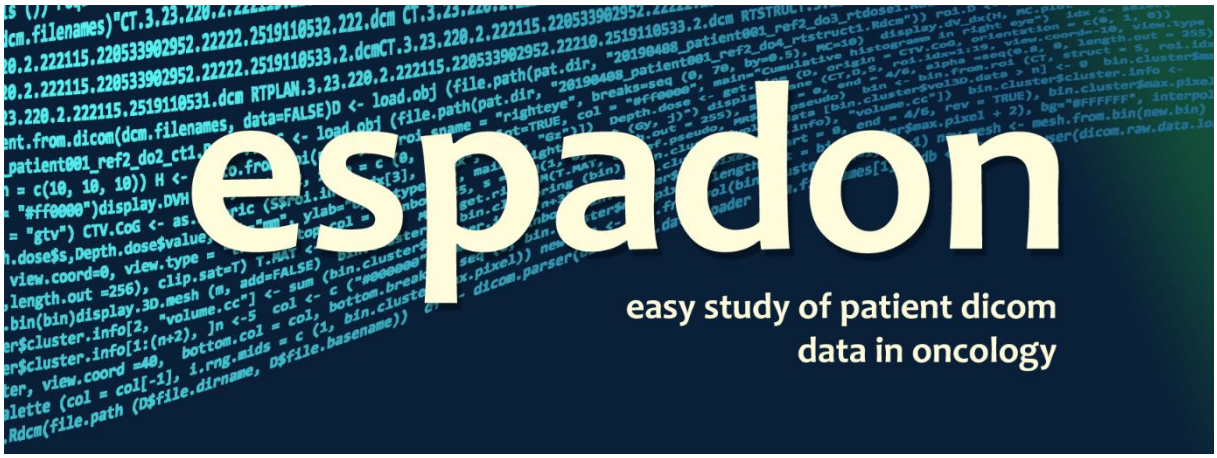


Creation of test objects

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

The development of robust algorithms requires much attention and adjustment. Of course these algorithms are supposed to work on real cases of CT, MR, dosimetry. However, understanding their behaviour, depending on the sampling, for example, and their possibilities is often easier on simple examples. We will see in this document how to create test objects allowing to confront the algorithms with simple but realistic cases.

We start by loading a patient's CT and we will see how to replace its contents with our test objects:

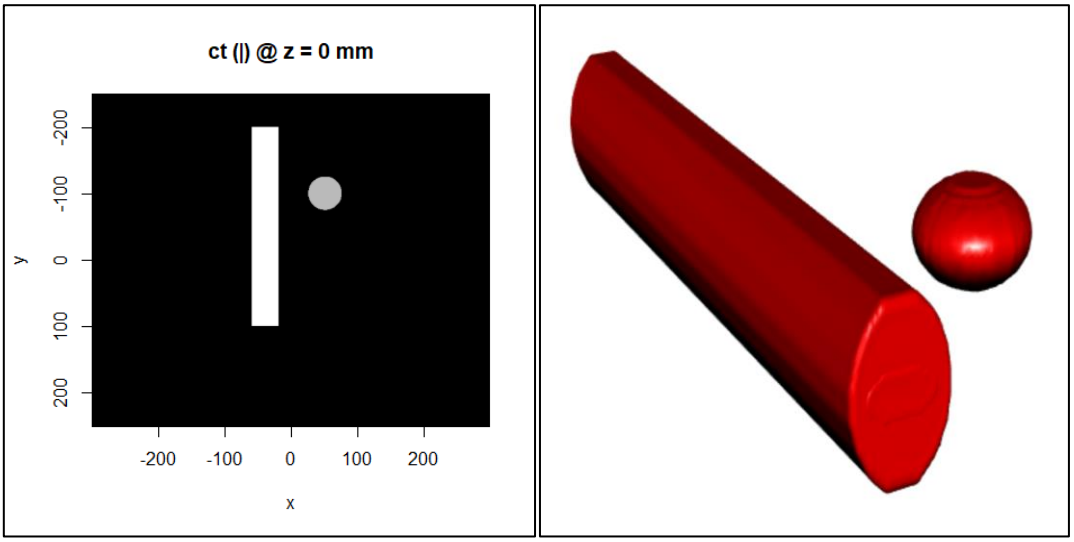
```
require (espadon)

pat.dir <- choose.dir ()
pat <- load.patient.from.dicom (pat.dir)
CT <- load.obj.data (pat$ct[[1]])
my.CT <- CT

CT$dxyz

[1] 0.976562 0.976562 2.500000
```

Firstly, we will see how to create simple objects and integrate them into an image. We will then see how to move them in space by means of rotation and translation, and how to modulate the images produced in intensity. Finally, we will see how to generate surfaces. All of these possibilities can be combined at will to build objects of varying complexity that will allow you to test the performance of your favourite algorithms.

