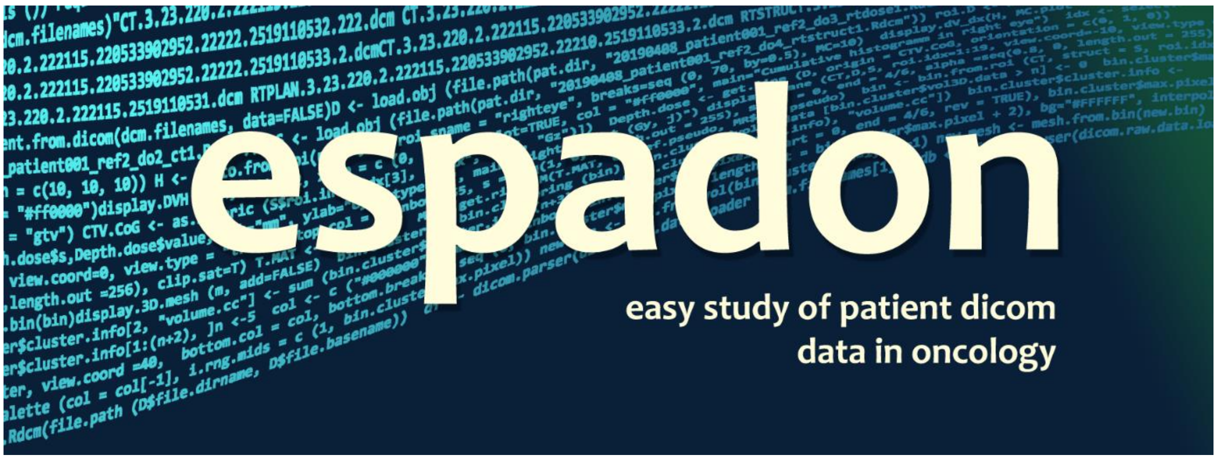


Creation of a CT-scan with test objects

C. Fontbonne and J.-M. Fontbonne | LPC Caen
Université de Caen Normandie, ENSICAEN, CNRS/IN2P3, LPC Caen UMR6534, F-14000 Caen, France Contact.espadon@lpccaen.in2p3.fr
https://espadon.cnrs.fr/

Table of content

1 Introduction ... 3

2 Background volume ... 3

3 Shapes available in the espadon package ... 4

4 Create your own shapes ... 5

4.1 Creation of "binary" object by selecting voxels ... 5

4.2 Intensity modulation and noise ... 8

5 Working with surfaces ... 9

6 Exporting espadon objects to DICOM files ... 10

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.

Firstly, we will create a background volume of modality "ct", name `my_CT`, in which we added all the shapes we want.

Espadon package provides simple objects. So, we will explain how to integrate this simple objects in `my_CT` scanner. Then, we will see two ways of generating shapes, integrating them into an image, and how to move then in space by means of rotation and translation. We will see how to modulating the images produced in intensity, then 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.

Finaly we will see how to export the espadon object my_CT into DICOM files.

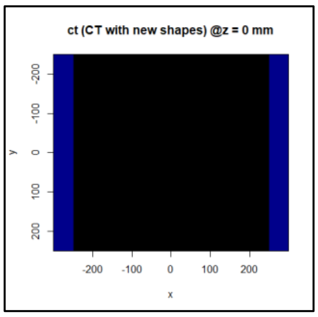
This document uses the espadon package version 1.9.0 or higher.

2 Background volume

We start by loading a patient's scanner and see how to replace its contents with our test objects. There are two ways to create a new CT, whose voxel values are set to -1000 HU.

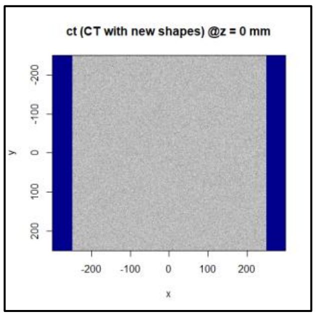
- Either by loading a patient's scanner and then copying it with the vol.copy() function: In this case, most of scanner feature are copied in the new CT (patient information , frame of reference…) : we simply change the content of the voxels and the min and max values to -1000.

```
library (espadon)

pat.dir <- "my_patient_dir"
pat <- load.patient.from.dicom (pat.dir)
CT <- load.obj.data (pat$ct[[1]])
my_CT <- vol.copy(CT, alias = "myCT", description = "CT with new shapes")
my_CT$vol3D.data[] <- -1000
my_CT$min.pixel <- my_CT$max.pixel <- -1000

plot (my_CT, bg="darkblue")
my_CT$dxyz

[1] 0.9765625 0.9765625 2.0000000
```



