input image, p_i, is assigned to one of N superpixels p_s which minimizes

$$d(p_i, p_s) = d_c(p_i, p_s) + m\sqrt{\frac{N}{M}}d_p(p_i, p_s) \tag{4.1}$$

where d_c is the color difference in L*a*b* color space [L'Eclairage, 1978] and d_p is the positional difference (both measured using Euclidean distance). The constant m is some value in the range $[0, 20]$ that is used to control the relative weight between the color and position components, which have different scales that cannot be otherwise normalized.

The second step of each iteration is to update the position and color values of the superpixels. These are simply set to the average position and average color of all the input pixels assigned to each superpixel. The algorithm continues until the difference between the previous and recomputed superpixel values is below some threshold.

4.1.2 Deterministic Annealing (DA)

Another component of our problem is to determine which colors to use in the palette and how these colors should be assigned to pixels in the output. To solve this component, we build upon the algorithm of deterministic annealing (DA) [Rose, 1998], which is a global optimization method for clustering inspired by the process of annealing in material science. In our implementation each cluster corresponds to a color in the palette.

Unlike SLIC, deterministic annealing is a fuzzy clustering algorithm, which probabilistically assigns objects to clusters based on their distance to the cluster. In other words, each object belongs, with some probability, to each cluster. Drawing on the physical annealing process, deterministic annealing uses a temperature value T, which can be viewed as proportional to the expected variance of the clusters. Initially, T is set to a high value T_0 such that each object is equally likely to belong to any cluster. At each value of T, the system is allowed to locally converge. The temperature is then lowered, which decreases the variance of each cluster, and objects begin to favor some clusters over others. As T approaches the final temperature (T_f), objects become associated with a single cluster with a probability of nearly one, and with other clusters with a probability of nearly zero. In the theoretical limit of $T_f = 0$, the deterministic annealing effectively becomes equivalent to k-means clustering.

In Section 4.6, we will provide a formal definition of this process in the context of our problem. Specifically, we will use Mass Constrained Deterministic Annealing (MCDA). MCDA seeks to make DA more computationally efficient by realizing that at high temperatures each object is equally likely to belong to every cluster, and therefore there is effectively only one cluster. MCDA saves computation by beginning with a single cluster, and internally represents each cluster throughout the algorithm with two sub-clusters. At the start of each new temperature T, each cluster's sub-clusters are set to a slight permutation of their mean. At a high T, these sub-clusters will converge to the same value, but as the temperature is lowered, they begin to naturally separate. When this occurs, the cluster is split into two distinct clusters (each represented by their own sub-clusters). This process continues recursively until some maximum number of clusters is reached. In our implementation, this equates to a palette that starts as a single color and grows over time. As our algorithm iterates, colors will split along the directions of highest variance until the maximum palette size is reached.

4.2 Overview

A visual overview of our automated algorithm is shown in Figure 4.1. The algorithm begins with three inputs: an image of width w_{in} and height h_{in}, the size of the output palette (K), and the size of the output image ($w_{\text{out}} \times h_{\text{out}}$). While the size of the output image is technically two parameters, we enforce that the output image has the same aspect ratio as the input image, and therefore only require as input the length of the longest dimension of the output image.

As shown in Figure 4.1, after initialization (a), our algorithm begins a two step iterative process (b) that is the core of our algorithm. The first step (c) of each iteration is solve for an optimal palette, and the second step (d) is to segment the input image into regions that correspond to pixels in the output. These steps are intertwined; solving for the palette uses the most recent segmentation, and segmenting uses the most recent

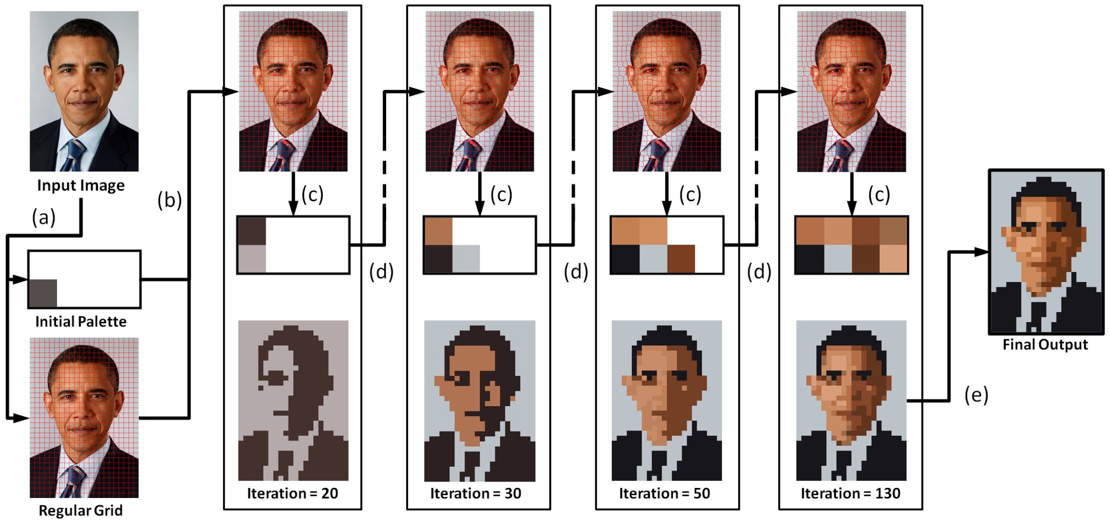


Figure 4.1: The pipeline of the algorithm, demonstrated using intermediate results. The superpixels are initialized (a) in a regular grid across the input image and the palette is initialized as a single color equal to the average color of the M input pixels. After initialization the algorithm begins iterating (b). During each iteration, the algorithm performs two main steps: refining superpixel segmentation of the input image (c) and refining the palette (d), which updates the values of the colors, their assignment to pixels in the output, and may introduce new colors to the palette. After convergence, the palette is saturated (e) and the final output is produced.

palette. Over time, more colors are added to the palette until the maximum, K, is reached. Once iteration converges to a solution, the palette is saturated (e), and the final output is produced.

The following is a list of terms which we will use throughout the rest of the thesis:

Input Pixels The set of pixels in the input image, denoted as p_i where $i \in [1, M]$, and $M = w_{\text{in}} \times h_{\text{in}}$.

Ouput Pixels The set of pixels in the output image, denoted as p_o where $o \in [1, N]$, and $N = w_{\text{out}} \times h_{\text{out}}$.

Superpixel A segment of the input image, denoted as p_s where $s \in [1, N]$. The superpixels are a segmentation of the input image.

Palette A set of K colors c_k, $k \in [1, K]$ in L*a*b* space [L'Eclairage, 1978].

The output image is constructed by mapping each superpixel of the input image to a pixel in the output. Figure 4.2 gives a low resolution example of this mapping. Each

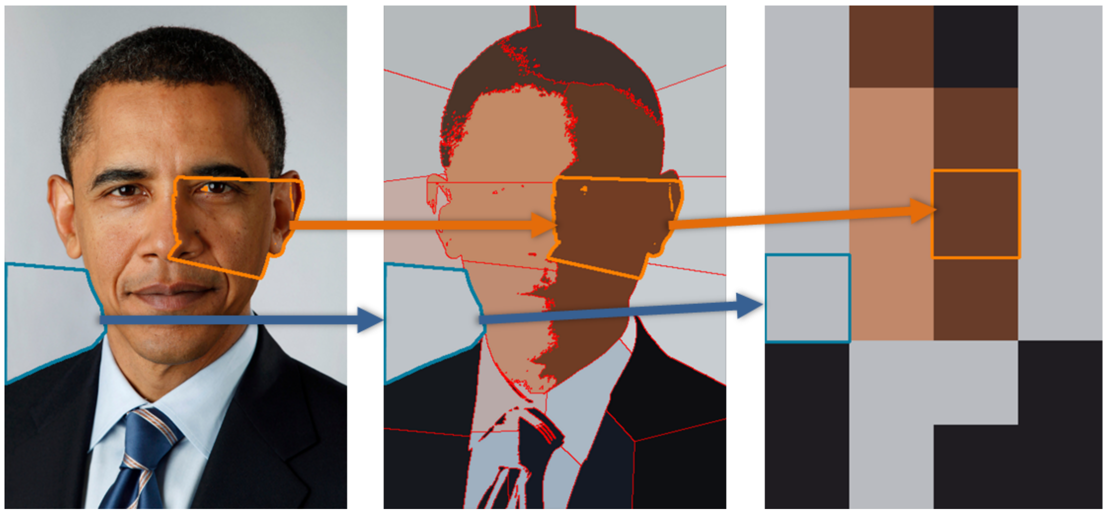


Figure 4.2: Pixels in the input image (left) are associated with superpixel regions (middle). Each superpixel region corresponds to a single pixel in the 4×6 output image (right).

superpixel is also associated with a color in the palette, which is also assigned to the superpixel's corresponding pixel in the output. Our algorithm is structured similarly to an MCDA solution of clustering objects (in our case superpixels) into a set of colors (the palette). However, during each iteration we also perform an additional step of refining the superpixel segmentation of the input image using our modified version of the SLIC algorithm. A high level pseudo-code is shown in Algorithm 1.

4.3 Initialization

Before our algorithm begins to iterate, we need to initialize both the palette and the superpixels. As part of the MCDA process, the palette is initialized to a single color

Algorithm 1

```
▷ initialize superpixels, palette and temperature $T$ (Section 4.3)
▷ while ($T \geq T_f$)
    ▷ refine superpixels segmentation with 1 step of modified SLIC (Section 4.5)
    ▷ associate superpixels to colors in the palette (Section 4.6)
    ▷ refine colors in the palette (Section 4.6)
    ▷ if (palette converged)
        ▷ reduce temperature $T = \alpha T$, ($\alpha < 1$)
        ▷ expand palette (Section 4.6)
▷ post-processing (Section 4.7)
```

c_1, which is set to the mean value of the M input pixels in L*a*b* space. N superpixels are created and their (x, y) positions are initialized in a regular grid (with the same dimensions as the output image) across the input image. Each superpixel's color component is set to c_1. Since each superpixel has the same color, each input pixel is assigned to the closest superpixel in (x, y) space.

