Meeting Andy Warhol Somewhere Over the Rainbow: RGB Colormapping and ITK

Nicholas J. Tustison[1], Hui Zhang[1], Gaëtan Lehmann[2], Paul Yushkevich[1] and James C. Gee[1]

January 7, 2009

[1]Penn Image Computing and Science Laboratory, University of Pennsylvania, USA
[2]Unité de Biologie du Développement et de la Reproduction, Institut National de la Recherche Agronomique, 78350 Jouy-en-Josas, France

Abstract

Although greyscale intensity values are primarily used in image data visualization oftentimes, due to the requirements of aesthetics (whether they be self-imposed or collaborator-suggested), mapping the greyscale image to a user-defined colormap is desired. In this paper, we propose a framework for inclusion in the ITK library for converting intensity-valued images to user-defined RGB colormap images. We also include several colormaps that can be readily applied for visualization of images in such programs as ITK-SNAP or can be used as examples for creating new colormaps.

Latest version available at the Insight Journal [http://hdl.handle.net/1926/1452]

Contents

1 Introduction **2**

2 Implementation **2**
2.1 Colormap Functor Base Class . . . 3
2.2 Derived Colormap Functor Classes . . . 4
2.3 Custom Colormap Image Functor Class . . . 5
2.4 The Coordinating Scalar Image To RGB Image Filter . . . 5
Predefined Colormaps . . . 5
Custom Colormaps . . . 7

