Semi-automatic colorization of images

Image and video colorization based on prioritized source propagation
Jun-Hee Heu, Dae-Young Hyun, Chang-Su Kim, and Sang-Uk Lee

Fast image and video colorization using chrominance blending
Liron Yatziv and Guillermo Sapiro

Guillaume Afanou, Nicolas Couture, Gladys Fallourd, and Anas Waalam

January 20, 2012

Persons in charge: Aurélie Bugeau and Vinh-Thong Ta
Teaching assistant: Pascal Desbarats
English teacher: Jean-Jacques Bernaulte

Abstract

Semi-automatic colorization represents a computer-assisted process of adding colors to grayscale still images or monochromatic videos. This process is used for example for the restoration of old movies, cartoons or special effects. A user has to scribble the color of some selected pixels in a source image and the program will automatically propagate to the remaining regions. The main goal of this project is to analyze two articles [1, 2], implement colorization algorithms and design an interactive interface in order to experiment these algorithms.
Contents

1 Introduction ......................................................... 3

2 State of the art .................................................... 3
  2.1 Color transfering schemes ...................................... 3
  2.2 Propagation-based schemes .................................... 4

3 Articles chosen ...................................................... 6
  3.1 Notations ........................................................ 6
  3.2 Fast image and video colorization using chrominance blending [1] .............................................. 6
    3.2.1 Principle of the article .................................. 6
    3.2.2 Implementation details .................................. 7
    3.2.3 Study of the complexity .................................. 8
    3.2.4 Algorithm ................................................ 9
  3.3 Image and video colorization based on prioritized source propagation [2] .................................. 11
    3.3.1 Principle of the article .................................. 11
    3.3.2 Implementation details .................................. 11
    3.3.3 Study of the complexity .................................. 11
    3.3.4 Algorithm ................................................ 12

4 Architecture ......................................................... 12
  4.1 Libraries ......................................................... 12
  4.2 Implementation ................................................ 12

5 Experimental results ............................................... 14
  5.1 Performance tests .............................................. 14
  5.2 Quality of algorithms ......................................... 16

6 Conclusion .......................................................... 16

7 Acknowledgements .................................................. 17

A Annexe : Color spaces .............................................. 23
1 Introduction

Colors play an important role in the human perception of visual information. Colorization is the art of adding colors to a monochromatic image or movie. This process of coloring photos and films is not recent. Hand coloring of photographs has been around since 1839 with the painter and printmaker Johann Baptist Isenring, who used a mixture of gum arabic and pigments to color daguerreotypes [3]. It was practiced in motion pictures in the early 1900s by the French Company Pathe, where many films were colored by hand. It was also practiced for filmstrips in the 1930s.

However, manual colorization is too time-consuming and expensive, especially in the case of video sequences, consisting of a huge number of still images. Thus, a semi-automatic colorization algorithm is required. A computer-assisted process was introduced in 1970 for adding colors to grayscale movies. The process was done by tracing the original grayscale frames onto new animation cels, and then adding color to the new cels [4].

It exists now many colorization methods in the literature. Generally, they can be differentiated in two types: the color-transfering schemes, which use the luminance keying approach, and the propagation-based schemes, which use propagation methods. Our project addresses the following problematic: “How to colorize quickly a monochromatic image, while obtaining high-quality results and reliable performances?”

To address this issue, two articles will be studied [1, 2]. Each one deals with a colorization method. The algorithm of each method will be implemented and an interactive interface will be designed in order to experiment and test these algorithms. Then, the results of colorization methods of articles will be analysed and compared.

This research paper contains all information on our project. Section 2 presents the state of the art of semi-automatic colorization. In Section 3, the two articles are described. A detailed explanation of the methods and a description of the algorithms is given. This is the heart of this research paper. Then Section 4 deals with the architecture used in order to implement these algorithms and the user interface. The final state of our software is presented in Section 5. This part contains the performed tests, experimental results and comparisons between the two methods implemented. Finally, Section 6 is devoted to the conclusion of this project. This conclusion is a synthesis of the project and suggests some improvements.

2 State of the art

Semi-automatic colorization represents a computer-assisted process of adding color to grayscale still images or monochromatic movies. This process is used to modernize grayscale movies, old documentary movies or to restore color films. It also can be found in cartoons. Our project consists in the semi-automatic colorization of grayscale images. It exists many colorization methods in the literature and generally, they can be differentiated into two types: the color-transfering schemes and the propagation-based schemes.

2.1 Color transferring schemes

These types of algorithms use the luminance keying approach. Everything in the image over, or below, a set brightness level is keyed out and replaced by either another image, or a color from a color generator. So, in algorithms using color transferring schemes, there is a color transfer from
Figure 1 illustrates an image colorization using a color transferring scheme. Different such transfering schemes have been implemented, among which:

- **Reinhard et al** [6] introduced a method for a more general form of color correction that borrows one image’s color characteristics from another. This method transfers colors from a color source image to a gray target image by matching the luminance components of the two images.

- **Welsh et al** [5] improved the previous algorithm by exploiting the luminance values and textures rather than just the grayscale values.

- **Chen et al** [7] used manual segmentation to divide the grayscale image into a set of layers. Then Bayesian image matting is used to estimate an alpha channel. This decomposition allows to apply colorization using Welsh’s method [5]. Finally, alpha-blending is used to construct the final image.

- **Sykora et al** [8] presented a color-by-example technique which combines image segmentation, patch-based sampling and probabilistic reasoning. This method is able to automate colorization when new color information is applied on the already designed grayscale cartoon.

These color transferring schemes provide acceptable colorization performance but provided that an input image has distinct luminance values or textures across object boundaries.

### 2.2 Propagation-based schemes

In these schemes, algorithms use propagation methods. They have the following operating mode: a user assigns colors to some pixels (source pixels) and then the method propagates those colors to the other remaining pixels. These methods define an other approach which assumes that the geometry of the grayscale information provides the geometry of the image. Figure 2 shows a grayscale image with scribbles drawn by a user and the result image colorized using a propagation-based scheme.

The main propagation-based schemes can be listed as follows:

- **Horiuchi** [9] used a probabilistic relaxation method by assuming a restricted condition to minimize the total of the color difference defined among adjacent pixels.
Levin et al [10] formulated the propagation problem as the minimization of a quadratic cost function. They solved this problem by computing the difference between a pixel and its weighted average neighborhood colors. Then they assumed that neighboring pixels with similar intensities should have similar colors. For video colorization, they estimated an optical flow between consecutive frames and defined the matching pixels as temporal neighbors.

Sapiro [11] proposed to inpaint the colors constrained both by the geometry, the structure of the monochromatic image and the provided color samples. The first one is given by the image’s gradient information and is represented as the geometry and structure of the whole colored version. Then, the color is obtained by solving a partial differential equation that propagates the color of source pixels.

Horiuchi and Hiriano [12] presented an algorithm that propagates colored seed pixels in all directions by minimizing the color difference among connected 4-pixels. This algorithm, unlike the previous ones, is fast and does not imply intensive computational cost to obtain good-quality results. But the method produces visible artifacts or block distortion since no color blending is performed.

Our project focuses on the following algorithms.

Yatziv and Sapiro [1] blended the colors of source pixels to paint a target pixel based on geodesic distances. These distances measure the variation of luminances along the path, from the source pixels to the target pixel. The authors considered temporal neighbors to extend these algorithms to video colorization.

Heu et al [2] proposed an efficient colorization algorithm for images and videos based on prioritized source propagation. The algorithm identifies the non-source pixel with the highest priority, the one that can be more reliably colorized. Its color is interpolated from the neighboring pixels. This is repeated until all non-source pixels are colorized.

Generally, these schemes provide more reliable performances than the previous ones. However, they may yield color blurring errors and their performances are affected significantly by the location of scribbles.
3 Articles chosen

During this project, we have to study the methods proposed in [1] and [2]. They use propagation methods. For each method, an algorithm will be implemented in order to compare its qualitatively and quantitatively.

3.1 Notations

To describe methods and algorithms, the following notations are used:

- $Y(p)$: luminance of the pixel $p$
- $\text{chrominance}(p)$: chrominance associated to the pixel $p$
- $N_p$: set of neighbors of $p$
- $d(s,t)$: intrinsic distance between two pixels $s$ and $t$
- $\text{MaxDistance}$: an arbitrary maximum value
- $\text{MaxBlendColor}$: maximum number of colors used for the blending
- $l\text{Chrominance}$: array of size $\text{MaxBlendColor}$ of chrominances
- $l\text{Distance}$: array of corresponding intrinsic distances to each chrominance in $l\text{Chrominance}$
- $l\text{Distance}_c(p)$: corresponding intrinsic distance to the chrominance $c$
- $C(s,t)$: curve linking the two pixels $s$ and $t$
- $\nabla Y$: luminance derivative (gradient)
- $d_c(t)$: minimum intrinsic distance from the chrominance $c$ to the pixel $t$
- $\Omega_c$: set of given colors with the chrominance $c$
- $W$: weighting factor defined by
  \[ W(r) = r^{-b} \quad (1) \]
  where $b$ represents a blending factor and $r$ the intrinsic distance
- $a(p)$: accuracy of pixel $p$ where $a(p) = 1$ if $p$ is colorized, 0 otherwise
- $\pi(p)$: priority of the non-source pixel $p$
- $w(p,q)$: weight for two pixels $p$ and $q$

3.2 Fast image and video colorization using chrominance blending [1]

3.2.1 Principle of the article

In this article, the method proposed is based on the concept of color blending. Color blending is a fast propagation technique that colorizes a grayscale image, achieving satisfactory results. This process uses $Y'CbCr$ color space, detailed in Annexe A. A user first scribbles colors on some selected pixels (set of source color pixels) as shown in Figure 3.
Then, the process automatically propagates those colors to the remaining regions which are not colorized. The objective of this method is to propagate the color of source pixels to their neighbors. This propagation uses intrinsic distances computed with the luminance channel of a pixel and the minimum distance $d$ is computed as follows:

$$d(s, t) = \min_{C(s, t)} \int_0^1 |\nabla Y.C(p)| dp$$

(2)

where $s$ and $t$ represent two arbitrary pixels in the image and $C(p)$ is a point belonging to $C(s, t)$. $\nabla Y$ and $C(s, t)$ are defined in Section 3.1. The intrinsic distance between two adjacent pixels is defined as the absolute value of the intensity difference. If two pixels have different luminances, the gradient is large, which involves that there is a contour line between two pixels. Otherwise, the gradient is low and the two pixels have similar chrominances.

