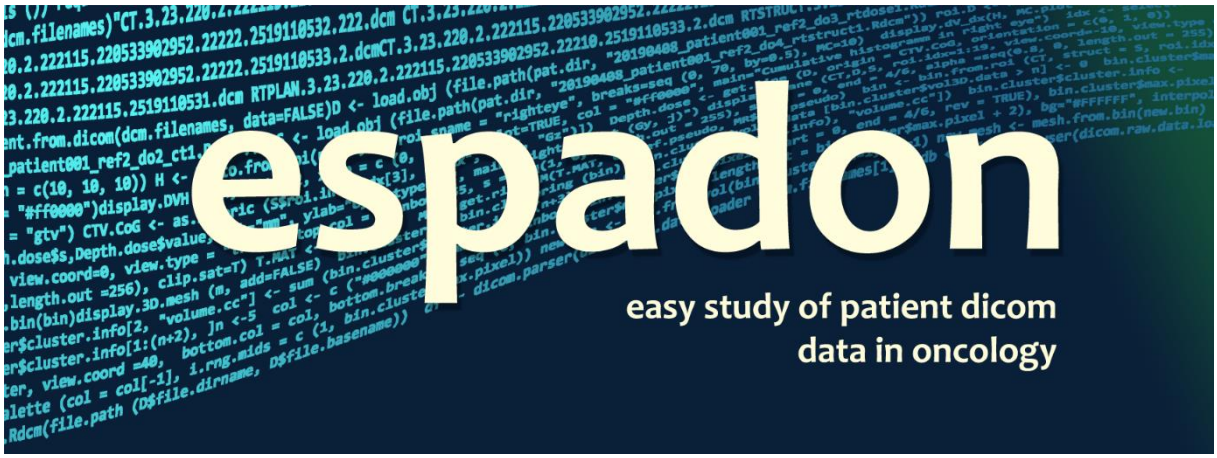


3D representation

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?

Representation is the basics of any study. The fact is that, in medical imaging, we often manipulate 3D stacks of images, binary or structure sets. Displaying these data in 2D is a first level of representation, but they truly are in 3D and need specific visualization tools we will explain in this vignette. Moreover, we will see in lots of examples that this kind of representation gives outstanding results. We will see we have to learn thinking in 3D in order to understand some properties of imaging objects, as expansions, for instance.

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

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

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

pat <- load.patient.from.Rdcm(pat.dir)
display.obj.links(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]])
```

The first section will deal with basics of 3D representations as wires or surface. The second section will introduce sophistications and include a new kind of surface data representation.

Espadon makes extensive use of rgl package of Duncan Murdoch for representation and Rvcg package of Stefan Schlager for mesh computation. Both include many many possibilities, and only few are effectively used here. Consider looking at these packages for more sophisticated images.

2 Basic 3D representation

In fact, in medical imaging, we manipulate two kinds of specific 3D objects:

1) Contours, that are basically curves in given planes of the image stack
2) Volumes, that are assemblies of voxels more or less belonging to given regions of interest (RoI)

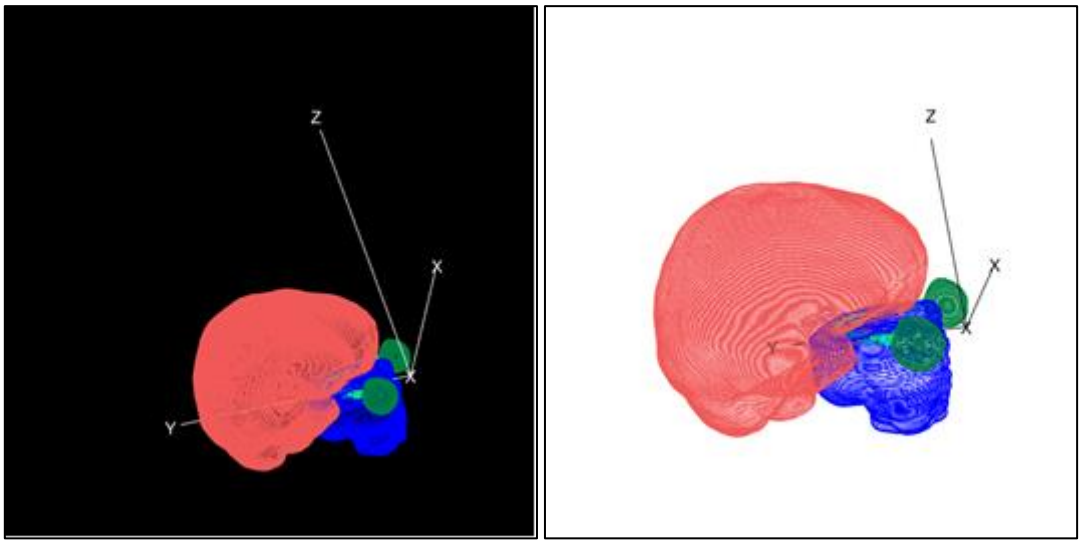
The main advantage of 3D representation is that it naturally preserves patient's orientation and voxel size. No risk to make right/left, top/bottom or face/back inversions that are a common source of troubles in 2D imaging.

2.1 Displaying 3D Regions of Interest (RoIs)

Displaying RoI is direct in espadon, using the `display.3D.contour` instruction. No interpretation and no modification are made on data. It simply displays the contour lines. For instance:

```
display.my.rois <- function (FoR=TRUE, FoR.col = "black") {
  display.3D.contour(S, roi.sname = "oeil", FoR.axis = FoR, FoR.col= FoR.col)
  display.3D.contour(S, roi.name = "chiasma", FoR.col= FoR.col)
  display.3D.contour(S, roi.name = c ("nod", "nog", "chiasma", "encephale"), FoR.col= FoR.col)
  display.3D.contour(S, roi.name = "ptv", FoR.col= FoR.col)
}
```

```
open3d (windowRect = c (100, 100, 600, 600))
bg3d("black")
display.my.rois (FoR.col = "white")

open3d (windowRect = c (100, 100, 600, 600))
display.my.rois (FoR = 100)
rgl.snapshot ("my 3D view.png")
```



Display of ROIs contours, with black/white background and default/100mm axis

It is a good practice to define the displaying window, using for instance `open3d (windowRect <- c (100, 100, 600, 600))` instruction of the rgl package. All subsequent displays will have the same size, which is practical for articles.

The background color is then set to black, thanks to the `bg3d` instruction of the rgl package.

The `display.my.rois` function simply defines what we want to see. As you can see, we have defined an optional display of the reference frame; its color is black by default. In this function, you can display your favorite RoIs. Note you can change the size of FoR axes just by specifying their length.

Last, when you found an appropriate view, a good practice consists in using `rgl.snapshot` instruction to capture it. This way, all your images will have the same size (actually, that is what we did for this vignette).

The default color of RoI is that was defined in the TPS. If you want to change that, remember default colors are set in `S$roi.info$color`. Feel free to change them as you wish.

2.2 Displaying 3D volumes

For the moment, we just displayed 3D contours as they are stored in DICOM. This is very practical as in 3D, this information is direct with no interpretation. But, in fact, contours define the surface of a shape, and this surface is absolutely meaningful. So, we will deal about volume/surface representation, now.

2.2.1 Basics of volume representation

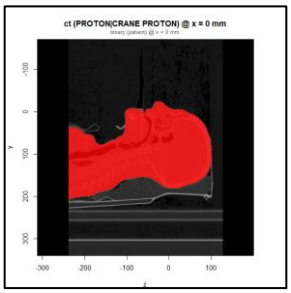
Volume representation is one of the most complicated thing, not especially in espadon, but generally speaking. It can be an exciting volume density, complicated to render; or iso-surface, far more “simpler” using

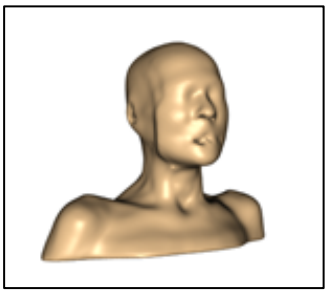
transparencies; or more generally, a single surface, the point of view we will discuss here. Moreover, this surface can be drawn from RoI, and even from volumes content, with more or less sophistications as they both rely on binary conversions and mesh representation.

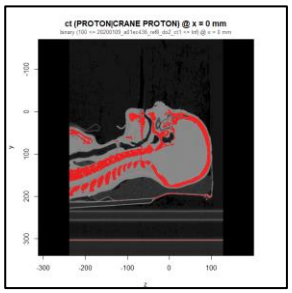
Mesh are probably the most powerful and "simplest" representation system for surfaces. We will discuss its exciting possibilities in a separate vignette. In fact, it is the basics of volume representation. For the moment, we just have to know **a volume can be represented as a mesh**, and **a mesh is just** a "simple" "equation" of **the volume surface** (not so simple, in fact, and not just an equation, but with so many possibilities… we will see later).

The final step of the process is getting a mesh. In espadon, the instruction is `mesh.from.bin`, suggesting you first need to have a binary description of your volume. Depending on what you want to do, this binary representation comes from `bin.from.roi` (if you have a specific RoI), `bin.from.vol` (if you work on an "analogical" volume), or any combination of binaries. We will shortly explore every possibilities; see the appropriate vignette for more information. As an example, the "patient" RoI, and a selection of CT>100 (possibly bones):

```
bin.pat <- bin.from.roi(CT, S, roi.name = "patient")
display.plane (CT, bin.pat, view.type = "sagi")                              # 1
mesh.pat <- mesh.from.bin (bin.pat)
open3d (windowRect = c (100, 100, 600, 600))
display.3D.mesh (mesh.pat, color = "burlywood2", specular = "#404040")       # 2

