

# Visualization of Images

John Muschelli

## MS Lesion

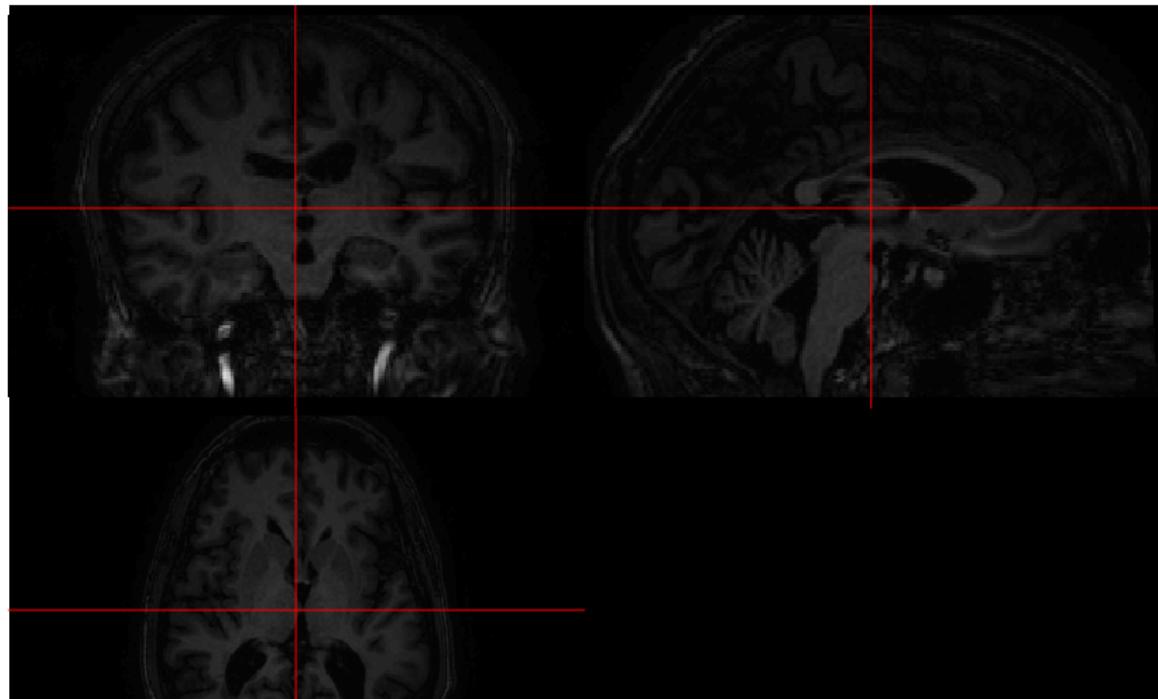
Let's read in the T1 image from a MS lesion data set to visualize:

```
library(ms.lesion)
library(neurobase)
files = get_image_filenames_list_by_subject()$training01
t1_fname = files["MPRAGE"]
t1 = readnii(t1_fname)
```

## Orthographic view

The `oro.nifti::orthographic` function provides great functionality on displaying nifti objects in 3 different planes.

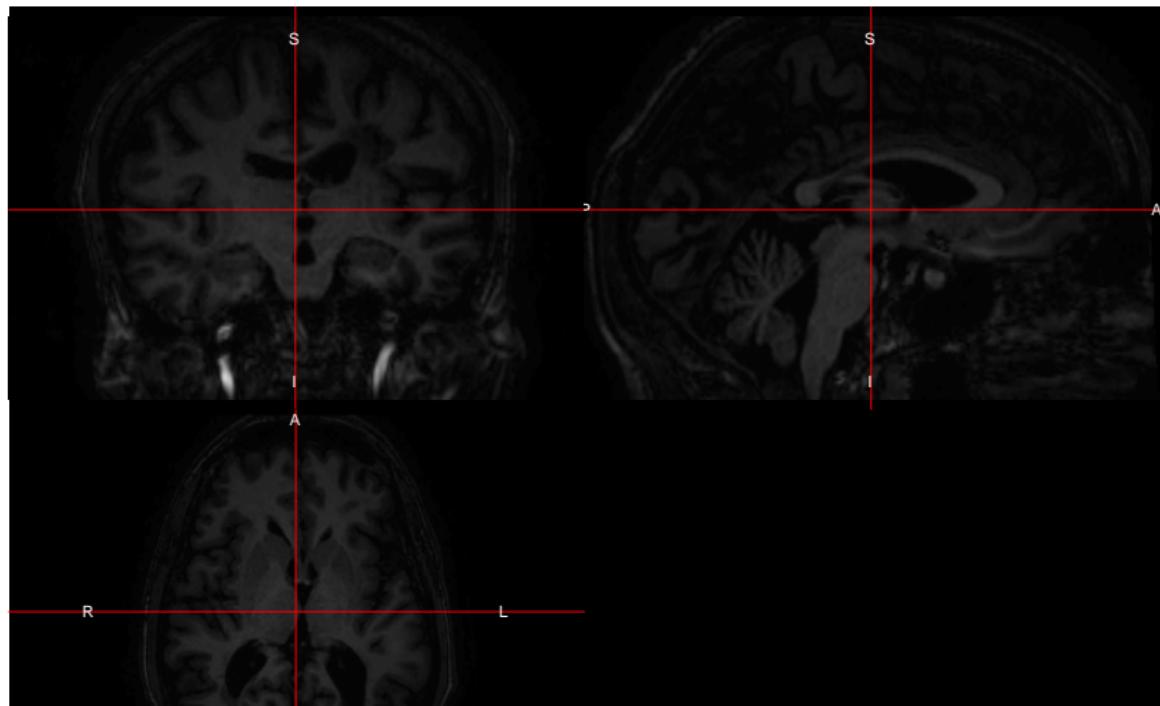
```
oro.nifti::orthographic(t1)
```



## Orthographic view with additions

The `neurobase::ortho2` function expands upon this with some different defaults.

