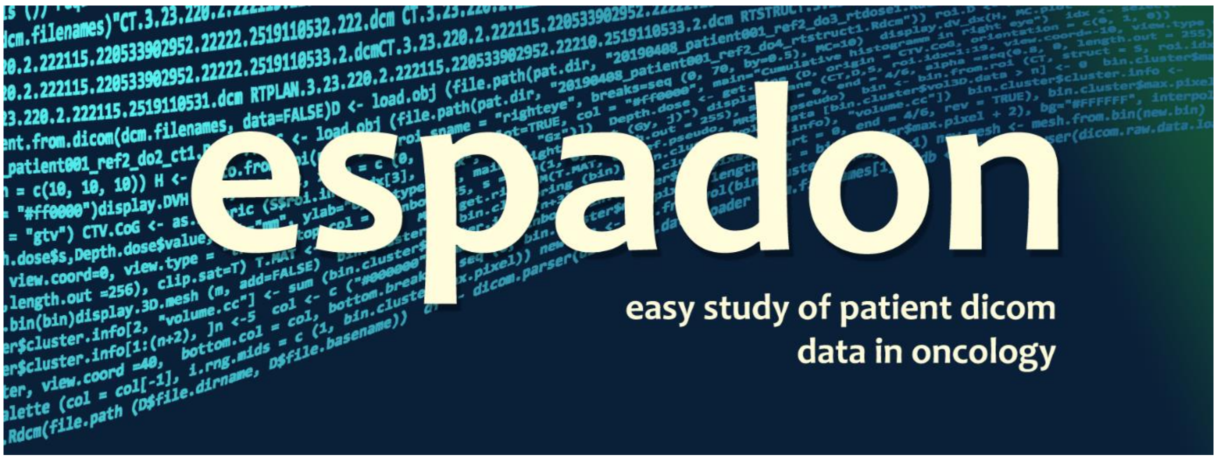


2D representation

C. Fontbonne and J.-M. Fontbonne | LPC Caen
Normandie Univ, ENSICAEN, UNICAEN, CNRS/IN2P3, LPC Caen, 14000 Caen, France
Contact.espadon@lpccaen.in2p3.fr
https://espadon.cnrs.fr/

Table of content

1 Introduction. What we will learn?

This vignette is devoted to 2D representation in espadon. Espadon is not intended to replace any TPS, but it is often useful to have independent tools to display images, dose and structures to check data and to prepare research activities. We already mentioned the display.kplane function in the geometry vignette. It is used for displaying acquired images in the **cut plane** frame of reference (FoR) (with lot of options, see the reference manual). Espadon incorporates a more powerful function for displaying images in the **patient** frame of reference: `display.plane`. This instruction incorporates lot of options, including color palettes, overlays, automatic computation on frame of reference, and representation of structures. In fact, this vignette will focus almost on the `display.plane` instruction and its usage.

In order to run the examples, you should have at hand a CT, a MR (and its Reg file, connecting it to the CT), a rt-Dose and a rt-Struct files. In our case, data were previously converted to Rdcm, and we selected the appropriate files:

```
rm (list=ls ())
require (espadon)

pat.dir <- "E:\\Rdicom data\\patient001"

pat <- load.patient(pat.dir)
View (pat)

D <- load.obj.data(pat$rtdose[[1]])
CT <- load.obj.data(pat$ct[[1]])
S <- load.obj.data(pat$rtstruct[[1]])
MR <- load.obj.data(pat$mr[[1]])
T.MAT <- pat$T.MAT   # or T.MAT <- load.T.MAT(pat.dir)
```

In this example, CT, dose and structures are in `CT$ref.pseudo = "ref8"`. MR is in `MR$ref.pseudo = "ref6"`.

The first section will deal with basic usage of `display.plane` and how to fine-tune a representation. The second section will explore advanced functionalities using overlays of dose, MR or binaries (more or less transparent) and structures.

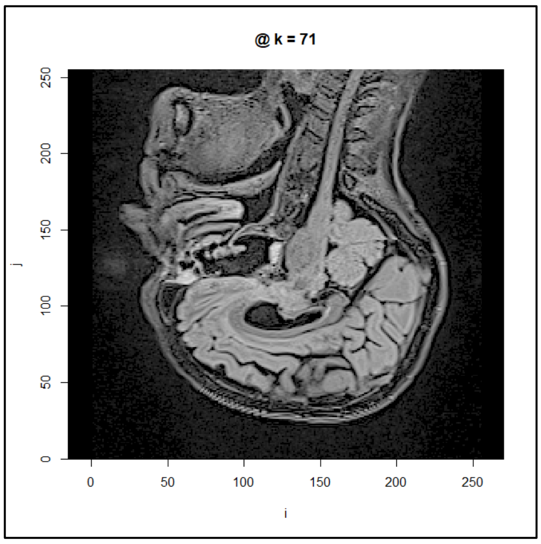
2 Basic 2D representations

2.1 As usually, a story of Frame of Reference…

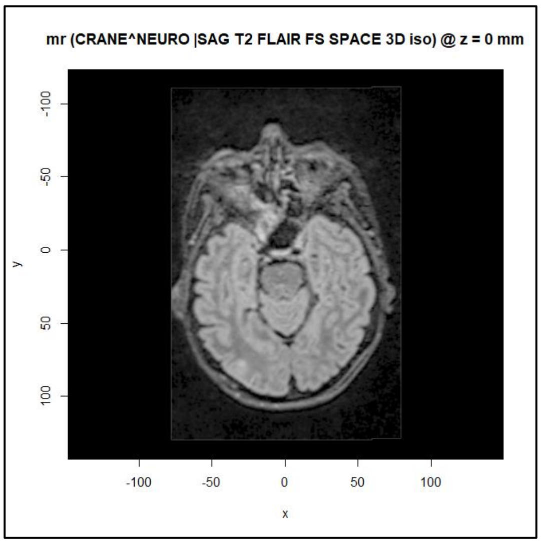
Frames of Reference are handled a user-friendly way in espadon. The following example illustrates this simplicity:

```
dev.new (width = 7, height = 5, noRStudioGD = TRUE)
display.kplane (MR) # top-left
dev.new (width = 7, height = 5, noRStudioGD = TRUE)
display.plane (MR)  # top-right
dev.new (width = 7, height = 5, noRStudioGD = TRUE)
display.plane (MR, display.ref= CT$ref.pseudo, T.MAT = T.MAT)              # bottom-left
dev.new (width = 7, height = 5, noRStudioGD = TRUE)
```

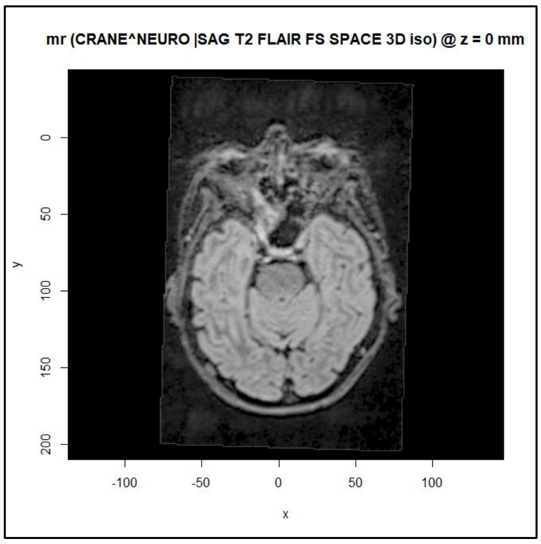
```
display.plane (CT, abs.rng = c(-120, 120), ord.rng = c(-40, 205))    # bottom-right
```



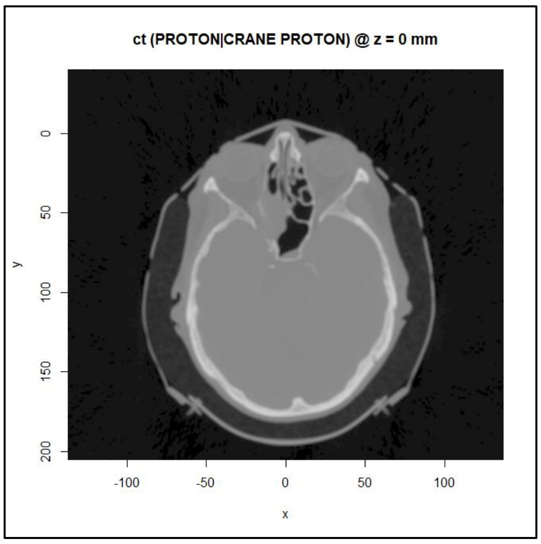
Original MR image



MR image in MR patient FoR



MR image in CT patient FoR



CT image

The top-left image comes from `display.kplane`. It is the exact display of the acquired image in the frame of reference of the MR cut planes. Note that, in this case,i and j coordinates are just the indices of the pixels scaled by the pixel size. They are not directly related to the frame of reference coordinates.