But the intrinsic distance between two adjacent pixels is not sufficient. For this algorithm, we would like to get an idea of a distance from a certain chrominance $c$ to any pixel in the image. The intrinsic distance of two arbitrary pixels is defined as the minimum accumulative sum of intrinsic distances:

$$d_c(t) = \min_{\forall s \in \Omega_c:chrominance(s)=c} d(s, t)$$

(3)

where $chrominance(s)$, $\Omega_c$ and $d(s, t)$ are defined in Section 3.1.

### 3.2.2 Implementation details

In order to compute the minimum accumulative sum of intrinsic distances (3), the algorithm uses the Dijkstra graph algorithm [13]. Let us recall that Dijkstra’s algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph. At the end of Dijkstra’s algorithm, a shortest path tree is obtained. For a given source vertex in the graph, the algorithm finds the path with the lowest cost (i.e., the shortest path) between that vertex and every other vertex.
Concerning the method of this article, the process presents similarities with the Dijkstra’s algorithm. A grayscale image is considered as a graph. On this graph, minimum intrinsic distances to each color in \( \Omega_c \) are computed for each pixel. Pixels have two arrays \( l\text{Chrominance} \) and \( l\text{Distance} \). Table 1 and Table 2 show the initialization of these arrays.

<table>
<thead>
<tr>
<th>( l\text{Distance} )</th>
<th>0</th>
<th>( \text{MaxDistance} )</th>
<th>( \text{MaxDistance} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l\text{Chrominance} )</td>
<td>( \text{Chrominance}(p) )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Initialization of a source pixel \( p \)

<table>
<thead>
<tr>
<th>( l\text{Distance} )</th>
<th>( \text{MaxDistance} )</th>
<th>( \text{MaxDistance} )</th>
<th>( \text{MaxDistance} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l\text{Chrominance} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Initialization of a non-source pixel \( p \)

At the beginning, source pixels are pixels with scribbles. They are added into a set \( LSP \). For each color \( c \) in the array \( l\text{Chrominance} \) of a pixel \( p \) in \( LSP \), an intrinsic distance is computed between each pixel of its neighboring and \( c \). Let’s suppose that \( c \) is present in \( l\text{Chrominance} \)’s array of \( q \). If this distance added with the distance corresponding to the color \( c \) in \( l\text{Distance} \) of \( p \) is smaller than the intrinsic distance corresponding to \( c \) in the array \( l\text{Chrominance} \) of \( q \), this distance is updated by:

\[
l\text{Distance}[c] = l\text{Distance}[c] + d(p,q).
\]

Then, \( q \) is added into the set \( LSP \). Conversely, suppose that \( c \) is not present in \( l\text{Chrominance} \)’s array of \( q \). If the intrinsic distance is smaller than the highest distance in \( l\text{Distance} \) of \( q \), this distance replaces the highest one in the array. The chrominance in \( l\text{Chrominance} \) of \( q \) is also updated. Otherwise, no update is done. After, the pixel \( p \) is removed from \( LSP \). The process is repeated until the set \( LSP \) is empty.

The idea for colorization is to compute the \( Cb \) and \( Cr \) components of a pixel. For a pixel which is not colorized, this process uses a blending on the chrominance in \( l\text{Chrominance} \) according to their geodesic distance in \( l\text{Distance} \). In order to compute \( Cb \) and \( Cr \) components, the following expression is used:

\[
\text{chrominance}(p) = \frac{\sum_{c \in l\text{Chrominance}(p)} W(l\text{Distance}_c(p)) \times c}{\sum_{c \in l\text{Chrominance}(p)} W(l\text{Distance}_c(p))}.
\]

This blending expression assigns to any pixel a color that is a weighted average of the different colors in \( l\text{Chrominance} \). \( W \) is a function that provides the weighting factor from an geodesic distance.

### 3.2.3 Study of the complexity

The complexity of algorithm presented in this article is \( O(N \cdot |\Omega_c|) \) with \( N \) the number of pixels in the input image. The algorithm passes over the image for each chrominance observed in \( \Omega_c \) and needs a memory in the order of the number of chrominances observed in \( \Omega_c \) times grayscale image size. If there is a large number of chrominance scribbles, the algorithm could be relatively slow and pricey in memory.

However, the human perception of color blending is limited and high blending accuracy is not fully necessary to obtain satisfactory results. It is enough to blend just the two or three most
significant chrominances (the chrominance with the closest geodesic distance to their observed source) to get satisfactory results. With this fact, both time and space complexity are reduced to $O(N)$.

3.2.4 Algorithm

To complete analysis of the article [1], we provide an algorithm of possible ways to implement the method proposed. This algorithm, described in Algorithm 1, use the following structures:

link structure:

1. $ptr$: pointer to a pixel
2. $isSource$: boolean which returns true if the pixel is a source pixel, false otherwise
3. $Y$: luminance channel
4. $bl$: blend structure of the pixel

blend structure:

1. $lChrominance$: array of $MaxBlendColor$ chrominances
2. $lDistance$: array of corresponding geodesic distances to each chrominance in $lChrominance$

The function $ModifyArrayDistance(p, q)$ returns true if arrays of neighbor $q$ have been modified by the compute of geodesic distances between $p$ and $q$ as detailed in Section 3.2.2.
Algorithm 1 Fast image colorization using chrominance blending [1]

Input: Initial grayscale image with color scribbles
Output: Colored image

1: create a list of source pixels $LSP \leftarrow \emptyset$
2: for each pixel $p$ do
3:   create a link structure $\lambda_p$ for $p$ ($\lambda_p.ptr \leftarrow p$)
4:   create a blend structure $bl$ for $\lambda_p.bl$ ($\lambda_p.bl \leftarrow bl$)
5:   initialize the array of chrominances $bl.lChrominance$ with 0 values
6:   initialize the array of distances $bl.lDistance$ with $MaxDistance$ values
7: $\lambda_p.isSource \leftarrow False$
8: if $p$ is a source pixel then
9:   $\lambda_p.isSource \leftarrow True$
10: $\lambda_p.bl.lChrominance[0] \leftarrow chrominance(p)$
11: $\lambda_p.bl.lDistance[0] \leftarrow 0$
12: $LSP \leftarrow LSP \cup \lambda_p$
end if
13: end for
14: while $LSP \neq \emptyset$ do
15: $\lambda_p \leftarrow$ the first link in LSP
16: $LSP \leftarrow LSP \setminus \lambda_p$
17: for all pixels $q$ neighboring $p$ do
18:   if $ModifyArrayDistance(p, q)$ then
19:     $LSP \leftarrow LSP \cup \lambda_q$
end if
end for
20: end while
21: for each pixel $p$ in the source image do
22:   compute $chrominance(p)$ with (4)
end for
3.3 Image and video colorization based on prioritized source propagation [2]

3.3.1 Principle of the article

Given a grayscale image with color scribbles, the objective of this algorithm is to propagate the source pixels color onto their neighbors. To achieve a high quality result, color accuracy is defined for each pixel to differentiate source and non-source pixels. Then, priorities of non-source pixels are computed. At the beginning, a higher priority is assigned to a pixel if it is a neighbor of a source pixel and if they have similar luminances. The priorities are sorted in the decreasing order. The first non-source pixel in the list is colorized with a weighted sum of the chrominance of its neighbors. This is repeated until all non-source pixels are colorized.

The color space used is $YUV$, detailed in A, because the propagation of colors is based on the luminance. Moreover, it allows to preserve the image’s luminance and modify the color only by changing the values of the chrominance channels.

3.3.2 Implementation details

To define the colorization order of pixels, priority $\pi(p)$ of each non-source pixel $p$ is computed as

$$\pi(p) = \sum_{q \in N_p} a(q)e^{-|Y(p)-Y(q)|}$$

where $N_p$ is the set of 4-neighbors of $p$. A high priority $\pi(p)$ is assigned to a pixel $p$ if a neighboring pixel $q$ has a high accuracy $a(q)$ and the luminances $Y(p)$ and $Y(q)$ are similar.

