Pixelated Image Abstraction with Integrated User Constraints

Timothy Gerstner[a], Doug DeCarlo[a], Marc Alexa[c], Adam Finkelstein[b], Yotam Gingold[a,d], Andrew Nealen[a]

[a] *Rutgers University*
[b] *Princeton University*
[c] *TU Berlin*
[d] *Columbia University*

Abstract

We present an automatic method that can be used to abstract high resolution images into very low resolution outputs with reduced color palettes in the style of pixel art. Our method simultaneously solves for a mapping of features and a reduced palette needed to construct the output image. The results are an approximation to the results generated by pixel artists. We compare our method against the results of two naive methods common to image manipulation programs, as well as the hand-crafted work of pixel artists. Through a formal user study and interviews with expert pixel artists we show that our results offer an improvement over the naive methods. By integrating a set of manual controls into our algorithm, we give users the ability to add constraints and incorporate their own choices into the iterative process.

Keywords: pixel art, image abstraction, non-photorealistic rendering, image segmentation, color quantization

1. Introduction

We see pixel art every day. Modern day handheld devices such as the iPhone, Android devices and the Nintendo DS regularly utilize pixel art to convey information on compact screens. Companies like Coca-Cola, Honda, Adobe, and Sony use pixel art in their advertisements [1]. It is used to make icons for desktops and avatars for social networks. While pixel art stems from the need to optimize imagery for low resolution displays, it has emerged as a contemporary art form in its own right. For example, it has been featured by Museum of Modern Art, and there are a number of passionate online communities devoted to it. The "Digital Orca" by Douglas Coupland is a popular sight at the Vancouver Convention Center. France recently was struck by a "Post-it War"[1], where people use Post-It notes to create pixel art on their windows, competing with their neighbors across workplaces, small businesses, and homes.

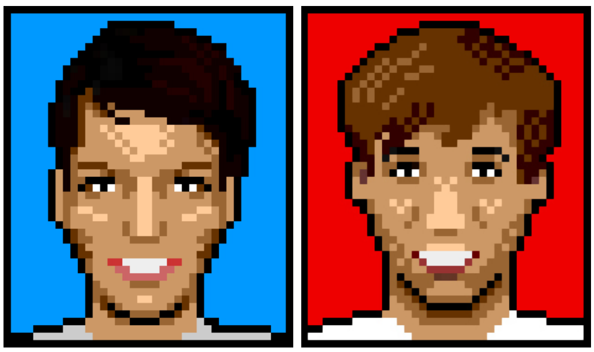
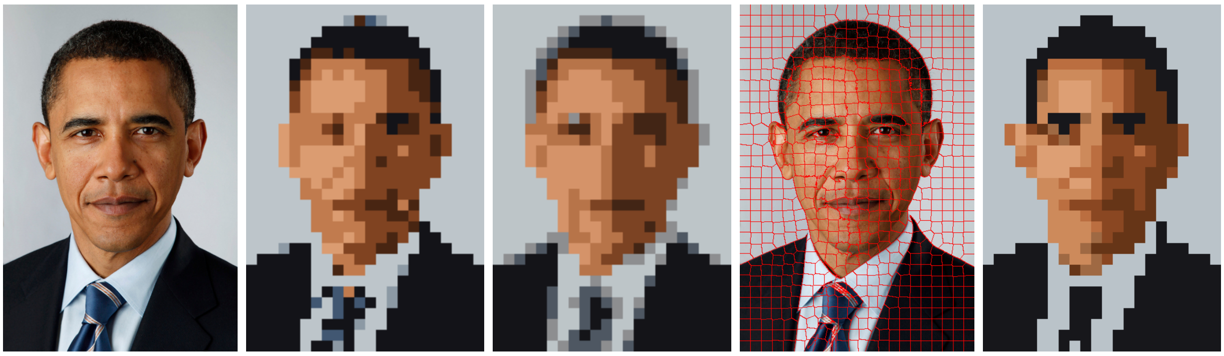


Figure 1: Examples of pixel art. "Alice Blue" and "Kyle Red" by Alice Bartlett. Notice how faces are easily distinguishable even with this limited resolution and palette. The facial features are no longer proportionally accurate, similar to deformation in a caricature.

What makes pixel art both compelling and difficult is the limitations imposed on the medium. With a significantly limited palette and resolution to work with, the task of creating pixel art becomes one of carefully choosing the set of colors and placing each pixel such that the final image best depicts the original subject. This task is particularly difficult as pixel art is typically viewed at a distance

Email address: `timgerst@cs.rutgers.edu` (Timothy Gerstner)

[1] http://www.postitwar.com/



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

Figure 2: Pixel art images simultaneously use very few pixels and a tiny color palette. Attempts to represent image (a) using only 22×32 pixels and 8 colors using (b) nearest-neighbor or (c) cubic downsampling (both followed by median cut color quantization), result in detail loss and blurriness. We optimize over a set of superpixels (d) and an associated color palette to produce output (e) in the style of pixel art.

where the pixel grid is clearly visible, which has been shown to contribute to the perception of the image [2]. As seen in Figure 1, creating pixel art is not a simple mapping process. Features such as the eyes and mouth need to be abstracted and resized in order to be represented in the final image. The end product, which is no longer physically accurate, still gives the impression of an identifiable person.

However, few, if any methods exist to automatically or semi-automatically create effective pixel art. Existing downsampling methods, two of which are shown in Figure 2, do not accurately capture the original subject. Artists often turn to making pieces by hand, pixel-by-pixel, which can take a significant amount of time and requires a certain degree of skill not easily acquired by novices of the art. Automated and semi-automated methods have been proposed for other popular art forms, such as line drawing [3, 4] and painting [5]. Methods such as [6] and [7] not only abstract images, but do so while retaining salient features.

We introduce an entirely automated process that transforms high resolution images into low resolution, small palette outputs in a pixel art style. At the core of our algorithm is a multi-step iterative process that simultaneously solves for a mapping of features and a reduced palette to convert an input image into a pixelated output image. In the first part of each iteration we use a modified version of an image segmentation proposed by Achanta et al. [8] to map regions of the input image to output pixels. In the second step, we utilize an adaptation of mass-constrained deterministic annealing [9] to find an optimal palette and its association to output pixels. These steps are interdependent, and the final solution is an optimization of both the spatial and palette sizes specified by the user. Throughout this process we utilize the perceptually uniform CIELAB color space [10]. The end result serves as an approximation to the process performed by pixel artists (Figure 2, right).