The top-right image is the MR displayed in the `MR$ref.pseudo = "ref6"` frame of reference (corresponding to the MR equipment), using `display.plane`. By default, in espadon, it is a transverse view. This time, x and y coordinates are related to space (expressed in millimeters) and by default espadon selected the $z = 0$ coordinate.

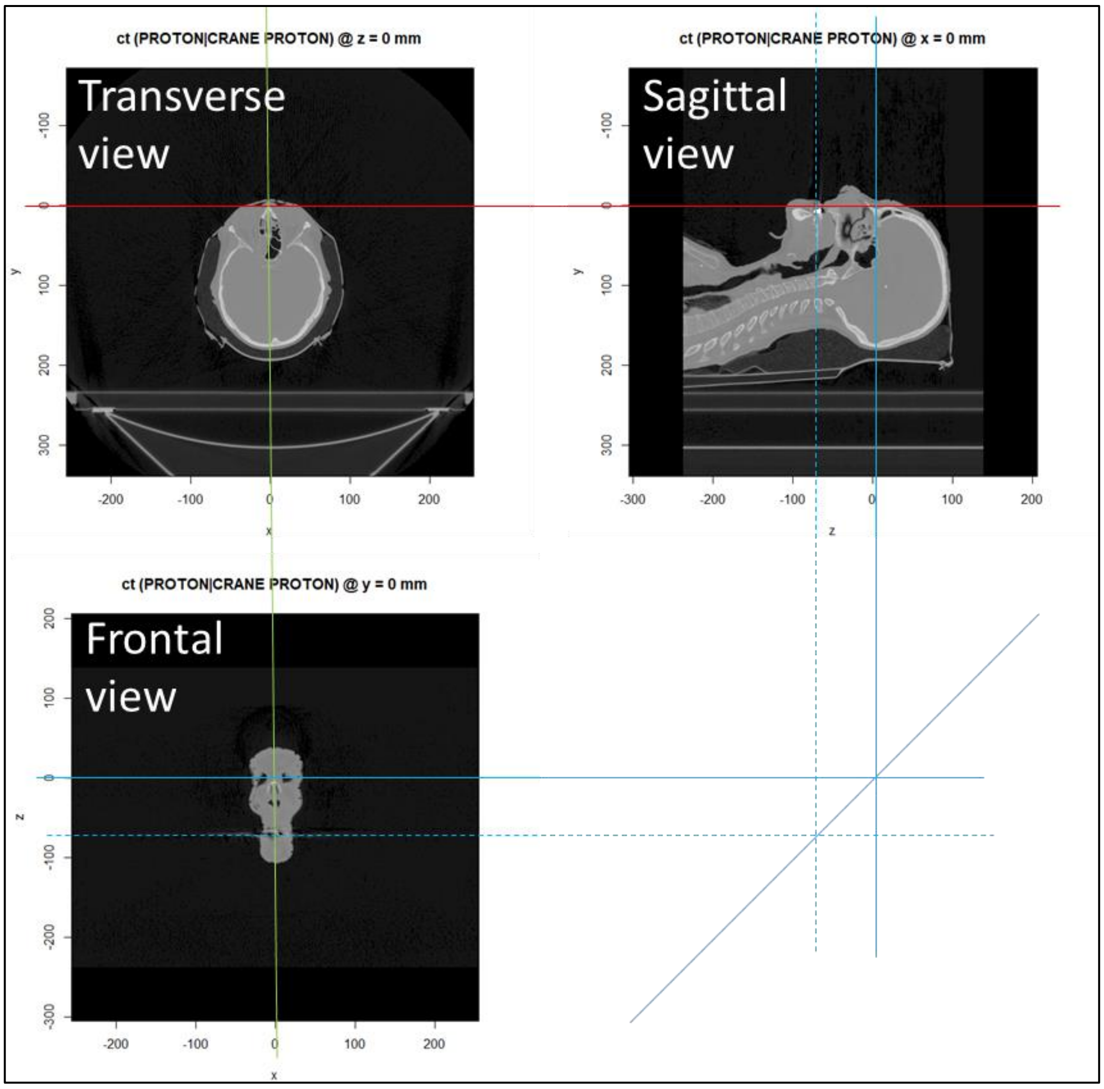
The bottom-left image is the MR displayed in `CT$ref.pseudo = "ref8"`, the frame of reference for the treatment. As the head of the patient is not in the same position in the MR machine and in the treatment machine, this MR was registered and aligned with the CT. We can see that the $z = 0$ plane (displayed by default) is not the same (due to change of frame of reference) and the registration slightly tilted the image clockwise (due to patient head position). In order to display an image in a specific FoR, your just have to provide the name of the target FoR using the `display.ref` option. The `display.ref` parameter also needs the registration list. It is the role of the `T.MAT=T.MAT` option.

Last, the bottom-right image is the CT (in its own frame of reference). Here, the patient was equipped with a contention mask. As the size of the CT is not the same as for the MR, we requested espadon to restrict the display for having (about) the same view. This is the role of `abs.rng` and `ord.rng` options. They need respectively the (left, right) and (top, bottom) coordinates. Note that by default (like in any TPS), in the **transverse view**, $x < 0$ is on the **left of the image**. It **corresponds to the right hand of the patient.** The patient's slice is seen from feet to head. Beware that, in this view, the top of the image has always a lower ordinate than the bottom of the image. No check is made about coordinates ordering, and espadon will do exactly what you requested…

2.2 Views and navigation

It is often useful to switch views from transverse, to sagittal or front:

```
display.plane (CT, view.type = "trans")   # top-left
display.plane (CT, view.type = "sagi")    # top-right
display.plane (CT, view.type = "front")   # bottom-left
```


ct (PROTON|CRANE PROTON) @ z = 0 mm
Transverse view
ct (PROTON|CRANE PROTON) @ x = 0 mm
Sagittal view
ct (PROTON|CRANE PROTON) @ y = 0 mm
Frontal view
x
y
z

Transverse, sagittal and frontal view of a CT

The transverse view is set by default (no need to express the view-type option). Note that, abscissa and ordinates are automatically adjusted for having the right representation. The patient is seen:

- From feet to head in transverse view
- From its left side in sagittal view
- Facing you in front view

As mentioned earlier, when needed, it is possible to zoom on specific places of the image using `abs.rng` and `ord.rng`. Abscissa and ordinates refer to the requested representation. **A simple rule of thumb, in order to preserve image orientation: when specifying a range, the first element is ALWAYS lower than the second element**.

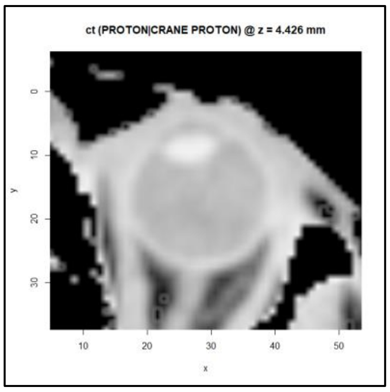
Navigating along the third axis of an image is possible using the `view.coord` option. For instance, by default in a transverse section, the third axis is along z coordinate, at $z = 0$. If one want to see several sections, at different heights, and save them in a pdf file for documentation:

```
pdf ("my_patient.pdf")
display.plane (CT, view.coord = seq (-50, 50, by=5))
dev.off ()
```

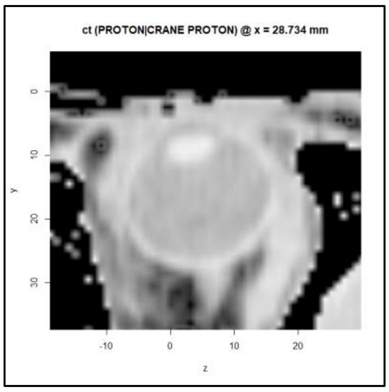
Note that `view.coord` is vectored meaning you can get several images just with one call to the function.