The chrominance vector $\text{chrominance}(p)$ of the non-source pixel $p$ with the higher priority is updated by

$$\text{chrominance}(p) = a(p).\text{chrominance}(p) + (1-a(p)).\sum_{q \in N_p} w(p, q).\text{chrominance}(q)$$

where the weight $w(p, q)$ is defined as

$$w(p, q) = \frac{a(q)e^{-|Y(p)-Y(q)|}}{\sum_{r \in N_p} a(r)e^{-|Y(p)-Y(r)|}}.$$ 

After coloring pixel $p$, its accuracy $a(p)$ is updated to 1. Besides, the priority of the neighboring pixels are brought up to date and all non-source pixels are sorted by priority. The pixel with higher priority is colorized and this is repeated until the image is colorized.

3.3.3 Study of the complexity

The algorithm presented in this article has complexity of $O(N^2)$ with $N$ the number of pixels in the grayscale image. Indeed, all non-source pixels are processed once and the number of source pixels is negligible. Then, for each non-source pixel, all priorities are sorted in the decreasing order.
3.3.4 Algorithm

To complete the analysis of the method described in the article [2], we proposed a possible algorithm which implements this method. This algorithm, described in Algorithm 2, uses the following structure:

\[ \text{link structure:} \]

1) \( \text{ptr} \): pointer to a pixel
2) \( a \): color accuracy \( \in \{0, 1\} \)
3) \( Y \): luminance channel
4) \( C \): chrominance vector
5) \( \pi \): priority

4 Architecture

4.1 Libraries

In order to implement algorithms, Qt and OpenCV libraries are used.

- **OpenCV** [14] is a library of programming functions mainly aimed at real time computer vision, developed initially by Intel and now supported by Willow Garage. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. We use it with C++ programming language.

- **Qt** [15] is a cross-platform application framework developed in C++ and produced by Nokia’s Qt Development Frameworks division. Qt is widely used for developing software with a graphical user interface (GUI) and also used for developing non-GUI programs such as command-line tools and consoles for servers.

4.2 Implementation

Two methods: the common features

In order to test the two algorithms described in [1] and [2], a software has been implemented with Qt. This simple interface is composed of a menu and a widget we created, \textit{QMat}, which allows to display the image and draw on it.

The class \textit{QMat} contains the following attributes:

- \texttt{mat} : an openCV Mat object which represents the current image that is the original grayscale image with color scribbles

- \texttt{matMarkers} : an openCV Mat object which is an image with the scribbles drawn by the user or a loaded mask

- \texttt{originalMat} : an openCV Mat object which corresponds to the original grayscale image

- \texttt{label} : a Qt QLabel object used to display the image in the interface
Algorithm 2 Image colorization based on prioritized source propagation [2]

Input: Initial grayscale image with color scribbles
Output: Colored image

1: create a list of link: \( L \leftarrow \emptyset \)
2: for each pixel \( p \) in the image do
3: create a link structure \( \lambda : \lambda.\text{ptr} \leftarrow p \)
4: \( \lambda.a \leftarrow 1 \) if the chrominance vector of \( p \) is defined, 0 otherwise
5: \( \lambda.Y \leftarrow \) grayscale of the pixel \( p \)
6: \( \lambda.C \leftarrow C(p) \)
7: \( L \leftarrow L \cup \lambda \)
8: end for
9: for each link \( \lambda \) in \( L \) do
10: \( \lambda.\pi \leftarrow 0 \)
11: if \( \lambda.a = 0 \) then
12: for all link \( \lambda_q \) in \( N_{\lambda.\text{ptr}} \) do
13: \( \lambda.\pi \leftarrow \lambda.\pi + \lambda_q.a \times e^{-|\lambda.Y - \lambda_q.Y|} \)
14: end for
15: end if
16: end for
17: create a queue \( f \) of non-source pixels sorted by priority
18: while \( f \neq \emptyset \) do
19: \( \lambda \leftarrow \) first link in \( f \)
20: \( \text{sum} \leftarrow 0 \)
21: for all link \( \lambda_q \) in \( N_{\lambda.\text{ptr}} \) do
22: \( w \leftarrow 0 \)
23: for all link \( \lambda_r \) in \( N_{\lambda.\text{ptr}} \) do
24: \( w \leftarrow w + \lambda_r.a \times e^{-|\lambda.Y - \lambda_r.Y|} \)
25: end for
26: \( \text{sum} \leftarrow \text{sum} + \frac{\lambda_q.a 	imes e^{-|\lambda.Y - \lambda_q.Y|}}{w} \times \lambda_q.C \)
27: end for
28: \( \lambda.C \leftarrow \lambda.a \times \lambda.C + (1 - \lambda.a) \times \text{sum} \)
29: \( \lambda.a \leftarrow 1 \)
30: \( f \leftarrow f \setminus \lambda \)
31: update priority of neighboring pixels and sort \( f \) by priority
32: end while
- `currentColor` and `penSize`: respectively the current color and size of the brush for drawing scribbles.

and the following methods:

- **getter and setter methods** for `_mat`, `_matMarkers`, `_originalMat`, `currentColor` and `penSize`
- `mousePressEvent()`, `mouseReleaseEvent()`, `mouseMoveEvent()`: functions to manage mouse events
- `reset()`: clear markers and reset the original image
- `setLabel()`: draw the current image in the interface
- `LaunchAlgoYatziv()`: function to launch the algorithm in [1]
- `LaunchAlgoHeu()`: function to launch the algorithm in [2]

5 Experimental results

We now present results of the two different techniques. Firstly, in order to propagate colors on a grayscale image, it is necessary to position scribbles on this image. A scribble represents a set of connected color pixels. They are considered as source pixels. Scribbles can be obtained in two ways. Either the user marks chrominance scribbles directly on the monochromatic image or he can load a mask (monochromatic image, with the same size of the grayscale image, containing scribbles colors). In this case, the initialization step of algorithms recovers colored pixels (all pixels with different values of R,G,B channels) and sets them on the grayscale image at the same position. It is also possible to paint other scribbles after loading a mask. Figure 4 illustrates an example of mask application on a grayscale image. Thus, the parameter of these algorithms is the grayscale image with colored scribbles.

Figure 4: Mask application on a grayscale image

5.1 Performance tests

According to the article [1], the CPU run-time of the process based on chrominance blending is fast and achieves a higher quality result. Whereas in the article [2], the colorization process using prioritized source propagation yields more reliable performance than the previous one. However, the run-time is never mentioned. For timing, algorithms have been tested on various images with an Intel Core Quad Core 2.66 GHz with 4-Go RAM running under Ubuntu 11.04. This part
presents performance tests obtained with the two algorithms. All tests have been experimented with the left image in Figure 5. The parameter $b$ used for launching the Yatziv-Sapiro’s algorithm has been set at 4 in Section 3.1. All the tests have been run with a 4-neighborhood. The 8-neighborhood gives very slightly better results but the increase of run-time is too high. The right image corresponds to a final result after a colorization process.

![Figure 5: Image used for the tests](image)

Performance criteria used are the image size, the percentage of source pixels in the image and the neighbor connectivity. Measurements were made once the images were loaded into memory. With our implementation of the method in the article [1], the algorithm run-time for our examples is fast, as shown in Table 3 and Figure 6. In fact, we observe that even if the image size increases, the run-time remains low (under than 0.3 seconds). On the opposite, the algorithm presented in the article [2] is slower than the previous one. For a small image (100x83 pixels), the run-time exceeds 1.5 seconds and rises very quickly when the size increases.

<table>
<thead>
<tr>
<th>Size of image</th>
<th>100x83</th>
<th>160x132</th>
<th>190x157</th>
<th>230x190</th>
<th>260x215</th>
<th>290x240</th>
<th>320x265</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yatziv &amp; Sapiro</td>
<td>0.019</td>
<td>0.049</td>
<td>0.075</td>
<td>0.115</td>
<td>0.158</td>
<td>0.180</td>
<td>0.216</td>
</tr>
<tr>
<td>Heu et al.</td>
<td>1.516</td>
<td>10.569</td>
<td>21.064</td>
<td>48.296</td>
<td>89.430</td>
<td>145.575</td>
<td>237.281</td>
</tr>
</tbody>
</table>

Table 3: Run-time with variation of the size of image (unity: s)

On the same way, Table 4 and Figure 7 show the run-time when the number of source pixels increases. For the first algorithm, even if the percentage of source pixels taken into account rises, the run-time varies very slowly. On the other hand, results of the second algorithm show that it depends more on the number of source pixels (run-time is divided by two when the number of source pixels increases by 3% to 29%).

<table>
<thead>
<tr>
<th>Percentage of source pixels</th>
<th>3%</th>
<th>5%</th>
<th>7%</th>
<th>14%</th>
<th>20%</th>
<th>29%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yatziv &amp; Sapiro</td>
<td>0.230</td>
<td>0.199</td>
<td>0.187</td>
<td>0.159</td>
<td>0.135</td>
<td>0.116</td>
</tr>
<tr>
<td>Heu et al.</td>
<td>121.491</td>
<td>121.131</td>
<td>109.980</td>
<td>98.939</td>
<td>89.430</td>
<td>65.591</td>
</tr>
</tbody>
</table>

Table 4: Run-time with variation of the percentage of source pixels (unity: s)

Previous results show a significant run-time difference between the two colorization processes. It is due to the complexity of each algorithm. The Yatziv and Sapiro algorithm makes a computation of the three closest intrinsic distances for each pixel. This process is not computationally costly because just the three most significant chrominances are considered for the blending. So, the blending colorization algorithm has an average time and space complexity of $O(M)$ where $M$
is the number of pixels. While the Heu et al. algorithm computes the chrominance vector for each non-source pixels and each time sorts the list of non-source pixels by priority. The complexity of the sorting of a list is $O(N \log(N))$ with $N$ the size of the list. Thereby, the complexity of the Heu et al. algorithm is $O(M^2)$.