This paper presents an extended edition of Pixelated Image Abstraction [11]]. In addition to an expanded results section, we have added a set of user controls to bridge the gap between the manual process of an artist and the automated process of our algorithm. These controls allow the user to provide as much or as little input into the process as desired, to produce a result that leverages both the strengths of our automated algorithm and the knowledge and personal touch of the user.

Aside from assisting a class of artists in this medium, applications for this work include automatic and semi-automatic design of low-resolution imagery in handheld, desktop, and online contexts like Facebook and Flickr, wherever iconic representations of high-resolution imagery are used.

2. Related Work

One aspect of our problem is to reproduce an image as faithfully as possible while constrained to just a few output colors. Color quantization is a classic problem wherein a limited color palette is chosen based on an input image for indexed color displays. A variety of methods were developed in the 1980's and early 1990's prior to the advent of inexpensive 24-bit displays, for example [12, 13, 14, 15]. A similar problem is that of selecting a small set of custom inks to be used in printing an image [16]. These methods rely only on the color histogram of the input image, and are typically coupled to an independent dithering (or halftoning) method for output in a relatively high resolution image. In our problem where the spatial resolution of the output is also highly constrained, we optimize simultaneously the selection and placement of colors in the final image.

The problem of image segmentation has been extensively studied. Proposed solutions include graph-cut techniques, such as the method proposed by Shi and Malik [17], and superpixel-based methods QuickShift [18], Turbopixels [19], and SLIC [8]. In particular, SLIC (Simple Linear Interative Clustering) produces regular sized and spaced regions with low computational overhead given very few input parameters. These characteristics make SLIC an appropriate starting point for parts of our method.

Mass-constrained deterministic annealing (MCDA) [9] is a method that uses a probabilistic assignment while clustering. Similar to k-means, it uses a fixed number of clusters, but unlike k-means it is independent of initialization. Also, unlike simulated annealing [20], it does not randomly search the solution space and will converge to the same result every time. We use an adapted version of MCDA for color palette optimization.

Puzicha et al. [21] proposed a method that reduces the palette of an image and applies halftoning using a model of human visual perception. While their method uses deterministic annealing and the CIELAB space to find a solution that optimizes both color reduction and dithering, our method instead emphasizes palette reduction in parallel with the reduction of the output resolution.

Kopf and Lischinski [22] 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.

Finally, we show that with minor modification our algorithm can produce "posterized" images, wherein large regions of constant color are separated by vectorized boundaries. To our knowledge, little research has addressed this problem, though it shares some aesthetic concerns with the *artistic thresholding* approach of Xu and Kaplan [23].

3. Background

Our method for making pixel art builds upon two existing techniques, which we briefly describe in this section.

SLIC. Achanta et al. [8] proposed an iterative method to segment an image into regions termed "superpixels." The algorithm is analogous to k-means clustering [24] in a five dimensional space (three color and two positional), discussed for example in Forsyth and Ponce [25]. Pixels in the input image p_i are assigned to superpixels p_s by minimizing

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

where d_c is the color difference, d_p is the positional difference, M is the number of pixels in the input image, N is the number of superpixels, and m is some value in the range $[0, 20]$ that controls the relative weight that color similarity and pixel adjacency have on the solution. The color and positional differences are measured using Euclidean distance (as are all distances in our paper, unless otherwise noted), and the colors are represented in LAB color space. Upon each iteration, superpixels are reassigned to the average color and position of the associated input pixels.

Mass Constrained Deterministic Annealing. MCDA [9] is a global optimization method for clustering that draws upon an analogy with the process of annealing a physical material. We use this method both for determining the colors in our palette, and for assigning one of these palette colors to each pixel—each cluster corresponds to a palette color.

MCDA is a fuzzy clustering algorithm that probabilistically assigns objects to clusters based on their distance from each cluster. It relies on 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, which makes each object equally likely to belong to any cluster. Each time the system locally converges T is lowered (and the variance of each cluster decreases). As this happens, objects begin to prefer favor particular clusters, and as T approaches zero each object becomes effectively assigned to a single cluster, at which point the final set of clusters is produced.

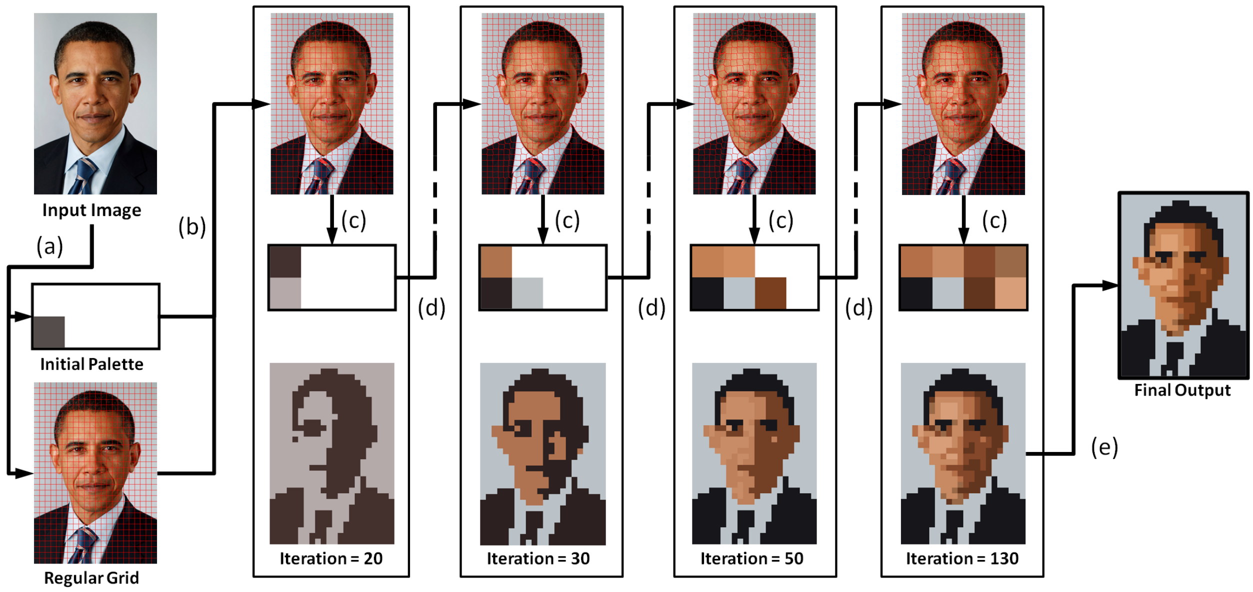


Figure 3: The pipeline of the algorithm. The superpixels (a) are initialized in a regular grid across the input image, and the palette is set to the average color of the M input pixels. The algorithm then begins iterating (b). Each iteration has two main steps: (c) the assignment of input pixels to superpixels, and (d) the assignment of superpixels to colors in the palette and updating the palette. This not only updates each color, but may also add new colors to the palette. After convergence, the palette is saturated (e) producing the final output.