3 Examples **8**
3.1 Visualizing RGB color images with ITK SNAP . . . 8
3.2 An ITK Homage to Andy Warhol . . . 8

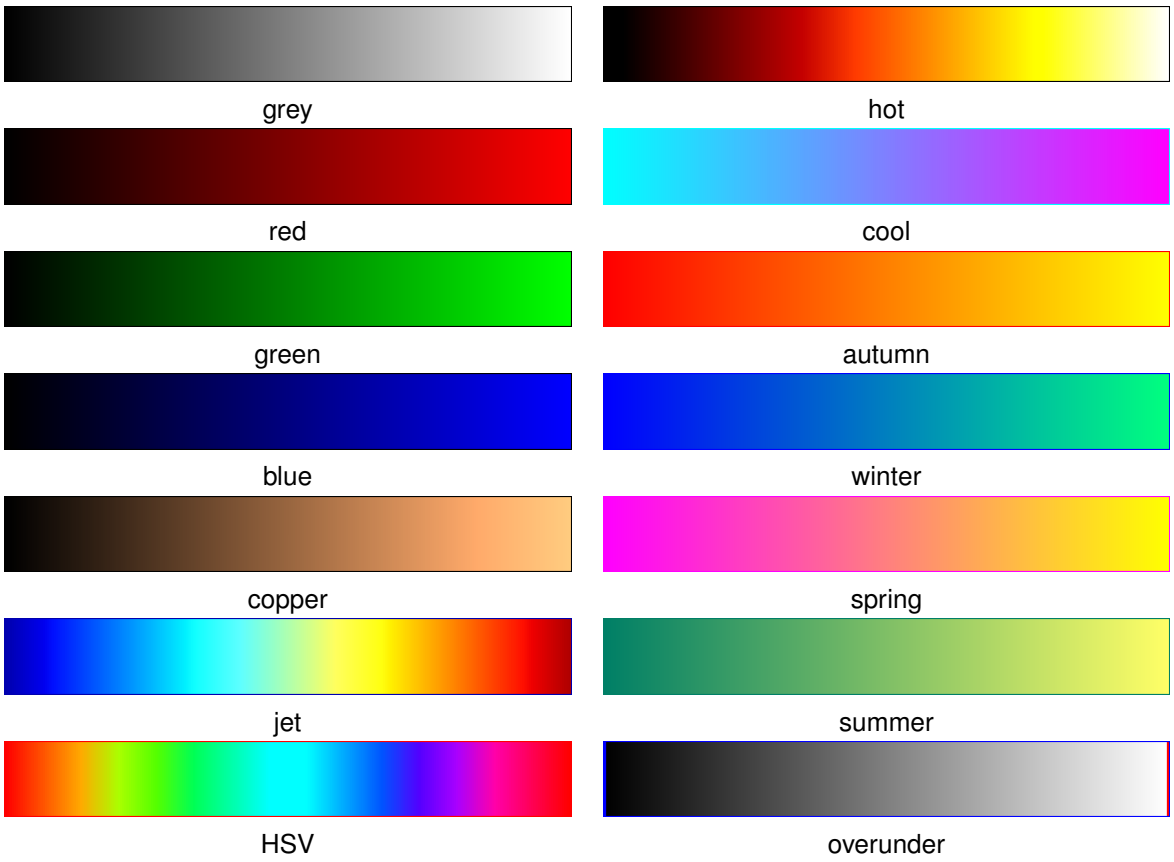


Figure 1: Implemented colormaps.

1 Introduction

Visualization of medical imagery can be enhanced through the use of mapping the scalar intensity values to RGB values. These images can then be visualized in such programs as ITK-SNAP. Given in Figure 1 is a list of colormaps that have been implemented and are included with this submission. We have also designed the framework to facilitate the development of other user-defined colormaps.

2 Implementation

Various classes exist to deal with RGB images. In fact, based on the title of one of the existing classes, we initially believed that the functionality we propose in this submission already existed in the toolkit. However, the class `ScalarToRGBPixelFunctor` performs a specific type of mapping "for visualizing labeled images which cannot be mapped successfully into grayscale images" which is not what we are proposing. In addition, Gaëten Lehmann has also provided the `LabelOverlayImageFilter` which has coloring and opacity capabilities but is based on the binary input (requiring a label image and input image).

The overall use of our proposed framework is that the user would have various colormap functor classes which are already defined or defined by the user (and subsequently submitted to the Insight Journal, of course). The user could then "plug in" any one of these colormaps to the image-to-image filter we created modeled after the `UnaryFunctorImageFilter` class to map an scalar image to an RGB image.

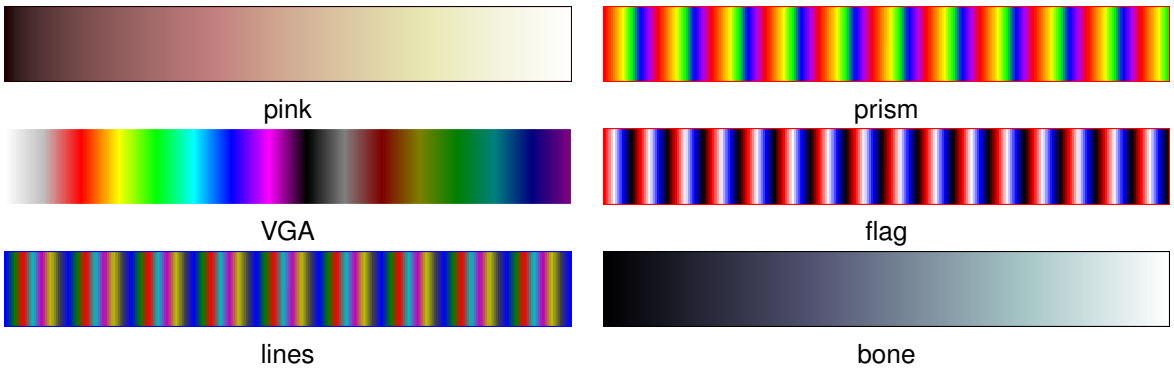


Figure 2: Custom colormaps created using a sampled piecewise description.

2.1 Colormap Functor Base Class

We introduce a new abstract base functor class called `ColormapFunctor`. In mapping the set of input grey pixel intensity values to RGB values, it is often helpful to rescale the input values to the range $[0, 1]$ before performing the mapping to normalized RGB values also in the range $[0, 1]$. One can then rescale the normalized RGB values to the desired range. Therefore, the abstract base colormap functor class defines two helper rescaling functions: `RescaleInputValue` and `RescaleRGBComponentValue`. The former takes the input value and linearly rescales it to the range $[0, 1]$ according to the parameters

- `m_MinimumInputValue`,
- `m_MaximumInputValue`.