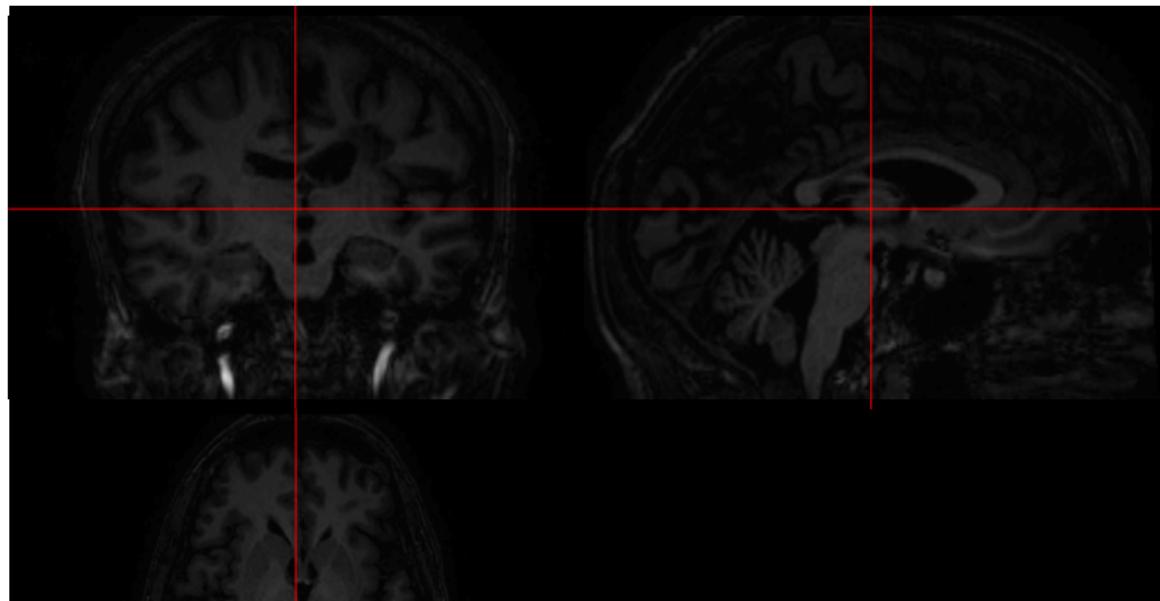
```
neurobase::ortho2(t1)
```



## Orthographic view with additions

We see that in `ortho2` there are annotations of the orientation of the image. Again, if the image was not reoriented, then these many not be correct. You can turn these off with the `add.orient` argument:

```
neurobase::ortho2(t1, add.orient = FALSE)
```



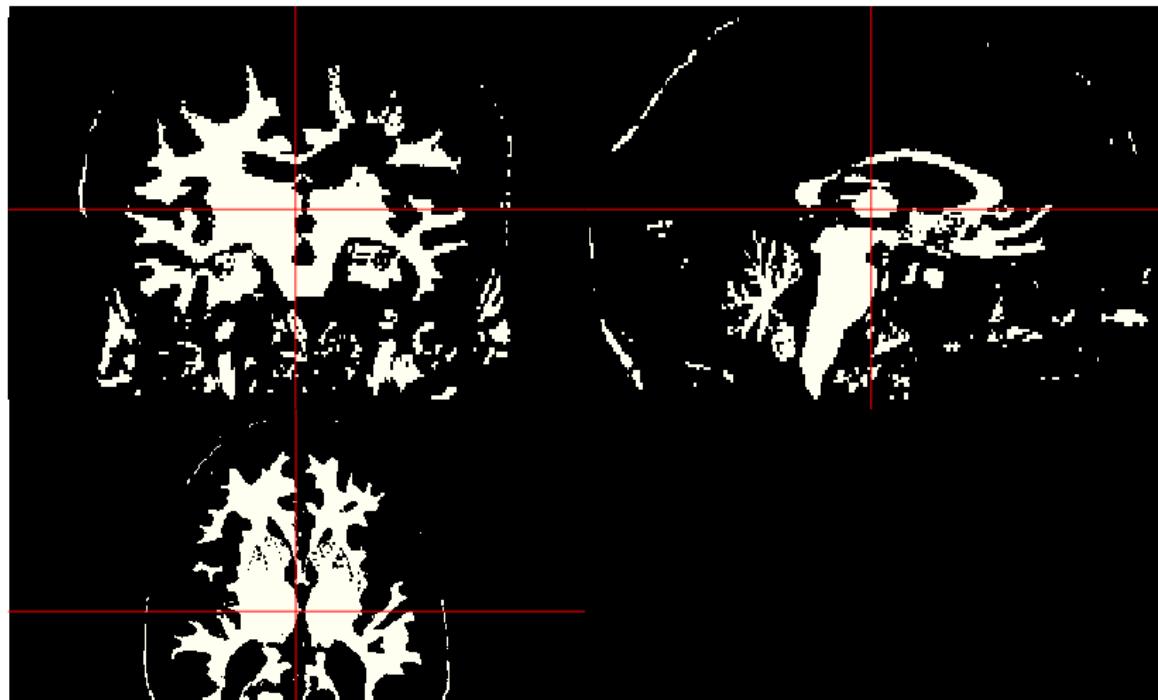
## Differences between orthographic and ortho2

The above code does not fully illustrate the differences between orthographic and ortho2. One marked difference is when you would like to “overlay” an image on top of another in an orthographic view.