In Section 4.3 we provide a formal definition of the conditional probability we use to assign superpixels to colors in the palette.

Since at high T having multiple clusters is redundant, MCDA begins with a single cluster, represented internally by two sub-clusters. At the beginning of each iteration these sub-clusters are set to slight permutations of their mean. At a high T these clusters converge to the same value after several iterations, but as the temperature is lowered they begin to naturally separate. When this occurs, the cluster is split into two separate clusters (each represented by their own sub-clusters). This continues recursively until the (user specified) maximum number of clusters is reached.

4. Method

Our automated algorithm is an iterative procedure—an example execution is shown in Figure 3. The process begins with an input image of width w_{in} and height h_{in} and produces an output image of width w_{out} and height h_{out} which contains at most K different colors—the palette size. Given the target output dimensions and palette size, each iteration of the algorithm segments the pixels in the input into regions corresponding to pixels in the output and solves for an optimal palette. Upon convergence, the palette is saturated to produce the final output. In this section, we describe our algorithm in terms of the following:

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 region of the input image, denoted as p_s where $s \in [1, N]$. The superpixels are a partition of the input image.

Palette A set of K colors c_k, $k \in [1, K]$ in LAB space.

Our algorithm constructs a mapping for each superpixel that relates a region of input pixels with a single pixel in the output, as in Figure 4. The algorithm proceeds similarly to MCDA, with a superpixel refinement and palette association step performed upon each iteration, as summarized in Algorithm 1. Section 5.1 describes how the algorithm can be expanded to allow a user to indicate important regions in the input image.

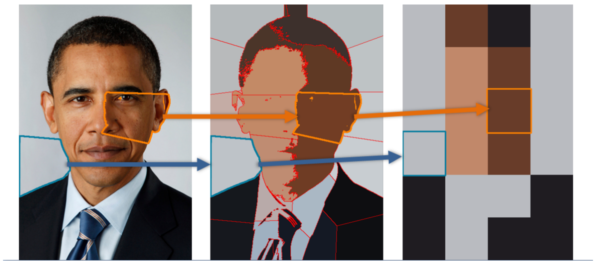


Figure 4: Pixels in the input image (left) are associated with superpixel regions (middle). Each superpixel region corresponds to a single pixel in the output image (right).

Algorithm 1

- ▷ **initialize** superpixels, palette and temperature T (Section 4.1)
- ▷ **while** $(T > T_f)$
  - ▷ **refine** superpixels with 1 step of modified SLIC (Section 4.2)
  - ▷ **associate** superpixels to colors in the palette (Section 4.3)
  - ▷ **refine** colors in the palette (Section 4.3)
  - ▷ **if** (palette converged)
    - ▷ **reduce** temperature $T = \alpha T$
    - ▷ **expand** palette (Section 4.3)
- ▷ post-process (Section 4.4)

4.1. Initialization

The N superpixel centers are initialized in a regular grid across the input image, and each input pixel is assigned to the nearest superpixel (in (x, y) space, measured to the superpixel center). The palette is initialized to a single color, which is set to the mean value of the M input pixels. All superpixels are assigned this mean color. See Figure 3, step (a).

The temperature T is set to $1.1T_c$, where T_c is the critical temperature of the set of M input pixels, defined as twice the variance along the major principal component axis of the set in LAB space [9]. The T_c of a set of objects assigned to a cluster is the temperature at which a cluster will naturally split. Therefore, this policy ensures that the initial temperature is easily above the temperature at which more than one color in the palette would exist.

4.2. Superpixel refinement

This stage of the algorithm assigns pixels in the input image to superpixels, which correspond to pixels in the output image—see steps (b) and (d) in Figure 3.

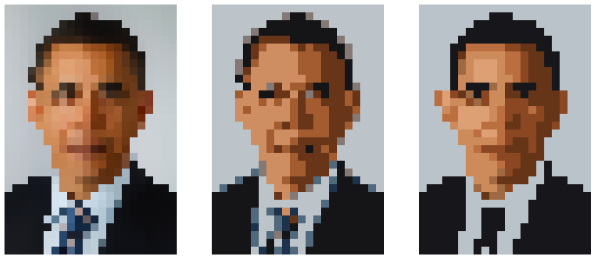


Figure 5: Our method uses palette colors when finding superpixels. Using the mean color of a superpixel works when the palette is unconstrained (left), but fails when using a constrained palette (middle). This is because the input pixels cluster into superpixels based on colors that do not exist in the final image, which creates a discrepancy. Using the palette colors to represent the superpixels (right) removes this discrepancy.

To accomplish this task, we use a single iteration of our modified version of SLIC. In the original SLIC algorithm, upon each iteration, every input pixel is assigned to the superpixel that minimizes $d(p_i, p_s)$, and the color of each superpixel is set to the mean color value of its associated input pixels, m_s. However, in our implementation, the color of each superpixel is set to the palette color that is associated with the superpixel (the construction of this mapping is explained in Section 4.3). This interdependency with the palette forces the superpixels to be optimized with respect to the colors in the palette rather than the colors in the input image. Figure 5 shows the results of using the mean color value instead of our optimized palette used in Figure 2.

However, this also means the color error will be generally higher. As a result, we've found that minimizing $d(p_i, p_s)$ using a value of $m = 45$ is more appropriate in this case (Achanta et al. [8] suggest $m = 10$). This increases the weight of the positional distance and results in a segmentation that contains superpixels with relatively uniform size.

Next, we perform two steps, one modifies each superpixel's (x, y) position for the next iteration, and one changes each superpixel's representative color. Each step is an additional modification to the original SLIC method and significantly improves the final result.