2 Creating simple objects

The simplest way to create test objects with a known external surface equation is to test the set of points, relative to that equation, as illustrated in the following example, with a sphere of radius 25 mm, centred at (50, -100, 0), and an elliptical cylinder whose axis is parallel to the Y-axis, running from y=-200 mm to y=100 m, and of radius 20 mm along X-axis, 40 mm along Z-axis, centered at x=40, z=0:

```
my.CT$vol3D.data[,,] <- 0               # reset of my.CT content

idx <- which (my.CT$vol3D.data == 0)
xyz <- get.xyz.from.index (idx, my.CT)  # get x,y,z coordinates of every point in my.CT

sphere <- (xyz[, 1] - 50)^2 + (xyz[, 2] + 100)^2 + xyz[, 3]^2 < 25^2
my.CT$vol3D.data[sphere] <- 1           # the sphere
cylinder <- ((xyz[, 1] + 40) / 20)^2 + ((xyz[, 3]) / 40)^2 < 1 & xyz[, 2] >= -200 & xyz[, 2] <= 100
my.CT$vol3D.data[cylinder] <- 2         # the cylinder

my.CT$min.pixel <- 0                    # set min and max value
my.CT$max.pixel <- 2

display.plane (my.CT)

bin <- bin.from.vol(my.CT, min=0.5)
m <- mesh.from.bin(bin)
display.3D.mesh(m, col="#FF0000")
```



Two test objects created with the code above. They are sampled with the original CT grid.

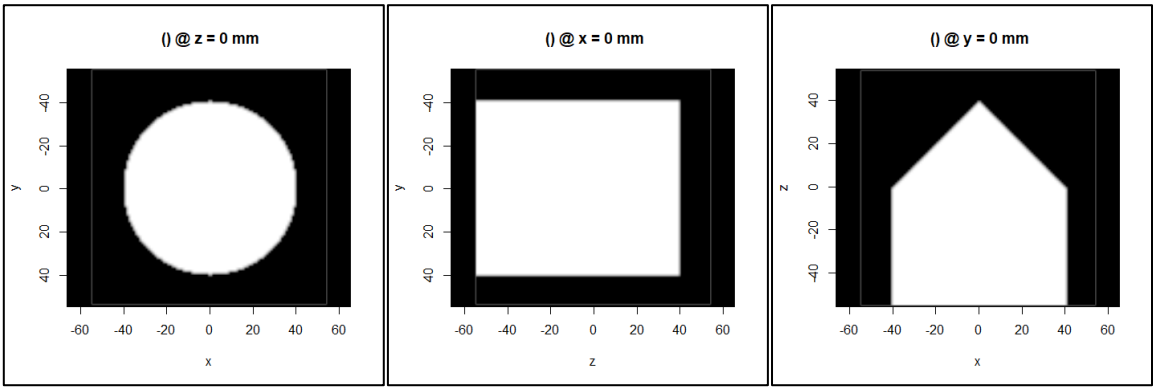
3 Adding rotations and translations and image modulation

The solution we have explored above has the merit of being simple. However, it requires the calculation of the volume membership of tens of millions of points. Here we will illustrate a solution that exploits the possibilities, in terms of changing reference frames and sampling, of espadon.

3.1 Rotations & translations

Let's build an object by limiting the amount of calculation to what is necessary:

```
obj <- vol.create (n.ijk = c(110, 110, 110), dxyz=c(1, 1, 1),
                   default.value = 0, ref.pseudo = "my.mysterious.object")
idx <- which (obj$vol3D.data == 0)
xyz <- get.xyz.from.index (idx, obj)

mysterious  <- (xyz[, 1] / 40)^2 + (xyz[, 2] / 40)^2 <= 1 &
  xyz[, 1] + xyz[, 3] < 40 & -xyz[, 1] + xyz[, 3] < 40
obj$vol3D.data[mysterious] <- 1
obj$min.pixel <- 0
obj$max.pixel <- 1

display.plane (obj)
display.plane (obj, view.type = "sagi")
display.plane (obj, view.type = "front")
```



The transverse, sagittal and frontal view of our mysterious test object. Can you guess the 3D object that gives a circle, a rectangle and a house in these three views? Answer below...

The main difference, compared to what we have already done, is to use the `vol.create` instruction by specifying the number of slice planes, the spatial steps, and also `ref.pseudo` which will be used in the following.