In the following example, we get a zoom (extended by 10mm) of the three views centered on the left eye ("oeil gauche" in French, stored as "oeilg" in the rt-Struct) using the fact that structure file gives us information about its location and extension:

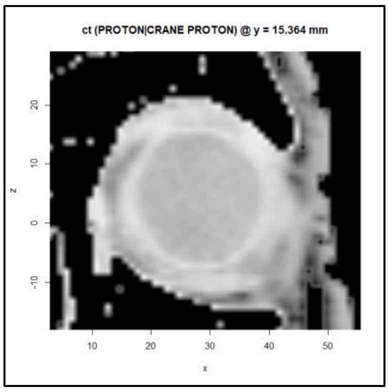
```
idx <- which (S$roi.info$roi.pseudo=="oeilg")
x.rng <- as.numeric (S$roi.info[idx, c("min.x", "max.x")]) + c(-10, 10)
y.rng <- as.numeric (S$roi.info[idx, c("min.y", "max.y")]) + c(-10, 10)
z.rng <- as.numeric (S$roi.info[idx, c("min.z", "max.z")]) + c(-10, 10)
G <- as.numeric (S$roi.info[idx, c("Gx", "Gy", "Gz")])
display.plane (CT, view.type = "trans", bottom.breaks = seq (-100, 100, length.out=256),
               abs.rng=x.rng, ord.rng=y.rng, view.coord = G[3])
display.plane (CT, view.type = "sagi", bottom.breaks = seq (-100, 100, length.out=256),
               abs.rng=z.rng, ord.rng=y.rng, view.coord = G[1])
display.plane (CT, view.type = "front", bottom.breaks = seq (-100, 100, length.out=256),
               abs.rng=x.rng, ord.rng=z.rng, view.coord = G[2])
```



Transverse view



Sagittal view



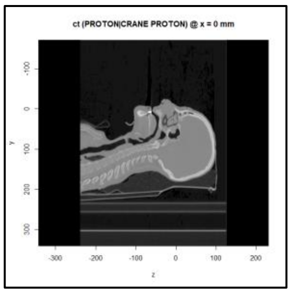
Front view

The first line gives the index of the left eye in the rt-Struct file. Then we get the box enclosing the eye in each direction. Note the use of `as.numeric`: as `roi.info` is a dataframe, data are in fact stored in a list, `as.numeric` converts them into a numeric vector. The next line gets the center of gravity. Then all these information are gathered for selecting abscissa and ordinates ranges, and finally the view coordinate (last argument).

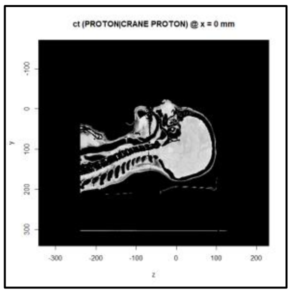
2.3 Contrast, background and saturation

We did not mentioned it, but in fact, `display.plane` uses several layers to display images. We will explore this feature later, but for now, we just worked on the bottom plane and we will see how to adjust background, contrast and colors of this plane:

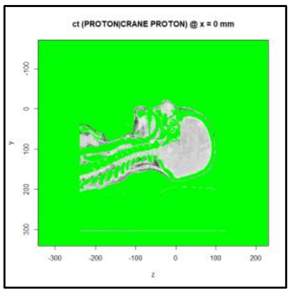
```
display.plane (CT, view.type = "sagi")                                    # 1-left
display.plane (CT, view.type = "sagi",
               bottom.breaks = seq (-100, 100, length.out = 256))         # 2
display.plane (CT, view.type = "sagi",
               bottom.breaks = seq (-100, 100, length.out = 256), bg="green")       # 3
display.plane (CT, view.type = "sagi",
               bottom.breaks = seq (-100, 100, length.out = 256), bg="green",
               sat.transp = TRUE)                                                     # 4-right
```



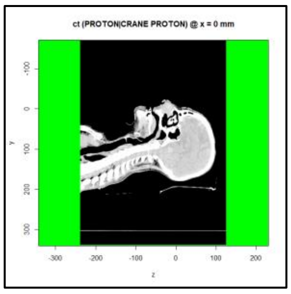
#1



#2



#3



#4

First image is the original CT, poorly contrasted as black is -1000 Hounsfield units (HU) and white, probably over 1000 HU.

As the CT is expressed in HU, we know that fat is about in the range [-100,0] HU and soft tissue in about [0,100] HU, using the `bottom.breaks` option, we restricted the black and white range in [-100,100] HU. By default, `display.plane` uses 255 colors, that is why we specified 256 breaks using `length.out=256` in the second picture.

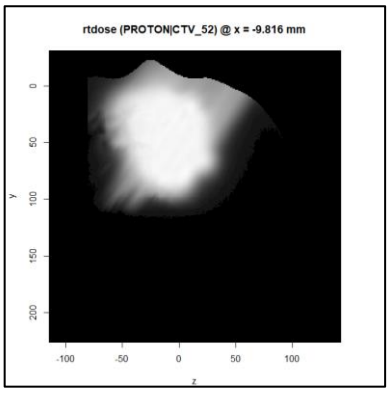
In the third image we requested a green background using bg=”green”. It illustrates that, image areas bellow or above the requested breaks are left transparent (on purpose, in order to detect them, in fact).

If you do not want to see these areas, you can use the `sat.transp = TRUE` option (figure 4) that set pixel values above the range at the top range value, and pixel values under the range at the minimum range value. The remaining green bands show you that the CT extension was not the same on x and y axes. If you do not want these bands to appear, just set the background to black! Note the CT artifact due to teeth implants at $x = -70$. Its appearing depends on your choice…

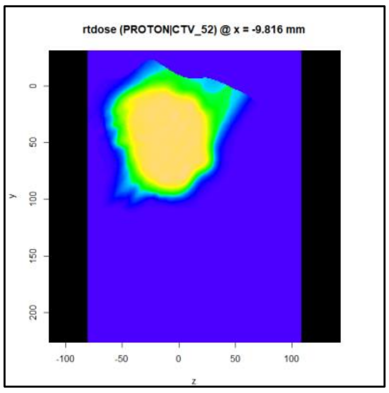
2.4 Color palettes

Color palettes are easy to create and adjust using the `bottom.col` option. By default, `bottom.col = grey.colors (255, start = 0, end = 1)`, meaning we go from black to white. You can use any kind of R color palette, provided the number of colors you use in `bottom.col` is one unit lower the number of breaks you define in `bottom.breaks`. Some examples:

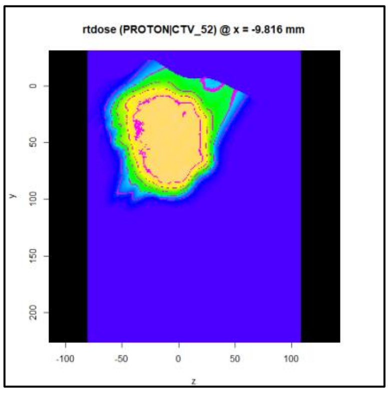
```
idx <- which (S$roi.info$roi.pseudo=="ptv")
G.x <- S$roi.info[idx, c("Gx")]
display.plane (D, view.type = "sagi", view.coord = G.x)      # 1-left
display.plane (D, view.type = "sagi", view.coord = G.x,
               bottom.breaks = seq (0, 60, length.out=61),
               bottom.col = topo.colors (60))                # 2-middle

my.cols <- topo.colors (60)
my.cols[seq (10, 60, by=10)] <- "magenta"
display.plane (D, view.type = "sagi", view.coord = G.x,
               bottom.breaks = seq (0, 60, length.out=61),
               bottom.col = my.cols)                          # 3-right
```



#1



#2

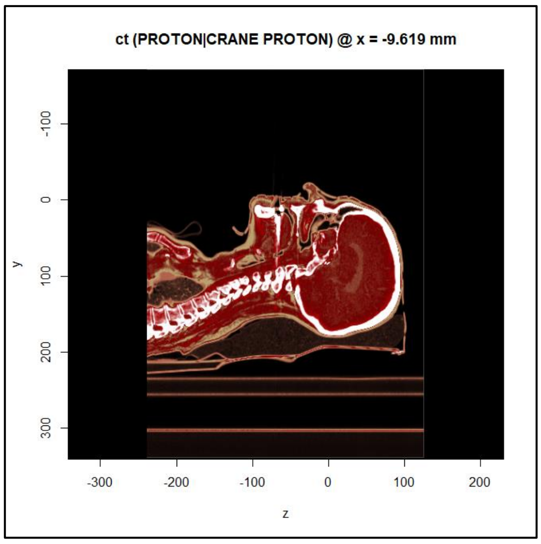


#3

Image on the left is simply the dose in black and white, which is not very expressive. In the middle, the dose using topographic colors. Yellow is now the maximum dose, and green, about the 50% isodose. In the right, we can also define and use homemade palettes; here for instance, we colored every multiple of 10 Gy in magenta to have a fast isodose representation.