As seen in Figure 6 (left), SLIC results in superpixel regions which tend to be organized in 6-connected neighborhoods (i.e. a hexagonal grid). This is caused by how the (x, y) position of each

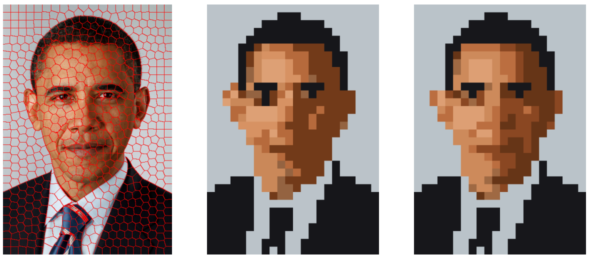


Figure 6: Without the Laplacian smoothing step, the superpixels (left) tend to have 6-connected neighborhoods. This causes small distortions in the output (center), which are particularly noticeable on the ear, eye and mouth, when compared to original output that uses the superpixels that included the smoothing step (right).

superpixel is defined as the average position of the input pixels associated with it. This hexagonal grid does not match the neighborhoods of the output pixels, which are 8-connected (i.e. a rectangular grid) and will give rise to undesirable distortions of image features and structures in the output, as seen in Figure 6(center).

We address this problem with Laplacian smoothing. Each superpixel center is moved a percentage of the distance from its current position to the average position of its 4-connected neighbors (using the neighborhoods at the time of initialization). We use 40%. As seen in Figure 2 (d), this improves the correspondence between the superpixel and output pixel neighborhoods. Specifically, it helps ensure that superpixel regions that are adjacent in the input map are also adjacent pixels in the output. To be clear, it is only in the next iteration when the superpixels will be reassigned based on this new center, due to the interleaved nature of our algorithm.

In our second additional step, the color representatives of the superpixels are smoothed. In the original SLIC algorithm, the representative color for each superpixel is the average color m_s of the input pixels associated with it. However, simply using the mean color can become problematic for continuous regions in the image that contain a color gradient (such as a smooth shadowed surface). While this gradient appears natural in the input image, the region will not appear continuous in the pixelated output.

To remedy this, our algorithm adjusts the values of m_s using a bilateral filter. We construct an image of size $w_{\text{out}} \times h_{\text{out}}$ where each superpixel is assigned the same position as its corresponding output pixel, with value m_s. The colors that results from bilaterally filtering this image, $m_s{}'$ are used while iterating the palette.

4.3. Palette refinement

Palette iteration is performed using MCDA [9]. Each iteration of the palette, as seen in step (c) in Figure 3, can be broken down into three basic steps: **associating** superpixels to colors in the palette, **refining** the palette, and **expanding** the palette. The associate and refine steps occur every iteration of our algorithm. When the palette has converged for the current temperature T, the expand step is performed.

It is important to note how we handle the sub-clusters mentioned in Section 3: we treat each sub-cluster as a separate color in the palette, and keep track of the pairs. The color of each c_k is the average color of its two sub-clusters. When the maximum size of the palette is reached (in terms of the number of distinct colors c_k), we eliminate the sub-clusters and represent each color in the palette as a single cluster.

Associate. The MCDA algorithm requires a probability model that states how likely a particular superpixel will be associated with each color in the palette. See Figure 7. The conditional probability $P(c_k|p_s)$ of a superpixel p_s being assigned color c_k depends on the color distance in LAB space and the current temperature, and is given by (after suitable normalization):

$$P(c_k|p_s) \propto P(c_k)\, e^{-\dfrac{||m_s{}' - c_k||}{T}} \tag{2}$$

$P(c_k)$ is the probability that color c_k is assigned to any superpixel, given the existing assignment.

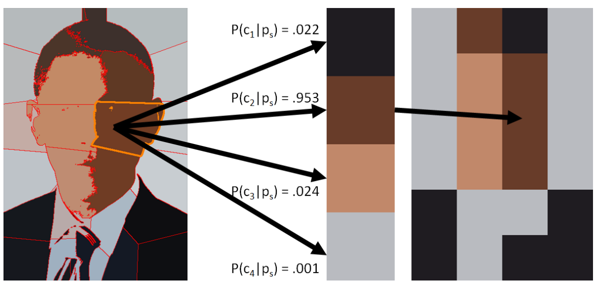


Figure 7: Each superpixel (left) is associated by some conditional probability $P(c_k|p_s)$ to each color in the palette (middle). The color with the highest probability is assigned to the superpixel and its associated output pixel in the final image (right).

Upon initialization, there is only one color, and thus this value is initialized to 1. As more colors are introduced into the palette, the value of this probability is computed by marginalizing over p_s:

$$P(c_k) = \sum_{s=1}^{N} P(c_k|p_s)P(p_s) \tag{3}$$

For the moment, $P(p_s)$ simply has a uniform distribution. This will be revisited in Section 5.1 when incorporating user-specified importance. The values of $P(c_k)$ are updated after the values of $P(c_k|p_s)$ are computed using Equation 2. Each superpixel is assigned to the color in the palette that maximizes $P(c_k|p_s)$. Intermediate results of this assignment can be seen in Figure 3 (bottom row). The exponential distribution in Equation 2 tends towards a uniform distribution for large values of T, in which case each superpixel will be evenly associated with every palette color. As T decreases, superpixels favor colors in the palette that are less distant. At the final temperature, the generic situation after convergence has $P(c_k|p_s) = 1$ for a single color in the palette and $P(c_k|p_s) = 0$ for the rest. In this case, deterministic annealing is equivalent to k-means clustering.

Refine. The next step is to refine the palette by reassigning each color c_k to a weighted average of all superpixel colors, using the probability of association with that color:

$$c_k = \frac{\sum_{s=1}^{N} m_s{'} P(c_k|p_s)P(p_s)}{P(c_k)} \tag{4}$$

This adapts the colors in the existing palette given the revised superpixels. Such changes in the palette can be seen in Figure 3, as the computation progresses.

Expand. Expansion only occurs during an iteration if the palette has converged for the current temperature T (convergence is measured by the total change in the palette since last iteration being less than some small value $\epsilon_{\text{palette}}$). First, the temperature is lowered by some factor α (we use 0.7). Next, the palette is expanded if the number of colors is less than the number specified by the user. For each c_k we check to see if the color needs to be split into two separate colors in the palette. As per MCDA, each color in the palette is represented by two cluster points c_{k_1} and c_{k_2}. We use $||c_{k_1} - c_{k_2}|| > \epsilon_{\text{cluster}}$ (where $\epsilon_{\text{cluster}}$ is a sufficiently small number), to check for palette separation. If so, the two cluster points are added to the palette as separate colors, each with its own pair of cluster points. As seen in Figure 3, over the course of many iterations, the palette grows from a single color to a set of eight (which is the maximum number specified by the user in this example).