We also need to define a starting temperature for MCDA. Rose [Rose, 1998] demonstrates that the temperature at which cluster will naturally split(the "critical temperature", T_c), is defined as twice the variance along the major principal component axis of the set of objects assigned to the cluster. We therefore initialize the temperature T to $1.1T_c$, which ensures that the initial temperature is above the point at which more than one color would exist in the palette.

4.4 Iteration

After initialization, our algorithm begins iteratively refining superpixel assignment and the palette. The iterative process alternates between performing a single step of refining the superpixel assignments, and a single step of refining the palette. As mentioned previously, a step of refining the superpixels uses the most recent palette, and similarly a step of refining the palette uses the most recent superpixels. This alternation and interdependency allows us to solve for superpixels and a palette that are optimized with respect to each other.

4.5 Superpixel Refinement

In the superpixel refinement step, our goal is to find the next best assignment of input pixels to superpixels, and to define a value for each superpixel in (x, y, L^*, a^*, b^*) space that represents the input pixels assigned to it. We accomplish this task by using an implementation of SLIC with several critical modifications. These changes are made to account for the constraints of a limited palette and the fact that the output will be a regular grid of pixels, neither of which SLIC was originally designed for.

The first alteration we make to the SLIC algorithm is to the color assigned to each superpixel at the end of every iteration. In the original SLIC algorithm, each input pixel is assigned to the superpixel that minimizes Equation 4.1 and the color of each superpixel is set to the mean color value of its associated input pixels, m_s. In our application, only a limited number of colors can be used in the output. However, is is very unlikely that m_s is in the output palette, and we would therefore be optimizing for a set of colors that do not exist. We would instead like the superpixels to form regions of input pixels that are optimized with respect to the actual palette. Therefore, rather than using m_s as the color component of each superpixel, we instead use each superpixel's associated color in the current palette (we will explain how colors are associated to superpixels beyond initialization in Section 4.6). A demonstration of how this improves the results is shown in Figure 4.3.

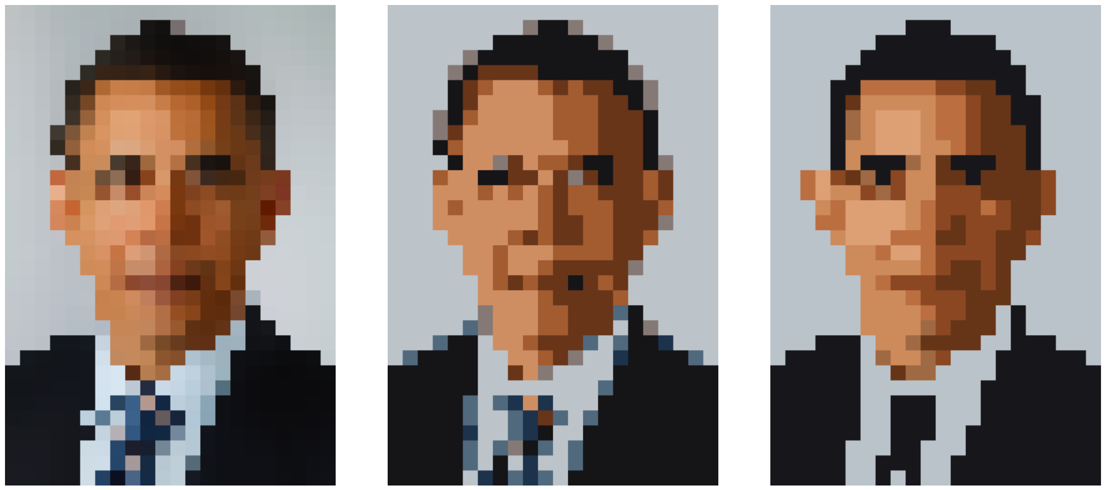


Figure 4.3: As part of our modified implementation of SLIC, our method uses palette colors to represent superpixels when clustering input pixels. Using the mean color of a superpixel works when the palette is unconstrained (left), but fails when using a constrained palette (middle). This is because the input pixels cluster into superpixels based on colors that may not exist in the final image, which creates a discrepancy. Using the palette colors to represent the superpixels removes this discrepancy (right).

The second modification we make to the SLIC algorithm is to compensate for the fact that the pixels in the output are in a regular grid. Using the original SLIC algorithm, the superpixels tend to form a hexagonal grid with 6-connected neighborhoods, as seen in Figure 4.4(a). This behavior is typical of typical of clustering algorithms which utilize Voronoi regions [Secord, 2002]. However, for our application the hexagonal grid does not match the 4-connected neighborhoods of the pixel grid, which creates unwanted distortions such as those seen in Figure 4.4(b). Ideally we would like adjacent output pixels to map to adjacent superpixels. To improve the correspondence between the superpixel and output pixel neighborhoods, we apply a step of Laplacian smoothing after the input pixels have been assigned and the superpixel locations have been updated. Once every iteration, each superpixel's (x, y) position is moved a percentage of the distance (we use 40%) from its current position to the average position of its 4-connected neighbors. The neighbors are established during the initialization of the regular grid and stay constant throughout iteration. The resulting superpixels of this modification are shown in Figure 4.4(c), which produces the output seen in Figure 4.4(d). These new positions will be used during the next step of superpixel refinement.

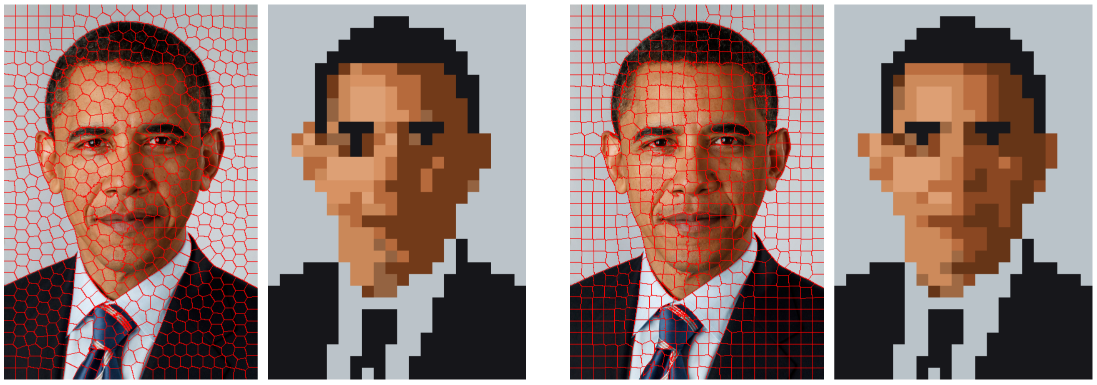

(a) (b) (c) (d)

Figure 4.4: Without the Laplacian smoothing step, the superpixels (a) tend to have 6-connected neighborhoods. This causes small distortions in the output (b), which are particularly noticeable on the ear, eye and mouth, when compared to our results (d) that use the superpixels (c) generated using the smoothing step.

Finally, we perform a similar smoothing step after the mean color value, m_s, is calculated for each superpixel. This step is used to help decrease artifacts in the output that result from different colors being used over continuous regions in the input that contains a smooth gradient. While we do not use m_s to assign input pixels to superpixels, it is still used in the palette refinement step (Section 4.6). We modify the value of m_s using a bilateral filter treating each superpixel as if it had the same position and neighborhood in an image as its corresponding pixel in the output image. The modified values $m_s{}'$ are used during the palette refinement step.

4.6 Palette Refinement

Each step of palette refinement is performed using a step of MCDA [Rose, 1998], and can be broken down into three basic steps: **associate** superpixels with some probability to each color in the palette, **refine** the palette based on these associations, and **expand** the palette by splitting existing colors. However, it is important to note that while **associate** and **refine** occur in every iteration, **expand** is only performed when the palette converges for the current temperature.

4.6.1 Associate

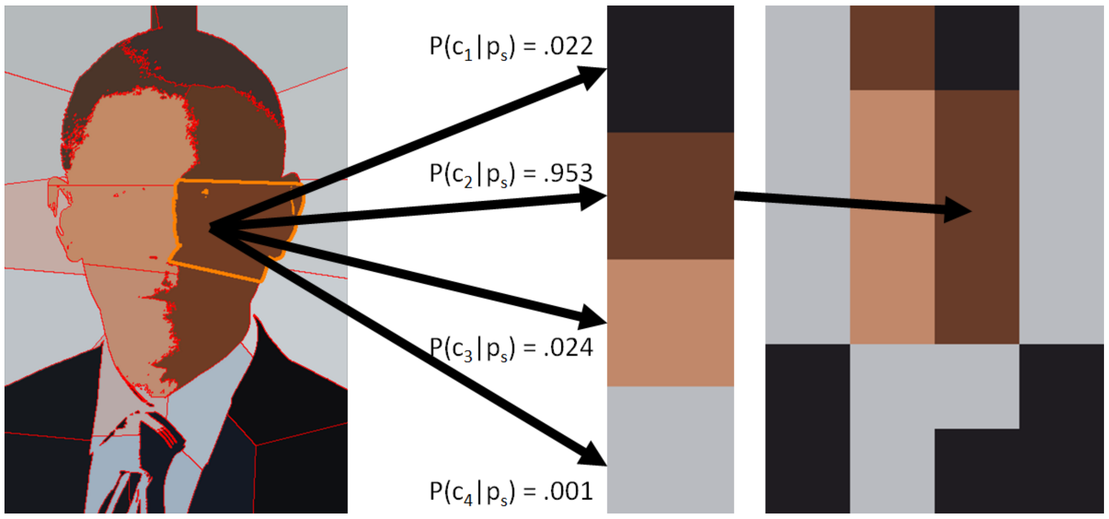


Figure 4.5: Each superpixel (left) is associated by some conditional probability $P(c_k|p_s)$ to each color in the palette (middle). The color with the highest probability is assigned to the superpixel and the superpixel's associated output pixel in the final image (right).