Last but not least, espadon incorporates the wonderful Realistic Volume Visualization palette developed by Silverstein, Parsad and Tsirline[1]:

```
display.plane (CT, view.type = "sagi", view.coord = G.x,
               bottom.breaks = seq (-1000, 1000, length.out=256),
               bottom.col = pal.RVV(255), sat.transp = TRUE)
```



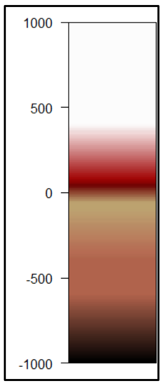
Sagittal view, with palette RVV

[1] Jonathan C. Silverstein, Nigel M. Parsad, Victor Tsirline, "*Automatic perceptual color map generation for realistic volume visualization*", Journal of Biomedical Informatics 41 (2008) 927–935

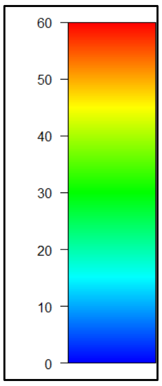
In this palette (more to follow), soft tissue are colored in red levels, fat, in marron levels, bones in white. It is very impressive for most tissue, except maybe the brain, as it is interpreted like “normal” soft tissue. Note that this palette is explicitly defined for CT expressed in Hounsfield Units. It must be used in the range [-1000, 1000] HU (exactly as in the code above). It would have no meaning if used with a MR, for instance.

The problem, when playing with color palettes, is that colored images should be accompanied with the color levels. Espadon incorporates a little tool to create color scales that can be copied/pasted to produce documents:

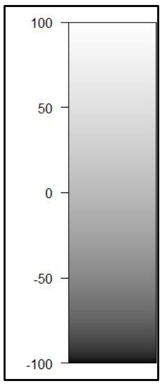
```
# color scale in RVV palette
display.palette (pal.RVV (256), c(-1000, 1000))                          # 1

# a palette for dose, for instance
display.palette (rainbow(256, start = 0, end = 4/6, rev = TRUE), c(0, 60))  # 2

# black & white palette for CTs or MRs
display.palette (grey.colors (256, start=0, end=1), c(-100, 100))          # 3

# transparency affects colors depending on background (black in first exemple,
# white in the second one)
display.palette (rainbow(256, s = seq(1, 0, length.out = 256), start = 0, end = 4/6,
                         alpha = seq(0.8, 0, length.out = 256), rev = TRUE), c(0, 60))            # 4
display.palette (rainbow(256, s = seq(1, 0, length.out = 256), start = 0, end = 4/6,
                         alpha = seq(0.8, 0, length.out = 256), rev = TRUE), c(0, 60), bg="white")# 5
```



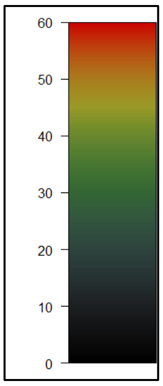
#1



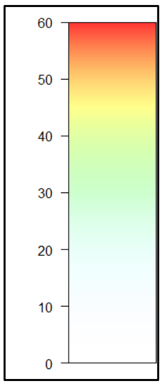
#2



#3



#4



#5

First palette on the left is the RVV palette extending from -1000 to 1000 HU. Next ones are examples using the rainbow palette and the black and white palette. The fourth and fifth examples show how the colorful rainbow palette is affected when using variable transparency on black (#4) (by default) or white (#5) backgrounds. Colors (`col`, the first argument of `display.palette`) are supposed to run linearly from the first to the second value of `i.rng.breaks` argument. If colors are defined by their mids (like for tissue identification and segmentation, for instance), consider using `i.rng.mids` instead.

3 Advanced 2D representation

As mentioned earlier, using `display.plane`, the main image is displayed on bottom. We can add an overlay image (consider adding transparency) on top, and structures if needed.

3.1 Adding structures

Adding structures is very easy, you just have to provide a structure set, and tell `display.plane` what structure you want to see. Structures can be accessed three ways:

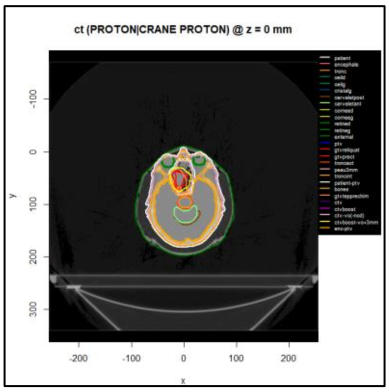
- By the exact correspondence with their `roi.pseudo` and names you request, for instance `roi.name = “heart”`.
- By a partial match between their `roi.pseudo` and names you could give, for instance `roi.sname = ”eye”` would probably give you left AND right eyes.
- Or by their index, for instance, `roi.idx = 1` is usually the patient contour.

Despite its apparent complexity, this solution is very practical when working with several patients having different names for the same region of interest (RoI). You just have to rename the `roi.pseudo`, for instance using “`my.personnal.roi.name`”, then for all subsequent code, you can access the RoI with its new name.

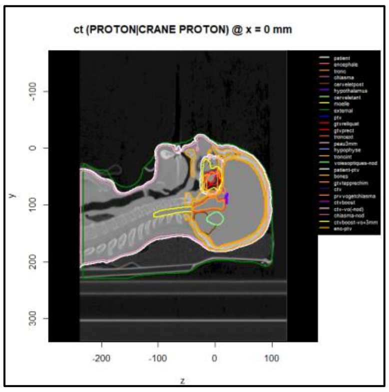
If none is requested, every RoI will be displayed. If you do not want the RoI legend to appear just set `legend.plot` option to `FALSE`. We will see later a way to get a separate legend.

As an example, the structure set on top of the CT:

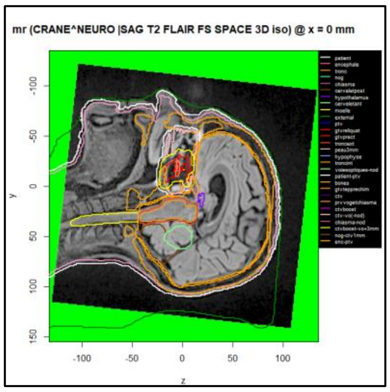
```
Par (mar = c(4, 4, 4, 6))   # To increase the margin on the right
display.plane (CT, struct = S)                                              # 1-left
display.plane (CT, struct = S, view.type = "sagi")                          # 2-middle
display.plane (MR, struct = S, view.type = "sagi", T.MAT = T.MAT, bg="green")       # 3-right
```



#1



#2



#3

Figure #1 does not need comments. It is just the structure set displayed onto of the CT.

On figure #2, you can see that espadon displayed the structure set as a continuous contour, even if the requested view is not the transverse one. In fact, every RoI and every plane is recomputed in order to build a new RoI representation in this view.

Figure #3 illustrates the same process in the case of the MR. Note that, **the default frame of reference is always the bottom image's one**. That is why we just had to provide `T.MAT=T.MAT` option. In this case, the structure set has been recomputed in the MR FoR. Had we provided the `display.ref` option, espadon would have displayed both MR and structure set in the requested FoR.

Note that, changing the frame of reference of the structure set or the view (except transverse), leads to computations for the display that can be time consuming.