### 5.2 Quality of algorithms

In order to achieve good quality results, the position and the luminance of source pixels are important. In fact, if scribbles are incorrectly positioned, some of image regions will not be colorized or the final image will not be natural. Likewise, if the luminance of source pixels is too different from the luminance of the pixels in the grayscale image at the same position, algorithms can cause erroneous color propagation and color blurring in the results. We observed this with the Yatziv and Sapiro algorithm particularly. Figure 8 shows examples of an erroneous color propagation. Thus we have to choose the source pixels accurately. Also, their luminance value must be close to that of the grayscale image at the same location. If these criteria are respected, a good quality image is obtained.

The two algorithms give satisfactory results but the second is more reliable than the first. As shown in Figure 9, in some regions, the color propagation is more accurate with the second algorithm. On the cat’s image the Yatziv and Sapiro algorithm (left image) creates some artifacts around the head of the cat because of the color blending effect. Conversely, the Heu et al. algorithm (right image) has well-defined boundaries. Similarly, the lamp’s image presents the same kind of artifacts. They can be explained by the fact that in these regions, the luminance values of both sides of the boundary are close. Figure 10 illustrates this case: the pixel $s$ is in the region A and the pixel $s'$ is in the region B. The pixel $t$ is in the region $B$ and it should have the color of $s'$. But the geodesic distance $d(s, t)$ between $s$ and $t$ is smaller than that between $s'$ and $t$. Thus, it will have the color of $s$, which will cause erroneous color propagation. On the other hand, the Heu et al. algorithm provides a more natural reconstruction with crisp color boundaries. This algorithm repeatedly identifies the pixel, which can be colorized most reliably, paints it, and the pixel becomes a source pixel.

### 6 Conclusion

During this project, we studied two articles about semi-automatic colorization of images [1, 2] and we implemented algorithms defined in these articles. Thereafter, we compared experimental results for the two methods in terms of quality and run-time. During these tests, we saw the limitations of these algorithms. The number, the position and the color of scribbles play an important role in the result. Indeed, the two algorithms use the luminance to propagate colors that is why a scribble should have a similar luminance to the corresponding region in the grayscale image.

Besides, our tests confirm that the Yatziv ans Sapiro algorithm is fast (within a second) and provides a good quality result but it can create some visual artifacts on boundaries. Contrary, the Heu et al. algorithm gives a better quality result with well-defined boundaries on the colored image. However, it is very slow (several minutes) and the run-time increases quickly with the image size.

As shown previously, choosing scribbles which fit the initial image's luminance is an effective constraint for the user. Therefore, it would be interesting to ensure that the user can choose any color regardless of the initial image's luminance. An other extension of this project could
be the study of the video colorization proposed in the articles analysed and the implementation of the corresponding algorithms. Moreover, we could adapt our algorithms in order to recolorize several parts in a color image. The user could put a scribble into a region to recolorize, and the algorithm could propagate the new color into this area.

This project provided us with an insight in the world of research and allowed us to study several research papers in depth. We had to understand the method defined in each article and to be able to implement an algorithm from this process.

Otherwise, meetings with persons in charge and with the teaching assistant gave us many tips for both the understanding of the articles, the implementation of the algorithms and the writing of this paper. Besides, this project allowed us to be better organized and more rigorous with respect to deadlines.

To conclude, the Project of Study and Research was beneficial to us in many points. Indeed, it allowed us to achieve a complete project from the analysis of the articles to the comparison of experimental results, through the implementation of algorithms and the development of software.

7 Acknowledgements

We acknowledge firstly our persons in charge Aurélie Bugeau and Vinh-Tong Ta which brought us many advices and took a lot of time with us throughout the project. Thanks to Pascal Desbarats which followed us for writing this research paper and for preparing the oral presentation particularly. We acknowledge also Jean-Jacques Bernaulte for the provided corrections to our research paper.
Figure 6: Run-time of Yatziv-Sapiro’s algorithm (top) and Heu et al. algorithm (bottom) with variation of size.
Figure 7: Run-time of Yatziv-Sapiro algorithm (top) and Heu et al. algorithm (bottom) with variation of number of source pixels.
Figure 8: Examples of erroneous color propagation due to wrong luminances and positions of source pixels.

Figure 9: Differences between Yatziv-Sapiro (left) and Heu et al (right) algorithms.
Figure 10: Explanation of an example erroneous color propagation
References


A Annexe : Color spaces

Color propagation is realized if two pixels present the same luminous intensity. Generally images are encoded in RGB space. However, this color space does not quantify the luminous intensity of a pixel. Therefore, an RGB to YCbCr or YUV conversion is needed before launching algorithms. Likewise, for rendering images on screen, reverse conversion is made.

• RGB space is an additive color model in which red, green, and blue light is added together in various ways to reproduce a broad array of colors. This color model is based on the human perception of colors. It is used for sensing, representation, and display of images in electronic systems, such as televisions and computers.

The only problem is that we cannot perform operations in this space because the eye is more sensitive to certain components, especially the green. The human vision has finer spatial sensitivity to luminance (grayscale) differences than chromatic differences. So we have to do a conversion into space YUV or YCbCr.

• YUV space is a color model like RGB but the difference is that YUV takes human perception into account. YUV color model is used in the PAL and SECAM composite color video standards. It is defined in terms of one luminance component Y and two chrominance components UV. The luminance is a component which represents the intensity of a color whereas the chrominance defines the color.

The representation of an image in YUV space is obtained from an RGB image by applying to each of the components Y, U and V a weighted sum of each R, G and B. In fact, to go from one representation in RGB to YUV space, it is necessary to apply to each pixel of the image the following transition matrix:

\[
\begin{pmatrix}
    Y \\
    U \\
    V
\end{pmatrix} =
\begin{pmatrix}
    0.299 & 0.587 & 0.114 \\
    -0.1471 & -0.2886 & 0.436 \\
    0.615 & -0.5149 & -0.1001
\end{pmatrix}
\begin{pmatrix}
    R \\
    G \\
    B
\end{pmatrix}
\]

This operation is totally reversible. It is possible to move from one representation of colors in the YUV space to a representation in RGB space. The transition matrix that allows this transformation is:

\[
\begin{pmatrix}
    R \\
    G \\
    B
\end{pmatrix} =
\begin{pmatrix}
    1 & 0 & 1.1398 \\
    1 & -0.3946 & -0.5806 \\
    1 & 2.0321 & 0
\end{pmatrix}
\begin{pmatrix}
    Y \\
    U \\
    V
\end{pmatrix}
\]

• YCbCr space is a color model similar to YUV. Y is the luminance component and Cb and Cr are the chrominance components. The luminance matches the grayscale information. Cb and Cr are the blue-difference and red-difference chroma components of the image. This color space is used as part of a color image pipeline. Like the YUV space, there is a transition formula which permits to convert an RGB space into YCbCr space:

\[
\begin{align*}
Y &= 0.299 \times R + 0.587 \times G + 0.114 \times B \\
Cb &= -0.1687 \times R - 0.3313 \times G + 0.5 \times B + 128 \\
Cr &= 0.5 \times R - 0.4187 \times G - 0.0813 \times B + 128
\end{align*}
\]
This operation is also reversible. It is possible to move from one representation of colors in the $YCbCr$ space to a representation in $RGB$ space. The transition formula that allows this transformation is:

\[
\begin{align*}
R &= Y + 1.402 \times (Cr - 128) \\
G &= Y - 0.34414 \times (Cb - 128) - 0.71414 \times (Cr - 128) \\
B &= Y + 1.772 \times (Cb - 128)
\end{align*}
\]
Fast Image and Video Colorization
Using Chrominance Blending
Liron Yatziv and Guillermo Sapiro

Abstract—Colorization, the task of coloring a grayscale image or video, involves assigning from the single dimension of intensity or luminance a quantity that varies in three dimensions, such as red, green, and blue channels. Mapping between intensity and color is, therefore, not unique, and colorization is ambiguous in nature and requires some amount of human interaction or external information. A computationally simple, yet effective, approach of colorization is presented in this paper. The method is fast and it can be conveniently used “on the fly,” permitting the user to interactively get the desired results promptly after providing a reduced set of chrominance scribbles. Based on the concepts of luminance-weighted chrominance blending and fast intrinsic distance computations, high-quality colorization results for still images and video are obtained at a fraction of the complexity and computational cost of previously reported techniques. Possible extensions of the algorithm introduced here included the capability of changing the colors of an existing color image or video, as well as changing the underlying luminance, and many other special effects demonstrated here.

Index Terms—Chrominance blending, colorization, gradient, interpolation, intrinsic distance, recolorization, special effects.

I. INTRODUCTION

COLORIZATION is the the art of adding color to a monochrome image or movie. The idea of “coloring” photos and films is not new. Ironically, hand coloring of photographs is as old as photography itself. There exists such examples from 1842 and possibly earlier [19]. It was practiced in motion pictures in the early 1900s by the French Company Pathe, where many films were colored by hand. It was also widely practiced for filmsstrips into the 1930s. The computer-assisted process was first introduced by Markle in 1970 for adding colors to black and white movies [3].