These values are calculated automatically from the input image by default in the coordinating filter class `itkScalarToRGBImageFilter` discussed in a subsequent section. However, this automatic behavior can be overridden by setting the parameter `UseInputImageExtremaForScaling` to `false` and specifying these input extrema parameters manually.

The function `RescaleRGBComponentValue` then takes the normalized RGB values assumed to be in the range $[0, 1]$ and linearly rescales them to the user-specified range [`m_MaximumRGBComponentValue`, `m_MaximumRGBComponentValue`]. By default, these are set to the minimum and maximum values of the RGB component type. We demonstrate the use of these functions when we discuss one of the derived RGB colormap functor classes.

Also, to be consistent with the various other functors that have been defined in ITK, we define the following operator functions:

- `virtual bool operator!=( const ScalarToRGBColormapFunctor & ) const`
- `virtual bool operator==( const ScalarToRGBColormapFunctor & other ) const`
- `virtual RGBPixelType operator()( const ScalarType & ) const = 0`

The reader will note the pure `operator()` function which is the only function we redefined in each of our derived colormap functor classes.

2.2 Derived Colormap Functor Classes

Each derived class defines a unique colormap functor class. For example, we include the following files defining the different colormaps in Figure 1:

- GreyColormapFunctor.h, GreyColormapFunctor.txx
- RedColormapFunctor.h, RedColormapFunctor.txx
- GreenColormapFunctor.h, GreenColormapFunctor.txx
- BlueColormapFunctor.h, BlueColormapFunctor.txx
- CopperColormapFunctor.h, CopperColormapFunctor.txx
- HotColormapFunctor.h, HotColormapFunctor.txx
- CoolColormapFunctor.h, CoolColormapFunctor.txx
- AutumnColormapFunctor.h, AutumnColormapFunctor.txx
- WinterColormapFunctor.h, WinterColormapFunctor.txx
- SpringColormapFunctor.h, SpringColormapFunctor.txx
- SummerColormapFunctor.h, SummerColormapFunctor.txx
- JetColormapFunctor.h, JetColormapFunctor.txx
- HSVColormapFunctor.h, HSVColormapFunctor.txx
- OverUnderColormapFunctor.h, OverUnderColormapFunctor.txx
- CustomColormapFunctor.h, CustomColormapFunctor.txx

To demonstrate the facility of creating a new colormap functor class, we invite the reader to inspect the CopperColormapFunctor class, specifically the code defining the operator() function which describes how to perform the mapping between scalar intensity values and the copper colormap.

```
26 template <class TScalar, class TRGBPixel>
27 typename CopperColormapFunctor<TScalar, TRGBPixel>::RGBPixelType
28 CopperColormapFunctor<TScalar, TRGBPixel>
29 ::operator()( const TScalar & v ) const
30 {
31   // Map the input scalar between [0, 1].
32   RealType value = this->RescaleInputValue( v );
33
34   // Apply the color map.
35   RealType red = 1.2 * value;
36   red = vnl_math_min( 1.0, red );
37
38   RealType green = 0.8 * value;
39
40   RealType blue = 0.5 * value;
41
42   // Set the rgb components after rescaling the values.
43   RGBPixelType pixel;
44
```

```
45    pixel[0] = this->RescaleRGBComponentValue( red );
46    pixel[1] = this->RescaleRGBComponentValue( green );
47    pixel[2] = this->RescaleRGBComponentValue( blue );
48
49    return pixel;
50  }
```

The input scalar intensity value is remapped to the range $[0,1]$ (line 32). The colormap is then applied to obtain the red, green and blue values. The resulting RGB pixel is then returned. The reader will note that this is the only function that was redefined to implement each colormap.