If you want to get the color legend of the structures separately, consider using the `display.legend` instruction. The colors are the one stored in the rt-Struct file. If you want to change them you should modify the `S$roi.info$color` associated with the appropriate RoI.

3.2 Image overlays

It is often interesting to "superpose" images. Depending on your needs you could want to:

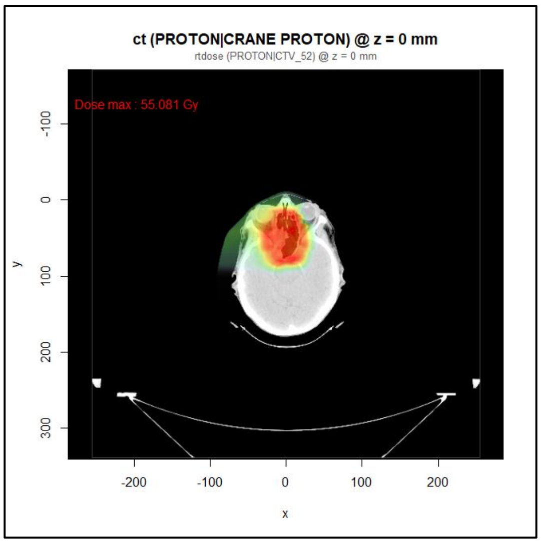
- Add a dose on top of a CT (or a MR for instance, we saw the modality of bottom image does not matter), to see if the target volume is appropriately covered.
- Add a binary modality on top of a CT to see if your favorite segmentation algorithm is right.
- Add an image on top of another image so see if your favorite registration algorithm performed well.
- We probably forgot lot of usages…

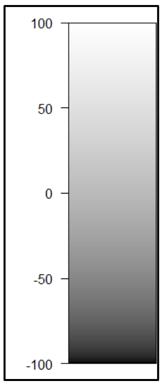
In all examples above, the master keyword is "seeing". Espadon provides simple ways to add continuous volumes on top of other continuous volumes. In fact, for continuous volumes, we will make a distinction between "analog" data (as for the dose) and "binary" data (as for selections for instance), and we will include some interesting representation tools for images over images.

3.2.1 Analog data on top of an image

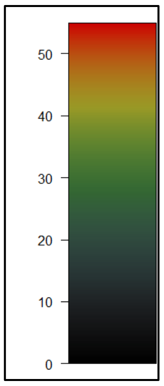
This kind of representation is mainly interesting for objects giving different information about the problem, for instance, superposition of a dose over a CT. As we would like to see the CT (at the bottom), we will apply some transparency on the dose (at the top). A good start could be something like that:

```
display.plane (bottom = CT, top = D,
               bottom.breaks = seq (-100, 100, length.out = 256),
               bottom.col = grey.colors(255, start = 0, end = 1),
               top.breaks = seq (0, 55, length.out = 256),
               top.col = rainbow(255, s = seq(1, 0, length.out = 255), start = 0, end = 4/6,
                                 alpha = seq(0.8, 0, length.out = 255), rev = TRUE),
               sat.transp = TRUE)
display.palette(grey.colors (255, start = 0, end = 1), c(-100, 100))
display.palette(rainbow(255, s = seq(1, 0, length.out = 255), start = 0, end = 4/6,
                        alpha = seq(0.8, 0, length.out = 255), rev = TRUE), c(0, 55))
```


ct (PROTON|CRANE PROTON) @ z = 0 mm
rtdose (PROTON|CTV_52) @ z = 0 mm
Dose max : 55.081 Gy
-100
0
100
200
300
y
-200
-100
0
100
200
x
Overlay of CT and rt-Dose



CT palette



rt-Dose palette

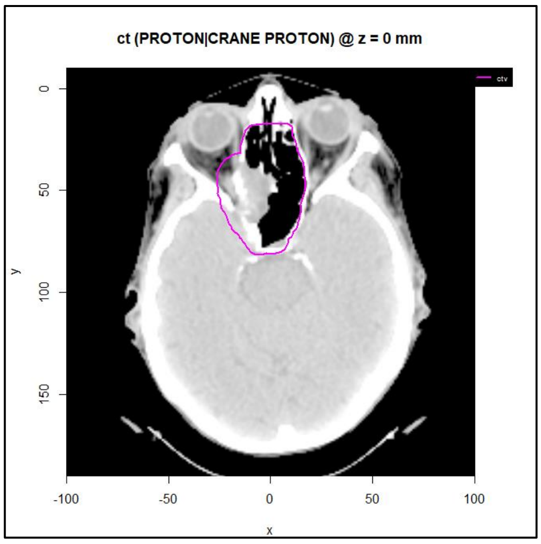
As espadon knows we are displaying a dose, it plots the maximum dose value, in this plane on top of the image. By default, the bottom palette is grey scale and the top palette rainbow (with adapted transparency). We are not obliged to specify them. As we changes the bottom range (from -100 to 100 HU), and the top scale (from 0 to 55 Gy), it is useful to display the color scales for both (on the right).

The main trick here was to use transparency. It is the role of the alpha value in a color palette. Parameter `alpha = 0` means the color is totally transparent, and `alpha = 1`, the color is totally opaque (we cannot see below). Here, we stopped the transparency for the red at `alpha=.8`, that is why we can see the nasal cavities on the CT, even if they receive the maximum dose. Low dose values are nearly transparent.

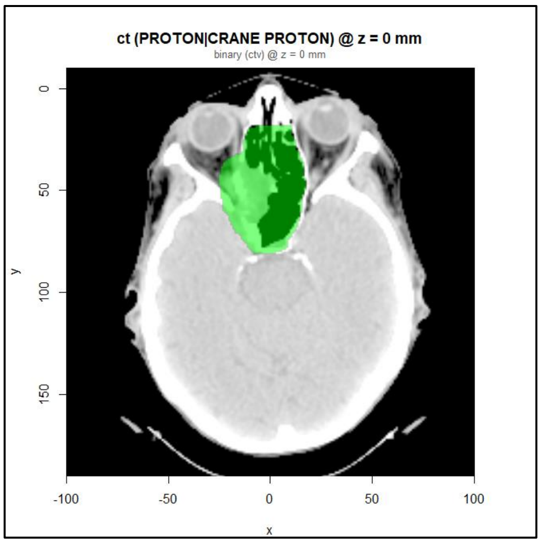
3.2.2 Binary data on top of an image

Binary modality volumes come from volume segmentation in espadon. They can have only three values, `TRUE` or `FALSE`, including `NA` for unspecified voxels. The usual meaning is `TRUE` for voxels belonging to a selected volume and `FALSE` for voxels outside. The basic instruction for getting a binary representation of a volume is `bin.from.roi`, as illustrated below:

```
S$roi.info$color[S$roi.info$roi.pseudo == "ctv"] <- "magenta"
display.plane (CT, struct=S, roi.name = "ctv",
               bottom.breaks = seq (-100, 100, length.out = 256),
               abs.rng = c(-100, 100), ord.rng = c(-10, 190), sat.transp = TRUE)  # left

bin.ctv <- bin.from.roi (CT, S, roi.name = "ctv")
display.plane (bottom=CT, top=bin.ctv,
               top.col = c("#00000000", "#00ff0080"),
               top.breaks = c(-0.5, 0.5, 1.5),
               bottom.breaks = seq (-100, 100, length.out = 256),
               abs.rng = c(-100, 100), ord.rng = c(-10, 190), sat.transp = TRUE)  # right
```



CT and CTV RoI

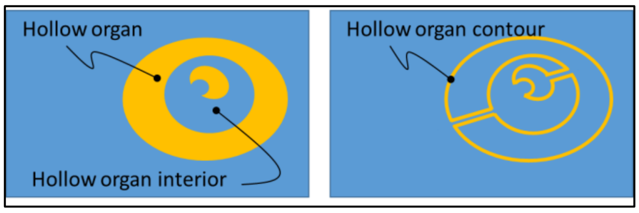


CT and binary selection of the CTV superposed

On the left, the representation as a contour of the CTV RoI on top of the CT. Note here the way we changed the color of the CTV to magenta on the first line.