- Or by using vol.create() function. Thanks to this, the background intensity can be noised with value.sd argument. We specify the number of slice planes, the spatial steps, and also `ref.pseudo` (pseudonym of the frame of reference) which will be used in the following. Patient information will be added later, if required.

```
library (espadon)

my_CT <- vol.create(n.ijk =c(512,512,301),
                    dxyz = c(0.9765625, 0.9765625, 2.0000000),
                    mid.pt = c(0,0,0), modality = "ct",
                    default.value = as.integer(-1000), value.sd = 5.0,
                    ref.pseudo = "ref1",
                    alias = "myCT", description = "CT with new shapes")

plot (my_CT, bg="darkblue")
```



3 Shapes available in the espadon package

Suppose that we want to create:

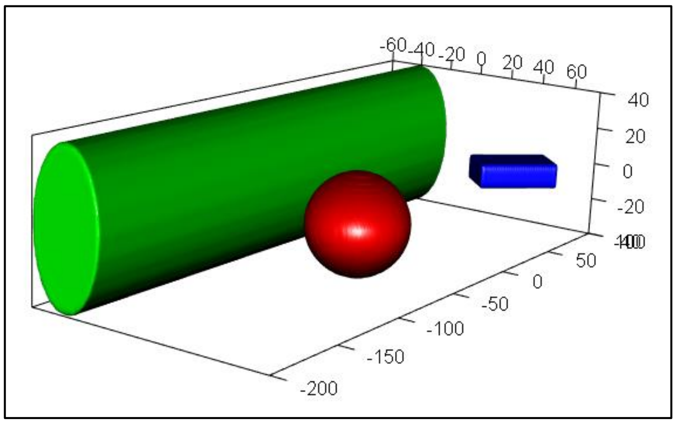
- a sphere of radius 25 mm, centred at (50, -100, 0) mm,
- an elliptical cylinder whose axis is parallel to the Y-axis, centered at (40, -50, 0) mm, with a height of 300 mm and of radius 20 mm along X-axis, 40 mm along Z-axis.
- A box of sides LxWxH = 42x25.3x12.6 mm, centered at (50.7, 50, 0) mm. Its length is oriented along the diagonal of the X and Y axes.

The first step is to create espadon object with a ‘binary’ or ‘weight’ modality. The binary modality contains TRUE voxels whose voxel center belongs to the shape, FALSE elsewhere. The ‘weight’ modality contains the percentage of voxel volume included in the shape. This gives results closer to reality.

```
sphere.center <- c (50,-100,0)
sphere.radius <- 25
bsphere <- bin.ellipsoid (my_CT, center = sphere.center, radius = sphere.radius)

cylinder.center <- c (-40,-50,0)
cylinder.height <- 300
cylinder.radius <- c (20,40)
cylinder.base.orientation <- c(1,0,0,0,0,1)
bcylinder <- bin.cylinder (my_CT, center = cylinder.center,
                          radius = cylinder.radius, height = cylinder.height,
                          orientation = cylinder.base.orientation)
box.center <- c(50.7, 50, 0)
box.sides <- c(42, 25.3, 12.6)
box.orientation <- c(1, 1, 0, -1, 1, 0) / sqrt(2)
bbox <- bin.cuboid(my_CT, side = box.sides, center = box.center,
                   orientation = box.orientation)
```

To visualize the shapes you've created, it's convenient to generate the meshes associated with the weight or binary objects, then display them in 3D.

