Generating Quality Assurance Images using FSL `overlay` and `slicer`

(click here for PDF version)

Johnathan W. Hill

Introduction

Single subject neuroimaging data are generally inspected using tools such as `fslview`. However, checking each image one at a time using this method can be tedious when performing quality assurance checks on multiple subjects. To optimize the neuroimaging quality assurance process, FSL tools such as `overlay` and `slicer` are used instead.

`overlay`

FSL's `overlay` is a program that generates a Neuroimaging Informatics Technology Initiative (NifTI) formatted image with one or more *overlay images* superimposed on a *background image*. This program is useful for quality assurance purposes because it allows a researcher to compare two or more versions of the same data.

A simplified syntax version for `overlay` is as follows (type `overlay` without arguments to get the whole story):

```
overlay <colour_type> <output_type> [-c] <background_image> <bg_min>
<bg_max> <stat_image_1> <s1_min> <s1_max> <output_file> [stat_image_2
s2min s2max]
```

Explanation of arguments:

`colour_type:` this number indicates if the overlay is solid (0) or transparent (1).
`output_type:` this number indicates if the output file will be encoded as floating points (0) or integer values (1).
`bg_min` & `bg_max`: range of contrast intensity of background image. The intensity range units are the same as those used in the `fslview` intensity bar. Use the -a flag to automatically estimate the intensity range or -A to use the entire range of the image.
`background_image`: image to be placed in background.
`-c`: optional flag that creates an output image with a checkered background
`stat_image`: overlay image sitting on top of background.
`s1_min` & `s1_max`: this is the intensity value range used by a color look-up table (see *More on Look-Up Tables*). Values 15-25, for example, will reassign intensity values under 15 to 15 and over 25 to 25. The range values specified here are the same as those used by the `fslview` contrast and illuminance toolbar. The look-up table values converted from intensity values are proportional to the specified intensity range.
`output_file`: output image.
`stat_image_2 s2min s2max`: specifies a second overlay image

Demo: using `overlay` for quality assurance of skull-stripped images

Reorienting raw neuroimaging data to standard radiological orientation and skull-stripping these reoriented images (i.e., removing non-brain anatomical structures such as the eyes and skull)

are two of the first steps in neuroimaging preprocessing. In this stage of preprocessing, `overlay` can be used to *overlay* a skull-stripped brain over its pre-skull-stripped version. Therefore, `overlay` permits researchers to compare neuroimaging data before and after processing.

Another FSL command, `fslreorient2std`, is used to reorient a raw NifTI image to radiological orientation. For instance, `T1_raw.nii.gz`, a raw structural image, is reoriented, renamed, and outputted as `T1_reoriented.nii.gz` using:

```
fslreorient2std T1_raw.nii.gz T1_reoriented.nii.gz
```

and `T1_reoriented.nii.gz` is skull-stripped using:

```
bet T1_reoriented.nii.gz T1_bet.nii.gz -R
```

The output will be a reoriented skull-stripped brain, `T1_bet.nii.gz.`

To generate a quality assurance image, the skull-stripped brain is placed on top of the reoriented brain. An image (`rendered.nii.gz`) with the two images superimposed is created using:

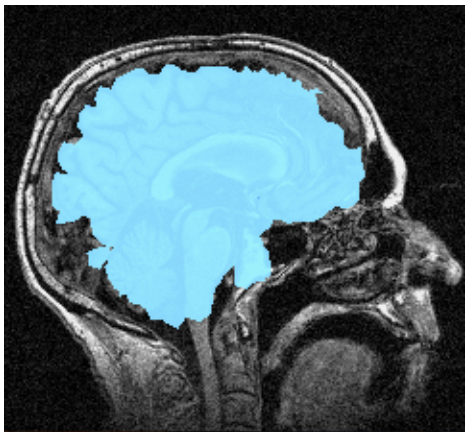
```
overlay 1 1 T1_reoriented.nii.gz -a T1_bet.nii.gz 1
10 rendered.nii.gz
```

The range 1 to 10 is used here to decrease the contrast and, therefore, emphasize the overlay image.