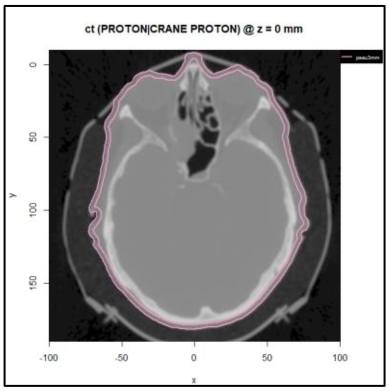
Binary modality is especially useful for getting image values. It can be displayed on top of the CT, as on the right image. We used two colors, black for `FALSE` (between -0.5 and 0.5), totally transparent, and green for `TRUE` (between 0.5 and 1.5), slightly transparent. This is respectively the meaning of the two last digits of “#00000000” and “#00ff0080”. Colors are defined by their (red, green, blue) triplet ranging from 0 to 255, i.e., #00 to #ff in hexadecimal. The last digit is the alpha channel also ranging from #00 (transparent) to #ff (opaque).

Binary modality suffers a **general issue** you have to keep in mind for **hollow organs**. In fact “hollow” has a clear definition in DICOM norm. Unfortunately, the way it is rendered by RoI is TPS-specific (it should not in fact). The norm is explicit. Hollow organs contours should be represented as on the right image:

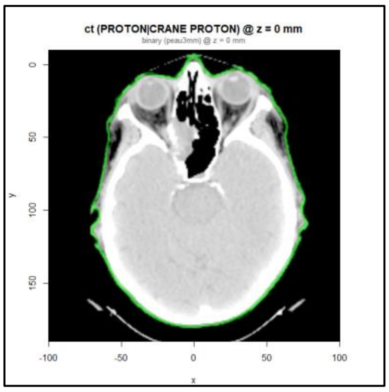


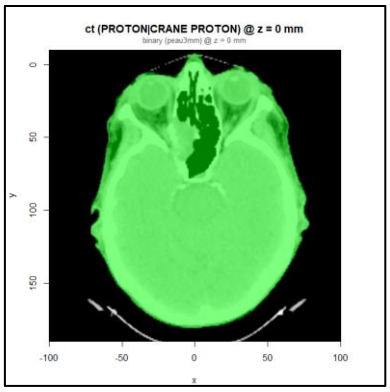
In fact, some TPS use a kind of volume algebra we illustrate in the following example where we are interested in the skin dose (for instance). Note the computation of binary volumes can be time consuming, especially for large volume as the patient one’s:

```
display.plane (CT, struct=S, roi.sname = "peau",
               abs.rng = c(-100, 100), ord.rng = c(-10, 190))                       # left

bin.skin1 <- bin.from.roi (CT, S, roi.sname = "peau")
display.plane (bottom=CT, top=bin.skin1,
               top.col = c("#00000000", "#00ff0080"),
               top.breaks = c(-0.5, 0.5, 1.5),
               bottom.breaks = seq (-100, 100, length.out = 256),
               abs.rng = c(-100, 100), ord.rng = c(-10, 190), sat.transp = TRUE)      # middle

bin.skin2 <- bin.from.roi (CT, S, roi.sname = "peau", within=FALSE)
display.plane (bottom=CT, top=bin.skin2,
               top.col = c("#00000000", "#00ff0080"),
               top.breaks = c(-0.5, 0.5, 1.5),
               bottom.breaks = seq (-100, 100, length.out = 256),
               abs.rng = c(-100, 100), ord.rng = c(-10, 190), sat.transp = TRUE)      # right
```



Skin RoI stored in the rt-Struct


ct (PROTON|CRANE PROTON) @ z = 0 mm
binary (peau3mm) @ z = 0 mm
y
x
Binary of the skin RoI rendered by espadon


ct (PROTON|CRANE PROTON) @ z = 0 mm
binary (peau3mm) @ z = 0 mm
y
x
Binary of the skin RoI as it shloud be rendered officialy

On the right, the skin contour as computed by the TPS. Note that the connecting line (mandatory in DICOM format) is not present.

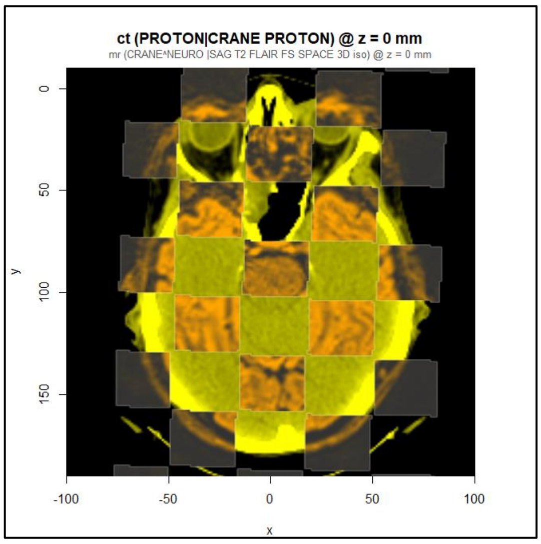
In the middle, the skin volume rendered by default by `bin.from.roi`. espadon sees that a contour line is within another and interprets this as the definition of a hollow volume. If TPS were DICOM compliant, we should not operate like that!

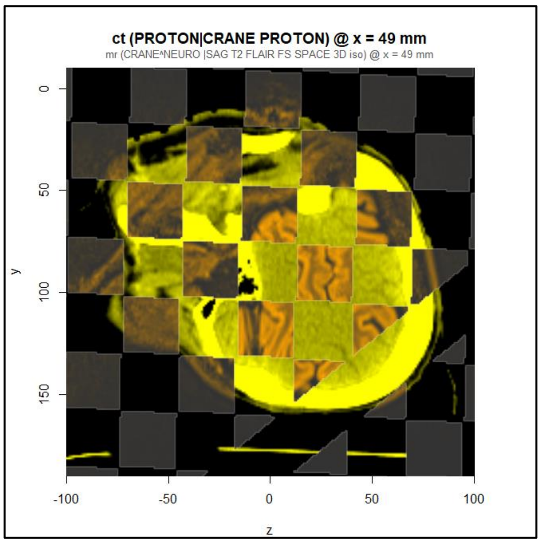
If you do not want espadon to interpret contour lines, just set the within option to `FALSE`, as was made on the right image.

3.2.3 An image on top of an image

Suppose we want to check the registration of the MR on the CT. It is difficult to superpose both images as they both are in grey-scales. A usual way to do this job consists in adding a chessboard on the top image where black squares are opaque and white squares transparent. This is not directly included in espadon, but with simple manipulations on the top object, it is easy to do.

```
MR.chess <- MR
i <- 1:(dim(MR.chess$vol3D.data)[1])
j <- 1:(dim(MR.chess$vol3D.data)[2])
k <- 1:(dim(MR.chess$vol3D.data)[3])
delta <- 60
ijk <- as.matrix (expand.grid(i, j, k))
ijk.xor <- xor (xor (ijk[, 1]%%delta >= delta / 2, ijk[, 2]%%delta >= delta / 2),
                ijk[, 3]%%delta >= delta / 2)
MR.chess$vol3D.data[ijk.xor] <- NA

display.plane(CT, MR.chess, T.MAT = T.MAT,
              abs.rng = c(-100, 100), ord.rng = c(-10, 190),
              bottom.breaks = seq (-100, 100, length.out = 256),
              bottom.col = colorRampPalette(c("black", "yellow"))(255),
              top.breaks = seq (0, 150, length.out = 256),
              top.col = colorRampPalette(c("grey20", "orange"))(255),
              sat.transp = TRUE)  # left-image
display.plane(CT, MR.chess, T.MAT = T.MAT, view.type = "sagi",
              abs.rng = c(-100, 100), ord.rng = c(-10, 190), view.coord = 49,
              bottom.breaks = seq (-100, 100, length.out = 256),
              bottom.col = colorRampPalette(c("black", "yellow"))(255),
              top.breaks = seq (0, 150, length.out = 256),
              top.col = colorRampPalette(c("grey20", "orange"))(255),
              sat.transp = TRUE)
```

