Decolorize: fast, contrast enhancing, color to grayscale conversion

Mark Grundland and Neil A. Dodgson
Computer Laboratory, University of Cambridge, United Kingdom
Algorithm Documentation ☙ mark@eyemaginary.com ❧ September 8, 2005

Implementation

MatLab: `tones= decolorize(picture,effect,scale,noise)` Here is a link to the code: 

Objective

Convert a color image into a grayscale image by representing color contrasts in grayscale in a visually pleasing way.

Priorities

The algorithm's design aims for simplicity, speed, and scalability without overly sacrificing quality. The algorithm is intended for real time performance.

Goals

Contrast Magnitude: The magnitude of the grayscale contrasts should visibly reflect the magnitude of the color contrasts.

Contrast Polarity: The positive or negative polarity of gray level change in the grayscale contrasts should visibly correspond to the polarity of luminance change in the color contrasts.

Dynamic Range: The dynamic range of the gray levels in the grayscale image should visibly accord with the dynamic range of luminance values in the color image.

Constraints

Continuous mapping: The transformation from color to grayscale is a continuous function. This constraint reduces image artifacts, such as false contours in homogeneous image regions.

Global Consistency: When two pixels have the same color in the color image, they will have the same gray level in the grayscale image. This constraint assists in image interpretation by allowing the ordering of gray levels to induce a global ordering relation on image colors.

Grayscale preservation: When a pixel in the color image is gray, it will have the same gray level in the grayscale image. This constraint assists in image interpretation by enforcing the usual relationship between gray level and luminance value.

Luminance ordering: When a sequence of pixels of increasing luminance in the color image share the same hue and saturation, they will have increasing gray levels in the grayscale image. This constraint reduces image artifacts, such as local reversals of image polarity.

Properties

Saturation Ordering: When a sequence of pixels having the same luminance and hue in the color image has a monotonic sequence of saturation values, its sequence of gray levels in the grayscale image will be a concatenation of at most two monotonic sequences.

Hue Ordering: When a sequence of pixels having the same luminance and saturation in the color image has a monotonic sequence of hue angles that lie on the same half of the color circle, its sequence of gray levels in the grayscale image will be a concatenation of at most two monotonic sequences.

Usage

Algorithm: $T = \text{decolorize}(X, \lambda, \sigma, \eta)$
(in MatLab) `tones= decolorize(picture,effect,scale,noise)`

Input: X is a color image, where each pixel X_i stores a linear RGB color value $(R_i, G_i, B_i) \in [0,1]^3$.

Output: T is a grayscale image, where each pixel stores a graylevel value $T_i \in [0,1]$.

Color Standard: The commonly used NTSC-Rec.601 standard is taken as the default grayscale conversion of the input RGB color values. The algorithm assumes linear color and grayscale values, so that gamma correction need not be performed in either preprocessing or postprocessing. Nevertheless, the algorithm may be easily extended to incorporate suitable gamma correction (e.g. standard monitor gamma of 2.2) or a different color standard (e.g. the sRGB color model with the ITU-Rec.709 color standard).

Effect Parameter: $0 \le \lambda \le 1$ is an indicator of how much the image's achromatic content can be changed to accommodate its chromatic contrasts. Its typical value is $\lambda = 0.5$ for a conspicuous effect or $\lambda = 0.3$ for a more subtle effect. This parameter enables the user to control the intensity of the contrast enhancing effect, together with the resulting expansion of the dynamic range.

Scale Parameter: $0 < \sigma$ is the radius in pixels of relevant color contrast features. Its typical value is $\sigma = 25$ for a 300×300 image. This parameter enables the user to take account of image resolution, ensuring that the algorithm pays attention to image features at the correct spatial scale.