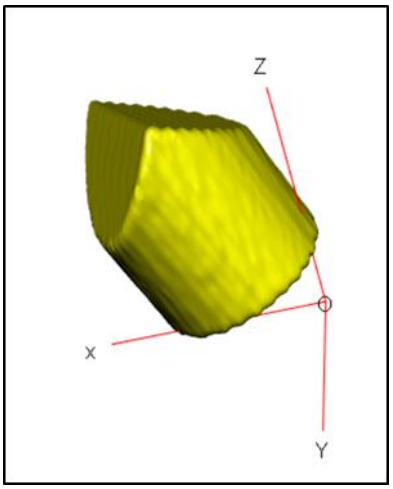
Now we might want to rotate this object along one or other of its axes, and translate it to finally sample it in the original CT frame of reference. To do this, we will construct a transfer matrix, as explained in the document "Computing cumulative dose" and use it to add it to the list of reference frames by exploiting the T.MAT mechanism. Finally, we will only have to resample the result on the CT grid.

```
build.T <- function (p) {
  Rx <- matrix (c(1, 0, 0, 0,
                  0, cos (p[1]), -sin (p[1]), 0,
                  0, sin (p[1]), cos (p[1]), 0,
                  0, 0, 0, 1), nrow = 4, byrow = TRUE)
  Ry <- matrix (c (cos (p[2]), 0, sin (p[2]), 0,
                   0, 1, 0, 0,
                   -sin (p[2]), 0, cos (p[2]), 0,
                   0, 0, 0, 1), nrow = 4, byrow = TRUE)
  Rz <- matrix (c (cos (p[3]), -sin (p[3]), 0, 0,
                   sin (p[3]), cos (p[3]), 0, 0,
                   0, 0, 1, 0,
                   0, 0, 0, 1), nrow = 4, byrow = TRUE)
  Tr <- matrix (c (1, 0, 0, p[4],
                   0, 1, 0, p[5],
                   0, 0, 1, p[6],
                   0, 0, 0, 1), nrow = 4, byrow = TRUE)
  Trans <- Tr %*% Rz %*% Ry %*% Rx

  return (Trans)
}

M <- build.T (p=c(c(0, 20, 30) * pi / 180, 50, 0, 70))

new.T.MAT <- ref.add (src.ref = CT$ref.pseudo,
                      orientation = as.vector (M[1:3, 1:2]), origin = as.vector (M[1:3, 4]),
                      new.ref.pseudo = "my.mysterious.object",
                      T.MAT = pat$T.MAT)

my.vol <- vol.regrid (obj, CT, T.MAT = new.T.MAT)
```

The three first paramters, when calling `build.T` concern rotations in degrees (around resp. X, Y and Z axes, in this order) and the three next are for translations (along X, Y and Z axes). When calling `ref.add`, we just have to provide the first two columns of the transformation matrix for `orientation`, and the last for `origin`. The new frame of reference must have the same name as we gave in `ref.pseudo` when creating the object. Then, we can resample the object on the CT grid using `vol.regrid` without forgetting to use the T.MAT mechanism.

It would be nice to see this mysterious object:

```
require (rgl)
open3d ()
lines3d (c (0, 100), c (0, 0), c (0, 0), col="red")
text3d (110, 0, 0, "x")
lines3d (c (0, 0), c (0, 100), c (0, 0), col="red")
text3d (0, 110, 0, "Y")
lines3d (c (0, 0), c (0, 0), c (0, 100), col="red")
text3d (0, 0, 110, "Z")
text3d (0,0,0, "O")

bin <- bin.from.vol (my.vol, min=0.5)
m <- mesh.from.bin (bin)
display.3D.mesh (m, col="yellow")
```

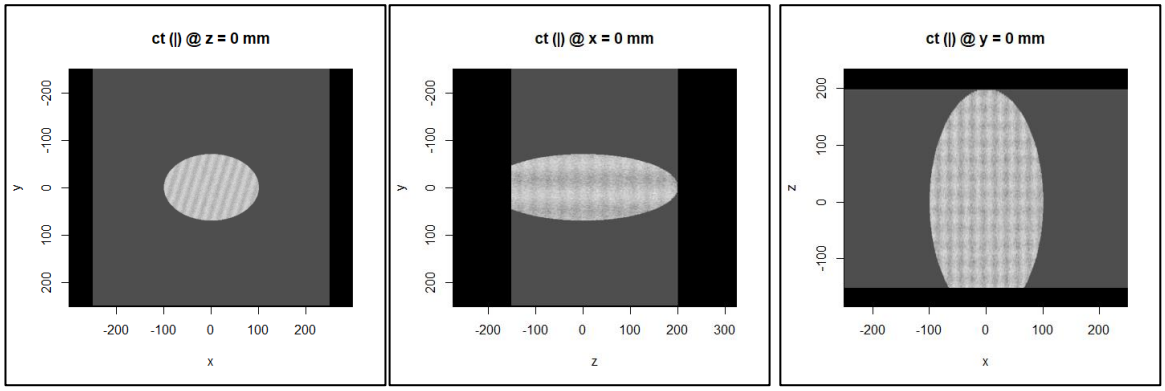


Well… It was simply an object that DIYers know well, a screwdriver head! This illustrates a great principle (of mathematics, physics, statistics...): the projection of an object often produces a biased vision of the problem, which is not so complicated in all its dimensions...