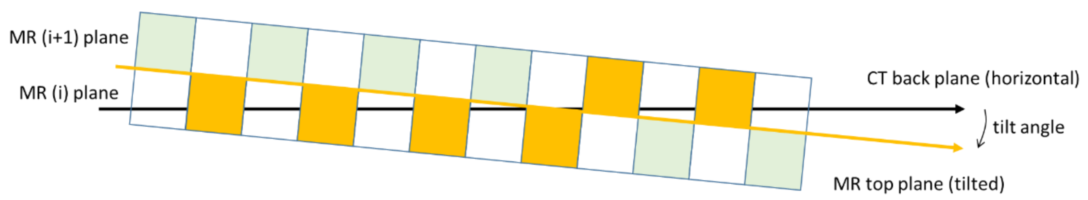

ct (PROTON|CRANE PROTON) @ z = 0 mm
mr (CRANE^NEURO |SAG T2 FLAIR FS SPACE 3D iso) @ z = 0 mm
y
0
50
100
150
-100
-50
0
50
100
x
Frontal view


ct (PROTON|CRANE PROTON) @ x = 49 mm
mr (CRANE^NEURO |SAG T2 FLAIR FS SPACE 3D iso) @ x = 49 mm
y
0
50
100
150
-100
-50
0
50
100
z
Sagittal view

In the first line, we just make a copy of the MR. then we get i, j, k indexes of the volume. Note the use of expand.grid function that gives every possible combinations of indexes. Then we create a 3D chessboard, each square being of size delta (in pixels, preferably an even number). The chessboard is simply a XOR of coordinates combinations modulo delta being superior or equal to delta divided by two. Then we simply set image content to NA (Not Available) at images positions that are TRUE. The bottom image (CT) is in yellow scale, and the top image (MR) in grey to orange scale. The skull appears in hyper-density on the CT, as bones are dense material; in hypo-density on the MR as bones contain less water (protons) than soft-tissue. As we can see, checking the chessboard content, the registration process was quite good.

An interesting thing appears on the bottom right part of the right image. There is a kind of oblique line switching the chessboard squares from transparent to opaque. Can you guess where it comes from?

Answer: this is a general issue when merging objects that do not share the same space sampling. This is not an artifact, just basics mathematics of sampling. Suppose the display plane defined by the CT is horizontal. Then see how this plane cuts the MR volume, tilted just a few degrees (in this case):



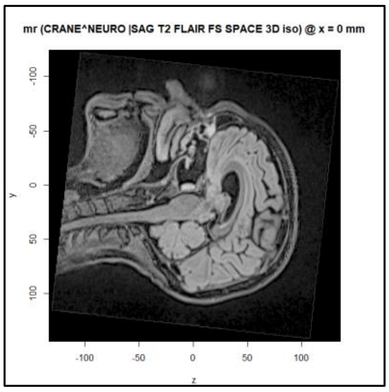
In fact, due to the tilt angle, first pixels of the MR are sampled in a plane and following pixels in the next plane. This is particularly visible in the case illustrated above as pixels were big, but keep in mid this is always what appends.

3.3 Image interpolation and sampling

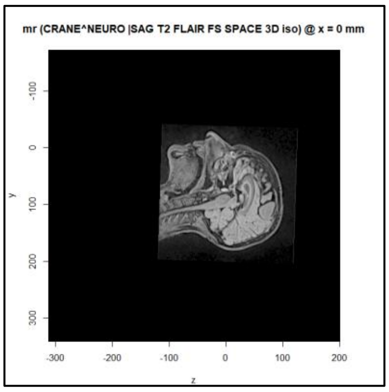
Up to now, we superposed images, using frame of reference transformations if needed. In fact, there is another way to do the job: perform the transformation BEFORE calling `display.plane`. Beware this functionality is memory consuming as we create a new object, for instance, a MR transported in the frame of reference of a CT. For this, we will use the `vol.regrid` instruction. This function uses the destination volume's grid (the CT), and interpolates the original volume (the MR) on this grid. There are several ways to do it, and we will examine some good or bad properties of interpolation. The choice depends on your goal!

First, let us use `vol.regrid` and see what it does:

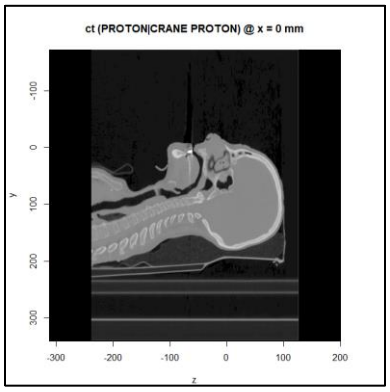
```
MR.regrid <- vol.regrid (MR, CT, T.MAT = T.MAT)

display.plane (MR , view.type = "sagi")                 # original MR in its own FoR-left
display.plane (MR.regrid , view.type = "sagi")          # MR grided on the CT-middle
display.plane (CT , view.type = "sagi")                 # original CT-right
```



Original MR



MR regrided on the CT



Original CT

The original MR is simply for memory. It is in its own frame of reference.

The regrided MR has now the size of the CT. In fact, the whole MR volume has first been transported in the CT FoR, and has been resampled on the grid of the CT.

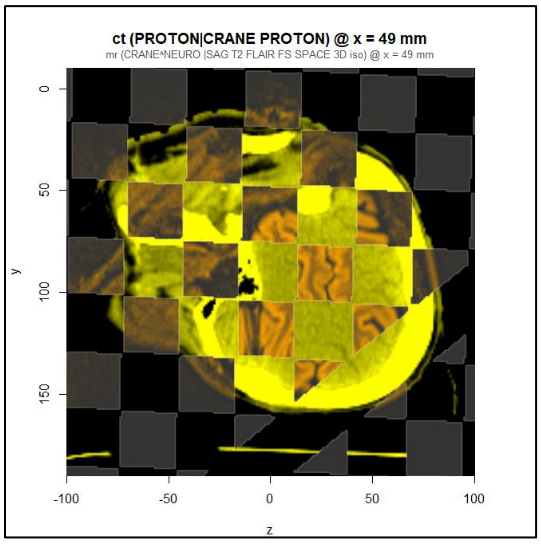
That is why the regrided MR is now directly stackable on the CT.

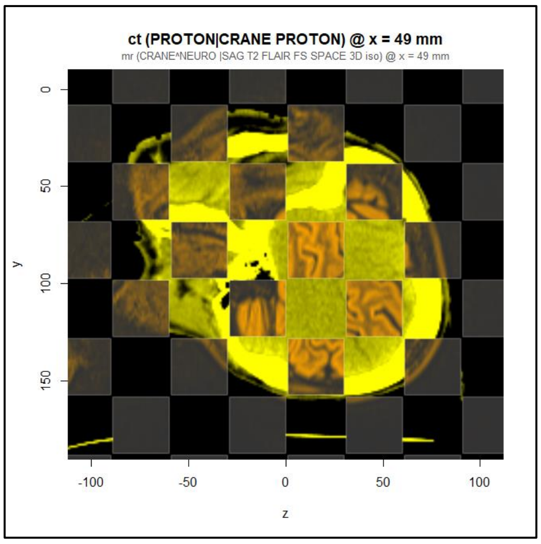
If we inspect original and regrided MR information, we can see the final volume shares the same space sampling than the CT (512x512x365 voxels):

```
str (MR$vol3D.data)

 num [1:256, 1:256, 1:144] 0 0 0 0 0 0 0 0 0 0 ...

str (MR.regrid$vol3D.data)

 num [1:512, 1:512, 1:365] NA NA NA NA NA NA NA NA NA NA ...

str (CT$vol3D.data)

 num [1:512, 1:512, 1:365] -1024 -1024 -1024 -1024 -1024 ...
```