2.3 Custom Colormap Image Functor Class

In addition to being able to derive other colormap classes, we provide a custom colormap functor class which allows one to specify a piecewise uniform sampling of the different RGB profile channels. Each channel (either red, green, or blue) is specified by a vector of scalar values **in the range** $[0,1]$. The mapping is performed by linearly interpolating between sample profile values. For example, the red, green and blue channels of the `cool` colormap class can be described, respectively, as

- $\{0,1\}$
- $\{1,0\}$
- $\{1\}$

where the red channel varies linearly from 0 to 1 over the entire domain of the input scalar image, the green channel varies linearly from 1 to 0 over the domain of the input scalar image, and the blue channel has a constant value of 1 over the entire domain. Other colormaps require more samples for adequate description. We provide several of these colormaps described by piecewise samples (and shown in Figure 2) in the accompanying `Source/CustomColormaps/` directory.

2.4 The Coordinating Scalar Image To RGB Image Filter

In previous sections, we described the various colormap functor classes. In this section, we describe the coordinating filter, `ScalarToRGBColormapImageFilter`, which accepts one of these colormap functor classes and an input scalar image and produces, as output, an RGB image where the pixel values have been mapped according to the functor class. We derive this class from the `ImageToImageFilter` class with a large portion of the code taken and adapted from the `UnaryFunctorImageFilter` class. In this way, the class has multi-thread capabilities. Also, if no colormap is specified, the output behavior is defaulted to a grayscale mapping. Usage is best illustrated by examining the test code which produced the images contained in the following sections.

Predefined Colormaps

Preliminaries include specifying the input/output image types and reading the input image. After these preliminaries, the filter is instantiated and supplied the input image.

```
29 int itkScalarToRGBColormapImageFilterTest( int argc, char *argv[] )
30 {
31   const unsigned int ImageDimension = 2;
32
33   typedef unsigned int PixelType;
34   typedef itk::RGBPixel<unsigned char> RGBPixelType;
35 //  typedef itk::RGBAPixel<unsigned char> RGBPixelType;
36
37   typedef itk::Image<PixelType, ImageDimension> ImageType;
38   typedef itk::Image<float, ImageDimension> RealImageType;
39   typedef itk::Image<RGBPixelType, ImageDimension> RGBImageType;
40
41   typedef itk::ImageFileReader<ImageType> ReaderType;
42   ReaderType::Pointer reader = ReaderType::New();
43   reader->SetFileName( argv[1] );
44   reader->Update();
45
46   std::string colormapString( argv[3] );
47
48   typedef itk::ScalarToRGBColormapImageFilter<ImageType,
49     RGBImageType> RGBFilterType;
50   RGBFilterType::Pointer rgbfilter = RGBFilterType::New();
51   rgbfilter->SetInput( reader->GetOutput() );
```

Subsequently the selected colormap functor is instantiated and supplied to the image filter. The next step concerns the selection of the desired colormap. For predefined colormaps, we have created an enumerated type such that setting the colormap is performed relatively easily. For example, selecting the “hot” colormap is performed via

```
73   else if ( colormapString == "hot"  )
74     {
75     rgbfilter->SetColormap( RGBFilterType::Hot );
76     }
```

Alternatively, we can instantiate the specific functor and set the colormap. For example, the “jet” colormap can be specified by the following snippet:

```
97   else if ( colormapString == "jet"  )
98     {
99 //    rgbfilter->SetColormap( RGBFilterType::Jet );
100    typedef itk::Functor::JetColormapFunctor<ImageType::PixelType,
101      RGBImageType::PixelType> ColormapType;
102    ColormapType::Pointer colormap = ColormapType::New();
103    rgbfilter->SetColormap( colormap );
104    }
```

where we have commented out the enumerated type call. We then specify the desired minimum and maximum RGB component values (lines 166-167). We then write the output to disk where we can visualize the result in ITK-SNAP or other software capable of viewing RGB images.

```
166  rgbfilter->GetColormap()->SetMinimumRGBComponentValue( 0 );
167  rgbfilter->GetColormap()->SetMaximumRGBComponentValue( 255 );
168
169  try
170    {
171    rgbfilter->Update();
172    }
```

```
  catch (...)
    {
    return EXIT_FAILURE;
    }

  typedef itk::ImageFileWriter<RGBImageType> WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName( argv[2] );
  writer->SetInput( rgbfilter->GetOutput() );
  writer->Update();
```