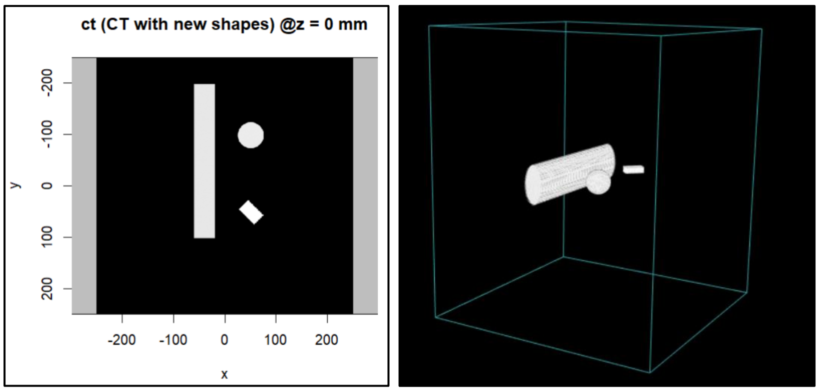
```
mesh.sphere <- mesh.from.bin(bsphere)
mesh.cylinder <- mesh.from.bin(bcylinder)
mesh.box <- mesh.from.bin(bbox)

display.3D.mesh(mesh.sphere, col ="#ff0000")
display.3D.mesh(mesh.cylinder, col ="#00ff00")
display.3D.mesh(mesh.box, col ="#0000ff")
rgl::axes3d()
```



Meshes of the 3 created shapes

Then we simply add the three shapes to the CT, giving each one a different intensity, with or without noise.

```
my_CT <- add.shape(my_CT, bsphere, shape.mean = 100, shape.sd = 5)
my_CT <- add.shape(my_CT, bcylinder, shape.mean = 50, shape.sd = 5)
my_CT <- add.shape(my_CT, bbox, shape.mean = 300, shape.sd = 2)

plot(my_CT, cut.interpolate = FALSE, bg="grey")

# plot all the slices in 3D
display.3D.stack(my_CT, k.id= my_CT$k.idx, border = FALSE)
rgl::bg3d("black")
```



Tranverse view, and slices stack of the new CT.

4 Create your own shapes

We have seen that to integrate a shape into your my_CT volume, we need to create a binary or weight modality object. There are two ways of generating this modality, either by creating it directly, or by first creating a "struct" modality, by defining the contours of our shape. The shape will be defined in the frame of reference named "my.mysterious.object.ref" as a pseudonym. It will be then translated and rotated to be defined in `my_CT` frame of reference.

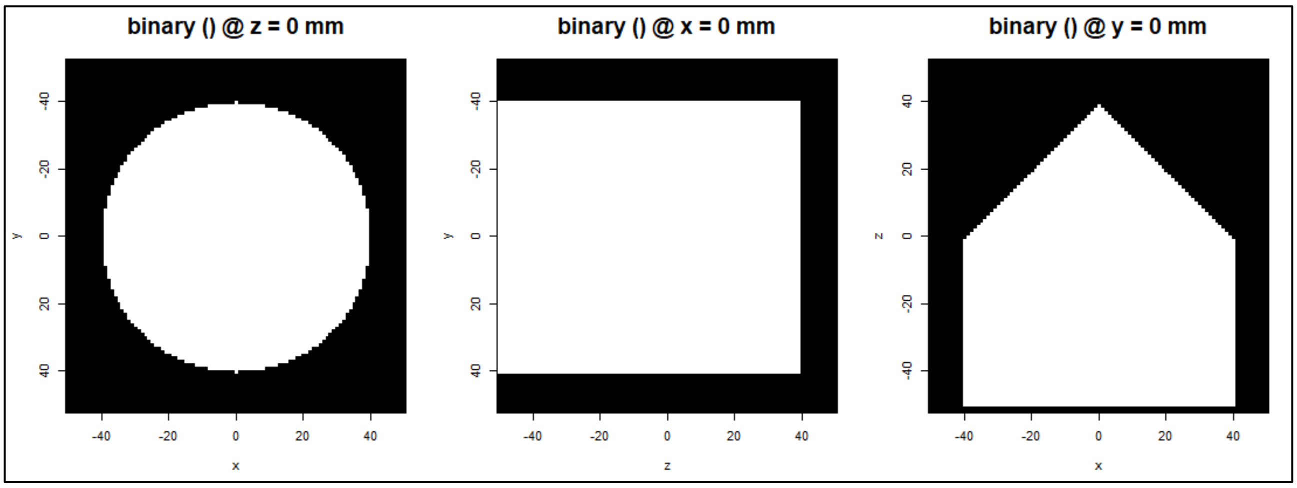
4.1 Creation of "binary" object by selecting voxels

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

```
obj <- vol.create (n.ijk = c(111, 111, 111), dxyz=c(1, 1, 1), default.value = FALSE,
                    ref.pseudo = "my.mysterious.object.ref")
obj$modality
# voxels coordinates
xyz <- get.xyz.from.index (1:length(obj$vol3D.data), obj)
```

We identify the voxels in our shape using the following conditions on the voxels' xyz coordinates.