To view the output of `overlay`, type

```
fslview rendered.nii.gz
```

The following window will be displayed:



`slices`

To automate the quality assurance of multiple subjects, another FSL tool—`slices` is used. `slices` is a program that creates a 2-dimensional image PNG (Portable Network Graphic) file—a commonly used image file—from a 3-dimensional NifTI file. This is useful for generating slices in multiple planes of a 3-dimensional image file.

The syntax for `slices` is as follows:

```
slices <input> -o <output>
```

`slices` will automatically convert the intensity values of the background image in the input image (which is the output of `overlay`) to color using a look-up table (LUT) with colors from red to yellow (see *`slicer`)*.

Demo: Using `slices` to generate a PNG-formatted quality assurance image of a skull-stripped image

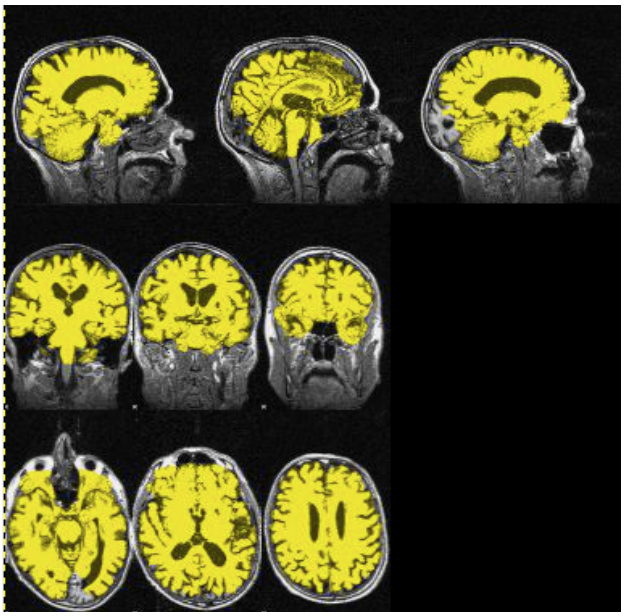
Type the following into your terminal:

```
slices rendered_bet.nii.gz -o foo.png
```

To display the output image, type

```
xdg-open foo.png
```

The following image will be displayed:



---

slicer

There are several circumstances where one may want to generate quality assurance images using different color schemes. So far, `overlay` and `slices` have used a default color range from red to yellow. The default LUT used by FSL in the cases so far is a file named `render1.lut`. This LUT is used as a default by FSL because it is alphabetically listed first in FSL's LUT directory. To specify a specific LUT table, another FSL program—`slicer`—can be used. `slicer` is similar to but more flexible than `slices`. `slicer` uses the following syntax:

```
slicer <input> [-l lut] [-x dim sagittal.png] [-y dim coronal.png] [-z dim axial.png]
```

Explanation of arguments:

<input>: rendered 3D image input

[-l lut]: use the -l flag and the absolute path to desired LUT file. `slicer` will use the specified LUT file to convert the overlay image's intensity values to color. A list of LUT files can be found in:

```
/usr/share/fsl/5.0/etc/luts
```

[-x dim sagittal.png] [-y dim coronal.png] [-z dim axial.png]: three options that specify the dimensions of each slice to be outputted as a PNG file. The -x flag is used to output a sagittal slice, the -y flag is used to output a coronal slice, and the -z flag is used to output an axial slice. `dim` is replaced with a value representative of slice to be outputted. At least one slice must be specified.