Each superpixel is associated with some probability to each color in the palette, as shown in Figure 4.5. The conditional probability $P(c_k|p_s)$ of superpixel p_s being assigned to color c_k is a function of the color distance between c_k and m'_s in L*a*b* space and the current temperature:

$$P(c_k|p_s) = \frac{P(c_k)e^{-\frac{||m_s{}'-c_k||}{T}}}{\sum_{j=1}^{K} P(c_j)e^{-\frac{||m_s{}'-c_j||}{T}}} \tag{4.2}$$

The numerator of this probability is an exponential measure of the distance based on the current temperature. The denominator is used to normalize the sum of probabilities over all c_k given p_s. $P(c_k)$ is the probability that color c_k is assigned to any given superpixel. At initialization, there is only one color in the palette, and so $P(c_1) = 1$. As more colors are added to the palette, the value for each color is defined as:

$$P(c_k) = \sum_{s=1}^{N} P(c_k|p_s)P(p_s) \tag{4.3}$$

where $P(p_s)$ can be seen as the prior probability of superpixel p_s. For now we assume that this value is uniform (i.e. 1) for all all superpixels as we do not favor any superpixel over another during palette generation. In Chapter 6.2 we will demonstrate how this value can be used to incorporate user-specified importance.

During each iteration of palette refinement, the values of $P(c_k|p_s)$ and then $P(c_k)$ are updated based on the most recent superpixel assignment. Each superpixel is assigned to the color in the palette that maximizes $P(c_k|p_s)$. Note that this assignment occurs each iteration, and can change as the temperature decreases, as superpixels assignment changes, and as more colors are introduced into the palette. The assigned color from the palette is used to represent the corresponding pixel p_o in the output image, as well as the superpixel color when computing the distance in LAB space from input pixels. Intermediate results of this process can be seen in the bottom row of Figure 4.1.

An advantage of this approach is that early in the process a superpixel has a nearly equal probability of being assigned to every color in the palette, due to the exponential factor in Equation 4.2. As the temperature decreases, each superpixel begins to favor colors in the palette that are closer in L*a*b* space, and therefore has a higher impact on it than colors it is more distant from (more on this in the next section). As the temperature approaches zero, each superpixel p_s effectively has a value of $P(c_k|p_s) = 1$ for the closest color, and $P(c_k|p_s) = 0$ for the rest. In other words, as the temperature approaches zero, MCDA becomes equivalent to k-means clustering, but without the problems of initialization inherent to k-means. Specifically, MCDA does not require an initial "guess" for each cluster, which can influence which local minimum the system converges to.

4.6.2 Refine

The next step is to refine the colors in the palette, by setting each color to the average of all superpixel colors, weighted by their probability of association computed in the previous step. The equation to compute each updated color is:

$$c_k = \frac{\sum_{s=1}^{N} m_s{}' P(c_k|p_s)P(p_s)}{P(c_k)} \tag{4.4}$$

Note that while this weighted average is calculated using *every* superpixel, superpixels with colors distant from c_k will have a weaker influence over the new color. In the the case of $T = 0$, colors with low probability of association will have effectively no influence at all. This can be seen in Figure 4.1 as the colors become more distinctive as the algorithm approaches convergence.

4.6.3 Expand

The palette expands when the algorithm has converged for the current temperature T and only if the maximum number of colors K has not been reached. We define convergence for a given temperature as the point at which the total change in the palette since the last iteration is less than some sufficiently small value ϵ_{P}.

The palette is expanded by splitting existing colors. To see if a color qualifies as one that should be split, we check to see if the distance between the sub-clusters of each color c_k exceeds some value ϵ_{C} (where ϵ_{C} is sufficiently small). If so, each sub-cluster is added to the palette as an actual color, with its sub-clusters both equal to its value prior to the split, and the original color is removed.

Once all splits have been resolved, each color is represented by two sub-clusters with the same value, either because the color never split or because it is a results of a previous color splitting. In order for any color to potentially separate in following iterations, each colors sub-clusters must be made distinctly different. To do so, we perturb the sub-clusters of each color by a small amount (less than ϵ_C) along the principal component axis of the cluster in L*a*b* space, which Rose [Rose, 1998] has shown to be the direction the cluster will split [Rose, 1998]. For each color, when T is greater than the critical temperature T_c, these clusters will converge to the same value, but when T is less than T_c they will separate. It is important to note that once the maximum number of colors in the palette has been reached, the palette will no longer expand, and we no longer need to check to see if a color qualifies to be split. Therefore, when the palette reaches size K, we no longer represent each color with sub-clusters.

4.7 Convergence

When the palette converges for a particular temperature T, the tempature is lowered by some factor α (we use 0.7). The algorithm continues to iterate until the temperature reaches the final temperature, T_f, and the palette has converged. While in theory the final temperature should be zero, this is not feasible in practice as the exponential component of Equation 4.2 becomes small enough to cause truncation errors. We use a value of $T_f = 1$ to avoid this issue in our implementation, which we have observed to be sufficiently small for our needs.

As a final post-processing step, we provide the option to saturate the palette, which is a typical technique in pixel art. To do so, we simply multiply the a^* and b^* channels of each color in the palette by a factor $\beta > 1$. For our results in Chapter 5 we use a value of $\beta = 1.1$. Finally, we convert our output image RGB space and output the final image.

Chapter 5

Automated Results

The following chapter contains several sets of examples at various output resolutions and color palette sizes generated using our automated algorithm. Unless otherwise specified, each result is generated using the parameter values mentioned in Chapter 4 and were produced in less than a minute on an Intel 2.67Ghz i7 Processor with 4GB of memory. For each example, we compare our method to results generated by two naive methods:

- *nearest method*: A bilateral filter is applied to the input image, followed by median cut color quantization to the palette size, and then the image is downsized using nearest neighbor downsampling.
- *cubic method*: The image is downsized to the output resolution using cubic downsampling, and then the palette is reduced using median cut color quantization.

Additionally, since it is not a significant contribution of our method, the naive method results are saturated using the same saturation process mentioned in Chapter 4.7. As a reminder to the reader, the results of our method and the naive methods should be viewed at a distance where the pixels are distinctly visible, as is typical of pixel art pieces.

In Figure 5.1, we demonstrate how each method's results vary as a function of the palette size, holding the output size constant. In this example, our algorithm exhibits several advantages over both naive methods. Across all palette sizes, our method introduces less speckles than the nearest method and looks less washed out than the cubic method. As the palette size shrinks, our method retains more salient colors within the palette, such as the green in the turban (middle column), and portrays

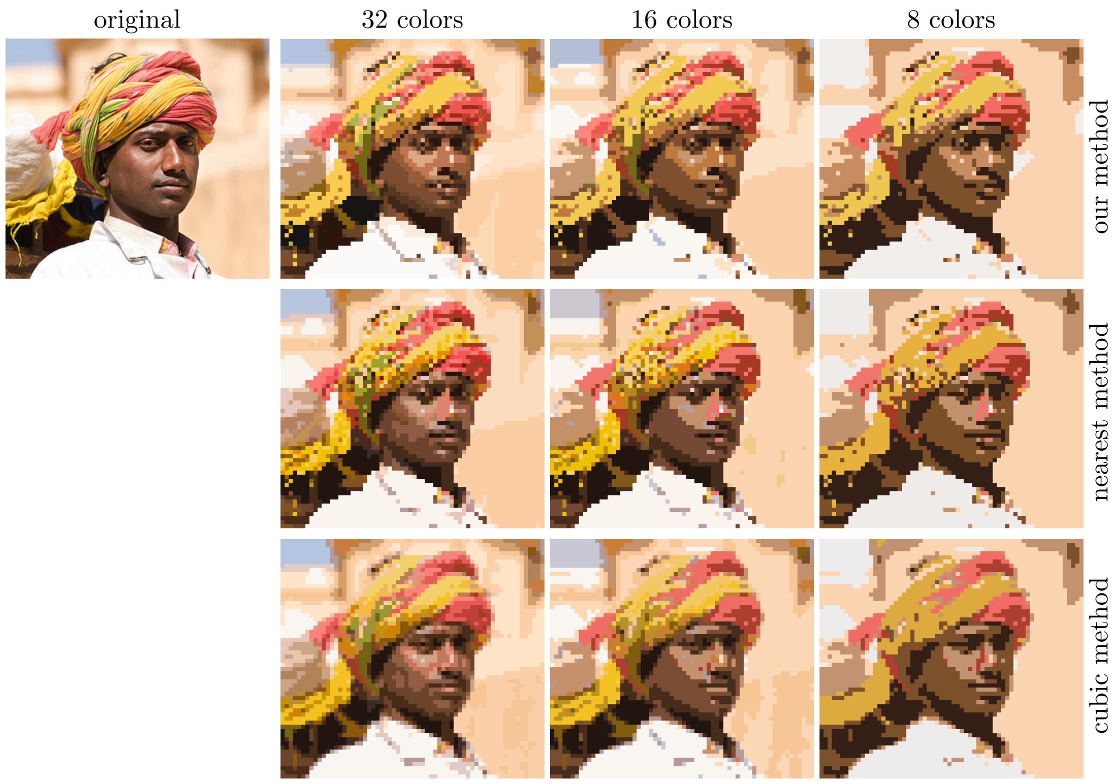


Figure 5.1: Varying the palette size. Each result is 64×58 pixels.

features such as the eyes in a manner that is more consistent with the original image. Our method also avoids visually jarring choices such as the use of pink tones in the faces of both naive methods.