## Differences between orthographic and ortho2

Here we will highlight voxels greater than the 90th quantile of the image:

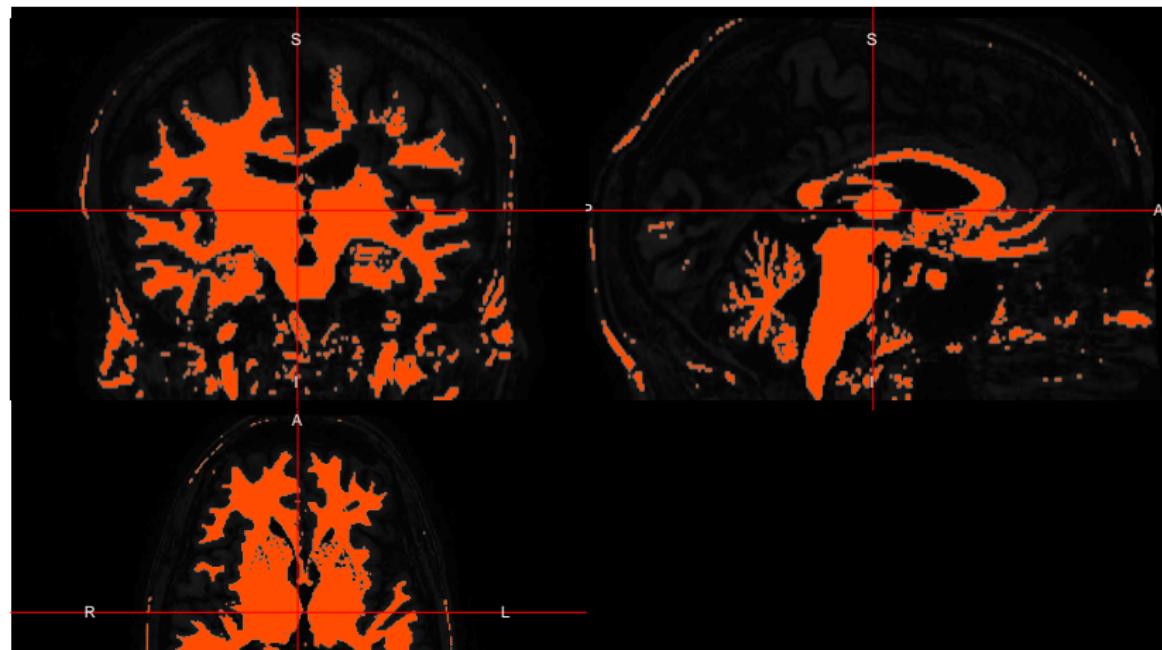
```
orthographic(t1, y = t1 > quantile(t1, 0.9))
```



## Differences between orthographic and ortho2

We see that the white matter is represented here, but we would like to see areas of the brain that are not over this quantile to be shown as the image. Let us contrast this with:

```
ortho2(t1, y = t1 > quantile(t1, 0.9))
```



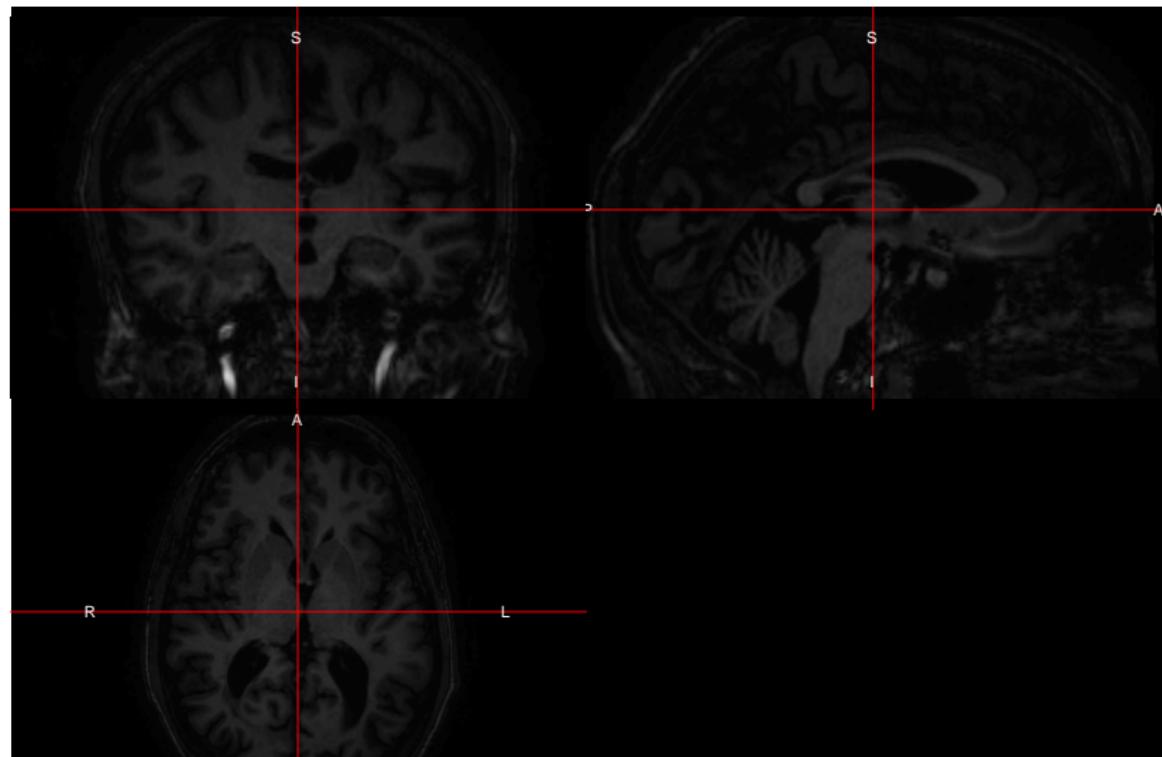
## Differences between orthographic and ortho2

We see the image where the mask is 0 shows the original image. This is due to the NA.y argument in ortho2. The ortho2 (and orthographic) function is based on the graphics::image function in R, as well as many other functions we will discuss below. When graphics::image sees an NA value, it does not plot anything there. The NA.y argument in ortho2 makes it so any values in the y argument (in this case the mask) that are equal to zero are turned to NA.