bin.bones <- bin.from.vol(CT, min=100)
display.plane (CT, bin.bones, view.type = "sagi")                            # 3
mesh.bones1 <- mesh.from.bin (bin.bones)
open3d (windowRect = c (100, 100, 600, 600))
display.3D.mesh (mesh.bones1, color = "burlywood2", specular = "#404040")    # 4
```


#1


#2

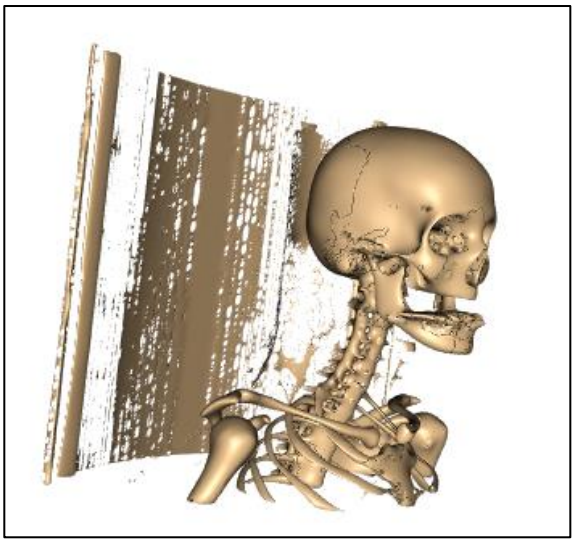

#3


#4

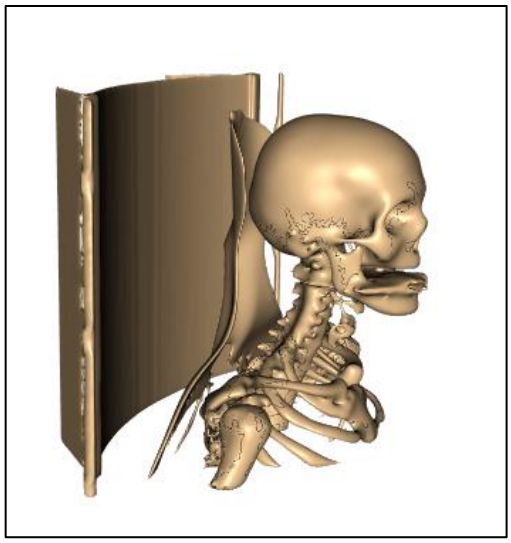
As can be seen, mesh display produces a pretty view. In fact, `mesh.from.bin` uses Rvcg package. Rcvg interpolate binary space using the "Marching Cubes" algorithm which made a good job (not enough for mathematics on surface, we will see later, but, good for now). As Marching Cubes interpolate space a rough way, it is often a good idea to smooth the result, using the `smooth.iteration` option (by default, set to 40) and consider the `tolerance` (set to 1mm by default). Both are Rvcg applications. See the manual for more information.

An embarrassing feature of binary from volume production is that it often leads to lot of isolated voxels or small clusters as illustrated on fig#4 (in fact, we do not want to see the couch, for instance). Fortunately, espadon includes a cluster filtering based on cluster volumes that solves (at least partially) this issue. It is time to play with clusters. We just have to remark bones are generally connected (even if not on the CT due to limited resolution):

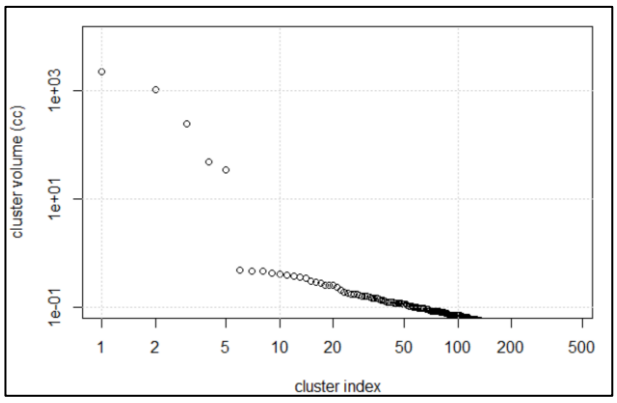
```
bin.bones <- bin.from.vol (CT, min=100)
display.plane (CT, bin.bones, view.type = "sagi")
mesh.bones <- mesh.from.bin (bin.bones)
open3d (windowRect = c (100, 100, 600, 600))
display.3D.mesh (mesh.bones, color = "burlywood2", specular = "#404040")        # 1
rgl.snapshot("3D representation.png")

dil.bones <- bin.dilation (bin.bones, max (bin.bones$dxyz))
mesh.dil.bones <- mesh.from.bin (dil.bones)
open3d (windowRect = c (100, 100, 600, 600))
display.3D.mesh (mesh.dil.bones, color = "burlywood2", specular = "#404040")    # 2

clust.bones <- bin.clustering (dil.bones)
par (mar = c (4, 4, 1, 1))
plot (clust.bones$cluster.info$value,
      clust.bones$cluster.info$volume.cc, log = "xy", ylim = c (.1, 10000),
      xlab = "cluster index", ylab = "cluster volume (cc)")                     # 3
grid ()
clust.sel <- bin.from.vol (clust.bones, min = .5, max = 1.5)
mesh.sel <- mesh.from.bin (clust.sel)
open3d (windowRect = c (100, 100, 600, 600))
display.3D.mesh (mesh.sel, color = "burlywood2", specular = "#404040")          # 4