3.2 Intensity modulation and noise

Of course, our representations have for the moment consisted of using fixed quantities 0, 1, 2... but nothing prevents us from simulating continuous quantities and adding noise in order to have more realistic examples whose defects we can control. For instance:

```
my.CT$vol3D.data[,,] <- 0

idx <- which (my.CT$vol3D.data == 0)
xyz <- get.xyz.from.index (idx, my.CT)

spheroid <- (xyz[, 1]/100)^2 + (xyz[, 2]/70)^2 + (xyz[, 3]/200)^2 <= 1
xyz.s <- xyz[spheroid, ]
my.CT$vol3D.data[spheroid] <- 100 + 20 * sin (2*pi / 20 * (xyz.s[, 1] + .3 * xyz.s[, 2])) +
    10 * cos (2*pi / 30 * xyz.s[, 3]) + 5 * rnorm (nrow (xyz.s), 0, 4)


my.CT$min.pixel <- min (my.CT$vol3D.data)
my.CT$max.pixel <- max (my.CT$vol3D.data)

display.plane (my.CT)
display.plane (my.CT, view.type = "sagi")
display.plane (my.CT, view.type = "front")
```



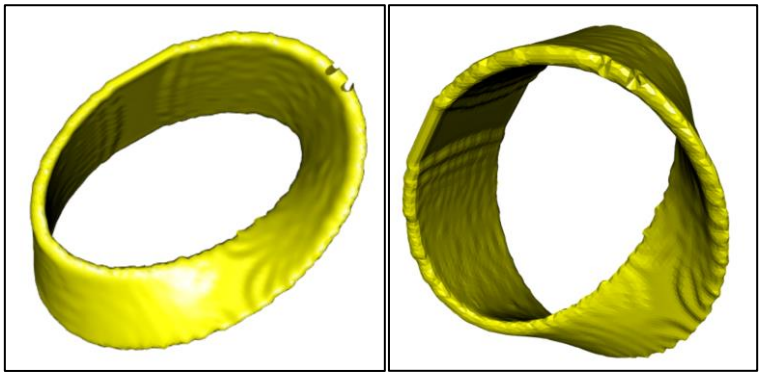
The three views of our "CT" with varying intensity and noise

4 Working with surfaces

The illustrated technics work as well on surfaces with subtle differences. Let's create a Moebius strip… The Moebius strip can be seen as a parametric surface whose x, y, z coordinate points depend on u and θ and a third parameter r which defines the radius.

```
r <- 29:31
u <- seq (-10, 10, by=.2)
theta <- seq (0, 360, by=.1) * pi / 180
rut <- as.matrix (expand.grid (r, u, theta))

moebius.strip <- function (rut) {
  x <- (rut[1] + rut[2] * cos (rut[3] / 2)) * cos (rut[3])
  y <- (1.5 * rut[1] + rut[2] * cos (rut[3] / 2)) * sin (rut[3])
  z <- 2 * rut[2] * sin (rut[3] / 2)
  return (c(x, y, z))
}

xyz <- apply (rut, 1, moebius.strip)

obj <- vol.create (n.ijk = c(200, 200, 200), dxyz=c(1, 1, 1),
                   default.value = 0, ref.pseudo = "my.mysterious.object")
ijk <- round (get.ijk.from.xyz (t (xyz), obj))
obj$vol3D.data[sweep (ijk, 2, c(-1,-1,-1))] <- 1

bin <- bin.from.vol (obj, min=0.5)
m <- mesh.from.bin (bin)
display.3D.mesh (m, col="yellow")
```

In this example, we fill the `vol3D.data` at locations provided by the x, y, z coordinates, themselves calculated using the parametrisation thru r, u, θ. As a surface, only u, θ are necessary, we added r in order to expand it a little. Note the use of `expand.grid` which calculates all possible triplets of r, u, θ. Once we have x, y, z coordinates, we have to convert them into i, j, k (DICOM raster coordinates), and add 1 to get espadon indexes (this is the role of `sweep`). Then, we get the Moebius strip:



The Moebius strip as calculated above (left image). Note the holes (top rigth) created by the meshing algorithm. They can be corrected using the mesh.repair instruction (right image).

This example explicitly shows the role of test objects, as the meshing algorithm has some problems that need to be corrected to go further. There, we just had to call `mesh.repair` to correct them as we can.