Custom Colormaps

For a custom colormap defined by piecewise samples, the following code which uses the accompanying text files in the `Source/CustomColormaps/` directory, illustrates its use. The reader will note that the channels do not necessarily need to be of the same length. For example, earlier we discussed the `cool` colormap class where the red channel varies linearly from 0 to 1 over the entire domain of the input scalar image, the green channel varies linearly from 1 to 0 over the domain of the input scalar image, and the blue channel has a constant value of 1 over the entire domain. The contents of the colormap file `Source/CustomColormaps/cool.txt` are, appropriately enough,

```
0 1
1 0
1
```

where the first line describe the red channel, the second line describes the green channel, and the third line describes the blue channel. A slightly more intricate example is the VGA colormap defined in the file `Source/CustomColormaps/vga.txt`.

```
1 0.75 1 1 0 0 0 1 0 0.5 0.5 0.5 0 0 0 0.5
1 0.75 0 1 1 1 0 0 0 0.5 0 0.5 0.5 0.5 0 0
1 0.75 0 0 0 1 1 1 0 0.5 0 0 0 0.5 0.5 0.5
```

Again, the first line defines, in a piecewise fashion, the red channel, the second the green channel, and the third line defines the blue channel.

These custom files are read into the custom colormap functor as follows:

```
  else if ( colormapString == "custom"  )
    {
    typedef itk::Functor::CustomColormapFunctor<ImageType::PixelType,
      RGBImageType::PixelType> ColormapType;
    ColormapType::Pointer colormap = ColormapType::New();

    ifstream str( argv[4] );
    std::string line;

    // Get red values
    {
    std::getline( str, line );
    std::istringstream iss( line );
    float value;
    ColormapType::ChannelType channel;
    while ( iss >> value )
      {
```

```
      channel.push_back( value );
      }
    colormap->SetRedChannel( channel );
    }

    // Get green values
    {
    std::getline( str, line );
    std::istringstream iss( line );
    float value;
    ColormapType::ChannelType channel;
    while ( iss >> value )
      {
      channel.push_back( value );
      }
    colormap->SetGreenChannel( channel );
    }
    // Get blue values
    {
    std::getline( str, line );
    std::istringstream iss( line );
    float value;
    ColormapType::ChannelType channel;
    while ( iss >> value )
      {
      channel.push_back( value );
      }
    colormap->SetBlueChannel( channel );
    }
    rgbfilter->SetColormap( colormap );
    }
```

3 Examples

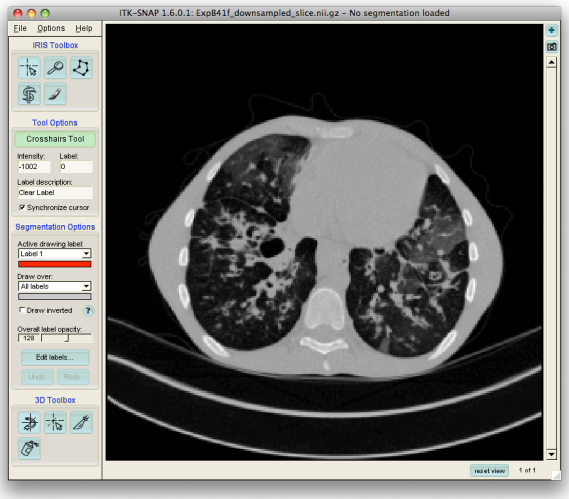
3.1 Visualizing RGB color images with ITK SNAP

In Figure 3 we showcase recently added functionality to ITK-SNAP which allows the user to visualize RGB image layers. After converting the RGB image shown in the top left of Figure 3 to an RGB image using the spring colormap, we can visualize the two images with varying levels of opacity.[1]