bin.keep <- bin.intersection (clust.sel, bin.bones)
mesh.keep <- mesh.from.bin (bin.keep)
open3d (windowRect = c (100, 100, 600, 600))
display.3D.mesh (mesh.keep, color = "burlywood2", specular = "#404040")         # 5
```



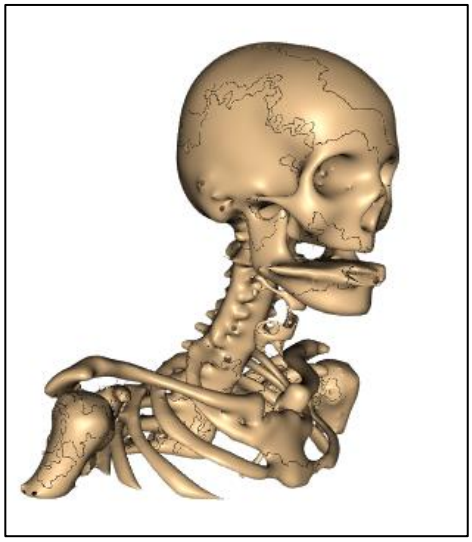
1 thresholded CT



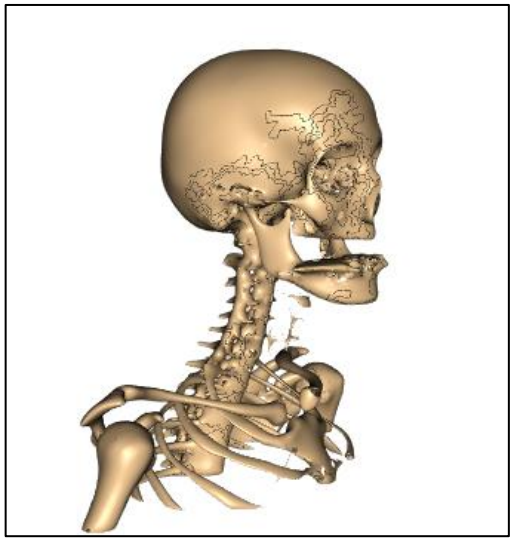
2 after volume expansion
(max (bin.bones$dxyz) = 2mm)



3 volumes of cluster



4 first cluster selection



5 intersection between original selection and 1st cluster. Bones have their original size and disconnected clusters are removed. Only the skeletton remains.

The different steps we experimented here are the following:

1) We make a selection on the CT>100 HU. It produces lot of clusters and the couch is apparent.
2) We make a volume expansion of 2 mm (max of volume field $dxyz). Bones that were disconnected are now connected, and the couch remains isolated from bones
3) We now make a clustering. Clustering computes every clusters of 8-connected voxels. It is an extension of 2D filling in 3D. As bones represent the major volume, they constitute a unique entity, the one with the biggest volume, here.
4) We make a selection of the main cluster. Clearly, it appears quite "fat" as we expanded the bone selection
5) Last, we compute the intersection between this selection (restricted to "fat" bones) and the original CT selection above 100 HU. We get the desired result, the skeleton of the patient.

Keep in mind that:

- volume expansion is time consuming as it involves several 3D FFTs (which remains much faster than a convolution, in fact),
- as well as clustering for other reasons…

We could have done this job faster and easier, using the fact that bones are inside the "patient" contour. We let it as an exercise. This example was just an occasion to explore volume expansion and cluster filtering. A last word, remember to use the par3d ()$userMatrix field if you want to keep the same viewpoint for each 3D representation:

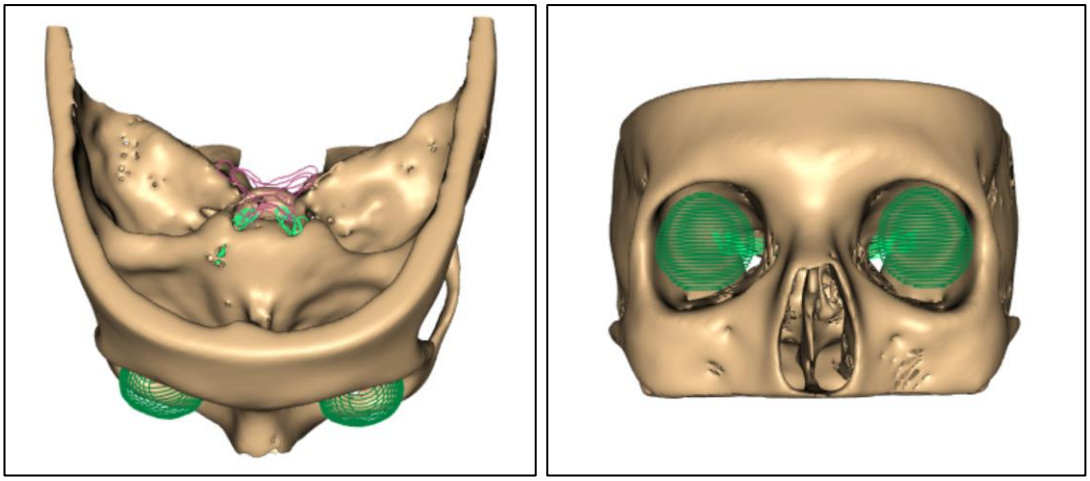
```
open3d (windowRect = c (100, 100, 600, 600))
display.3D.mesh (mesh.bones, color = "burlywood2", specular = "#404040")
# choose here your point of view in 3D representation
M <- par3d ()$userMatrix

open3d (windowRect = c (100, 100, 600, 600), userMatrix = M)
display.3D.mesh (mesh.keep, color = "burlywood2", specular = "#404040")
```

2.2.2 Volume restriction and transparency for 3D context

Even when we are interested in representing organs, it is often useful to add "patient" contour or bones as a context in a 3D image. For that purpose, consider using volume restriction in order to stay focused on the

important part of your representation, and volume transparency, if needed. Let us illustrate these features on simple examples:

```
to.keep <- select.names (S$roi.info$roi.pseudo,
                         roi.name = c ("nod", "nog", "chiasma", "oeilg", "oeild"))
boxes <- S$roi.info[to.keep, c("min.x", "max.x", "min.y", "max.y", "min.z", "max.z")]
pt.min <- c (min (boxes[, 1]), min (boxes[, 3]), min (boxes[, 5])) - 30
pt.max <- c (max (boxes[, 2]), max (boxes[, 4]), max (boxes[, 6])) + 30
CT.rest <- nesting.cube (CT, pt.min, pt.max)
bin.bones <- bin.from.vol (CT.rest, min = 100)
clust.bones <- bin.clustering (bin.bones)
sel.bones <- bin.from.vol (clust.bones, min = .5, max = 1.5)
mesh.bones <- mesh.from.bin (sel.bones, tol = .1, smooth.iteration = 10)

open3d (windowRect = c (100, 100, 600, 600))
display.3D.contour (S, roi.idx = to.keep)
display.3D.mesh (mesh.bones, color = "burlywood2", specular = "#404040")
```

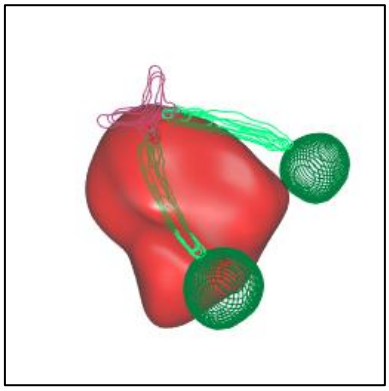


Meshes and contours

Both representation, wires for the RoI of optical path, and mesh for the skull can coexist in rgl. Here, we also illustrated the use of `nesting.cube` instruction whose role is to restrict the CT to a volume surrounding our volume of interest.

In this example, it would be interesting to see the Planed Tumor Volume in conjunction with optical pathway. Unfortunately, it surround the later. It is possible to add a slight transparency in order to make both coexist in the same drawing:

```
open3d (windowRect = c (100, 100, 600, 600))
display.3D.contour (S, roi.name = "oeild")
display.3D.contour (S, roi.name = "oeilg")
display.3D.contour (S, roi.name = "nod")
display.3D.contour (S, roi.name = "nog")
display.3D.contour (S, roi.name = "chiasma")
b.ptv <- bin.from.roi (CT, S, roi.name = "ptv")
m.ptv <- mesh.from.bin (b.ptv)
display.3D.mesh (m.ptv, col="#FF0000", alpha=.5)
```



Display of RoIs and PTV mesh with transparency

Beware that transparency in 3D is complex to handle, even for rgl. Use it with parsimony…

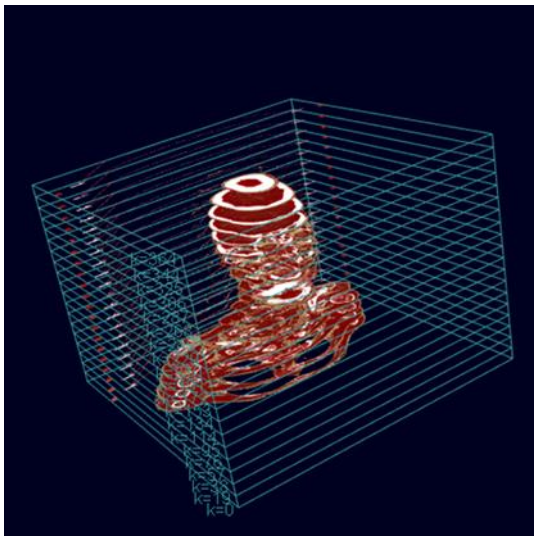
3 Advanced 3D representations

We will discuss, in this chapter, some advanced representations taking advantage of the power of rgl. Some technics are quite tricky but we can produce very expressive results with just a few lines of code…

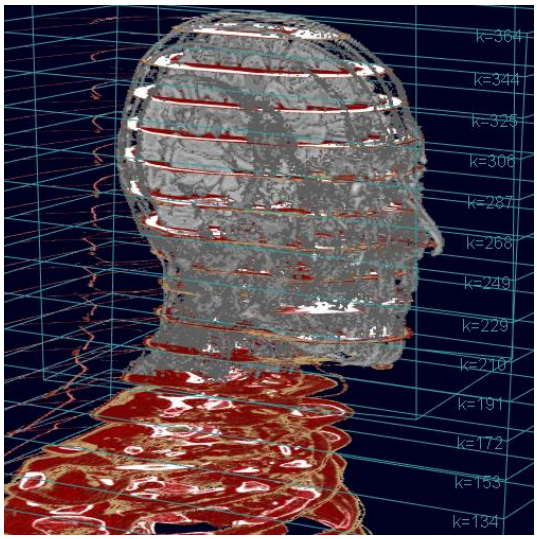
3.1 Displaying 3D stacks of images

As a first example, we can easily display 3D stacks of images as they were acquired in their frame of reference (FoR) and combine them into any FoR, for instance, the patient's one:

```
open3d (windowRect = c (100, 100, 600, 600))
bg3d ("#000020")
display.3D.stack (vol=CT, k.idx = CT$k.idx[seq(1, CT$n.ijk[3], length.out = 20)],
                  col= pal.RVV(200, alpha= c(rep(0,90), rep(1,110))),
                  breaks = seq(-1000, 1000, length.out = 201))                    # 1