After resolving any splits, each color is represented by two sub-clusters with the same value (unless the maximum number of colors have been reached). In order for any color's sub-clusters to separate in the following iterations, c_{k_1} and c_{k_2} must be made distinctly different. To do so, we perturb the sub-clusters of each color by a small amount along the principal component axis of the cluster in LAB space. Rose [9] has shown this to be the direction a cluster will split. This perturbation allows the sub-clusters of each color to merge when $T > T_c$ and separate when $T < T_c$.

Algorithm 1 is defined so that the superpixel and palette refinement steps are iterated until convergence. The system converges when the temperature has reached the final temperature T_f and the palette converges. We use $T_f = 1$ to avoid truncation errors as the exponential component of the Equation 2 becomes small.

4.4. Palette Saturation

As a post-processing step, we provide the option to saturate the palette, which is a typical pixel artist technique, by simply multiplying the a and b channels of each color by a parameter $\beta > 1$. This value used in all our results is $\beta = 1.1$. Lastly, by converting to from LAB to RGB space, our algorithm outputs the final image.

5. User Controls

The algorithm described in Section 4 completely automates the selection of the color palette. This stands in marked contrast to the traditional, manual process of creating pixel art, where the artist carefully selects each color in the palette and its placement in the image. Therefore, we propose a set of user controls that leverage the results of our algorithm and bridges the gap between these two extremes. These controls allow the user to have in as much or as little control over the process as they want. This combines the power and speed of our automated method with the knowledge and creativity of the user.

The first user control, originally proposed in Pixelated Image Abstraction [11], 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 we propose, pixel and palette constraints, are used after the automated algorithm initially converges. 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. After each set of edits, the user can choose to have our automated algorithm continue to iterate using the current result as its starting point with the user's edits as constraints (see Section 5.4). To demonstrate the effectiveness of these user controls, we developed a user interface that was used to generate the results in Figure 16.

5.1. Importance Map

As stated in Section 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. Thus, our system allows additional 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 4.3) by adjusting the prior $P(p_s)$.

Given the importance map, the value $P(p_s)$ for each superpixel is given by the average importance of all input pixels contained in superpixel p_s (and suitable normalization across all superpixels):

$$P(p_s) \propto \frac{1}{|p_s|} \sum_{\text{pixel } i \in p_s} W_i \tag{5}$$

$P(p_s)$ thus determines how much each superpixel affects the resulting palette, through Equations 3 and 4. This results in a palette that can better represent colors in the regions of the input image marked as important.

5.2. 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 in to our program, we can allow the user to work in the area between these two extremes. For each pixel in the output, we allow the user to select a subset of colors in the palette. For each color not in this subset, we set the conditional probability of that color for this pixel, $P(c_k|p_s)$, to 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.

As explained in Section 4.2, 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 naturally

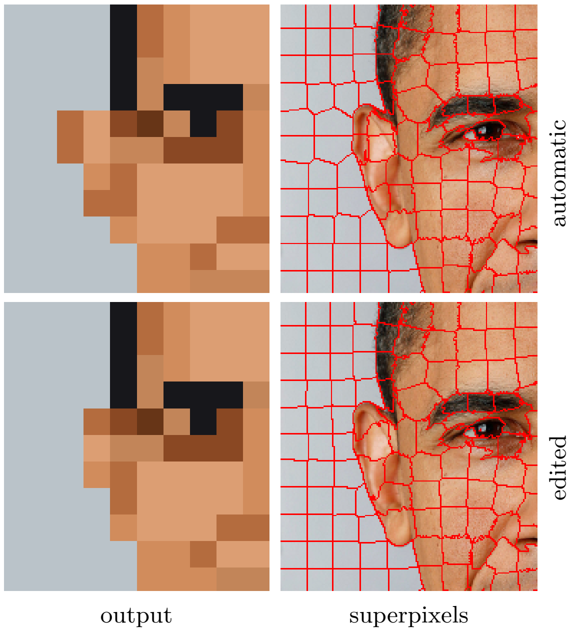


Figure 8: When the user provides constraints to the image, future iterations of the algorithm will update the superpixels in a way that seeks to decrease error under the new constraints. In this example, the original image (top left)), is modified by constraining several pixels of the ear to the background color (bottom left). As a result, the superpixels (top right) are redistributed to match the constraints (bottom right). 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.

compensate as the algorithm attempts to decrease error under these constraints, as seen in Figure 8.

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 the brush, they are able to choose the precise color of specific pixels, restrict the range of colors for others, and leave the rest entirely to our algorithm.

5.3. Palette Constraints

Similarly, in traditional pixel art the artist needs to manually choose each color of the palette. We again provide a set of constraints to give the user control over this process while using our algorithm. 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 4.2. 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, and 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. Fortunately, during any 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.

5.4. Reiterating

After using any of the tools described in this section, 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 4.3 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.

6. Results

We tested our algorithm on a variety of input images at various output resolutions and color palette sizes (Figures 9–16). For each example, we compare *our method* to two naive approaches:

- *nearest method*: a bilateral filter followed by median cut color quantization, followed by nearest neighbor downsampling,
- *cubic method*: cubic downsampling followed by median cut color quantization. Unless otherwise stated, the results are generated using only our automated algorithm, and no user input was integrated into the result.

All of our results use the parameter settings from Section 4. Each result was produced in generally less than a minute on an Intel 2.67Ghz i7 processor with 4GB memory. Each naive result is saturated using the same method described in Section 4.4. Please note it is best to view the results up-close or zoomed-in, with each pixel being distinctly visible.

In Figure 9, we show the effects of varying the number of colors in the output palette. Our automatic method introduces fewer isolated colors than the nearest method, while looking less washed out than the cubic method. As the palette size shrinks, our method is better able to preserve salient colors, such as the green in the turban. Our method's palette assignment also improves the visibility of the eyes and does not color any of the face pink.

Figure 9: Varying the palette size (output images are 64×58).