Demo: Using `slicer` and a specified LUT to capture a sagittal slice image from a skull-stripped image

Using the -l flag with `slicer`, you can specify a custom LUT (`rendersea.lut`) that uses a blue color scheme. Using the -x flag with 0.5 specifies that you would like to take a middle sagittal slice from `rendered.nii.gz`.

```
slicer rendered.nii.gz -l /usr/share/fsl/5.0/etc/luts/rendersea.lut -x 0.5 foo.png
```

The single output file, `foo.png`, will look like the following:

More on Look-Up Tables:

Look-up tables are used for converting color ranges from one another. For example, look-up tables can be used to convert a grayscale image to a "warm" colored image. Colors red, green, and blue are each stored with an intensity between 0 and 255. Larger ranges of color (in contrast to just three distinct colors) can be created this way by combining the intensity of the red, green, and blue values. A range needs to be specified in overlay to determine the intensity of the color in the output image.[1]

LUT file format:

The first 100 values in a LUT represent the RGB colors mapped to the background image intensities. These colors will be automatically mapped and are grayscale. The next 100 values represent the the colors to be mapped to the intensity values of the first overlay image (this is specified in the `s1_min` and `s1_max` options in `overlay`'s syntax).

For example, typing

```
cat /usr/share/fsl/5.0/etc/rendersea.lut | head -109 | tail -100
```

into your terminal will output a RGB value representation of a grayscale color range. Head and tail are used in the command above to hide the 9 lines of header information. LUT files used by `overlay` and `slicer` will output a range of values like this:

```
<-color{0.000000,0.000000,0.000000}->
<-color{0.010000,0.010000,0.010000}->
<-color{0.020000,0.020000,0.020000}->
<-color{0.030000,0.030000,0.030000}->
<-color{0.040000,0.040000,0.040000}->
                                    …
                  90 more values incrementing by:
              <-color{+0.010000, +0.010000, +0.010000}->
                                    …
<-color{0.950000,0.950000,0.950000}->
<-color{0.960000,0.960000,0.960000}->
```

```
<-color{0.970000,0.970000,0.970000}->
<-color{0.980000,0.980000,0.980000}->
<-color{0.990000,0.990000,0.990000}->
```

Each line represents one RGB value in the following format:

```
<-color{RED, GREEN, BLUE}->
```

There are 256 possible color intensities (0-255) that each RGB element can be represented with. Each RGB element in this specific LUT format represents a proportion of the range each element can take on. For example, `<-color{0.000000,0.000000,0.000000}->` will map an RGB value with all elements at an intensity of 0–"black"–and `<-color{0.990000,0.990000,0.990000}->` will map an RGB value with all elements at an intensity of about 253–"white". The elements do not increment all the way to 1.00000 because the range starts at 0.00000.

View the next 100 values by typing:

```
cat rendersea.lut | tail -105 | head -100
```

into your terminal. Head and tail are used in this instance to hide the extra 5 lines of header information at the bottom of the file. The following text will be outputted:

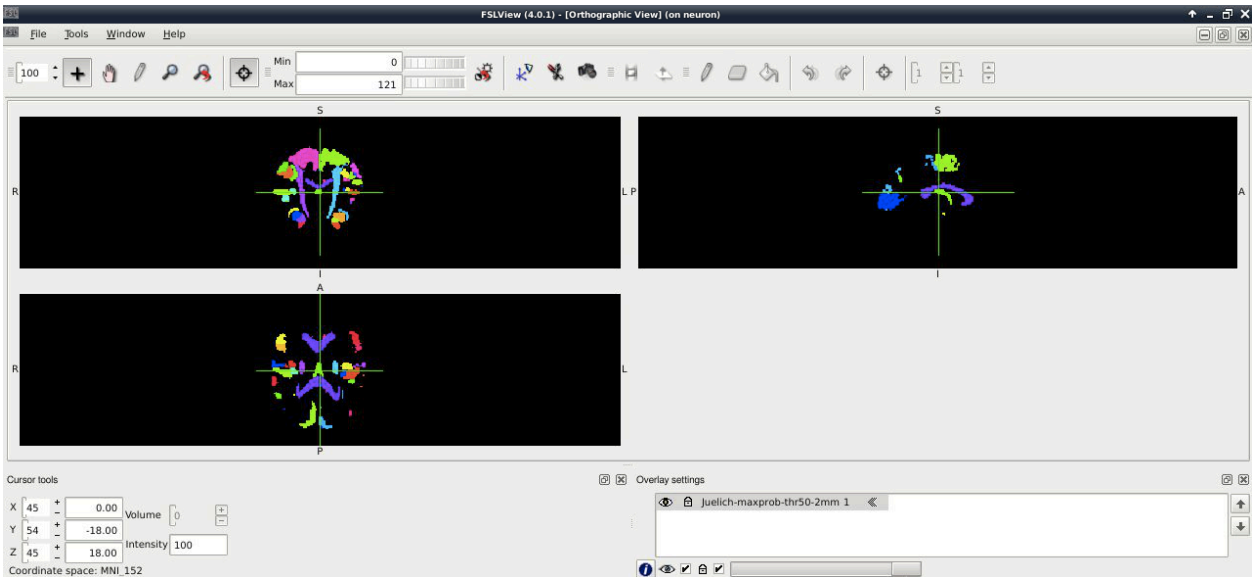
```
<-color{0.000000,0.000000,1.000000}->
<-color{0.005050,0.010101,1.000000}->
<-color{0.010101,0.020202,1.000000}->
<-color{0.015152,0.030303,1.000000}->
<-color{0.020202,0.040404,1.000000}->
                                    …
                  90 values, each incrementing by:
              <-color{+0.005050, +0.010101, +0.000000}->
                                    …
<-color{0.479800,0.959600,1.000000}->
<-color{0.484850,0.969700,1.000000}->
<-color{0.489900,0.979800,1.000000}->
<-color{0.494950,0.989900,1.000000}->
<-color{0.500000,1.000000,1.000000}->
```

Demo: Creating and using a custom LUT

To create an image with multiple anatomical masks, each a different color, we need to create a LUT with distinct color categories. To do this, we will use `fslmaths`–a tool that is used for performing arithmetic operations on NifTI image intensity values.

We will create a file that contains six separate masks–two for the primary motor cortices (right and left), one for the hippocampus, and two for the basolateral amygdalae (right and left). These anatomical masks come from the Juelich atlas. Using `fslview` to

open `/usr/share/fsl/data/atlases/Juelich/Juelich-maxprob-thr50-2mm.nii.gz`, the various masks in the Juelich atlas will be displayed as follows:



The following code will create a single file containing all six masks:

```
MASKDIRECTORY=/var/tmp
ATLAS=/usr/share/fsl/data/atlases/Juelich/Juelich-maxprob-thr50-
2mm.nii.gz

fslmaths ${ATLAS} -thr 9 -uthr 9 ${MASKDIRECTORY}/1amyg.nii.gz

fslmaths ${ATLAS} -thr 10 -uthr 10 ${MASKDIRECTORY}/2amyg.nii.gz

fslmaths ${ATLAS} -thr 48 -uthr 48 ${MASKDIRECTORY}/3mcor.nii.gz

fslmaths ${ATLAS} -thr 47 -uthr 47 ${MASKDIRECTORY}/4mcor.nii.gz

fslmaths ${ATLAS} -thr 95 -uthr 95 ${MASKDIRECTORY}/5hipp.nii.gz

fslmaths ${MASKDIRECTORY}/1amyg.nii.gz -add
${MASKDIRECTORY}/2amyg.nii.gz -add ${MASKDIRECTORY}/3mcor.nii.gz -add
${MASKDIRECTORY}/4mcor.nii.gz -add ${MASKDIRECTORY}/5hipp.nii.gz
${MASKDIRECTORY}/masks.nii.gz
```