display.3D.stack (MR, display.ref = CT$ref.pseudo, T.MAT = pat$T.MAT,
                  cube = TRUE, border = FALSE, ktext = FALSE)                     # 2
```



#1 CT image stack



#2 the MR stack added to the CT in the CT FoR

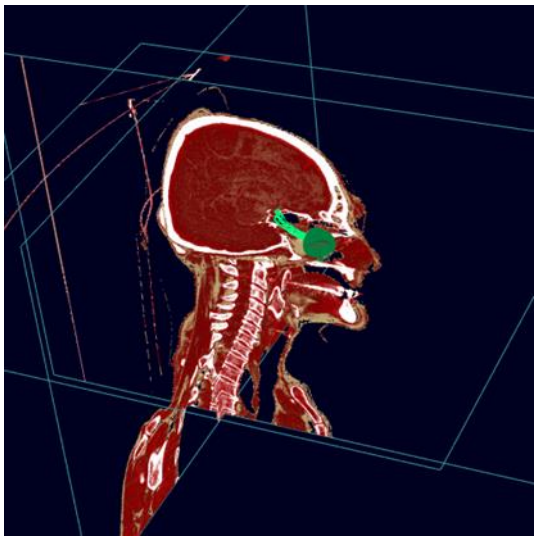
The first image (left) is the representation of CT stack ordered by plane index (k). By default, images are surrounded by their bounding box and the plane index (k) is displayed. Note the specific usage of transparency on the first colors of the RVV palette. It is used here to make the air invisible. On the second image (right), we added the MR stack. As the MR has its own FoR, we explicitly specify we want to display it in the FoR of the CT `CT$ref.pseudo` and transmit the `T.MAT` matrixes. We removed the surrounding image box (`border=FALSE`), images indexes (`ktext=FALSE`), and added the stack bounding box (`cube=TRUE`).

Of course, we can also add contours and so on. Beware that, too much information kills the information…

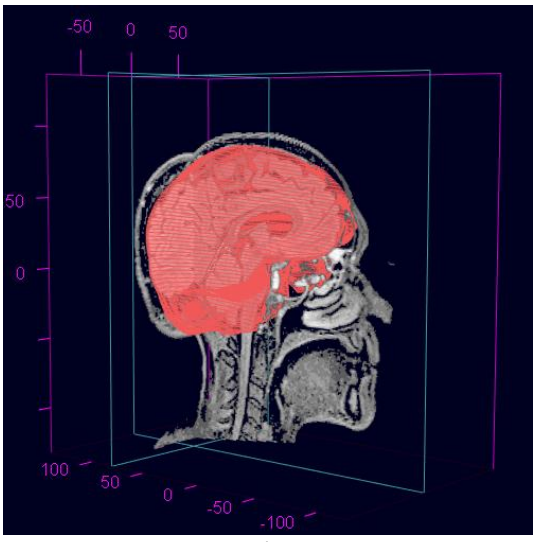
3.2 Incorporating images in 3D scenes

As mentioned earlier, 3D can be difficult to understand without context. In the following example, we explore the incorporation of images into the scene. These images can be CT slices, MR or dose, depending on your needs. In order to do that easily, we developed the `display.3D.sections` instructions. This instruction displays the transverse and/or sagittal and/or frontal views at a given intersection point:

```
open3d (windowRect = c (100, 100, 600, 600))
display.3D.sections(CT, cross.pt = c(0, 150, 0),
                    col= pal.RVV(200, alpha= c(rep(0,90), rep(1,110))),
                    breaks = seq(-1000, 1000, length.out = 201))
display.3D.contour (S, roi.name = c ("oeild", "nod")) #1
bg3d ("#000020")

open3d (windowRect = c (100, 100, 600, 600))
display.3D.sections(MR, cross.pt = c(0, 80, 0), trans = FALSE)
display.3D.contour(S, roi.name = "encephale", display.ref = MR$ref.pseudo, T.MAT = pat$T.MAT) #2
axes3d(col="magenta")
bg3d ("#000020")
```



#1 transverse, sagittal and frontal views



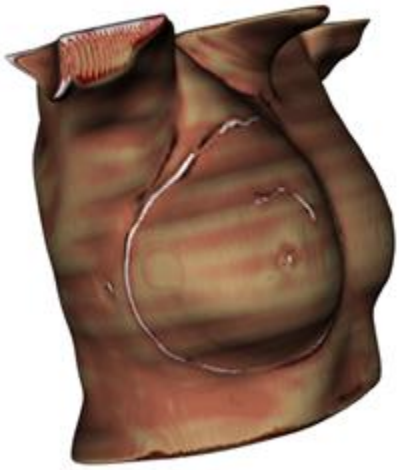
#2 representation in the MR FoR, transverse view removed

The color palette obeys the same scheme as `display.3D.stack` for transparency. We can select the desired view by setting `trans`, `sagi` and `front` arguments to `TRUE` (by default) or `FALSE`. On the left image, everything is displayed in the CT FoR. On the right image, we removed the transverse view and displayed the CT in its own FoR. For adding the brain, as it was contoured in the CT FoR, we just have to specify we want it displayed in the MR one using `display.ref = MR$ref.pseudo`. `axes3d` simply adds graduations on the representation. Note that, in `display.3d.sections`, the `cross.point` is expressed in x, y, z coordinates.

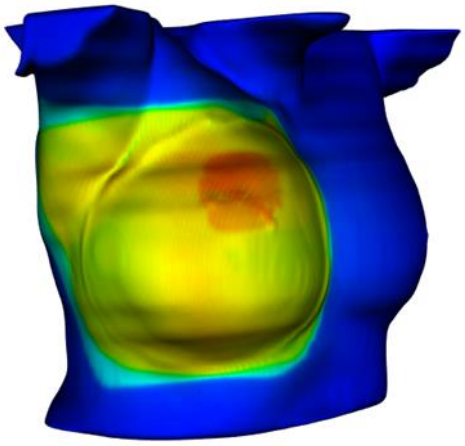
3.3 Surface coloring with information

We will introduce here a new kind of visualization specialized on surfaces. Organs are separated by interfaces, precisely their contours we can transform into mesh. It is tempting to display information on the mesh, not just only a uniform color, but for instance, the dose, or the CT. Roughly speaking, this kind of representation is what is the closest of skin dose, for instance. It can be extended to dose at a given depth, if needed, or even tissue identification under the skin. Let us illustrate this espadon feature by using the treatment of breast cancer. In this example, we will explain the whole code as it can give inspiration for new developments. In fact, most of it and far more is already implemented in espadon, as we will see in a vignette dedicated to meshes.