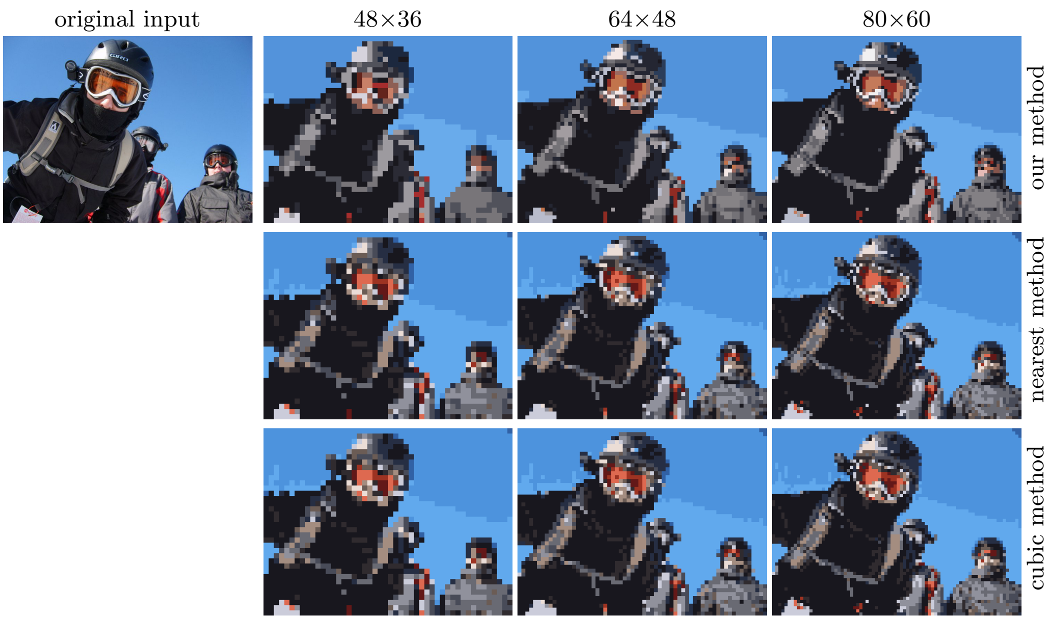


Figure 10: Varying the output resolution (palette has 16 colors).

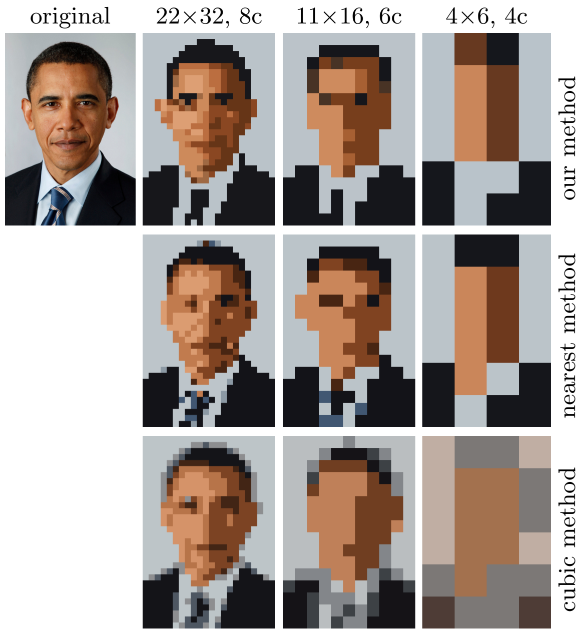


Figure 11: Examples of very low resolution and small palette sizes.

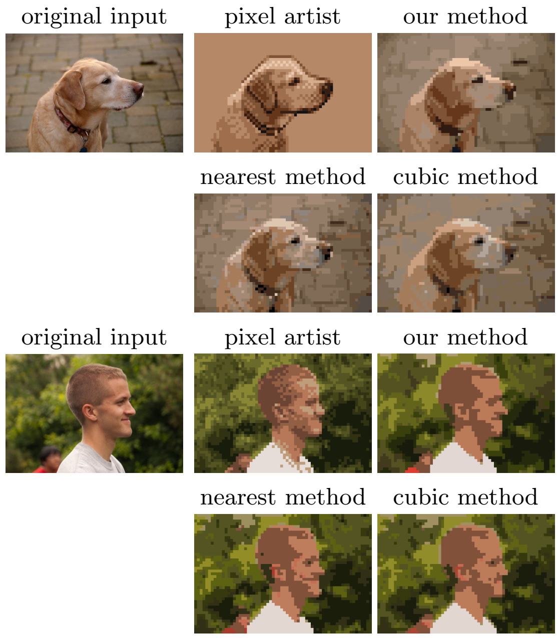


Figure 12: Comparing to the work of expert pixel artists (64×43). The results generate from our method and the naive methods use 16 colors in the first example, 12 in the second. The pixel artists use 8 colors in the first example, 11 in the second.

Similar results are seen in Figure 10 when we vary the output resolution. Again we see that the cubic method produces washed-out images and the nearest method has a speckled appearance. At all resolutions, our method preserves features such as the goggles more faithfully, and consistently chooses more accurate skin tones for the faces, whereas both naive methods choose gray.

Using our automated algorithm, the image of Barack Obama is recognizable even at extremely small output resolutions and palette sizes (Figure 11). At 22×32 and 11×16, our method more clearly depicts features such as the eyes while coloring regions such as the hair and tie more consistently. At 11×16, the nearest method produces a result that appears to distort facial features, while the cubic method produces a result that "loses" the eyes. At 6×4, results are very abstract, but our method's output could still be identified as a person or as having originated from the input.

In Figure 12, we compare our automated output to manual results created by expert pixel artists. While our results exhibit the same advantages seen in the previous figures over the naive methods, they do not match the results made by artists. Expert artists are able to heavily leverage their human understanding of the scene to emphasize and deemphasize features and make use of techniques such as dithering and edge highlighting. While there are many existing methods to automatically dither an image, at these resolutions the decision on when to apply dithering is nontrivial, and uniform dithering can introduce undesired textures to surfaces (such as skin).

Figure 13 contains additional results computed using various input images. Overall, our automated approach is able to produce less noisy, sharper images with a better selection of colors than the naive techniques we compared against.

To verify our analysis, we conducted a formal user study with 100 subjects using Amazon Mechanical Turk. Subjects were shown the original image and the results of our automated method and the two naive methods. The results were scaled to approximately 256 pixels along their longest dimension using nearest neighbor upsampling, so that users could clearly see the pixel grid. We asked subjects the question, "Which of the following best represents the image above?" Subjects responded by choosing a result image. The stimulus sets and answer choices were randomized to remove bias. The study consisted of the example images and parameters shown in our paper, excluding the results gen-

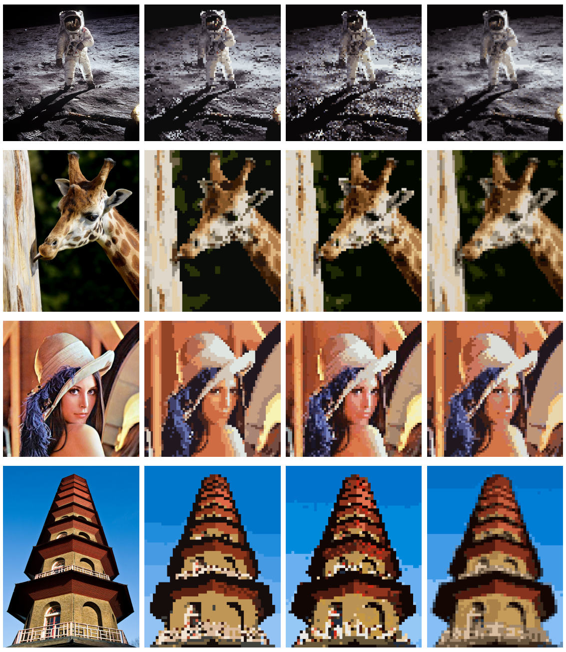


Figure 13: Additional results at various resolution and palette sizes. Columns (left to right): input image, output of our algorithm, output of the nearest method, output of the cubic method.

erated using user input, and each stimulus was duplicated four times (sixty total).