One interesting property of our automated method that our results in Figure 5.1 highlight is that the convergence to the final result is relatively independent of the maximum number of colors allowed in the palette. To observe this property, we define the error of a given result as the mean Euclidean distance from each pixel to its corresponding superpixel center in LAB space. Figure 5.2 shows how this error decreases over the number of iterations for our method's results in Figure 5.1. As a reminder, all three cases have the same spatial size and the only difference is their maximum palette sizes. As each method iterates, the error (and intermediate results) are initially identical, and at iterations 20, 28, and 60 the palettes of each result split in the exact same way. The only time when the intermediate results may not be identical is when when a particular result reaches its maximum palette size K. At this point, the algorithm continues to the final temperature without splitting further. This is most noticeable in this example at iteration 115 where the result with $K = 16$ splits to 16 colors while the result with $K = 32$ splits to 28 colors. As expected, increasing the number of colors in the palette decreases the final error.

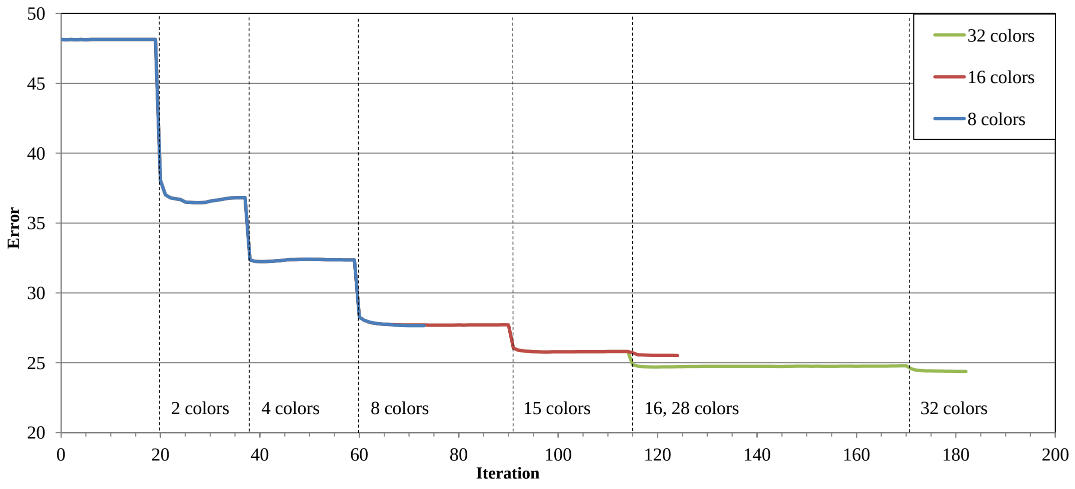


Figure 5.2: A comparison of the error per iteration of three results of our algorithm that vary only by palette size (The results are shown in Figure 5.1). The intermediate error (and the intermediate results) are identical each iteration, until a particular result reaches it's maximum palette size.

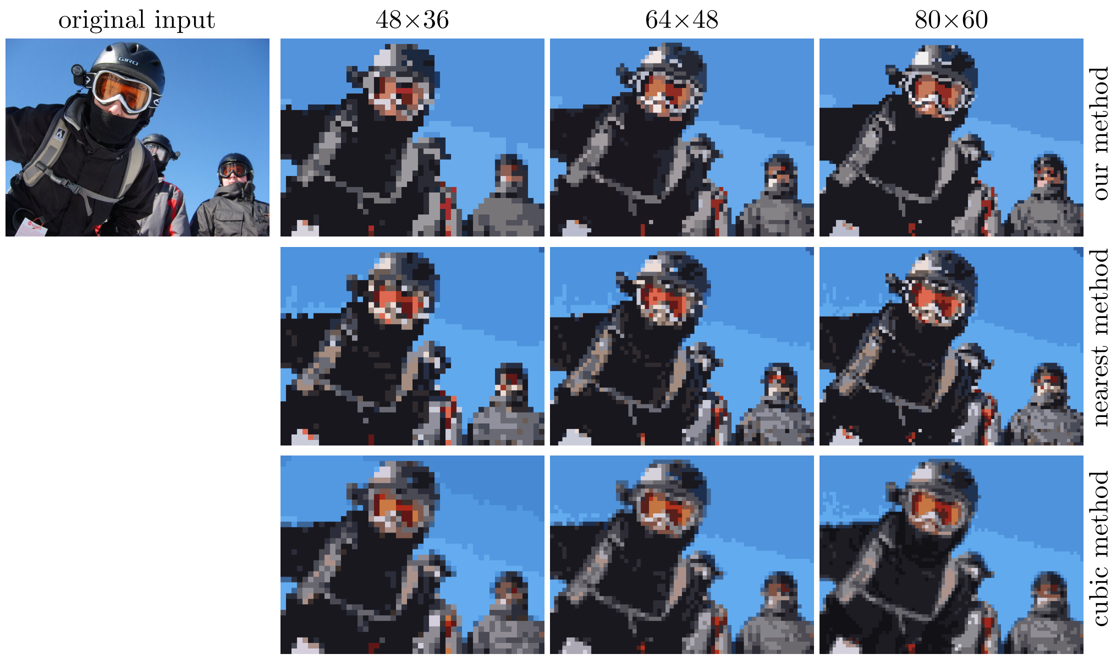


Figure 5.3: Varying the output resolution. Each result has a palette of 16 colors.

In Figure 5.3, we demonstrate how each method's results vary as a function of the resolution, this time holding the palette size constant. Similar to the results in Figure 5.1, our methods results are less speckled than the nearest method, and less washed out than the cubic method. Across all resolutions, our method makes better choices of color selection and placement, which is most noticeable in the skin tones on the face. While our method choose relatively natural colors, both naive methods choose colors such as gray, which no longer accurately represent the original subject. Additionally, at each resolution our method retains features such as the goggle rims more effectively than either naive method.

Our method also shows an improvement at extremely small output resolutions and palette sizes, as seen in Figure 5.4. At 22×32 and eight colors, our method more clearly depicts facial features such as the eyes, nose and mouth. The boundaries on areas such as the tie and hair are much more distinct than in either of the naive methods. Even at 11×16 and six colors, the facial features are retained much more distinctly than either naive method. At 4×6 and four colors, the results are very abstract, but the results of our method and the nearest method can still be identified as originating from the input. While our method arguably makes a poor choice in adding two colors to the hair, the nearest method erroneously uses a skin tone on the shirt. The results of the cubic method are blurred to the point of no longer being distinguishable, and neither the shape nor colors match the original image.

To verify our analysis, we conducted a formal user study consisting of 100 subjects using Amazon Mechanical Turk. In this study, subjects were shown the original, high resolution image and the results of our automated method and the two naive methods. The study consisted of the results shown in Figures 5.1, 5.3, 5.4, and 5.5 as well as the automated results in Figure 5.7, shown later in this chapter. The subjects were shown a single set that contained one of the original images and the corresponding results. The order of the results were randomized to remove bias and were scaled to approximately 256 pixels along their longest dimension using nearest neighbor upsampling, so that the users could see the pixel grid. The subjects were asked the question (in text) “Which of the following best represents the image above?” Each subject responded by choosing