## Bright values

Large intensities can “dampen” the viewing of an image.

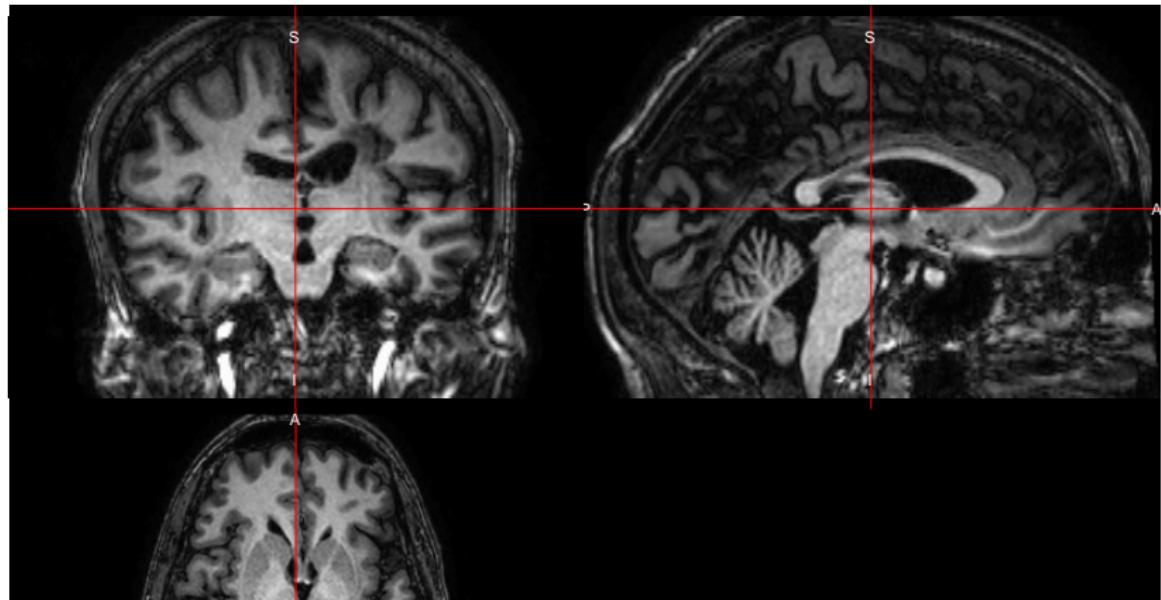
`ortho2(t1)`



## Bright values

We see a faint outline of the image, but this single large value affects how we view the image. The function `robust_window` calculates quantiles of an image, by default the 0 (min) and 99.9th quantile, and sets values outside of this range to that quantile.

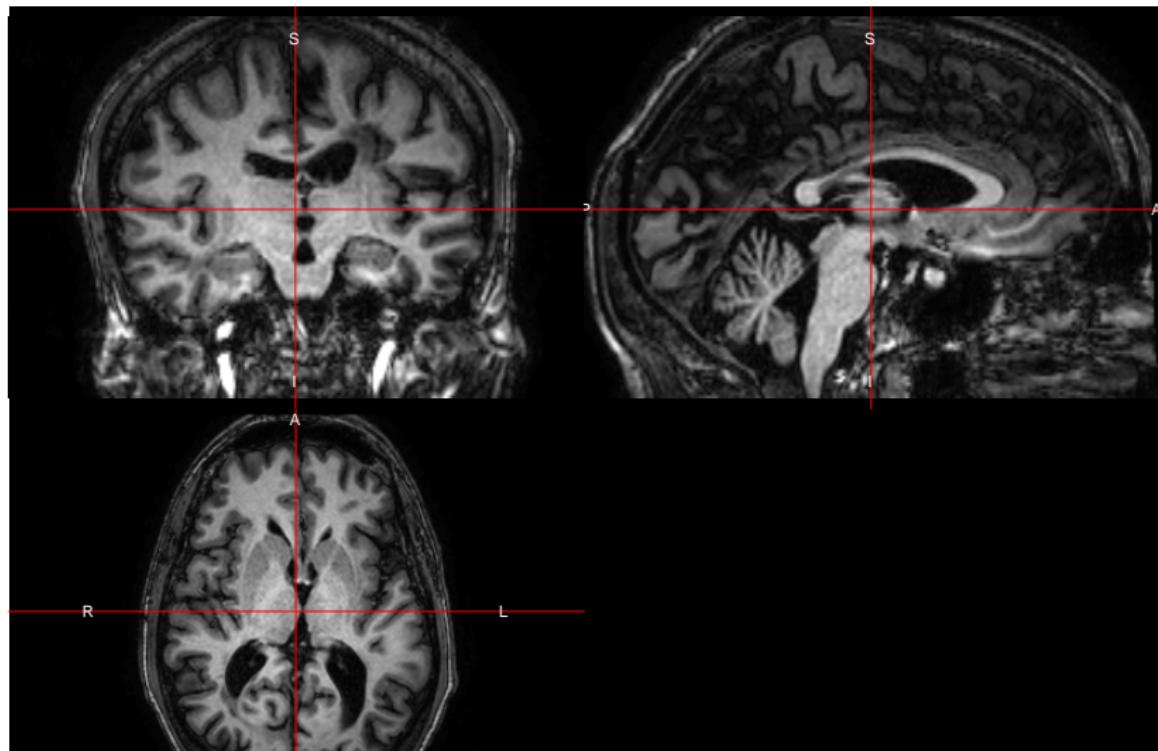
```
ortho2(robust_window(t1))
```



## Changing the Windowing

The `zlim` option can also map intensities that can be plotted:

```
ortho2(t1, zlim = quantile(t1, probs = c(0, 0.999)))
```