We can play the superposition game of two images of the previous chapter and see what happens now (the code is the same, the only difference relies on base objects):

```
MR.regrid.chess <- MR.regrid
i <- 1:(dim(MR.regrid.chess$vol3D.data)[1])
j <- 1:(dim(MR.regrid.chess$vol3D.data)[2])
k <- 1:(dim(MR.regrid.chess$vol3D.data)[3])
delta <- 60
ijk <- as.matrix (expand.grid(i, j, k))
ijk.xor <- xor (xor (ijk[, 1]%%delta >= delta / 2, ijk[, 2]%%delta >= delta / 2), ijk[, 3]%%delta >= delta / 
2)
MR.regrid.chess$vol3D.data[ijk.xor] <- NA

display.plane(CT, MR.chess, T.MAT = T.MAT, view.type = "sagi",
              abs.rng = c(-100, 100), ord.rng = c(-10, 190), view.coord = 49,
              bottom.breaks = seq (-100, 100, length.out = 256),
              bottom.col = colorRampPalette(c("black", "yellow"))(255),
              top.breaks = seq (0, 150, length.out = 256),
              top.col = colorRampPalette(c("grey20", "orange"))(255),
              sat.transp = TRUE)                                          # left
display.plane(CT, MR.regrid.chess, view.type = "sagi",
              abs.rng = c(-100, 100), ord.rng = c(-10, 190), view.coord = 49,
              bottom.breaks = seq (-100, 100, length.out = 256),
              bottom.col = colorRampPalette(c("black", "yellow"))(255),
              top.breaks = seq (0, 150, length.out = 256),
              top.col = colorRampPalette(c("grey20", "orange"))(255),
              sat.transp = TRUE)                                          # right
```


ct (PROTON|CRANE PROTON) @ x = 49 mm
mr (CRANE^NEURO |SAG T2 FLAIR FS SPACE 3D iso) @ x = 49 mm
0
50
100
150
y
-100
-50
0
50
100
z
Before resampling the MR


ct (PROTON|CRANE PROTON) @ x = 49 mm
mr (CRANE^NEURO |SAG T2 FLAIR FS SPACE 3D iso) @ x = 49 mm
0
50
100
150
y
-100
-50
0
50
100
z
After resampling the MR

Now, as de “chessboarding” was made on the regrided MR, the chessboard squares are perfectly aligned with the CT. That is why, squares are perfectly horizontal and vertical, and the sampling “problem” disappeared.

Beware there is no magic in the sampling process:

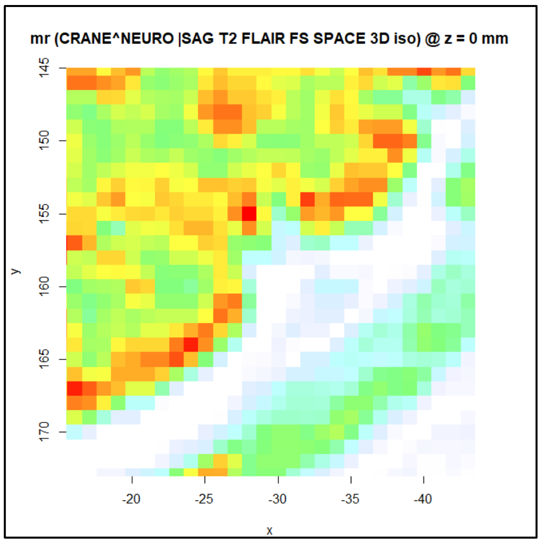
- If the destination grid is more widely spaced than the original grid, we will speak about under-sampling. **Under-sampling** inevitably leads to **loss of information**.
- If the destination grid is narrower than the original grid, we will speak about over-sampling. **Over-sampling does not create information**, it just makes up a story that nothing says is the right one.

Both rely on interpolation as we wonder what could be the value of an image (sampled on its own grid) at points where it was generally not measured (the destination grid). There are several ways to interpolate data, the simplest being:

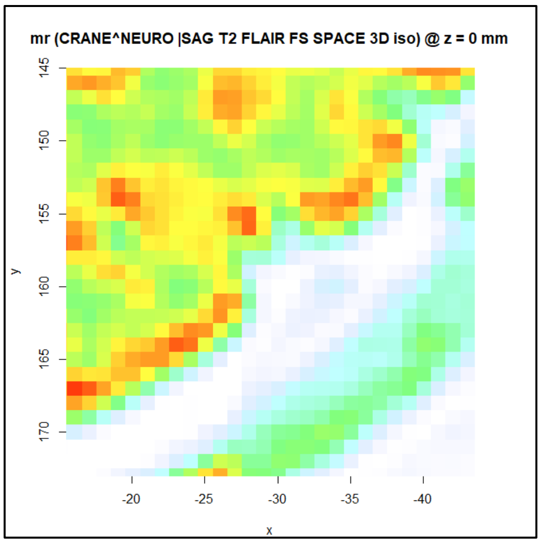
- The **zero order sampler** (or “sample and hold”). The same value is assumed at any point in the voxel volume. The zero order sampling presents a pixelated aspect but it **preserves sharper edges**.
- The first order sampler, here, in 3D, the **trilinear interpolation**. The value at a given point in space is a polynomial ponderation of the eight voxels values surrounding this point. The trilinear interpolation **gives a smooth aspect** at the expanse of edges.

To visualize that, consider the following example where we used a color palette to highlight details:

```
MR.0.order <- vol.regrid (MR, CT, T.MAT = T.MAT, interpolate = FALSE)
MR.1st.order <- vol.regrid (MR, CT, T.MAT = T.MAT, interpolate = TRUE)

display.plane (MR.0.order,
               bottom.breaks = seq (0, 150, length.out=256),
               bottom.col = rainbow(255, s = seq(1, 0, length.out = 255),
                                    start = 0, end = 4/6, rev = TRUE),
               abs.rng = c(-19, -40), ord.rng = c(145, 173), sat.transp = TRUE,
               interpolate = FALSE)                                         # left
display.plane (MR.1st.order,
               bottom.breaks = seq (0, 150, length.out=256),
               bottom.col = rainbow(255, s = seq(1, 0, length.out = 255),
                                    start = 0, end = 4/6, rev = TRUE),
               abs.rng = c(-19, -40), ord.rng = c(145, 173), sat.transp = TRUE,
               interpolate = FALSE)                                         # right
```



Zero-order interpolation (by default)

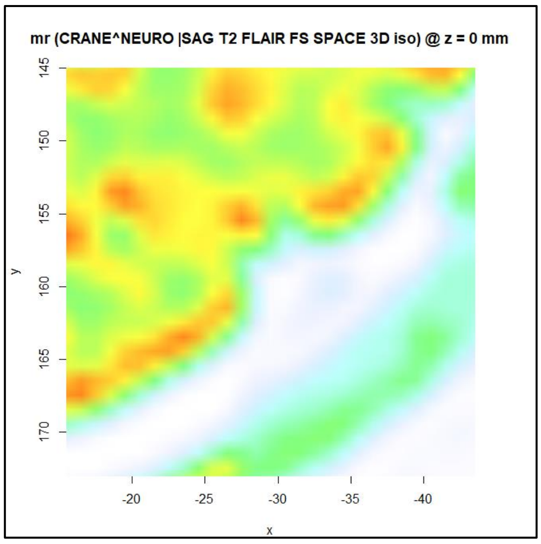


Trilinear interpolation

Well, the difference is slight, but it exists! In espadon, by default, we use interpolation, but if you want a zero-order sampling, just set `interpolation` to `FALSE`.

For instance, we did not mentioned it, but `display.plane` has one (purely cosmetic):

```
display.plane (MR.1st.order,
               bottom.breaks = seq (0, 150, length.out=256),
               bottom.col = rainbow(255, s = seq(1, 0, length.out = 255),
                                    start = 0, end = 4/6, rev = TRUE),
               abs.rng = c(-19, -40), ord.rng = c(145, 173),
               interpolate = TRUE, sat.transp = TRUE)
```



Trilinear interpolation sampling, with interpolation display

4 What we learned

We saw that the `display.plane` instruction is the Swiss Army knife of 2D representation in espadon. It can handle images and structure sets in any frame of reference (provided the registration exists). The bottom image defines the default FoR. The display FoR can be changed by providing the `T.MAT` and telling the target FoR. Then the top image is displayed (preferably using transparency) in the selected FoR. Then structure sets are added. If not expressed in the display FoR, they are recomputed to form continuous lines on the display.

For each image (bottom or top), user palettes and breaks can be defined for powerful representations.

If needed (for documentation purpose), structure set legends and color legends can be produced separately.

By the way, all these instructions (and others) have even more options (well, we explained most of them for `display.plane`), do not forget to have a look at the reference manual and other vignettes.

You can see also:

`display.kplane`	`Display of a plane of a volume`
`display.legend`	`Display of the RoI legend`
`display.palette`	`Display of the color palette`
`display.plane`	`Display the transverse frontal or sagittal view in the patient reference system`
`pal.RVV`	`Conversion of Hounsfied Units to Realistic Volume Vizualization colors`
`select.names`	`Regions of Interest (RoI) indices`
`vol.regrid`	`Transform the grid of a volume class object into the grid of another.`