As neatly presented by Sykora et al. [26] (their work also includes an outstanding overview of the literature on the subject), various early computer-based colorization techniques include straightforward approaches such as luminance keying [9]. This method is based on a user-defined lookup table which transforms grayscale into color. Welsh et al. [28], inspired by work of Reinhard et al. [21] and Hertzmann et al. [11], extended this idea by matching luminance and texture rather than just the grayscale values.

Chen et al. [5] used manual segmentation to divide the grayscale image into a set of layers. Then an alpha channel was estimated using Bayesian image matting. This decomposition allows to apply colorization using Welsh’s approach. The final image is constructed using alpha-blending. Recently, Sykora et al. [26] have similarly used a segmentation method optimized for the colorization of black and white cartoons.

Other approaches, including our own, assume that homogeneity of the grayscale image indicates homogeneity in the color. In other words, as detailed in [23], the geometry of the image is provided by the geometry of the grayscale information (see also [4], [6], and [15]). Often, in these methods, in addition to the grayscale data, color hints are provided by the user via scribbles. Horiuchi [12] used a probabilistic relaxation method while Levin et al. [17] solved an optimization problem that minimizes a quadratic cost function derived from the color differences between a pixel and its weighted average neighborhood colors. Sapiro [23] proposed to inpaint the colors constrained by the grayscale gradients and the color scribbles that serve as boundary conditions. The method reduces to solving linear or nonlinear Poisson equations.

The main shortcoming of these previous approaches is their intensive computational cost, needed to obtain good-quality results. Horiuchi and Hirano addressed this issue in [13], where they presented a faster algorithm that propagates colored seed pixels in all directions and the coloring is done by choosing from a preselected list of color candidates. However, the method produces visible artifacts of block distortion since no color blending is performed. While Horiuchi’s method colorizes a still image within a few seconds, we present in this paper a propagation method that colorizes a still image within a second or less, achieving even higher quality results. In contrast with works such as those in [17], the technique proposed here is easily extended to video without the optical flow computation, further improving in the computational cost, at no sacrifice in the image quality. Other special effects are also easily obtained using our framework, as demonstrated here and further exemplified at http://mountains.ece.umn.edu/~liron/colorization/.

The scheme proposed here is based on the concept of color blending. This blending is derived from a weighted distance function efficiently computed from the luminance channel. The underlying approach can be generalized to produce other effects such as recolorization. In the remainder of this paper, we describe the algorithm and present a number of examples.

II. FAST COLORIZATION FRAMEWORK

Similar to other colorization methods, e.g., [17] and [23], we use luminance/chrominance color systems. We present our
method in the YCbCr color space, although other color spaces such as YIQ or YUV could be used as well. Moreover, work can also be done directly on the RGB space. Let \( Y(x,y) \) be the given monochromatic image \((T = 0)\) or video \((T > 0)\) defined on a region \( \Omega \). Our goal is to complete the \( C_b \) and \( C_r \) channels \( C_b(x,y,\tau) : \Omega \times [0,T] \rightarrow \mathbb{R}^+ \) and \( C_r(x,y,\tau) : \Omega \times [0,T] \rightarrow \mathbb{R}^+ \), respectively. For clarity of the exposition, we refer to both channels as the chrominance.

The proposed technique also uses as input observed values of the chrominance channels in a region \( \Omega_c \subseteq \Omega \) which is significantly smaller than \( \Omega \) (see [17]). These values are often provided by the user or borrowed from other data.

Let \( s \) and \( t \) be two points in \( \Omega \) and let \( C(p) : [0,1] \rightarrow \Omega \) be a curve in \( \Omega \). Also, let \( C_{sf} \) represent a curve connecting \( s \) and \( t \) such that \( C(0) = s \) and \( C(1) = t \). We define the intrinsic (geodesic) distance between \( s \) and \( t \) by

\[
d(s,t) := \min_{C_{sf}} \int_0^1 \| \nabla Y \cdot \tilde{C}(p) \| dp,
\]

This intrinsic distance gives a measurement of how “flat” is the flattest curve between any two points in the luminance channel. The integral in the equation above is basically integrating the luminance \( Y \) gradient in the direction of the curve \( C(p) \) (as given by projecting the gradient \( \nabla Y \) into the tangent \( \tilde{C} \) to this curve).

When considering the minimum over all paths \( C_{sf} \), we then keep the one with the smallest overall gradient in this direction, thereby the flattest path connecting the two points \( s \) and \( t \) (the path that goes from \( s \) to \( t \) with minimal overall gradient). Note that the minimal path need not be unique, but we only care about the intrinsic length \( d(s,t) \) of this path, so this does not affect the algorithm.

Geodesic distances of this type can be efficiently and accurately computed using recently developed fast numerical techniques [10], [14], [24], [25], [27], [29], [31]. We found that for the application at hand, even simpler techniques such as a best first one (in particular, Dijkstra [8]), integrated in the pseudocode below, are sufficient. This will be further detailed in the Section III.

Even though a mapping between luminance and chrominance is not unique, a close relationship between the basic geometry of these channels is frequently observed in natural images (see, for example, [4], [6], and [15]). Sharp luminance changes are likely to indicate an edge in the chrominance, and a gradual change in luminance often indicates that the chrominance is also not likely to have an edge but rather a moderate change. In other words, as reported in the aforementioned works, often, there is a close relationship between the geometries of the luminance and chrominance channels. Exploring this assumption, a change in luminance causes a related change in chrominance. This has been used in different fashions in [17] and [23], as well as in [2] for super resolution. From this, for the proposed colorization approach we assume that the smaller the intrinsic distance \( d(s,t) \) between two points \( (s,t) \), the more similar chrominance they would have.\(^1\)

Since the chrominance data is often given in whole regions and not necessarily in single isolated points, we would like to get an idea of the distance from a certain known chrominance (“scribbles” with a given uniform color) to any point \( t \) in \( \Omega \) we then define the intrinsic distance \( d_c(t) \) from a point \( t \) (to be colored) to a certain chrominance \( c \), as the minimum distance \( d(s,t) \) from \( t \) to any point \( s \) of the same chrominance \( c \) in the set of given colors \( \Omega_c \).

\[
d_c(t) := \min_{\forall s \in \Omega_c, : \text{chrominance}(s) = c} d(s,t).
\]

This gives the distance from a point \( t \) to be colored to scribbles (from the provided set \( \Omega_c \)) with the same color \( c \), or, in other words, the distance from a point to a provided color.

Our idea for colorization is to compute the \( C_b \) and \( C_r \) components (chrominance) of a point \( t \) in the region where they are missing (\( \Omega \setminus \Omega_c \)) by blending the different chrominance in \( \Omega_c \), according to their intrinsic distance to \( t \)

\[
\text{chrominance}(t) = \frac{\sum_{\forall c \in \text{chrominase}(\Omega_c)} \text{W}(d_c(t)) c}{\sum_{\forall c \in \text{chrominase}(\Omega_c)} \text{W}(d_c(t))}
\]

where \( \text{chrominase}(\Omega_c) \) stands for all the different unique chrominance in the set \( \Omega_c \), and \( \text{W}(\cdot) \) is a function of the intrinsic distance that translates it into a blending weight. In other words, the above blending expression assigns to any point \( t \) to be colored, a color that is a weighted average of the different colors in the provided set of scribbles \( \Omega_c \). For every distinct color \( c \) in the set \( \Omega_c \), the distance to it from \( t \) is computed following (2) [which uses (1)], and this distance is used to define the weight of the color \( c \) at the point \( t \) (the blending proportion of this color). The denominator in the above equation is just a normalization factor.

The function \( \text{W}(\cdot) \) that provides the weighting factor from the distance \( d_c(t) \) should hold some basic properties:

1. \( \lim_{r \rightarrow 0} \text{W}(r) = \infty \);
2. \( \lim_{r \rightarrow \infty} \text{W}(r) = 0 \);
3. \( \lim_{r \rightarrow \infty} \text{W}(r + r_0) / \text{W}(r) = 1, r_0 \) a constant.

The first two requirements are obvious. Requirement 3) is necessary when there are two or more chrominance sources close by, but the blending is done relatively far from all sources. The desired visual result would be the even blending of all chrominance. For the experiments reported below, we used

\[
\text{W}(r) = r^{-b}
\]

where \( b \) is the blending factor, typically \( 1 \leq b \leq 6 \). This factor defines how smooth is the chrominance transition.

Note that, in theory, following the equations above, a point \( t \) to be colored will be influenced by all distinct colors \( c \) in \( \Omega_c \) since \( d_c(t) < \infty \).

A. Implementation Details

A key component of our algorithm is the computation of the intrinsic, gradient weighted, distance provided by (1). We should then first concentrate on how to compute this distance. Geodesic distances of this type have been widely studied in the scientific computing community, in general, as well as in the control and image-processing communities, in particular. For
example, expressions of this type have been used for edge detection in [16]. They constitute a modification of the popular geodesic active contours for object detection (see [22] and references therein for details). The popularity of these expressions in other disciplines means that their efficient computation has been widely studied, e.g., [10], [14], [24], [25], [27], [29], and [31], where the reader is referred for more details, including exact numerical computations of the gradient in (1) using up-wind schemes. These algorithms are inspired by the classical Dijkstra graph algorithm [8],2 fast sweeping techniques [7], and special data structures such as buckets and priority queues [29]. A straightforward and very fast and accurate implementation of our algorithm consists in computing for every point \( t \) to be colored in \( \Omega \setminus \Omega_c \), the distance \( d_c(t) \) to every distinct chrominance \( c \) in the given \( \Omega_c \), using these fast techniques, thereby obtaining the set of weights needed for the blending (3). We found that there is no real need to implement the computation of the weighted distance using the aforementioned accurate techniques, and simply using classical Dijkstra is sufficient. Thereby, in principle, we create a graph where each image pixel is a vertex connected to its neighbors with edges that, following (1), have weights equal to the absolute gray value derivatives between the neighbors, and work with Dijkstra’s algorithm on this graph. Following the results in [29] (see also the references therein), we could use more accurate distance computations at minimal additional cost, but we found this not to be necessary to obtain the high quality results reported here.