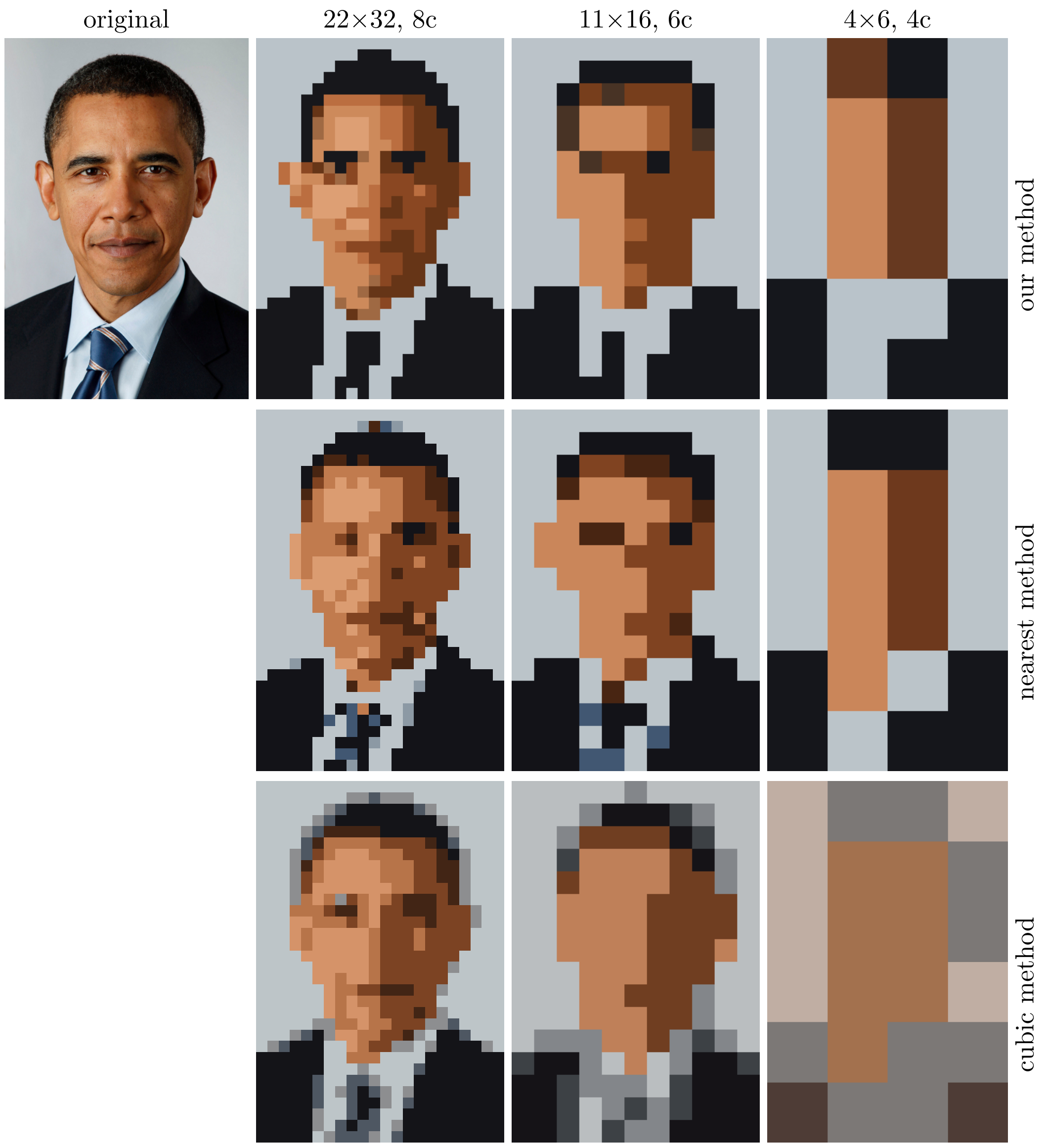


Figure 5.4: Examples of very low resolution and palette sizes.

one of the three result images. They then confirmed the choice and moved onto the next example. The images sets were shown in random order, and each set was duplicated four times to check for consistency.

To account for subjects who were answering randomly, we eliminated the results of every subject who gave inconsistent responses, which we defined as choosing the same answer for less than three of the four duplicates on more than a third of the stimuli. This reduced the number of valid responses to forty subjects. Of these responses, the subjects chose our results 41.49% of the time, the nearest method 34.52% of the

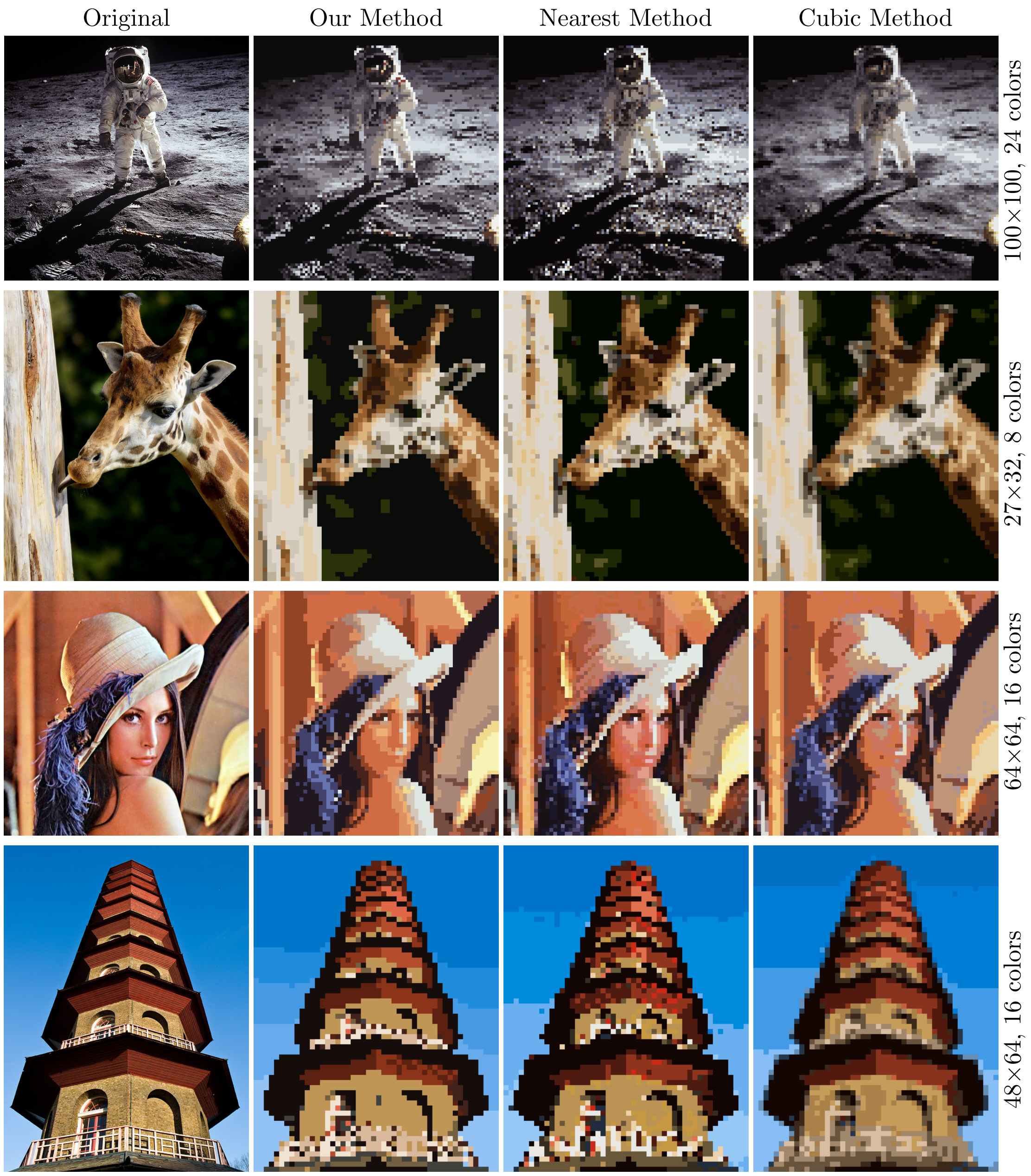


Figure 5.5: Additional results at various resolution and palette sizes used in our user study. Columns (left to right): input image, output of our algorithm, output of the nearest method, output of the cubic method.

time, and the cubic method 23.99% of the time. Using a one-way analysis of variance (ANOVA) on the results, we found a p value of 2.12×10^{-6}, which leads us to reject the null hypothesis that subjects all chose randomly. Using Tukey's range test we found that our automated method is significantly different from the nearest method with a 91% confidence interval, and from the cubic method with a 99% confidence interval.

In Section 3, we mentioned that the method of Kopf and Lischinski [Kopf and Lischinski, 2011] is essentially the inverse process of our method; it takes a pixel art piece and converts it to a smooth, vectorized output. Additionally, their method's process of reshaping pixels can be seen as a similar approach to our method of segmentation. Their algorithm computes a mapping between each pixel and a segment, and, like our method, does so by utilizing both neighborhood and color information.

To see how well our method actually serves as an inverse process to their depixeling algorithm, we took the vectorized output of their method as the input of our automated algorithm, setting the size of output image and palette to the same as their input, and compared the results to those of the naive methods. We used the 54 images found in their papers supplemental material. An example can be seen in Figure 5.6. Results were compared by taking the sum of the per-pixel euclidean distance in L*a*b* color space from each method's result and the original input to their algorithm. We found the mean squared per-pixel error and standard deviation for each method was 17.03 and 2.08 (our method), 18.0 and 2.85 (nearest), and 118.71 and 8.08 (cubic).

We hypothesize that the lack of difference between our method and the nearest method is due to the fact that the original input to the depixelizing algorithm was pixel art and while the method creates a smooth output, it does not offset the reshaped pixels far from their original color or position, which means the results still generally

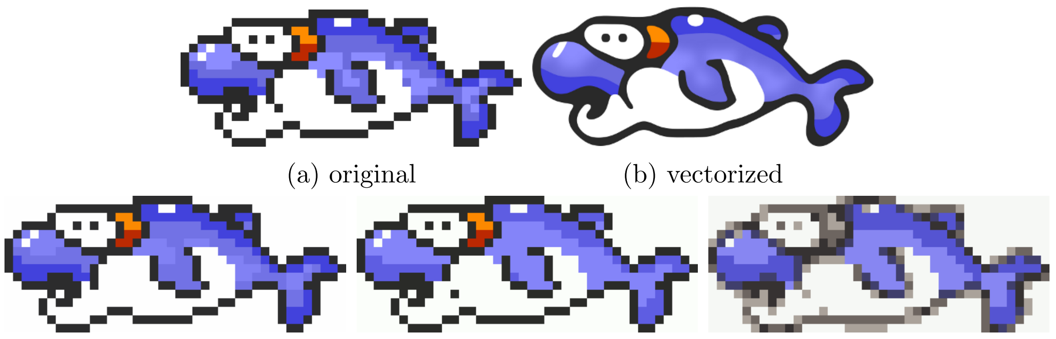

(a) original (b) vectorized
(c) our result (d) nearest result (e) cubic result

Figure 5.6: The original pixel art image (a) (© Nintendo Co., Ltd.) is converted to a vectorized version (b) using Kopf and Lischinski's method [Kopf and Lischinski, 2011]. The vectorized version is then converted back to a pixelated version using our automated method (c) and the two naive methods (d,e).

align with a pixel grid. The poor performance of the cubic method is attributed to its tendency to blur and wash out colors, as seen in Figure 5.6.