## Double orthographic view

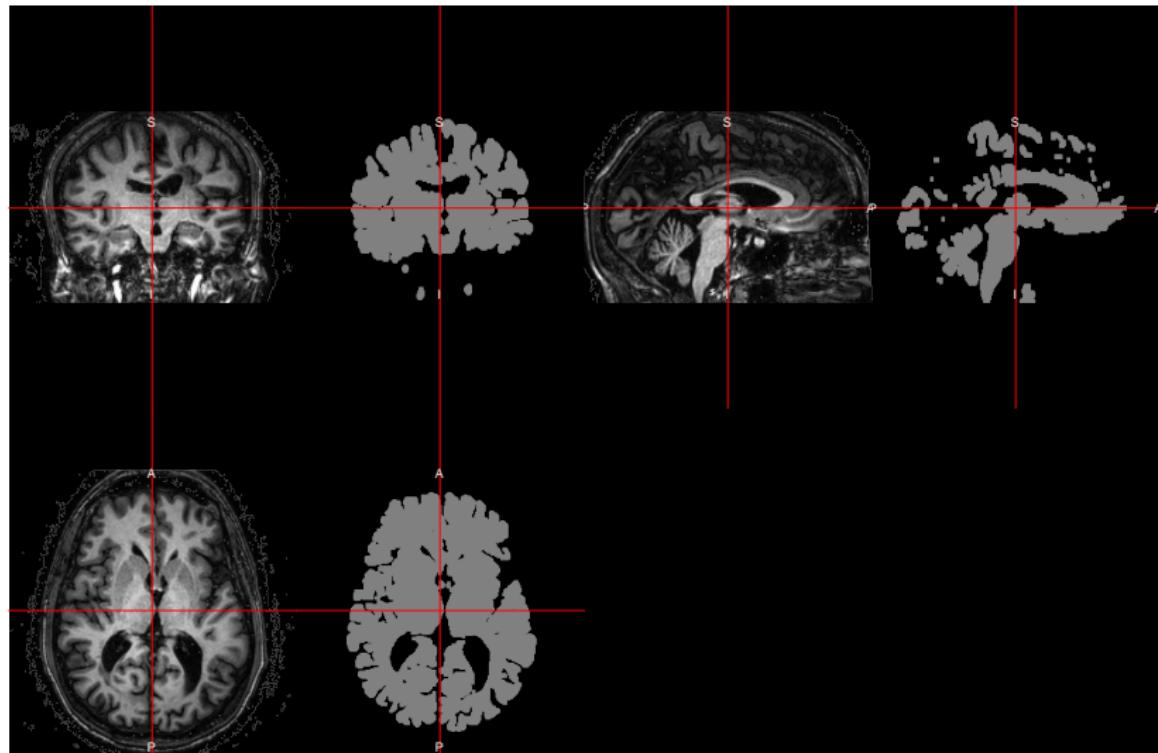
Sometimes you would like to represent 2 images side by side, of the same dimensions and orientation of course. The `double_ortho` function allows you to do this. Let's get a coarse mask of the image:

```
mask = oMask(t1)
```

## Double time

We can view the original T1 alongside the brain-extracted image:

```
double_ortho(t1, mask)
```