To complete the description of the proposed colorization technique, we provide a pseudocode that exemplifies one of the possible ways to efficiently implement the proposed algorithm [(1)–(3)] and further emphasizes the simplicity of the proposed technique. This is just a standard form of implementing the classical Dijkstra-based ideas mentioned above.

\* Input:
- Grayscale video/image.
- List of observed pixels \( \Omega_c \) and their chrominance (user provided scribbles).
\* Output:
- Colored video/image.
\* Definitions:
- For each pixel in \( \Omega \) a blend structure is created that contains:
  1. \( \chi \)—list of chrominances (the chrominances that reached this blend)
  2. \( \omega \)—list of distances (the corresponding intrinsic distance to each chrominance in \( \chi \))

  - A link structure contains:
    1. \( \text{ptr} \)—pointer to a pixel (the destination pixel of the link)
    2. \( \ell \)—a chrominance (the chrominance the link attempts to transfer)
    3. \( d \)—distance (the intrinsic distance from the chrominance origin in \( \Omega_c \))

\* The basic algorithm:
1) create an empty blend structure for each pixel in \( \Omega \)
2) \( L \leftarrow \emptyset \)
3) for each pixel \( p \) in \( \Omega_c \)
   a) let \( \ell_p \) be the chrominance of the scribble in \( p \)
   b) create link \( \lambda \)
   c) \((\lambda, \text{ptr}, \ell_p, \ell, d) \leftarrow (p, \text{ptr}, 0)\)

4) while \( L \neq \emptyset \)
   a) \( \lambda \leftarrow \) the link with smallest distance in \( L \) (see Footnote (4)).
   b) \( L \leftarrow L \setminus \lambda \)
   c) \( b \leftarrow \) the blend structure of \( \lambda \_	ext{ptr} \)
   d) if \( \lambda \_\ell \not\in b_\chi \)
      i) add \( \lambda \_\ell \) to \( b_\chi \)
      ii) add \( \lambda d \) to \( b_\omega \)
   e) for all pixels \( q \) neighboring \( p \)
      a) create link \( \lambda \)
      b) \((\lambda_{\_\text{ptr}}, \lambda_{\_\ell}, \lambda_{\_d}) \leftarrow (q, \text{ptr}, d + \text{distance}(p, q))^3\)
      c) \( L \leftarrow L \cup \lambda \)
5) for all pixels in \( \Omega \) generate the chrominance channels by applying (3) with the pixels’ blend structures.

\* B. Performance and Relaxation

The described colorization algorithm has an average time and space complexity of \( O(|\Omega| \cdot \text{chrominances}(\Omega_c)). \) The algorithm passes over the image/video for each different chrominance observed in \( \Omega_c \) and needs a memory in the order of the number of different chrominances observed in \( \Omega_c \), times the input image/video size. If there are a large number of scribbles of different chrominances, the algorithm could be relatively slow and pricey in memory (although still more efficient that those previously reported in the literature).

Fortunately, since human perception of color blending is limited, high blending accuracy is not fully necessary to obtain satisfactory results. Experimental results show that it is enough just to blend the most significant chrominance (the chrominance with the closest intrinsic distance to their observed source). We found that, in natural images, it is enough to blend just the two or three most significant chrominance to satisfy satisfactory results. Such a relaxation reduces both the time and space complexity to \( O(|\Omega|) \); thereby linear in the amount of data. Therefore, we do not include in the blend chrominances that their weight in the blending equation is small relatively to the total weight. Additional quality improvements could be achieved if an adaptive threshold, following results such as those from the MacAdam ellipses [18], is used. Any color lying just outside an ellipse is at the “just noticeable difference” (jnd) level, which is the smallest perceivable color difference under the experiment conditions. A possible use of this is to define an adaptive threshold that would filter out chrominance that if added to the blend would not cause a jnd.

This proposed algorithm relaxation of limiting the number of contributors to the blending equation gives a tight restriction on how far the chrominance will propagate to be included in the blend. The restriction can be easily implemented adding conditions to d) in 4) in the pseudocode.

\* The function distance \( (p, q) \) is the distance between \( p \) and \( q \). Following (1) and the Dijkstra algorithm, this is obtained by a numerical approximation of the absolute value of the corresponding luminance derivative; e.g., just absolute luminance difference for a simple approximation.

Using a priority queue with \( O(1) \) average complexity per operation, as done in [1], [27], and [29]; a heap sort data structure as used in the original Dijkstra algorithm would slightly increase the run-time complexity.

3 The Dijkstra algorithm is not consistent and has a bias when measuring distances on a grid. The aforementioned techniques are correct for this bias.
Fig. 1. Still-image colorization examples. Given a grayscale image, (left) the user marks chrominance scribbles, and (right) our algorithm provides a colorized image. The image size/run time from top to bottom are 230×345/less than 0.42 s, 256×256/less than 0.36 s, and 600×450/less than 1.73 s.

III. COLORIZATION RESULTS

We now present examples of our image and video colorization technique. Additional examples, comparisons, and movies, as well as software for testing our approach, can be found at http://mountains.ece.umn.edu/~liron/colorization/. This site also contains examples beyond colorization, such as video recoloring, depth effects, decolorizing, and depth-of-field effects.

The proposed algorithm has been implemented in C++ as a stand-alone win32 application so it could be tested for speed and quality. For timing we used an Intel Pentium III with 512-MB RAM running under Windows 2000. Fig. 1 shows examples of still image colorization using our proposed algorithm. The algorithm run time for all the examples in Fig. 1, measured once the images were loaded into memory, is less than 7 μs per pixel.

Figs. 2 and 3 compare our method with the one recently proposed by Levin et al. [17] (this work partially inspired our own). The method minimizes the difference between a pixel’s color and the weighted average color of its neighboring pixels. The weights are provided by the luminance channel. The minimization is an optimization problem, subject to constraints supplied by the user as chrominance scribbles. Solving this is computationally costly and slower than our proposed technique. First, in

Fig. 2, we observe that we achieve the same visual quality at a fraction of the computational cost (more comparisons are provided in the aforementioned web site, all supporting the same finding). Overall, the method proposed in [17] performs very well on many images, yet Fig. 3 demonstrates that it can perform poorly when colorizing relatively far from the provided color constraints. Following the equations in the previous section [see (3)], even far away pixels will receive color from the scribbles with our approach (in particular from the closest ones, in weighted distance, see section on relaxation above). In order to match visual quality with our technique, the method proposed in [17] needs more user input, meaning additional color scribbles. We also found that the inspiring technique developed in [17] has a sensible scale parameter and often fails at strong edges,
In Fig. 4, we study the robustness of our proposed approach with respect to the scribbles placement (location of the set $\Omega_k$). Before describing these examples, let us provide some basic comments that will help to understand the observed robustness of this technique. Assume that we know the “ideal” position to place a scribble (see Section IV for more on this). What happens if, instead of placing this scribble at the optimal place, we place it at a different location? If the ideal scribble and the placed one are both inside the same object, and the region between them is relatively homogenous, then using our gradient weighted metric, the distance between the two scribbles will be relatively small. From the triangle inequality, we can then bound the distance from the placed scribble to a given pixel to be colored, simply using the sum of the distance from the ideal scribble to such pixel and the distance between the scribbles. Since the latter is small, then the distance from the ideal...
scribble and the one from the placed one to the pixel to be colored are very similar, and, as such, the result of the colorization algorithm. If the placed scribble is located “on the wrong side of the edge” (or with high gradients between it and the ideal scribble), then, of course, the distance between the ideal and the placed scribbles will be large and as such the algorithm will produce very different results. Of course, this will almost mean an intentional mistake by the user. Moreover, interactive colorization is possible thanks to the fast speed of the proposed technique, thereby permitting the user to correct errors and to add or move scribbles as needed (this can be easily experimented with the public domain software mentioned above). The first row of Fig. 4 shows an example of an image colored with different sets of scribbles. Notice how the results are visually indistinguishable. This is repeated for a second example in the next row. The last row of the figure shows the evolution of the result as the user adds scribbles, a common working scenario for this type of application, which is allowed by the speed and robustness of our proposed technique (this is also part of the public domain implementation mentioned above).

Fig. 5 shows a simple example of special effects that can be easily obtained with our proposed framework, colorization with palettes. After the scribble is propagated, the gray value is used to key the palette. The original image is followed by the colored one on the first row. The second row shows the palettes for the mountain and the sky, respectively.

Figs. 6 and 7 show how our technique can be applied to video. Given a grayscale video and some chrominance scribbles anywhere in the video, our algorithm colors the whole video within seconds. This is a significant computational complexity reduction compared to [17] and [26], where not only is each frame computed significantly faster, but there is also no need for optical flow. Fig. 6 demonstrates the colorization of an animated scene from the movie Finding Nemo. Fig. 7 shows a colorization example of an old film. In both examples, we obtained very good results just by marking a few chrominance scribbles in a single frame.