We also compared the results of our automated method to the manual results created by expert pixel artists, shown in Figure 5.7. As seen, the pixel artist are able to achieve superior results and incorporate more advanced techniques such as dithering and edge highlighting. While there are many known methods to automatically dither an image,

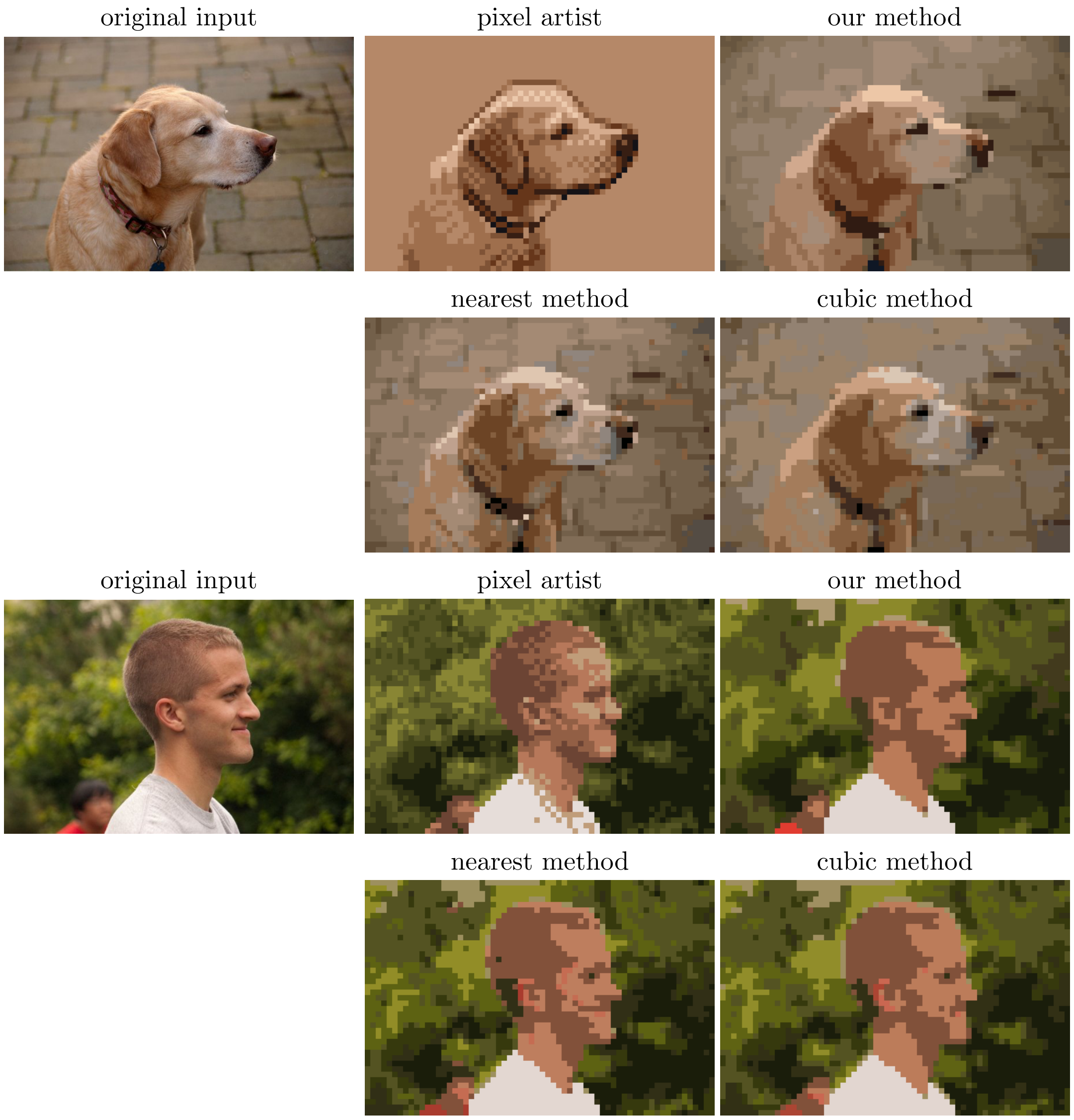


Figure 5.7: Comparing to the work of expert pixel artists. The resolution of all results are 64×43 pixels. (top example) The results generated by our algorithm and the naive methods use 16 colors, while the result generated by the pixel artist, Adam Saltsman, uses 8. (bottom example) The results generated by our algorithm and the naive method use 12 colors, while the result generated by the pixel artist, Ted Martens, uses 11 colors.

it is important to realize when it is appropriate to dither; at these resolutions dithering can often introduce the unwanted appearance of textures on smooth surfaces such as skin. Note how in the second example the artists uses dithering sparingly on the face, but in both examples dithering is intentionally used to give texture to hair and fur. Similarly, edge highlighting requires a nontrivial understanding of where the important edges of the scene are and where the light sources are. As such, the artists are able to heavily leverage their understanding of the scene to make use of these techniques in an effective manner, as well as selectively emphasize important features in the scene, to produce results that our automated method does not match.

However, the artists we interviewed also agreed that our automated method was an improvement over the naive approaches. Each artist was shown the results highlighted in this chapter. Adam Saltsman, creator of Canabalt, Flixel, and Figure 5.7(top), said that our results are “more uniform, more reasonable palette, better forms, more readable.” Ted Martens, creator of the Pixel Fireplace and Figure 5.7(bottom), stated that our algorithm “chooses better colors for the palette, groups them well, and finds shapes better.” Craig Adams, art director of Superbrothers: Sword & Sworcery EP (Figure 1.1(b)), noted that “essential features seem to survive a bit better [and] shapes seem to come through a bit more coherently. I think the snowboarder’s goggles are the clearest example of an essential shape—the white rim of the goggle—being coherently preserved in your process, while it decays in the ‘naive’ process.”

Chapter 6
User Guided Algorithm

6.1 Overview

The algorithm described in Chapter 4 completely automates the entire generation of the result, which comes with some disadvantages. The first is that the method does not make use of advanced techniques commonly used by pixels artists, which would require a greater understanding of the scene's composition. Furthermore, in their feedback, the expert pixel artists we interviewed felt that a purely automated algorithm was too limiting and expressed a desire to have a greater control over the process. In this chapter, we propose a set of user controls that are integrated into our iterative algorithm, and help bridge the gap between the purely manual and purely automated processes. These controls allow the user to have as much or as little control over the process as they want, and therefore take advantage of both the power and speed of the automated method and the knowledge and creativity of the user.

The first user control, proposed in our original paper [Gerstner et al., 2012], is an "importance map" that acts as an additional input to our algorithm and lets the user emphasize areas of the image they believe to be important. The second and third controls, as proposed in our extended paper [Gerstner et al., 2013], are pixel and palette constraints. Using these two controls, the user can directly edit the palette colors and their assignment in the output image, giving them full control over the result. These constraints are used after the automated algorithm initially converges. After each set of edits, the user can choose to have our automated algorithm continue to iterate, using the current result and the user's constraints as a starting point (see Section 6.5). To demonstrate the effectiveness of these user controls, we developed a user interface that was used to generate the results in Chapter 7.

6.2 Importance Map

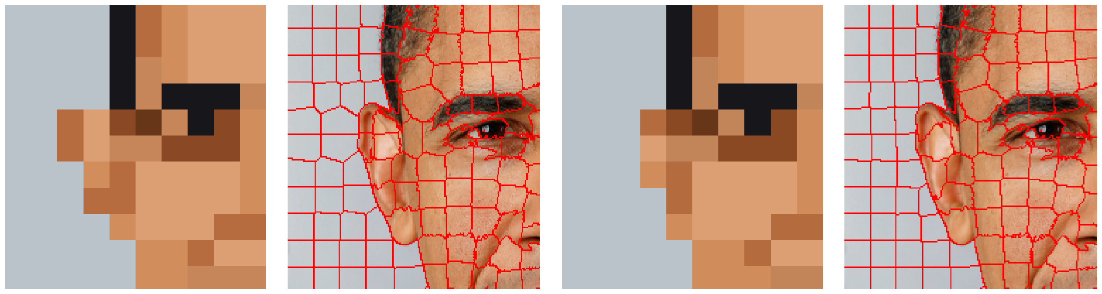
As stated in Chapter 4 our automated method does not favor any image content. For instance, nothing is in place that can distinguish between foreground and background objects, or treat them separately in the output. However, user input (or the output of a computer vision system) can easily be incorporated into our algorithm to prioritize the foreground in the output. This importance map serves as an additional, but optional, input at the beginning of our method. Users can supply a $w_{\text{in}} \times h_{\text{in}}$ grayscale image of weights $W_i \in [0, 1]$, $i \in [1, M]$, used to indicate the importance of each input pixel p_i. In our interface, this is done by using a simple brush to mark areas with the desired weight. We incorporate this map when iterating the palette (Section 4.6) by adjusting the prior $P(p_s)$ used in Equation 4.3. Given the importance map, the value $P(p_s)$ for each superpixel is the average importance of all input pixels contained in superpixel p_s and is given by the equation (normalized across all superpixels):

$$P(p_s) \propto \frac{1}{|p_s|} \sum_{p_i \in p_s} W_i \tag{6.1}$$

$P(p_s)$ thus determines how much each superpixel affects the resulting palette, specifically by emphasizing colors in regions marked as important. Note that while this additional control does not have a direct impact on the superpixel segmentation, due to the interleaved nature of our algorithm, the altered palette will result in a different superpixel configuration.

6.3 Pixel Constraints

In traditional pixel art, the artist needs to manually choose the color of each pixel in the output. In contrast, our automated algorithm makes the choice entirely for the user. By adding a simple constraint into our program, we allow the user to work in the area between these two extremes. For each pixel in the output, the user can choose a subset of colors in the palette that a pixel may select from. For each color not in this subset, the conditional probability with respect to this pixel, $P(c_k|p_s)$, is set to



(a) (b) (c) (d)

Figure 6.1: When the user provides constraints to the image, future iterations of the algorithm will update the superpixels in a way that seeks to decrease error under the new constraints. In this example, the automated output (a), is modified by constraining several pixels of the ear to the background color (c). As a result, the superpixels (b) are redistributed to match the constraints (d). The superpixels that used to be part of the ear now form segments of the background, and neighboring pixels in the output have changed to accommodate the new superpixel distribution.

zero. This restricts the color assigned to the output pixel to the color with the highest conditional probability within the subset. Note this has the convenient property of being equivalent to the manual process when the subset is a single color, and to the automatic process when the subset is the entire palette.