```
# patient loading. This patient has 2 rt-Doses that must be added together.
pat <- load.patient.from.dicom(choose.dir())
S <- load.obj.data (pat$rtstruct[[1]])
D1 <- load.obj.data (pat$rtdose[[1]])
D2 <- load.obj.data (pat$rtdose[[2]])
CT <- load.obj.data (pat$ct[[1]])

skin <- bin.from.roi(CT, S, roi.name = "body")
m.skin <- mesh.from.bin(skin, tol=.1, smooth.iteration = 20, smooth.type = "HClaplace")

t.center.normals <- apply (m.skin$mesh$it, 2, function (t) {
  A <- m.skin$mesh$vb[, t[1]]
  B <- m.skin$mesh$vb[, t[2]]
  C <- m.skin$mesh$vb[, t[3]]
  GC <- ((A + B + C) / 3)[1:3]
  N <- vector.product ((B-A)[1:3], (C-A)[1:3])
  S <- sqrt (sum (N^2))
  c (GC, N/S, S/2)
})
rownames (t.center.normals) <- c("G.x", "G.y", "G.z", "n.x", "n.y", "n.z", "S")
m.skin$mesh.param <- t.center.normals

get.mesh <- function (vol, mesh, depth) {
  xyz <- mesh$mesh.param[1:3, ] + depth * mesh$mesh.param[4:6, ]
  mesh$value <- get.value.from.xyz (t (xyz), vol, interpolate = TRUE)
  return (mesh)
}

m.skin <- get.mesh (CT, m.skin, 0)

cuts <- as.numeric (cut (m.skin$value, breaks=c (-Inf, seq (-1000, 1000, length.out=254), Inf)))
cuts[is.na (cuts)] <- 1
cols <- pal.RVV(255)[cuts]
open3d (windowRect = c (100, 100, 600, 600), zoom = .7)
shade3d (m.skin$mesh, meshColor = "faces", col=cols, specular = "#000000")                              #1
D <- vol.sum(D1, D2)
m.skin <- get.mesh (D, m.skin, -3)

cuts <- as.numeric (cut (m.skin$value, breaks=c (-Inf, seq (0, 52, length.out=254), Inf)))
cuts[is.na (cuts)] <- 1
cols <- hcl.colors(255, "YlOrBr", rev=TRUE)[cuts]
cols <- rainbow (255, rev=TRUE, start=0, end=4/6)[cuts]
open3d (windowRect = c (100, 100, 600, 600), zoom = .7)
shade3d (m.skin$mesh, meshColor = "faces", col=cols, specular = "#000000")                              #2
```



#1 Patient skin mesh with RVV palette.



#2 Skin dose at 3mm depth.

The beginning of the code is quite usual for the reader, now. We just get a binary of the patient RoI, then compute the mesh. A mesh in fact is the representation of a surface using small triangles. The computation is done by Rvcg:

```
str (m.skin)

List of 7
 $ frame.of.reference: chr "1.2.246.352.221.5186297184324858032688653652242671787"
 $ description       : chr " mesh"
 $ object.alias      : chr ""
 $ ref.pseudo        : chr "ref1"
 $ mesh              :List of 4
  ..$ vb     : num [1:4, 1:433164] -30 -109.6 -146.8 1 -30.6 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:4] "x0" "y0" "z0" ""
  .. .. ..$ : NULL
  ..$ it     : int [1:3, 1:866320] 1 2 3 3 4 1 4 5 1 4 ...
  ..$ normals: num [1:4, 1:433164] -0.125 -0.763 -0.634 1 -0.191 ...
  ..$ remface: int [1:866320] 0 0 0 0 0 0 0 0 0 0 ...
  ..- attr(*, "class")= chr "mesh3d"
```

What is important here is `mesh$it` which gives the index of the three vertices of each triangle. For each vertex, its coordinates are stored in the first three lines of `mesh$vb`, the last being filled with ones for reasons already explained. Normals to the vertex are also computed by Rvcg, but we will not use them. In fact, we need the normal to the triangles. For that, we exploit an important feature of meshes. Vertex composing a triangle are oriented. If A, B and C are the vertex (in the order stored in $mesh\$it$), then $\overrightarrow{AB} \wedge \overrightarrow{AC}$ points to the direction outside the volume. That is just what we need. In `t.center.normal`, we compute the center of gravity of each triangle, its normal and its surface and store them as new features in the mesh. Next step consists in getting the desired value at position $\vec{G} + depth \cdot \vec{n}$ using the `get.value.from.xyz` instruction we already explained. And that is all. If depth = 0, we are at the patient's skin, depth > 0, we go outside and for depth < 0, we go inside the patient.

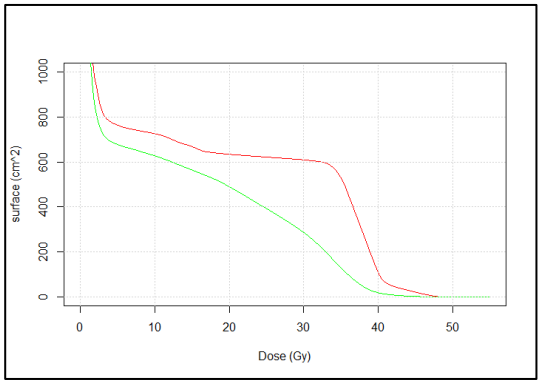
On the left figure, we are right on the skin and used the RVV palette. We can see the skin (in red), and fat (in yellow). The horizontal bands are probably due to surface sampling of the CT and resulting interpolation. We clearly see the ring (interpreted in white as its density is high), and dots used for positioning.

The treatment consisted in a uniform dose on the right breast with opposite fields, for a total of 43 Gy, and a complement of 10.7 Gy. The right picture shows the skin dose at 3mm depth. What is interesting here is the coupling between the irradiation technology and patient's morphology.

The technology is responsible for most of the pattern. Due to radiations build-up, the right and left parts of the breast receive "low" dose. There is an increase of dose on the vertical central part of the breast as the two opposite fields, here, have passed the build-up and combine. Note also the overdosing under the armpit. Depending on patient's morphology, such overdosing often appear at skin folds.

Represented as we did, skin dose contains a lot of information we did not exploit (more to come about this subject), but we can at least try to build the equivalent of DVH, namely, the Dose Surface Histogram. Maybe we could compare it to the "usual" skin DVH, restricted to 3 mm. Have a look at the following code where, we first compute the "negative expansion" of patient's ROI to get the DVH, and compare it to our new point of view:

```
skin.m3 <- bin.erosion (skin, 3)
inv.skin.m3 <- bin.inversion (skin.m3)
b.skin <- bin.intersection (skin, inv.skin.m3)
D.regrid <- vol.regrid (D, CT, interpolate = TRUE)
D.seq <- seq (0, 55, by=.1)
H <- histo.from.bin (D.regrid, b.skin, breaks = D.seq)
DVH <- histo.DVH (H)

D.cut <- as.numeric (cut (m.skin$value, D.seq))
surf <- sapply (1:length (D.seq), function (d) sum (m.skin$mesh.param[7, D.cut == d], na.rm = TRUE))

plot (D.seq, rev (cumsum (rev (surf))) / 100,
      ylab = "surface (cm^2)", xlab = "Dose (Gy)",
      type='l', ylim=c(1, 1000), col="red")
lines (DVH$mids, DVH$vol / .3, col="green")
grid ()
```



Dose-Surface Histogramm at 3 mm depth

The first six lines compute de "skin dose" the regular way. It consists in applying a "negative expansion" of (for instance) 3 mm to the "patient" RoI. Doing this, the volume is roughly the product of skin surface by selection thickness (3 mm). Our "true" skin dose is based on data produced earlier. We have, for every triangle, its surface, and the dose at the measurement point (here, we choose 1.5 mm under the skin, right in the middle of the regular method). To get the cumulative Dose Surface Histogram (DSH), we have to bin the measured dose using cut, for instance. Now, we know what bin is associated with a given dose. In order to measure the surface, we have to sum the surfaces of triangles sharing the same bin. Then we plot both cumulative DSH. In red, the exact surface (not volume) expressed in square centimeters, in green, the regular DVH we divided by 0.3 cm in order to get the approximate surface.

Strictly speaking, the measurement we performed on the mesh is possibly not the true surface; it depends on the `smooth.type` and `smooth.iteration` arguments of the `mesh.from.bin` function. In our example, the surface is 1.5 mm under the skin (it is in fact the skin surface). We could have done this measurement, but it involves beautiful mathematics far beyond this vignette. Maybe in a future one…

4 What we learned

Representation is always the basic for understanding data (in fact, R was initially created for that job). Using the powerful packages rgl and Rvcg, espadon is able to produce 3D figures in a few lines of codes. 3D representation is a little bit more complicated than 2D, but so much informative it worth to expense few hours to understand this world we try to make the easiest in espadon.

An interesting point of 3D representation is that it can be used to check for frame of reference or geometry problems frequently occurring during code development. As object of different families can be simultaneously displayed (contours, meshes, single images or stacks of images), it is easy to see if all are aligned or represent what the developer had in mind, for instance.

We took the opportunity to introduce the usage of mesh as a perfect description of surfaces. Meshes are far more complicated than usual objects as image stacks or binary volumes, but they open a new space for computation on volumes when their surface plays an important role (as the skin, for instance, but not only). A separate vignette will deal with this kind of representation. However, as representation is not the end of the story (just the beginning, in fact), we will see later how to get information using meshes, and produce features.

We illustrated lot of techniques for manipulating volumes, produce binary, clean them, and so on. We also made a dose accumulation as a patient had a boost on the tumor site. There are far more to say about all these subjects in future vignettes…

By the way, all these instruction (and others) have even more options, do not forget to have a look at the reference manual and other vignettes.

You can see also:

```
bin.clustering                  Binary volume clustering
bin.dilation                    Binary volume dilation.
bin.erosion                     Binary volume erosion
bin.from.roi                    Creation of a binary volume according to ROI
bin.from.vol                    Creation of a binary volume according to the voxel values of a volume
display.3D.contour              Display the 3D contours of the ROI
display.3D.mesh                 3D display of a mesh
display.3D.sections             Display 3D sections of a patient
display.3D.stack                Display in 3D the selected planes of an espadon volume
get.value.from.xyz              Voxel values on a selection of points.
mesh.from.bin                   Creation of a mesh according to a binary volume
vol.regrid                      Transform the grid of a volume class object into the grid of another.
vol.sum                         Sum of 2 volumes
```