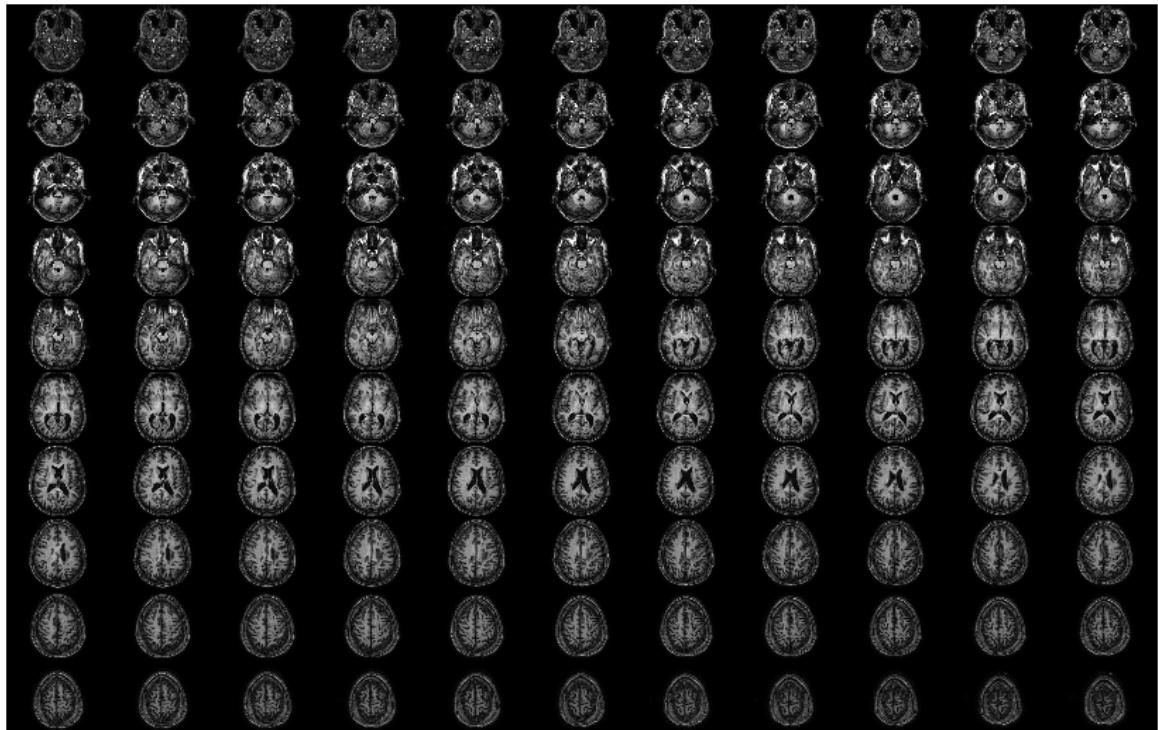
## Single slice view

We may want to view a single slice of an image. The `oro.nifti::image` function can be used here. Note, `graphics::image` exists and `oro.nifti::image` both exist. The `oro.nifti::image` allows you to just write `image(nifti_object)`, which performs operations and calls functions using `graphics::image`. This allows the user to use a “generic” version of `image`, which `oro.nifti` adapted specifically for `nifti` objects. You can see the help for this function in `?image.nifti`.

# Plotter

Let's plot an image of the 80th slice of the T1 image:

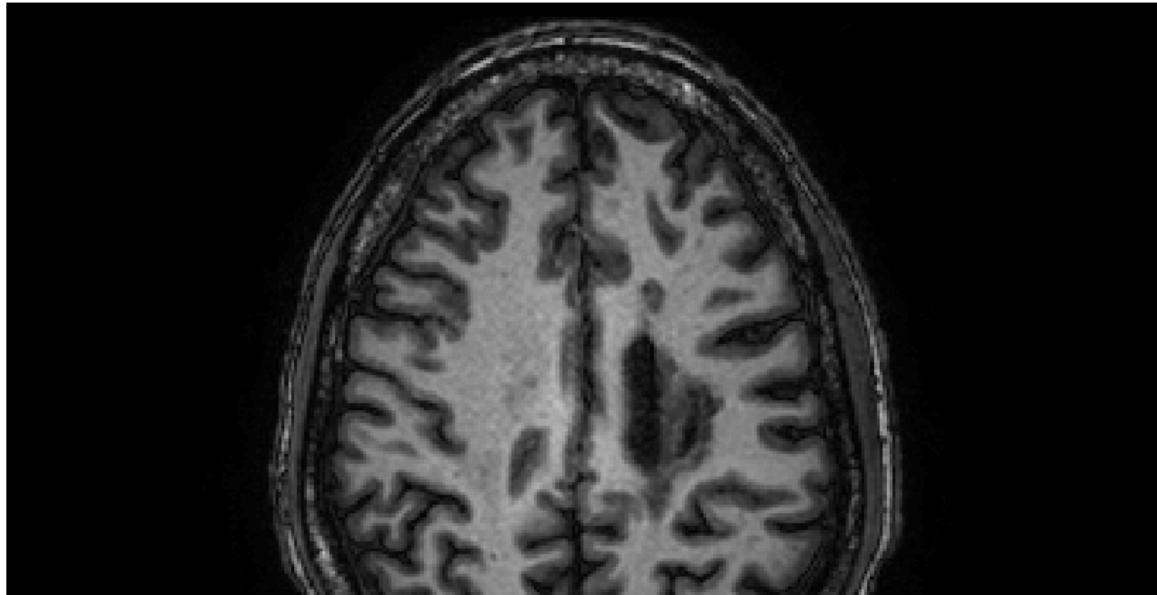
```
image(t1, z = 80)
```



## Bad Plot

What happened? Well, the default argument `plot.type` in `image.nifti` is set for "multiple", so that even if you specify a slice, it will plot **all** slices. Here, if we pass `plot.type = "single"`, we get the single slice we want.

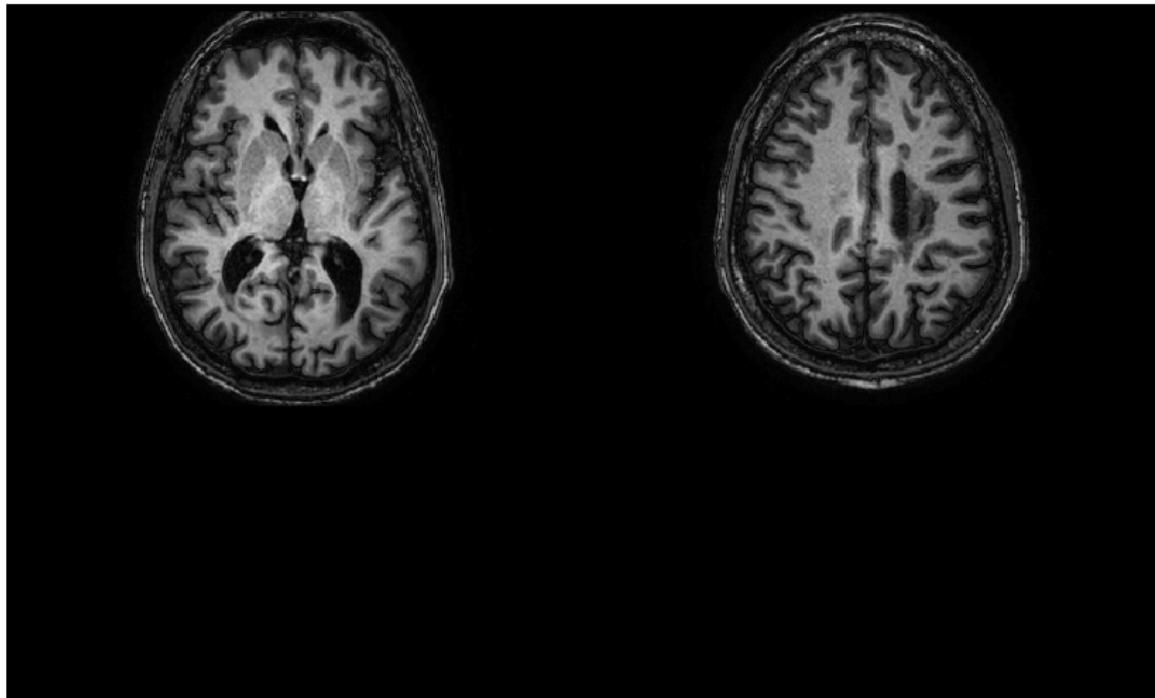
```
image(t1, z = 80, plot.type = "single")
```