The use of this tool can sometimes result in regions of the input image being under represented, which our algorithm is already built to correct. As explained in Chapter 4.5, superpixels are represented using the color in the palette with the highest conditional probability, $P(c_k|p_s)$. Therefore, adding these constraints will affect the assignment of input pixels to superpixels in future iterations. As a result, when constraints are added by the user, neighboring superpixels will automatically compensate as the algorithm attempts to decrease error under these constraints, as seen in Figure 6.1.

In our interface, we implement this tool as a paint brush, and allow the user to select one or more colors from the palette to form the subset as they paint onto the output image. Using this brush, they are able to choose an exact color for specific pixels, restrict the range of colors for others, and leave the rest entirely to our algorithm.

6.4 Palette Constraints

Similarly, We provide a set of constraints to allow the user control over the generation of the palette used in the output. After the palette has initially converged, the user has the option to edit and fix colors in the palette. This is done in one of two ways. The first is a trivial method; the user directly modifies a specific color in the palette. The second utilizes the information already gathered by our algorithm. By choosing a color in the palette c_k, and then a superpixel p_s formed by our algorithm, we set c_k to the mean color of that region, m_s, as found in Section 4.5. While the first method allows the user to have direct control, the second provides them with a way of selecting a relatively uniform area of the original image from which to sample a color, without having to specify specific values.

In addition to changing the color, the user has the option to keep these colors fixed or free during any future iterations of the algorithm. If they are fixed, they will remain the same color for the rest of the process. If they are not fixed, they will be free to converge to a new value as our algorithm iterates, starting with the initial color provided by the user's edit. This gives the users another dimension of palette control in addition to the ability to manually choose the colors.

Note that when a color is changed in the palette, areas of the original image may no longer be well represented in the palette. As in the case of pixel constraints, during future iterations our algorithm will naturally seek to reduce this discrepancy by updating the unfixed colors in the palette as it attempts to minimize error and converge to a new local minimum.

6.5 Reiterating

After applying pixel or palette constraints, the user has the option of rerunning our algorithm. However, rather than starting from scratch, the algorithm begins with the results of the previous iteration, subject to the constraints specified by the user. When rerunning the algorithm, the temperature remains at the final temperature T_f it reached at convergence, and continues until the convergence condition described in

Section 4.6 is met again. Note that while iterating, the algorithm maintains the user's constraints. Therefore the user can decide what the algorithm can and cannot update. Also note that since the algorithm is not starting from scratch, it is generally close to the next solution, and convergence occurs rapidly (usually less than a second). After the algorithm has converged, the user can continue making edits and rerunning the algorithm until satisfied. In this way the user becomes a part of the iterative loop, and both user and algorithm work to create a final solution.

Chapter 7

User Guided Results

In this chapter we present example results using the user controls specified in Chapter 6 compared to the results generated using the automated method described in Chapter 4. For the automated component of both versions of the algorithm, we again use the same parameters specified in Chapter 4.

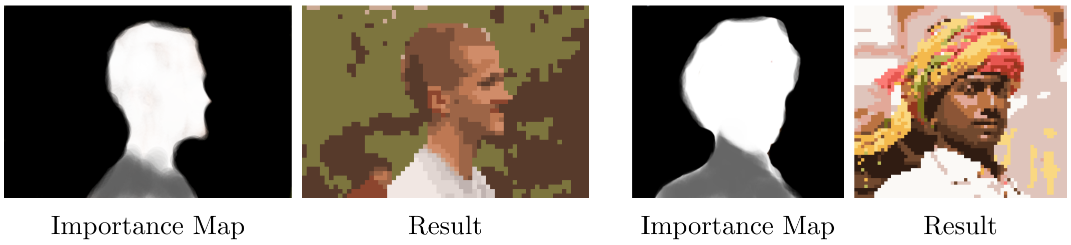


Figure 7.1: Results generated using an importance map. (left) 64×43, 12 colors (right) 64×58, 16 colors

In Figure 7.1, we present two examples from our method using only an importance map as an additional input to the algorithm. For each example we show the importance map used and the resulting output. As shown in both examples, by heavily weighting the figures in the importance map, our algorithm emphasizes the foreground in the resulting palette. The incorporation of an importance map achieves a result closer to those created by an expert pixel artist, which can be seen by comparing the first example to the manual and automated results generated from the same input image in Figure 7.1. The second example exhibits similar improvements to the original result in Figure 5.1.

In Figure 7.2, we demonstrate the results of also incorporating pixel and palette constraints during our iterative process. For each result, we recorded the total number of pixel and palette constraints used to create the final product. In each case, the user spent less than two minutes creating an importance map. Figure 7.2(top) was made using 18 pixel constraints (2.6% of the total number of pixels in the image) and one palette constraint. Figure 7.2(middle) was created using 766 pixel constraints (27.8%) and eight palette constraints. However, it should be noted that we count each new constraint during the process, even if it overwrites a previous constraint. In the final result, only 483 pixels (17.6%) and zero colors were constrained. This discrepancy is due to the user trying several colors for each pixel, which is typical when creating pixel art manually. Figure 7.2(bottom) was created using 2732 (66.7%) pixel constraints and zero color constraints. Of the 2732 pixel constraints, 2140 were created in 27 user operations to make the background a solid color, as can be seen in the last column.

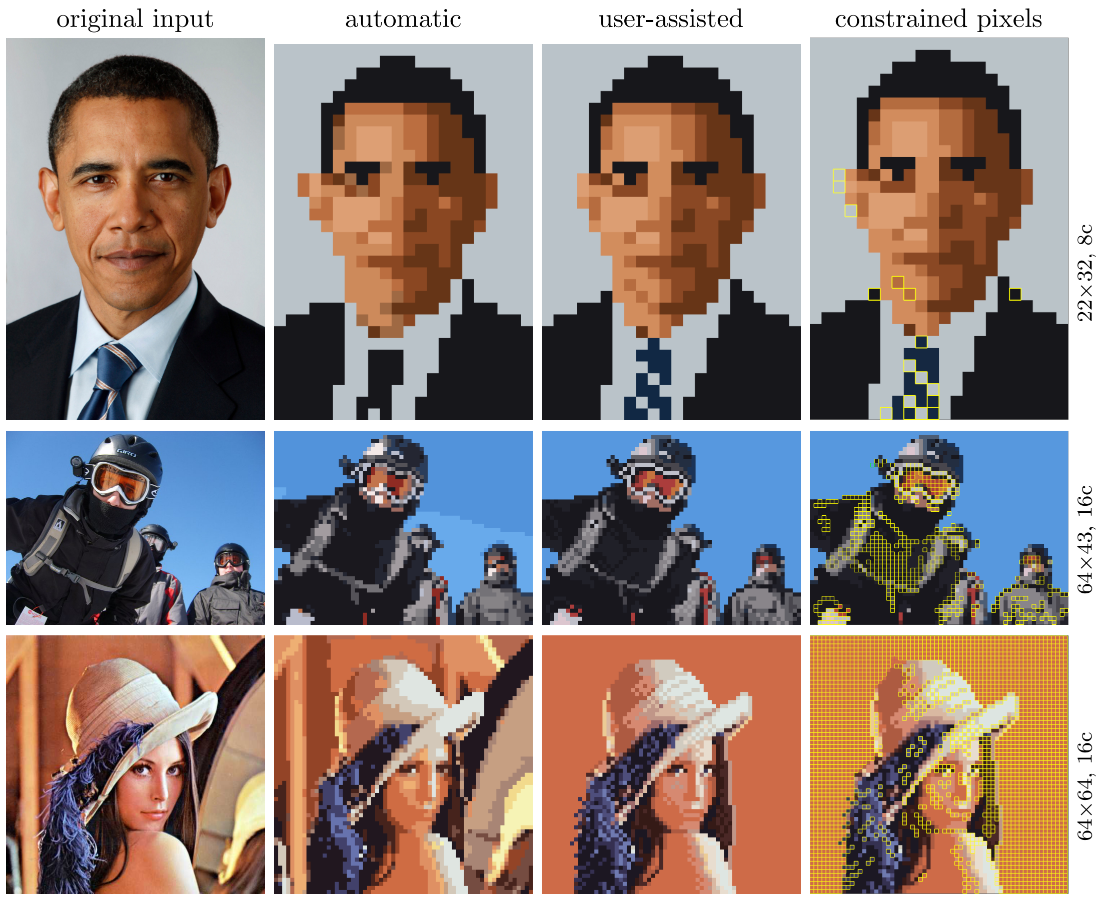


Figure 7.2: The results of the automatic method compared to the results obtained by integrating user input into the iterative process with our interface. The pixel and palette constraints give users the ability to incorporate high level information that the algorithm does not. The last column indicates with a yellow border which pixels were given constraints to produce the user-assisted result.

Incorporating pixel and palette constraints into the automated process give users the ability to work anywhere between a fully automated and a fully manual process. The advantage of this approach can be seen in the results of Figure 7.2 and Figure 7.3. In Figure 7.2(top), the user provides minimal, but effective changes, such as improving the jawline, and removing a skin color in favor of a blue in the palette for the tie. They also introduce a simple striped pattern into the tie, which still represents the original image but no longer has a direct correspondence. In Figure 7.3(left), the user incorporates several of the high level techniques used by the expert pixel artists in Figure 5.7 such as dithering and edge highlighting, and choices such as removing the background, none of which are natively built into our automated algorithm. The image in Figure 7.2(bottom) is a failure case for our automated algorithm, due to the lighting and high variation in the background. However, even with this initially poor output, interleaving the iterative process with user constraints significantly improves the results.