We accounted for users answering randomly by eliminating the results of any subject who gave inconsistent responses (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. The final results show that users choose our results 41.49% of the time, the nearest method 34.52% of the 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. While we acknowledge that the question asked is difficult one given that it is an aesthetic judgment, we believe the results of this study still show subjects prefer the results of our method over the results of either naive method.

We also received feedback from three expert pixel artists on our automated method; each concluded that the automated results are, in general, an improvement over the naive approaches. Ted Martens, creator of the Pixel Fireplace, said that our algorithm "chooses better colors for the palette, groups them well, and finds shapes better." Adam Saltsman, creator of Canabalt and Flixel, characterized our results as "more uniform, more reasonable palette, better forms, more readable." Craig Adams, art director of Superbrothers: Sword & Sworcery EP, observed 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."

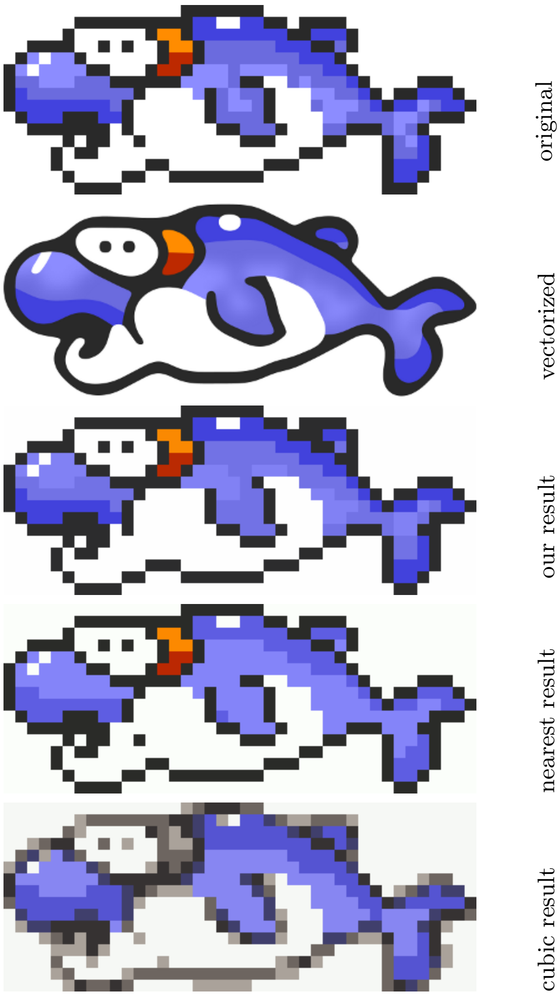


Figure 14: The original pixel art image (© Nintendo Co., Ltd.) is converted to a vectorized version using Kopf and Lischinski's method [22]. The vectorized version is then converted back to a pixelated version using our automated method and the two naive methods.

In Section 2, we mentioned that the method of Kopf and Lischinski [22] is essentially the inverse process of our method; it takes a pixel art piece and converts it to a smooth, vectorized output. To see how well our method actually serves as the inverse process, 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. The results, compared to those of the naive methods, are shown in Figure 14. Visually our method appears to outperform either naive method, and obtains a result that is similar to their original input. To quantify the effectiveness of our method, we took the sum of the Euclidean distance in LAB space of every pixel between our output and their input. We did the same for the naive methods. We found the total error for our method, the nearest method and the cubic method to be 1.63×10^3, 3.05×10^3 and 9.84×10^3, respectively. In other words, our method has 47% less error than the nearest method, and 83.5% less error than the cubic method.

In Figure 15, we present results from our method using an the importance map as an additional input to the algorithm. The results are closer to those created by expert pixel artist. Figure 15(left) allocates more colors in the palette to the face of the person, similar to the manual result in Figure 12. Figure 15(right) also shows an improvement over the non-weighted results in Figure 9. For both examples, the importance map emphasizes the face and de-emphasizes the background; consequently, more colors are allocated to the face in each example at the expense of the background.

In Figure 16, we demonstrate the advantage of allowing the user to also place pixel and palette constraints during our iterative process. In Figure 16(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, and would not be achievable by our algorithm alone. These changes took less than a minute to make.

The improved result in Figure 16(middle row) is achieved by interleaving multiple steps of user constraints and iterations of our algorithm. The user is also able to incorporate the advanced techniques observed in Figure 12 such as dithering and edge highlighting, which are not natively built into our algorithm.

Figure 15: Results using an importance map. (top)64×58, 16 colors (bottom) 64×43, 12 colors

The image in Figure 16(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, by interleaving the iterative process with user constraints (such as restricting the background to a single color) the results are significantly improved.

Finally, while not the direct goal of our work, we briefly mention a secondary application of our method, image *posterization*. This artistic technique uses just a few colors (originally motivated by the use of custom inks in printing) and typically seeks a vectorized output. Adobe Illustrator provides a feature called LiveTrace that can posterize the image in Figure 2(a), yielding Figure 17(a) with only 6 colors. To our knowledge, little research has addressed this problem, though it shares some aesthetic concerns with the*artistic thresholding* approach of Xu and Kaplan [23]. A simple modification to our optimization that omits the smoothing step (Figure 6-left) and then colors the original image via associated superpixels gives us Figure 17(b), which makes a more effective starting point for vectorization. The resulting Figure 17(c) offers improved spatial and color fidelity, based on a good faith effort to produce a similar style in Illustrator.