Noise Parameter: $0 < \eta \ll \frac{1}{2}$ is a quantile specifying the portion of image pixels that are outliers, displaying aberrant color values. Its typical value is $\eta = 0.001$ for a high quality photograph. This parameter enables the user to compensate for image noise, ensuring that the algorithm can robustly determine the extent of the image's dynamic range.

```
function [tones,recolor] = decolorize(picture,effect,scale,noise)
% Created: 4 June 2005
% Version: 8 September 2005
% Copyright 2005, Mark Grundland. All rights resaved.
% Licensed for noncommercial, academic use only.

% Developed by Mark Grundland.
% Computer Laboratory, University of Cambridge, UK.
% Technical Report UCAM-CL-TR-649, University of Cambridge.
% Web page: http://www.eyemaginary.com/Portfolio/Publications.html
% Any questions? Contact mark @ eyemaginary.com .

% Usage:   Coverts RGB picture to grayscale tones,
%             while endeavoring to express color contrasts as tone changes.
%          Can also recolor the original picture
%             to incorporate the enhanced grayscale tones.
% Options: effect specifies how much the picture's achromatic content
%             should be altered to accommodate the chromatic contrasts
%             (default effect = 0.5 )
%          scale in pixels is the typical size of
%             relevant color contrast features
%             (default scale = sqrt(2*min(size(picture.dimensions)) )
%          noise quantile indicates the amount of noise in the picture
%             enabling the dynamic range of the tones to be appropriately scaled
%             (default noise = 0.001)
% Assumes: valid RGB 0 <= picture <= 1
%          positive 0 <= effect <= 1
%          positive 1 <= scale << min(picture.dimensions)
%          small quantile 0 <= noise << 0.5
% Applies: Standard NTSC color to grayscale conversion
%             is used as the default achromatic color channel.
% Example: tones=decolorize(picture,0.5,25,0.001)


% Examine inputs
frame=[size(picture,1), size(picture,2)];
pixels=frame(1)*frame(2);
if nargin<2 || isempty(effect)
    effect=0.5;
end;
if nargin<3 || isempty(scale)
    scale=sqrt(2*min(frame));
end;
```

```
44  if nargin<4 || isempty(noise)
45      noise=0.001;
46  end;
47
48  % Reset the random number generator
49  randn('state',0);
50  tolerance=100*eps;
51
52  % Define the YPQ color space
53  colorconvert=[0.29893602129377538475271555, 0.58704307445112129093513270,  0.11402090425510332431215180;
54                0.5,                          0.5,                          -1;
55                1,                           -1,                            0]';
56  colorrevert= [1,                           0.11402090425510332431215180,  0.64405352657867295309120860;
57                1,                           0.11402090425510332431215180, -0.35594647342132704690879140;
58                1,                          -0.88597909574489667568784820,  0.14405352657867295309120860]';
59  colorspan=[ 0,  1;
60             -1,  1;
61             -1,  1];
62  maxluminance=1;
63  scaleluminance=0.66856793424088827189;
64  maxsaturation=1.1180339887498948482;
65  alter=effect*(maxluminance/maxsaturation);
66
67  % Covert picture to the YPQ color space
68  picture=reshape(picture,[pixels,3]);
69  image=picture*colorconvert;
70  original=image;
71  chroma=sqrt(image(:,2).*image(:,2)+image(:,3).*image(:,3));
72
73  % Pair each pixel with a randomly chosen sample site
74  mesh=reshape(cat(3,repmat((1:frame(1))',[1,frame(2)]),repmat((1:frame(2)),[frame(1),1])),[pixels,2]);
75  displace=(scale*sqrt(2/pi))*randn(pixels,2);
76  look=round(mesh+displace);
77  redo=find((look(:,1)<1));
78  look(redo,1)=2-rem(look(redo,1),frame(1)-1);
79  redo=find((look(:,2)<2));
80  look(redo,2)=2-rem(look(redo,2),frame(2)-1);
81  redo=find((look(:,1)>frame(1)));
82  look(redo,1)=frame(1)-1-rem(look(redo,1)-2,frame(1)-1);
83  redo=find((look(:,2)>frame(2)));
84  look(redo,2)=frame(2)-1-rem(look(redo,2)-2,frame(2)-1);
85  look=look(:,1)+frame(1)*(look(:,2)-1);
86
```

```
% Calculate the color differences between the paired pixels
delta=image-image(look,:);
contrastchange=abs(delta(:,1));
contrastdirection=sign(delta(:,1));
colordifference=picture-picture(look,:);
colordifference=sqrt(sum(colordifference.*colordifference,2))+eps;

% Derive a chromatic axis from the weighted sum of chromatic differences between paired pixels
weight=1-((contrastchange/scaleluminance)./colordifference);
weight(find(colordifference<tolerance))=0;
axis=weight.*contrastdirection;
axis=delta(:,2:3).*[axis, axis];
axis=sum(axis,1);

% Project the chromatic content of the picture onto the chromatic axis
projection=image(:,2)*axis(1)+image(:,3)*axis(2);
projection=projection/(quantiles(abs(projection),1-noise)+tolerance);

% Combine the achromatic tones with the projected chromatic colors and adjust the dynamic range
image(:,1)=image(:,1)+effect*projection;
imagerange=quantiles(image(:,1),[noise; 1-noise]);
image(:,1)=(image(:,1)-imagerange(1))/(imagerange(2)-imagerange(1)+tolerance);
targetrange=effect*[0; maxluminance]+(1-effect)*quantiles(original(:,1),[noise; 1-noise]);
image(:,1)=targetrange(1)+(image(:,1)*(targetrange(2)-targetrange(1)+tolerance));
image(:,1)=min(max(image(:,1),original(:,1)-alter.*chroma),original(:,1)+alter.*chroma);
image(:,1)=min(max(image(:,1),0),maxluminance);

% Return the results
tones=image(:,1)/maxluminance;
tones=reshape(tones,frame);
if nargout>1
    recolor=image*colorrevert;
    recolor=cat(3,reshape(recolor(:,1),frame),reshape(recolor(:,2),frame),reshape(recolor(:,3),frame));
    recolor=min(max(recolor,0),1);
end;
```

```
130  function r = quantiles(x,q);
131  % Usage:   Finds quantiles q of data x
132  % Assumes: quantiles 0 <= q <= 1
133  % Example: r=quantiles(x,[0; 0.5; 1]);
134  %           xmin=r(1); xmedian=r(2); xmax=r(3)
135
136  tolerance=100*eps;
137  q=q(:);
138  x=sort(x);
139  n=size(x,1);
140  k=size(x,2);
141  e=1/(2*n);
142  q=max(e,min(1-e,q));
143  q=n*q+0.5;
144  p=min(n-1,floor(q));
145  q=q-p;
146  q(find(tolerance>q))=0;
147  q(find(q>(1-tolerance)))=1;
148  q=repmat(q,[1,k]);
149  r=(1-q).*x(p,:)+q.*x(p+1,:);
```