Although this code is sufficient for placing the 5 masks into a single image, the intensity values of the masks need to be separated equidistant from one another so that the custom LUT can code these intensity values into different colors. The following code will modify the original intensity values:

```
MASKDIRECTORY=/var/tmp
ATLAS=/usr/share/fsl/data/atlases/Juelich/Juelich-maxprob-thr50-
2mm.nii.gz

fslmaths ${ATLAS} -thr 9 -uthr 9 -add 1 -thr 10 -uthr
10  ${MASKDIRECTORY}/1amyg.nii.gz

fslmaths ${ATLAS} -thr 10 -uthr 10 -add 20 -thr 30 -uthr 30
${MASKDIRECTORY}/2amyg.nii.gz

fslmaths ${ATLAS} -thr 48 -uthr 48 -add 2 -thr 50 -uthr 50
${MASKDIRECTORY}/3mcor.nii.gz

fslmaths ${ATLAS} -thr 47 -uthr 47 -add 23 -thr 70 -uthr 70
${MASKDIRECTORY}/4mcor.nii.gz

fslmaths ${ATLAS} -thr 95 -uthr 95 -sub 5 -thr 90 -uthr 90
${MASKDIRECTORY}/5hipp.nii.gz

fslmaths ${MASKDIRECTORY}/1amyg.nii.gz -add
${MASKDIRECTORY}/2amyg.nii.gz -add ${MASKDIRECTORY}/3mcor.nii.gz -add
${MASKDIRECTORY}/4mcor.nii.gz -add ${MASKDIRECTORY}/5hipp.nii.gz
${MASKDIRECTORY}/masks.nii.gz
```