## Multiple Slices but not All

If we put multiple slices with `plot.type = "single"`, then we will get a view of these 2 slices:

```
image(t1, z = c(60, 80), plot.type = "single")
```



## Overlaying slices

We can also overlay one slice of an image upon another using the `oro.nifti::overlay` function. Here we must specify `plot.type` again for only one slice.

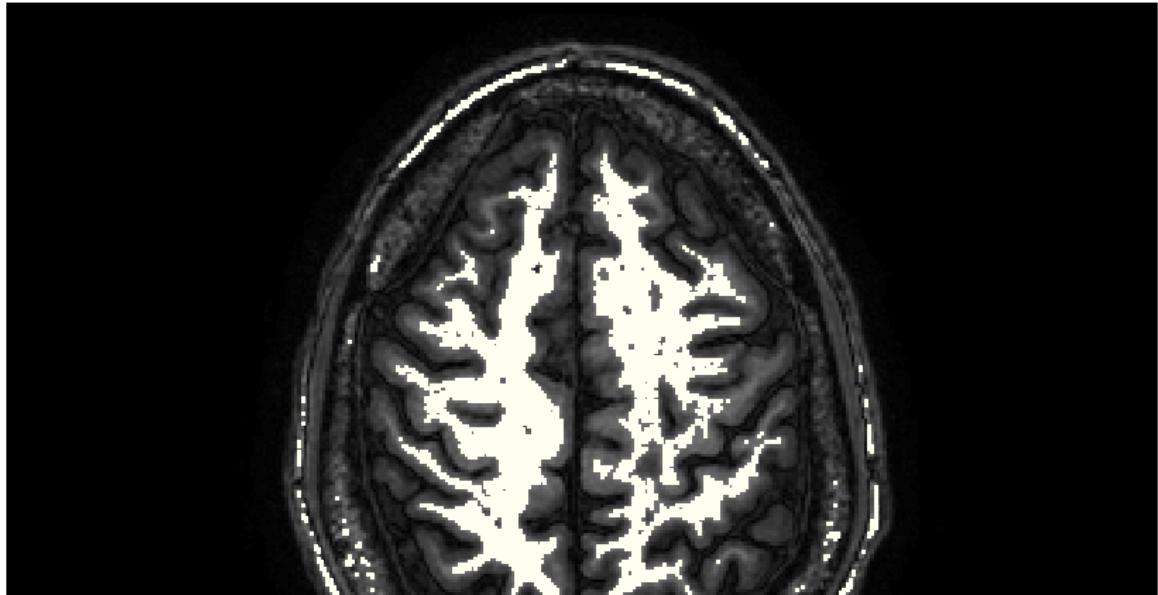
```
overlay(t1, y = t1 > quantile(t1, 0.9), z = 80, plot.type =
```



## Overlay2

We have not yet implemented `overlay2` (at the time of running this), which has the `NA.y` option, but will in the future. We can do this prior to plotting and pass in this `NA`'d mask:

```
mask = t1 > quantile(t1, 0.9); mask[ mask == 0 ] = NA  
overlay(t1, y = mask, z = 90, plot.type = "single")
```



## Dropping empty dimensions

In some instances, there are extraneous slices to an image. For example, in the Eve template image we read in, it is just the brain. Areas of the skull and extracranial tissue are removed, but the slices remain so that the brain image and the original image are in the same space with the same dimensions. For plotting or further analyses, we can drop these empty dimensions using the `neurobase::dropEmptyImageDimensions` function or `drop_empty_dim` shorthand function.