The blending factor is the only free parameter of the algorithm as currently implemented. We set it to be $b = 4$ for all examples in this paper and the additional ones in the aforementioned web page. Better results may have been archived selecting a different value per image and video, although we did not find this necessary to obtain the high quality results presented here.

A. Recolorization and Extensions

Recolorization is the art of replacing the colors of an image by new colors. Fig. 8 shows that it is possible to change colors of an existing image or video just by slightly modifying our original colorization algorithm. When colorizing grayscale images, we based the process on the assumption that homogeneity of the grayscale indicates homogeneity in the chrominance. In recolorization, we have more clues about how the chrominance should vary. We assume that homogeneity in the original chrominance indicates homogeneity in the new chrominance. The chrominance propagation can be based on the grayscale assumption or the original color assumption or both, as demonstrated in Fig. 9. The intrinsic distance can also be measured on the $Cb$ and $Cr$ channels rather than just on the intensity channel as done in (1). This is done simply by replacing $Y$ by $Cb$ or...
Fig. 7. Video colorization example. (a) Given a 21 frame grayscale sequence, (b) our algorithm provides a colorized video. (c) The first and last frame of the colorized video are enlarged to show the color content. (d) To colorize the 21 frames, all that was needed are a few chrominance scribbles on a single frame. The user marked chrominance scribbles on the eleventh frame of this sequence. The size of each frame is $320 \times 240$, and the algorithm total run time for the whole sequence is 14 s. The movie can be found in the web site for this project.

In this equation, thereby using the gradients of these (now available) chroma channels to control the blending weight of the new color. Video recolorization examples can be seen in the aforementioned web site.

Recolorization is just one possible extensions of our method. It is also possible to further generalize it by defining the measurement medium $M(x, y, \tau) : \Omega \times [0, T) \rightarrow \mathbb{R}$ on which the intrinsic distance is measured. $M$ can be any channel of the input image, a mix of the channels, or any other data that will make sense for weighting the intrinsic distance. The blending medium $B(x, y, \tau) : \Omega \times [0, T) \rightarrow \mathbb{R}$ is then the data that is actually blended. Both in colorization and recolorization, we selected $B$ to be the chrominance. Yet, $B$ can be any image processing effect or any data that makes sense to blend for a given application.

Fig. 10 follows this idea and gives an example of object brightness change.

IV. CONCLUDING REMARKS

In this paper, we have introduced a fast colorization algorithm for still images and video. While keeping the quality at least as good as previously reported algorithms, the introduced technique manages to colorize images and movies within a second, compared to other techniques that may reach several minutes. The proposed approach needs less manual effort than techniques such as those introduced in [17] and [23] and can be used interactively due to its high speed. We also showed that simple
Fig. 8. Recolorization example. (a) Original color image, (b) scribbles, and (c) image after recolorization (note the pants, glasses, and window). The white scribbles represent a “mask,” used to keep the original colors.

Fig. 9. Recolorization example using the Cr channel for measurement rather than the intensity channel. (a) Original color image; (b) intensity channel of the image. It can be clearly seen that the intensity image does not contain significant structural information on the red rainbow strip (this is quite a unique example of this effect). On the other hand, both the Cb and Cr do change significantly between the stripes and can, therefore, be used for recolorization. (c) Recolored image, the red stripe of the rainbow was replaced by a purple one.

modifications in the algorithm lead to recolorization and other special effects [30].

A number of directions can be pursued as a result of the framework introduced here. First, as mentioned before, other special effects can be obtained following this distance-based blending of image attributes approach. We are working on including in this framework a crude (fast) computation of optical flow, to use mainly when the motion is too fast for the weighted distance used in the examples presented here. Another interesting problem is to investigate the use of this colorization approach for image compression. As shown in the examples presented here, a few color samples are often enough to produce visually pleasant results. This can be exploited for compression, in the same spirit as done in [20], where the encoder voluntarily drops information that can be efficiently reconstructed by the decoder. In order to do this, we need to understand what is the simplest set of color scribbles that when combined with our algorithm, manages to reconstruct the color without any visual artifacts. In the same spirit, for the colorization of originally mono-chromatic data, or the recoloring of color data, it is important to understand the best position to place the scribbles so that the user’s effort is minimized. For example, an edge map on the luminance data might help to guide the user. Although, as demonstrated with the examples provided in this paper and the additional ones in our web site, the robustness and speed of the algorithm make it possible to work in an interactive fashion, reducing/minimizing user intervention is always an important goal. Results in these research directions will be reported elsewhere.
CB to the maximum preselected change. (e) It is possible to further process the image using the same matte and (f) demonstrate this by similarly adding the gray-level matte parts of the image. The darkening is done simply by subtracting the gray-level matte from the intensity channel, where white means no change and black is the gray level of the matte. In this case, we chose to change the brightness. (d) Image after applying the darkening. Note that the intensity only changed in the desired grayscale matte (we only keep the blending channel). With the matte, it is possible to apply an effect to the original image with a magnitude proportional to the effect we do so by placing scribbles or by just marking whole areas. (c) Using our colorization method, we propagate the markings (white and black colors) and get a car into a darker color. (b) We demonstrate further processing on the original image. Our goal is to change the color of the yellow car into a darker color. (a) We define the blending medium by roughly marking areas we do not wish to change in white and areas we do want to change in black; we do so by placing scribbles or by just marking whole areas.

ACKNOWLEDGMENT

The authors would like to thank G. Brown, A. Bartesaghi, and K. Patwardhan for their inspiring and useful remarks.

REFERENCES


Liron Yatziv was born in Haifa, Israel, on January 21, 1975. He received the B.Sc. degree (summa cum laude) from the Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa, in 2001, and the M.Sc. degree from the University of Minnesota, Minneapolis, in 2005. He is currently an Associate Member of Technical Staff, Siemens Corporate Research, Princeton, NJ. In his current position at Siemens, he is responsible for developing image processing algorithms that will be used in advanced medical equipment. His current interests are in the fields of image processing and algorithms, about which he published several papers.

Guillermo Sapiro was born in Montevideo, Uruguay, on April 3, 1966. He received the B.Sc. (summa cum laude), M.Sc., and Ph.D. degrees from the Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa, in 1989, 1991, and 1993, respectively. After postdoctoral research at the Massachusetts Institute of Technology, Cambridge, he became a Member of the Technical Staff at the research facilities of HP Labs, Palo Alto, CA. He is currently with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis. He works on differential geometry and geometric partial differential equations, both in theory and applications in computer vision, computer graphics, medical imaging, and image analysis. He recently coedited a special issue on image analysis in the Journal of Visual Communication and Image Representation. He has authored and coauthored numerous papers in image analysis and he has written a book. Dr. Sapiro is a member of SIAM. He recently co-edited a special issue of IEEE TRANSACTIONS ON IMAGE PROCESSING. He was awarded the Gutwirth Scholarship for Special Excellence in Graduate Studies in 1991, the Ollendorff Fellowship for Excellence in Vision and Image Understanding Work in 1992, the Rothschild Fellowship for Postdoctoral Studies in 1993, the Office of Naval Research Young Investigator Award in 1998, the Presidential Early Career Awards for Scientist and Engineers (PECASE) in 1998, and the National Science Foundation Career Award in 1999.
An efficient colorization scheme for images and videos based on prioritized source propagation is proposed in this work. A user first scribbles colors on a set of source pixels in an image or the first frame of a movie. The proposed algorithm then propagates those colors to the other non-source pixels and the subsequent frames. Specifically, the proposed algorithm identifies the non-source pixel with the highest priority, which can be most reliably colorized. Then, its color is interpolated from the neighboring pixels. This is repeated until the whole image or movie is colorized. Simulation results demonstrate that the proposed algorithm yields more reliable colorization performance than the conventional algorithms.

Index Terms— Image colorization, video colorization, and source propagation.

1. INTRODUCTION

Colorization is the process of adding colors, which play an important role in the human perception of visual information, to monochrome images or videos [1]. In addition to the monochrome-to-color conversion, colorization can be used in various applications, e.g., the restoration of faded colors in old movies and the modification and enhancement of color tones in images or videos. However, manual colorization is too time-consuming and expensive, especially in the case of video sequences, consisting of a huge number of still images. Thus, an automatic colorization algorithm is required.

Various attempts have been made to develop efficient colorization schemes. Reinhard et al. [2] introduced an early colorization scheme, which transfers colors from a color source image to a gray target image by matching the luminance components of the two images. Welsh et al. [3] improved the matching performance by exploiting the luminance values and textures of neighboring pixels. These color transferring schemes provide acceptable colorization performance, provided that an input image has distinct luminance values or textures across object boundaries.

Given the luminance information of an image or video signal, the proposed algorithm attempts to generate color information, which looks natural and realistic. We work in the YUV domain...
Fig. 1. Colorization of the “Flower” image: (a) the input image with color sources, (b) the Levin et al.’s algorithm, (c) the Yatziv and Sapiro’s algorithm, and (d) the proposed algorithm.

space, where \(Y\) is the luminance channel, and \(U\) and \(V\) are the chrominance channels. Let \(V(p)\) denote the luminance of pixel \(p\), and \(C(p) = (U(p), V(p))\) denote the chrominance vector of \(p\).