Opening `masks.nii.gz` with `fslview` will color-code the masks using a default LUT file. However, this will not be adequate for coding the masks in different colors. Therefore, a custom LUT file must be created to code the masks in different colors. The first 9 lines of the this custom LUT are header information. The next 100 lines represent a grayscale color range. The following 100 lines include 20 repetitions for each of the 5 colors used in the LUT. Finally, the last 5 lines are additional header information. The following LUT will code the masks in blue (intensities 1 to 20), yellow (intensities 21 to 40), green (intensities 41 to 60), red (intensities 61 to 80) and purple (intensities 81 to 100):

```
%!VEST-LUT
%%BeginInstance
<<
/SavedInstanceClassName /ClassLUT
/PseudoColorMinimum 0.00
/PseudoColorMaximum 1.00
/PseudoColorMinControl /Low
/PseudoColorMaxControl /High
/PseudoColormap [
<-color{0.000000,0.000000,0.000000}->
<-color{0.010000,0.010000,0.010000}->
<-color{0.020000,0.020000,0.020000}->
<-color{0.030000,0.030000,0.030000}->
<-color{0.040000,0.040000,0.040000}->
<-color{0.050000,0.050000,0.050000}->
<-color{0.060000,0.060000,0.060000}->
```

```
<-color{0.070000,0.070000,0.070000}->
<-color{0.080000,0.080000,0.080000}->
<-color{0.090000,0.090000,0.090000}->
<-color{0.100000,0.100000,0.100000}->
<-color{0.110000,0.110000,0.110000}->
<-color{0.120000,0.120000,0.120000}->
<-color{0.130000,0.130000,0.130000}->
<-color{0.140000,0.140000,0.140000}->
<-color{0.150000,0.150000,0.150000}->
<-color{0.160000,0.160000,0.160000}->
<-color{0.170000,0.170000,0.170000}->
<-color{0.180000,0.180000,0.180000}->
<-color{0.190000,0.190000,0.190000}->
<-color{0.200000,0.200000,0.200000}->
<-color{0.210000,0.210000,0.210000}->
<-color{0.220000,0.220000,0.220000}->
<-color{0.230000,0.230000,0.230000}->
<-color{0.240000,0.240000,0.240000}->
<-color{0.250000,0.250000,0.250000}->
<-color{0.260000,0.260000,0.260000}->
<-color{0.270000,0.270000,0.270000}->
<-color{0.280000,0.280000,0.280000}->
<-color{0.290000,0.290000,0.290000}->
<-color{0.300000,0.300000,0.300000}->
<-color{0.310000,0.310000,0.310000}->
<-color{0.320000,0.320000,0.320000}->
<-color{0.330000,0.330000,0.330000}->
<-color{0.340000,0.340000,0.340000}->
<-color{0.350000,0.350000,0.350000}->
<-color{0.360000,0.360000,0.360000}->
<-color{0.370000,0.370000,0.370000}->
<-color{0.380000,0.380000,0.380000}->
<-color{0.390000,0.390000,0.390000}->
<-color{0.400000,0.400000,0.400000}->
<-color{0.410000,0.410000,0.410000}->
<-color{0.420000,0.420000,0.420000}->
<-color{0.430000,0.430000,0.430000}->
<-color{0.440000,0.440000,0.440000}->
<-color{0.450000,0.450000,0.450000}->
<-color{0.460000,0.460000,0.460000}->
<-color{0.470000,0.470000,0.470000}->
<-color{0.480000,0.480000,0.480000}->
<-color{0.490000,0.490000,0.490000}->
<-color{0.500000,0.500000,0.500000}->
<-color{0.510000,0.510000,0.510000}->
<-color{0.520000,0.520000,0.520000}->
<-color{0.530000,0.530000,0.530000}->
<-color{0.540000,0.540000,0.540000}->
<-color{0.550000,0.550000,0.550000}->
<-color{0.560000,0.560000,0.560000}->
<-color{0.570000,0.570000,0.570000}->
```

```
<-color{0.580000,0.580000,0.580000}->
<-color{0.590000,0.590000,0.590000}->
<-color{0.600000,0.600000,0.600000}->
<-color{0.610000,0.610000,0.610000}->
<-color{0.620000,0.620000,0.620000}->
<-color{0.630000,0.630000,0.630000}->
<-color{0.640000,0.640000,0.640000}->
<-color{0.650000,0.650000,0.650000}->
<-color{0.660000,0.660000,0.660000}->
<-color{0.670000,0.670000,0.670000}->
<-color{0.680000,0.680000,0.680000}->
<-color{0.690000,0.690000,0.690000}->
<-color{0.700000,0.700000,0.700000}->
<-color{0.710000,0.710000,0.710000}->
<-color{0.720000,0.720000,0.720000}->
<-color{0.730000,0.730000,0.730000}->
<-color{0.740000,0.740000,0.740000}->
<-color{0.750000,0.750000,0.750000}->
<-color{0.760000,0.760000,0.760000}->
<-color{0.770000,0.770000,0.770000}->
<-color{0.780000,0.780000,0.780000}->
<-color{0.790000,0.790000,0.790000}->
<-color{0.800000,0.800000,0.800000}->
<-color{0.810000,0.810000,0.810000}->
<-color{0.820000,0.820000,0.820000}->
<-color{0.830000,0.830000,0.830000}->
<-color{0.840000,0.840000,0.840000}->
<-color{0.850000,0.850000,0.850000}->
<-color{0.860000,0.860000,0.860000}->
<-color{0.870000,0.870000,0.870000}->
<-color{0.880000,0.880000,0.880000}->
<-color{0.890000,0.890000,0.890000}->
<-color{0.900000,0.900000,0.900000}->
<-color{0.910000,0.910000,0.910000}->
<-color{0.920000,0.920000,0.920000}->
<-color{0.930000,0.930000,0.930000}->
<-color{0.940000,0.940000,0.940000}->
<-color{0.950000,0.950000,0.950000}->
<-color{0.960000,0.960000,0.960000}->
<-color{0.970000,0.970000,0.970000}->
<-color{0.980000,0.980000,0.980000}->
<-color{0.990000,0.990000,0.990000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
```

```
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{0.000000,0.000000,1.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{1.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
<-color{0.000000,1.000000,0.000000}->
```