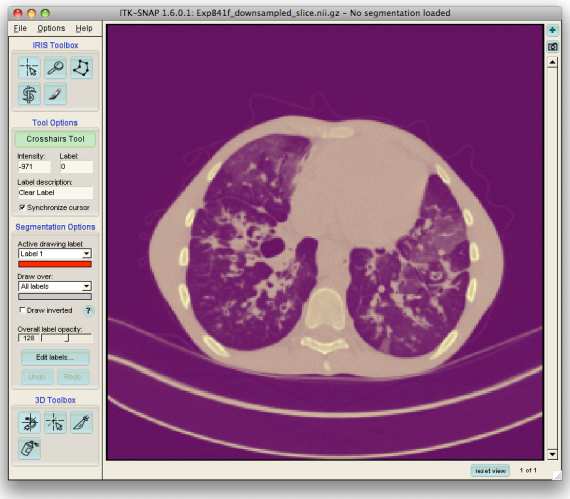
3.2 An ITK Homage to Andy Warhol

As a pioneer in the Pop Art movement Andy Warhol is famous for depicting various iconic figures (e.g. Marilyn Monroe) using different coloring schemes for artistic purposes. We use our new framework to produce our own creation very much in the spirit (although perhaps lacking Warhol's artistic sensitivities) of the Pop Art movement in Figure 5 from the image in Figure 4 (available at `http://en.wikipedia.org/wiki/Lenna`).

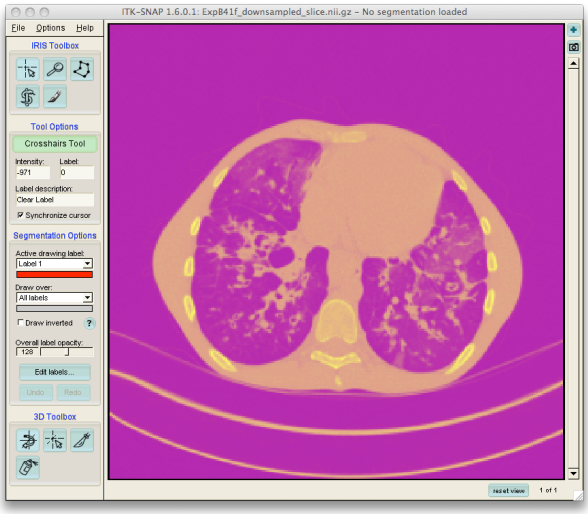
[1] The reader will note that only RGB images of the metaIO format (.mha) are currently readable in ITK-SNAP.



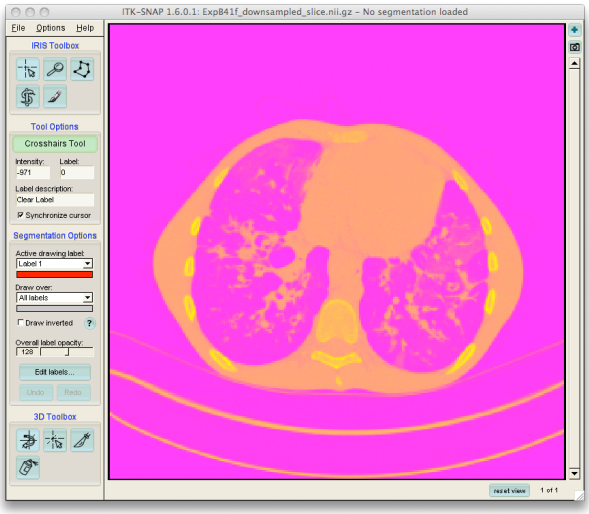
RGB overlay opacity = 0



RGB overlay opacity = 82



RGB overlay opacity = 164



RGB overlay opacity = 255

Figure 3: CT axial lung slice displayed with a spring colormap RGB overlay in ITK-SNAP with varying levels of opacity.

Figure 4: Subject for our ITK Pop Art contribution.

Figure 5: An ITK Homage to Andy Warhol.