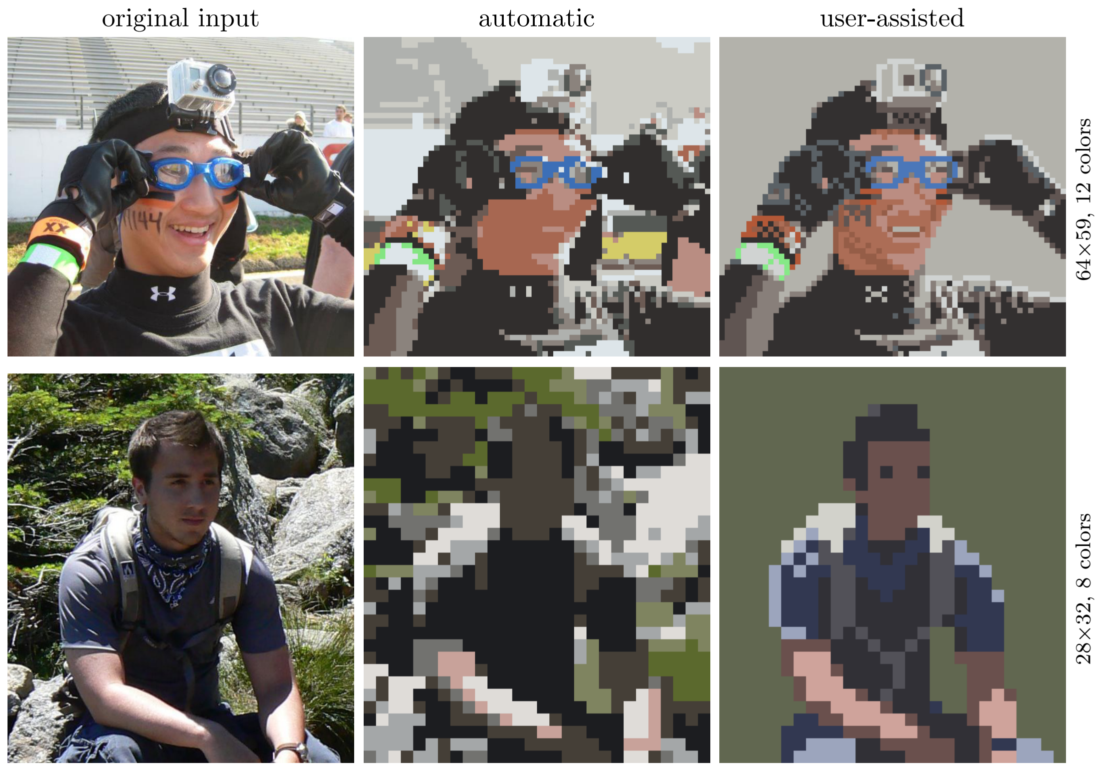


Figure 7.3: (top) Using our proposed tools, the user can incorporate high level techniques such as dithering, and edge highlighting into the final result. (bottom) An example of a failure case for the automated algorithm, the results of which are drastically improved when augmented with user constraints.

Chapter 8

Conclusion

In this thesis we presented an entirely automated method to transform an input image into an output in the style of pixel art. By interleaving iterative steps, our algorithm produces a palette and a mapping of segments of the input image to pixels of the output image that are optimized with respect to each other. We also extend the automated algorithm with a set of controls that allow the user to work seamlessly between the entirely manual and entirely automated processes of generating pixel art.

The results of our automated method exhibit several visual advantages over existing naive methods including the choice of colors in the palette and the retention of important features of the original image. We also demonstrate that it is able to produce recognizable images even at very low resolutions. Our user study indicates that our method is the preferred over the naive methods, which is further supported by the opinion of the expert pixel artists we reviewed.

The results of our semi-automated method demonstrate that incorporating the user's input can dramatically improve the results, even in failure cases, while still offloading much of the computation work to the automated system. Using the controls we proposed, users are able to provide feedback to the algorithm and incorporate their understanding of the scene and their own artistic creativity.

The effectiveness of incorporating the user into the process is encouraging. For future work, we believe it is worthwhile to continue to explore new ways to expand the algorithm by leveraging the user's feedback. Specifically, advanced techniques such as dithering and edge highlighting are difficult to implement in the algorithm without an understanding of the scene. While it may be possible to leverage object recognition and edge detection algorithms to implement these techniques, these are tasks that humans can perform quickly and easily, without requiring the skill of an artist. It therefore seems natural to utilize their knowledge and creativity to produce a final result.

Furthermore, the approach of our method is very general, and does not favor any image content. Future work on this problem may benefit from focusing on a particular type of image content, such as portraits or landscapes, which would have specific characteristics and expected features that an automated algorithm could leverage to produce a better result.

Other avenues for future work include incorporating greater flexibility into the current algorithm. This includes techniques to automatically resize the palette and canvas without reinitialization, and a method to perform palette transfers, which would be beneficial to artists who have a specific palette they would like to work with, as in the case of physical mediums such as post-it notes, cross stitching, building facades, and Lego bricks.

References

[Achanta et al., 2010] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2010). SLIC Superpixels. Technical report, IVRG CVLAB.

[Antonelli, 2013] Antonelli, P. (2013). Video games: 14 in the collection, for starters. `http://www.moma.org/explore/inside_out/2012/11/29/video-games-14-in-the-collection-for-starters`.

[DeCarlo et al., 2003] DeCarlo, D., Finkelstein, A., Rusinkiewicz, S., and Santella, A. (2003). Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855.

[DeCarlo and Santella, 2002] DeCarlo, D. and Santella, A. (2002). Stylization and abstraction of photographs. *ACM Trans. Graph.*, 21:769–776.

[Gerstner et al., 2012] Gerstner, T., DeCarlo, D., Alexa, M., Finkelstein, A., Gingold, Y., and Nealen, A. (2012). Pixelated image abstraction. In *Proceedings of the International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*.

[Gerstner et al., 2013] Gerstner, T., DeCarlo, D., Alexa, M., Finkelstein, A., Gingold, Y., and Nealen, A. (2013). Pixelated image abstraction with user constraints. *Computers & Graphics*.

[Gervautz and Purgathofer, 1990] Gervautz, M. and Purgathofer, W. (1990). Graphics gems. chapter A simple method for color quantization: octree quantization, pages 287–293.

[Gooch et al., 2002] Gooch, B., Coombe, G., and Shirley, P. (2002). Artistic vision: painterly rendering using computer vision techniques. In *Non-Photorealistic Animation and Rendering (NPAR)*, pages 83–90.

[Heckbert, 1982] Heckbert, P. (1982). Color image quantization for frame buffer display. *SIGGRAPH Comput. Graph.*, 16:297–307.

[Inglis and Kaplan, 2012] Inglis, T. C. and Kaplan, C. S. (2012). Pixelating vector line art. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '12, pages 21–28, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Judd et al., 2007] Judd, T., Durand, F., and Adelson, E. H. (2007). Apparent ridges for line drawing. *ACM Trans. Graph.*, 26(3):19–19.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

[Kopf and Lischinski, 2011] Kopf, J. and Lischinski, D. (2011). Depixelizing pixel art. *ACM Trans. Graph.*, 30(4):99–99.

[L'Eclairage, 1978] L'Eclairage, C. I. D. (1978). Recommendations on uniform color spaces, color-difference equations, psychometric color therms. CIE.

[Levinshtein et al., 2009] Levinshtein, A., Stere, A., Kutulakos, K. N., Fleet, D. J., Dickinson, S. J., and Siddiqi, K. (2009). Turbopixels: Fast superpixels using geometric flows.

[MacQueen, 1967] MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297.

[Marr and Hildreth, 1980] Marr, D. and Hildreth, E. (1980). Theory of edge detection. *International Journal of Computer Vision*.

[Nunneley, 2013] Nunneley, S. (2013). Minecraft sales hit 20 million mark for all platforms. `http://www.vg247.com/2013/01/22/minecraft-sales-hit-20-million-mark-for-all-platforms/`.

[Orchard and Bouman, 1991] Orchard, M. and Bouman, C. (1991). Color quantization of images. *IEEE Trans. on Signal Processing*, 39:2677–2690.

[Puzicha et al., 2000] Puzicha, J., Held, M., Ketterer, J., Buhmann, J. M., and Fellner, D. W. (2000). On spatial quantization of color images. *IEEE Transactions on Image Processing*, 9:666–682.

[Rose, 1998] Rose, K. (1998). Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11):2210–2239.

[Secord, 2002] Secord, A. (2002). Random marks on paper: Non-photorealistic rendering with small primitives.

[Shi and Malik, 1997] Shi, J. and Malik, J. (1997). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905.

[Vedaldi and Soatto, 2008] Vedaldi, A. and Soatto, S. (2008). Quick shift and kernel methods for mode seeking. In *In European Conference on Computer Vision, volume IV*, pages 705–718.

[Vermehr et al., 2012] Vermehr, K., Sauerteig, S., and Smital, S. (2012). eboy. http://hello.eboy.com.

[Winnemöller et al., 2006] Winnemöller, H., Olsen, S. C., and Gooch, B. (2006). Real-time video abstraction. *ACM Trans. Graph.*, 25:1221–1226.

[Wu, 1992] Wu, X. (1992). Color quantization by dynamic programming and principal analysis. *ACM Trans. Graph.*, 11:348–372.

Vita

Timothy Gerstner

2013	M.S. in Computer Science, Rutgers University
2010	B. Sc. in Mechanical and Aerospace Engineering, Rutgers University
2006	Graduated from Warren Hills Regional High School.