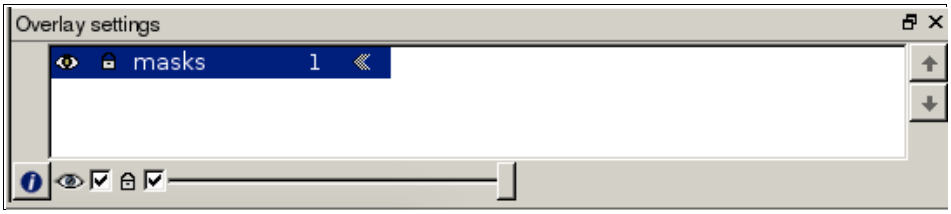
```
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{1.000000,0.000000,0.000000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
<-color{0.500000,0.000000,0.500000}->
]
>>
%%EndInstance
%%EOF
```

Save this custom LUT file in your current working directory with the extension ".lut" to use the file as a LUT in quality assurance images. Use the following overlay command to superimpose the masks over a standard brain:

```
overlay 1 1 /usr/share/fsl/data/standard/MNI152_T1_2mm_brain.nii.gz -
a masks.nii.gz 1 100 renderedMasks.nii.gz
```

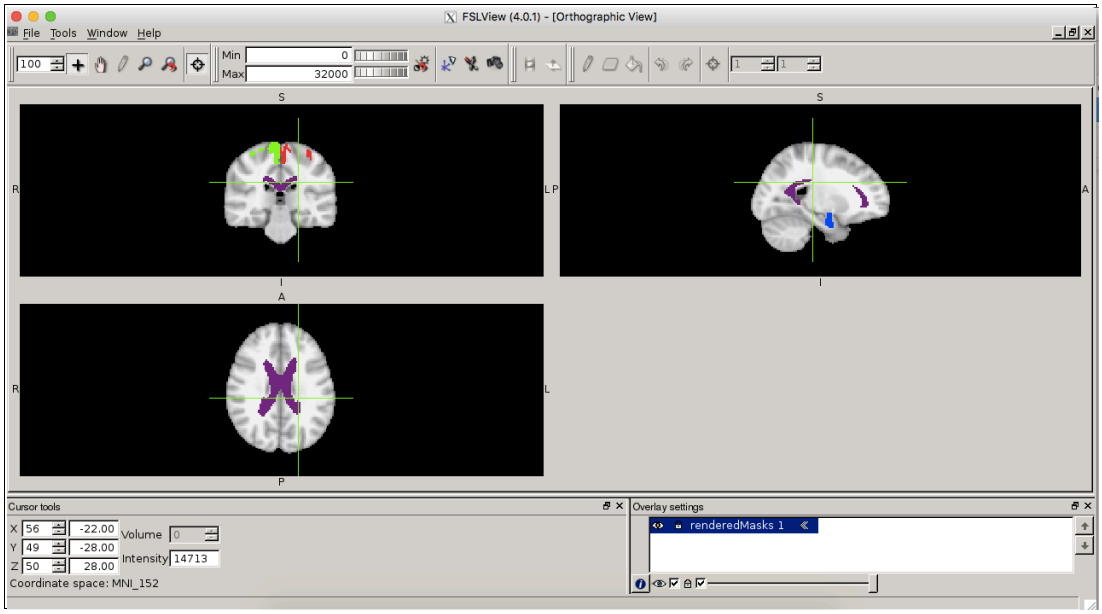
The range 1-100 is used because the intensity values of the masks are between 1 and 100. These values match proportionally to the 100 values of the LUT.

To select a custom LUT table in fslview, select the “i” at the bottom, left-hand corner of the overlay settings.



Then, select the folder icon on the right-hand side of the look-up table options.

Now, just select the custom lut table in your current directory and you should have an image with 5 anatomical masks, of 5 distinct colors, overlying on a standard brain.



Reference:

http://www.upstate.edu/radiology/education/rsna/intro/display.php[1]