2.1. Image Colorization

Given a luminance image, a user puts color values onto a selected set of pixels with a few brush strokes, as illustrated in Fig. 1 (a) or Fig. 2 (a). Then, the proposed algorithm propagates those color values to neighboring pixels to construct a color image.

The color accuracy \(a(p)\) of pixel \(p\) is defined as a number between 0 and 1, which indicates the reliability of the color \(C(p)\). For example, \(a(p) = 1\) when pixel \(p\) has the most accurate color \(C(p)\), whereas \(a(p) = 0\) means that \(C(p)\) is not reliable at all and thus should be updated using the color information of neighboring pixels. Initially, \(a(p)\) is set to 1 if \(p\) is assigned a color vector by the user, or set to 0 otherwise.

We call \(p\) a source pixel if \(a(p) = 1\). The color information of source pixels is propagated to non-source pixels. To this end, the priorities of non-source pixels are defined, and non-source pixels are colorized in the decreasing order of their priorities. Specifically, the priority \(\pi(p)\) of a non-source pixel \(p\) is defined as

\[
\pi(p) = \sum_{q \in N_p} a(q)e^{-|Y(p) - Y(q)|},
\]

where \(N_p\) denotes the 4-neighbor set of \(p\). In other words, if \(p = (x, y)\), \(N_p = \{(x-1, y), (x+1, y), (x, y-1), (x, y+1)\}\). Notice that \(p\) is assigned a higher priority \(\pi(p)\), if a neighboring pixel \(q\) has a high accuracy \(a(q)\) and the luminances \(Y(p)\) and \(Y(q)\) are similar to each other. In other words, if a source pixel has a neighbor and they have similar luminances, the neighbor is assigned a high priority.

Fig. 2. Colorization of the “Ocean” image: (a) the input image with color sources, (b) the Levin et al.’s algorithm, (c) the Yatziv and Sapiro’s algorithm, and (d) the proposed algorithm.

After assigning and sorting the priorities, we update the color vector \(C(p)\) of the non-source pixel \(p\) with the highest priority by

\[
C(p) \leftarrow a(p) \cdot C(p) + (1 - a(p)) \cdot \sum_{q \in N_p} w_{p,q} C(q),
\]

where the weight \(w_{p,q}\) is defined as

\[
w_{p,q} = \frac{a(q)e^{-|Y(p) - Y(q)|}}{\sum_{r \in N_p} a(r)e^{-|Y(p) - Y(r)|}}.
\]

In Eq. (2), \(C(p)\) is set as the weighted sum of neighboring colors, since the accuracy \(a(p)\) of the non-source pixel \(p\) is 0. In the weighted summation, \(C(q)\) is given a high weight \(w_{p,q}\) if \(q\) has a high accuracy and \(p\) and \(q\) have similar luminances.

After coloring pixel \(p\), its accuracy \(a(p)\) is updated to 1, i.e., \(p\) becomes a source pixel. The priorities of the neighboring pixels in \(N_p\) are also updated by Eq. (1). Then, the priorities of all non-source pixels are sorted, and the non-source pixel with the highest priority is colorized by Eq. (2). This is repeated until all pixels are colorized.

In the image colorization, the accuracy \(a(p)\) is binary: it is 1 if \(p\) is a source pixel, and 0 otherwise. However, in the video colorization, \(a(p)\) has a continuous value between 0 and 1, as will be described in the next subsection.

2.2. Video Colorization

In the video colorization, we colorize frames sequentially, and perform the motion-compensated prediction to use the colors in the previous frame to colorize the current frame.
The motion vector $v_p$ of pixel $p$ from frame $k$ to frame $k-1$ is estimated using the block matching algorithm by

$$v_p = \arg \min_{v \in \Lambda} \text{SAD}(p, v)$$  \hspace{1cm} (4)

where $\Lambda$ denotes a motion search range. SAD stands for the sum of absolute differences, given by

$$\text{SAD}(p, v) = \sum_{q \in B(p)} |Y_q(q) - Y_{k-1}(q + v)|$$  \hspace{1cm} (5)

where $B(p)$ is a block centered at $p$, and the subscript $k$ denotes the frame index. In this work, the motion search range $\Lambda$ is $[-7, 7] \times [-7, 7]$ and the size of $B(p)$ is $5 \times 5$. After the motion estimation, the color $C_k(p)$ of pixel $p$ in the current frame is initialized with the color $C_{k-1}(p + v_p)$ of the matching pixel in the previous frame.

However, the estimated motion vector $v_p$ may not represent the true motion, and the initial color may be erroneous. To overcome this problem, by investigating the characteristics of the SAD function in Eq. (4), we measure the color accuracy $a(p)$ of the initial color $C(p)$ as follows.

$$a(p) = \exp \left( -\frac{\text{SAD}(p, v_p)}{\text{SAD}(p, \bar{v}_p)} \right)$$  \hspace{1cm} (6)

where $\beta$ is a positive scaling factor and

$$\bar{v}_p = \arg \min_{v \in \Lambda, \|C_{k-1}(p + v) - C_{k-1}(p + v_p)\| > \theta} \text{SAD}(p, v)$$  \hspace{1cm} (7)

Note that, while $v_p$ is the optimal motion vector yielding the smallest SAD among the entire search range, $\bar{v}_p$ is the optimal motion vector among the vectors that point to different colors than $v_p$ by a threshold $\theta$. The scaling factor $\beta$ in Eq. (6) is fixed to 10, and the threshold $\theta$ in Eq. (7) is fixed to 5 in this work.

Fig. 3 (a) shows the initial colorization result of a frame in the “Funny Face” movie, in which the color of each pixel is copied from the previous frame using the motion vector field. Fig. 3 (b) is the corresponding color accuracy map, in which a brighter pixel depicts a higher accuracy. We see that the map estimates color accuracies faithfully, and pixels near object boundaries, where the colorization errors are observed, are assigned low color accuracies.

Fig. 4. The effects of the color source positions on the colorization results. (a) the input gray image with color sources, (b) the Levin et al.’s algorithm, (c) the Yatziv and Sapiro’s algorithm, and (d) the proposed algorithm.

After the initial colorization and the color accuracy computation, the priority of each pixel is obtained via Eq. (1). Then, as in the image colorization, the color of the pixel with the highest priority is updated via the rule in Eq. (2), its accuracy is set to 1, and the priorities of the neighboring pixels are updated. This is repeated until all colors are updated. Fig. 3 (c) shows the final colorization result of the “Funny Face” movie. Note that the proposed algorithm reduces the errors
Fig. 5. Video colorization of the “Truck” sequence. From top to bottom, the 1st frame before colorization, the 1st colorized frame, the 12th colorized frame, and an enlarged part of the 12th frame. (a) The Levin et al.’s algorithm, (b) the Yatziv and Sapiro’s algorithm, and (c) the proposed algorithm.

3. SIMULATION RESULTS

We evaluate the performance of the proposed algorithm on various test images and movies. For comparison, the colorization results of the Levin et al.’s algorithm in [4] and the Yatziv and Sapiro’s algorithm in [5] are also presented.

Fig. 1 compares the results on the “Flower” image. Since the petals and the background have similar intensities, the Levin et al.’s algorithm causes severe errors. Similarly, the Yatziv and Sapiro’s algorithm propagates the colors of the background to the petals erroneously. On the other hand, we see that the proposed algorithm provides more natural reconstruction with crisp color boundaries. Fig. 2 shows the results on the “Ocean” image. Whereas the conventional algorithms cause color blurring and erroneous color propagation, the proposed algorithm provides high-quality reconstruction without any noticeable artifacts.

Fig. 4. On the same gray image, we assign colors to different source pixels. Note that the performance of the Levin et al.’s algorithm is significantly affected by the source locations. This is because the smoothness constraint is included in their cost function, and thus a target pixel is more affected by a geometrically closer source pixel. Thus, color blurring is observed in their algorithm. The Yatziv and Sapiro’s algorithm is even more affected by the source locations. Their algorithm uses the geodesic distance for color blending. On the other hand, the proposed algorithm propagates colors sequentially with proper prioritization. Specifically, the proposed algorithm repeatedly identifies the pixel, which can be colorized most reliably, paints it, and then makes it another source pixel. Therefore, the performance of the proposed algorithm is much less sensitive to the initial source locations, as shown in Fig. 4 (d).

Fig. 5 shows video colorization results. After a user strokes color brushes on the 1st frame, those colors are propagated to the remaining pixels and also to the subsequent frames. In the Levin et al.’s algorithm, the colors of the truck and the background are blended together and the qualities degrade in later frames. Also, in the Yatziv and Sapiro’s algorithm, artifacts are observed around the truck. The proposed algorithm alleviates these artifacts and provides significantly better results. Simulation results on various other images and videos also confirm that the proposed algorithm requires much less human interactions than the conventional algorithms to achieve similar colorization qualities.

4. CONCLUSIONS

A simple but efficient colorization scheme for images and videos, which propagates the colors of source pixels to non-source pixels in a prioritized manner, was proposed in this work. The proposed algorithm derives the color accuracies and the colorization priorities, and colorizes the non-source pixels in the decreasing order of their priorities. Simulation results demonstrated that the proposed algorithm yields significantly better performance than the conventional colorization algorithms.

Future research topics include the inclusion of texture information in the colorizing process and the handling of occluded objects in video colorization.

5. REFERENCES