By default, if one `nifti` is passed to the function and `keep_ind = FALSE`, then the return is a `nifti` object.

```
reduced = dropEmptyImageDimensions(t1)
dim(t1)
```

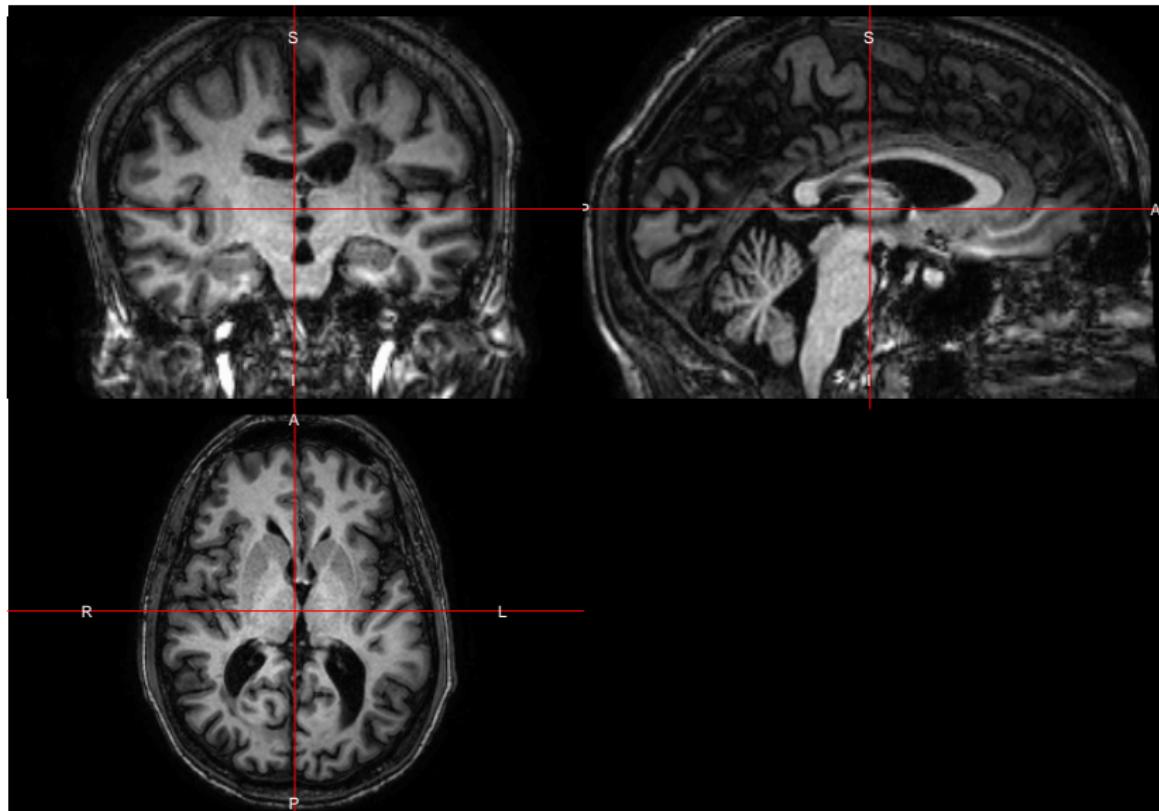
```
## [1] 256 256 120
```

```
dim(reduced)
```

```
## [1] 256 256 120
```

# Plotting the Reduced Image

ortho2(reduced)



## Vectorizing a nifti

To convert a nifti to a vector, you can simply use the `c()` operator:

```
vals = c(t1)  
class(vals)
```

```
## [1] "numeric"
```

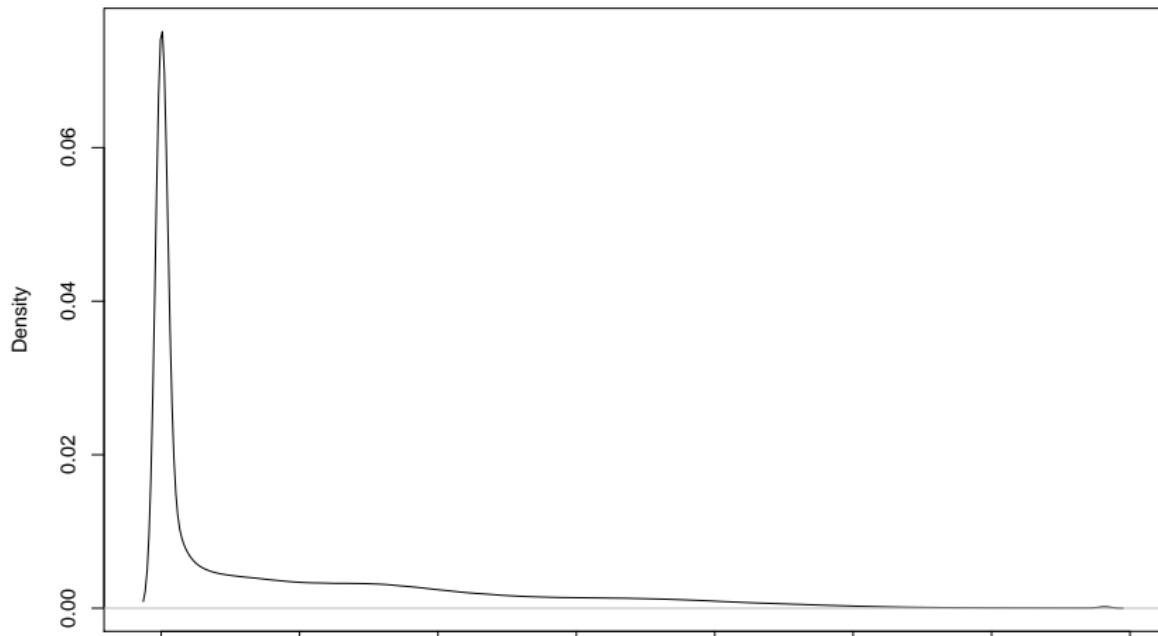
Note an array can be reconstructed by using `array(vals, dim = dim(eve))` and will be in the correct order as the way R creates vectors and arrays.

## Histogram

From these values we can do all the standard plotting/manipulations of data. For example, let's do a marginal density of the values:

```
plot(density(vals))
```

```
density.default(x = vals)
```



## Histogram

You can also pass in a mask to most standard functions:

```
plot(density(t1, mask = t1 > 0))
```

`density.default(x = x)`