7. Conclusion, Limitations and Future work

We present a multi-step iterative process that simultaneously solves for a mapping of features and a reduced palette to convert an input image to a pixelated output image. Our method demonstrates sev-

Figure 16: The results of the automatic method compared to the results obtained by integrating user input into the iterative process with our interface. Note that the user can choose to make only a few key edits (top), or they can leverage their understanding of the image to drastically improve images that are otherwise difficult for the automated algorithm (bottom).

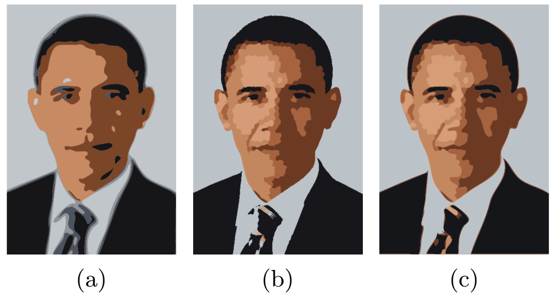


Figure 17: (a) vectorized photo posterized with Illustrator (6 colors). (b) Optimization without Laplacian smoothing, coloring associated input pixels (6 colors). (c) Vectorizing *b* in Illustrator yields similar style with better spatial and color fidelity than *a*.

eral advantages over the naive methods. Our results have a more vibrant palette, retain more features of the original image, and produce a cleaner output with fewer artifacts. While the naive methods produce unidentifiable results at very low resolutions and palette sizes, our approach is still able to create iconic images that conjure the original image. Thus our method makes a significant step towards the quality of images produced by pixel artists.

Nevertheless, our method has several limitations which we view as potential avenues for future research. While pixel artists view the results of our automated algorithm as an improvement, they also express the desire to have a greater control over the final product.

To address these concerns, we implemented several controls that allow the user to give as much or little feedback into the automated process as they desire. By incorporating an importance map we give the user the ability to guide the palette selection, and by giving the user the ability to provide pixel and palette constraints and interleave them with our algorithm, we remove the gap between the manual and automated methods of producing pixel art.

The results of combining these user constraints into our iterative algorithm are encouraging. For future work, we wish to expand on our proposed method and user controls to increase the interaction between the automated algorithm and the user. Our goal is to create a complete system that incorporates the speed and power of an automated method to assist artists in their entire process, without restricting the control of the artists over the final result.

As such, the next step is to explore how the user's feedback can help inform more advanced pixel art techniques in our algorithm, such as those that would produce edge highlighting and dithering. We'd also like to look into ways of automatically performing palette transfers, which would allow potential applications of this work to include, for example, reproduction of an image in repeating tiles like Lego, or design for architectural facades composed of particular building materials like known shades of brick. Currently, our algorithm is limited to working with colors that are similar to the original image due to the nature of how we minimize error, and such an application is not possible without the user applying a large number of constraints.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback. We also wish to thank pixel artists Craig Adams, Ted Martens, and Adam Saltsman for their advice and comments. 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 permission: Figure 1 by Alice Bartlett, Figure 9 by Louis Vest, Figure 13 (giraffe) by Paul Adams, and Figure 13 (pagoda) by William Warby. The pixel art in Figure 12 is copyright Adam Saltsman (top) and Ted Martens (bottom).

References

[1] Vermehr K, Sauerteig S, Smital S. eboy. http://hello.eboy.com; 2012.

[2] Marr D, Hildreth E. Theory of edge detection. International Journal of Computer Vision 1980;.

[3] DeCarlo D, Finkelstein A, Rusinkiewicz S, Santella A. Suggestive contours for conveying shape. ACM Trans Graph 2003;22(3):848–55.

[4] Judd T, Durand F, Adelson EH. Apparent ridges for line drawing. ACM Trans Graph 2007;26(3):19–.

[5] Gooch B, Coombe G, Shirley P. Artistic vision: painterly rendering using computer vision techniques. In: Non-Photorealistic Animation and Rendering (NPAR). ISBN 1-58113-494-0; 2002, p. 83–90. doi:http://doi.acm.org/10.1145/508530.508545. URL `http://doi.acm.org/10.1145/508530.508545`.

[6] DeCarlo D, Santella A. Stylization and abstraction of photographs. ACM Trans Graph 2002;21:769–76. doi:http://doi.acm.org/10.1145/566654.566650. URL `http://doi.acm.org/10.1145/566654.566650`.

[7] Winnemöller H, Olsen SC, Gooch B. Real-time video abstraction. ACM Trans Graph 2006;25:1221–6.

[8] Achanta R, Shaji A, Smith K, Lucchi A, Fua P, Süsstrunk S. SLIC Superpixels. Tech. Rep.; IVRG CVLAB; 2010.

[9] Rose K. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. Proceedings of the IEEE 1998;86(11):2210–39. doi:10.1109/5.726788.

[10] Sharma G, Trussell HJ. Digital color imaging. IEEE Transactions on Image Processing 1997;6:901–32.

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

[12] Gervautz M, Purgathofer W. Graphics gems. chap. A simple method for color quantization: octree quantization. ISBN 0-12-286169-5; 1990, p. 287–93.

[13] Heckbert P. Color image quantization for frame buffer display. SIGGRAPH Comput Graph 1982;16:297–307.

[14] Orchard M, Bouman C. Color quantization of images. IEEE Trans on Signal Processing 1991;39:2677–90.

[15] Wu X. Color quantization by dynamic programming and principal analysis. ACM Trans Graph 1992;11:348–72.

[16] Stollnitz EJ, Ostromoukhov V, Salesin DH. Reproducing color images using custom inks. In: Proceedings of SIGGRAPH. ISBN 0-89791-999-8; 1998, p. 267–74.

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

[18] Vedaldi A, Soatto S. Quick shift and kernel methods for mode seeking. In: In European Conference on Computer Vision, volume IV. 2008, p. 705–18.

[19] Levinshtein A, Stere A, Kutulakos KN, Fleet DJ, Dickinson SJ, Siddiqi K. Turbopixels: Fast superpixels using geometric flows. 2009.

[20] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220:671–80.

[21] Puzicha J, Held M, Ketterer J, Buhmann JM, Fellner DW. On spatial quantization of color images. IEEE Transactions on Image Processing 2000;9:666–82.

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

[23] Xu J, Kaplan CS, Mi X. Computer-generated papercutting. In: Proceedings of Pacific Graphics. 2007, p. 343–50.

[24] MacQueen JB. Some methods for classification and analysis of multivariate observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. 1967, p. 281–97.

[25] Forsyth DA, Ponce J. Computer Vision: A Modern Approach. Prentice Hall; 2002.