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

par (mfrow = c (1, 3))
display.plane (obj, interpolate = FALSE)
display.plane (obj, view.type = "sagi", interpolate = FALSE)
display.plane (obj, view.type = "front", interpolate = FALSE)
```



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...

4.2 Creation of "weight" object by defining the contours of the shape

The precedent shape can be defined by its contours. It will give results closer to reality, especially if we want to rotate the shape. The list `contour.list` contains 3-column matrices or dataframes providing contour coordinates in z steps.

```
z<-obj$xyz0[,3]
theta<- seq(0,2*pi,length.out = 200)
outline <- round(as.matrix(data.frame(x=40*cos(theta), y = 40*sin(theta), z=NA)),6)
contours.list <- lapply(z, function(zi){
  outline_ <- outline
  outline_[,3] <- zi
  if (zi>40) return(NULL)
  outline_[ outline_[, 1] + outline_[, 3] >= 40,1] <- 40 - zi
  outline_[-outline_[, 1] + outline_[, 3] >= 40,1] <- zi - 40
  outline_
})
contours.list <- contours.list[!sapply(contours.list, is.null)]


head(contours.list[[1]])


            x        y   z
[1,] 40.00000 0.000000 -55
[2,] 39.98006 1.262742 -55
```

```
[3,] 39.92027 2.524225 -55
[4,] 39.82069 3.783192 -55
[5,] 39.68141 5.038388 -55
[6,] 39.50258 6.288562 -55
```

From this contours, we can generate a struct espadon object, with a unique region of interest (ROI), and then generate a volume espadon object of modality “weight”.

```
struct.mysterious <- struct.create (contours.list, roi.name="mysterious",
                                    roi.nb = 1,
                                    roi.color = "#ff0000",
                                    roi.type = "",
                                    ref.pseudo = obj$ref.pseudo,
                                    frame.of.reference = obj$frame.of.reference,
                                    alias="", description = NULL)
obj <- bin.from.roi (obj, struct.mysterious, modality="weight")
```

4.3 Translation and rotation

We may now 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.

```
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 = my_CT$ref.pseudo,
                      orientation = as.vector (M[1:3, 1:2]), origin = as.vector (M[1:3, 4]),
                      new.ref.pseudo = obj$ref.pseudo)
```

The three first parameters, 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, If we only have the binary object `obj` created in § 4.1, we can resample the object on the CT grid using `vol.regrid` without forgetting to use the T.MAT mechanism, and the “Average” method, which involves calculating the average of the initial voxels included in a voxel in the new reference frame.

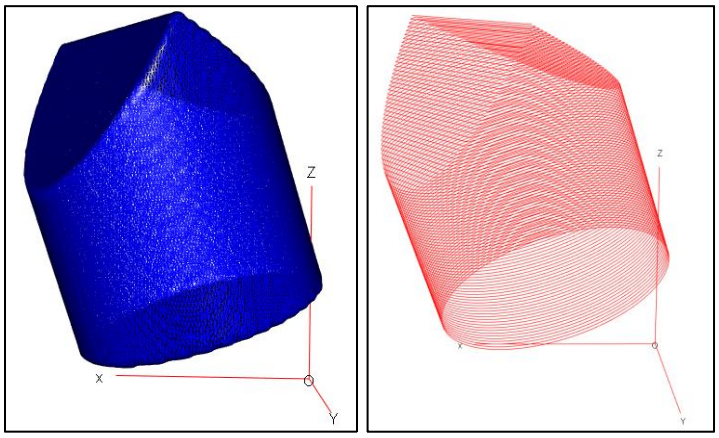
```
my.bvol <- vol.regrid (obj, my_CT, T.MAT = new.T.MAT, method = "Av")
```

If we have the struct object S created in § 4.2, it is more efficient to transfer it to the new frame of reference before creating a weight or binary object.

```
S <- struct.in.new.ref (struct.mysterious, new.ref.pseudo = my_CT$ref.pseudo, T.MAT = new.T.MAT )
my.bvol <- bin.from.roi (my_CT, S, modality="weight")
```

It would be nice to see this mysterious object:

```
require (rgl)
m <- mesh.from.bin (my.bvol)
open3d ()
lines3d (c (0, 60), c (0, 0), c (0, 0), col="red")
text3d (65, 0, 0, "x")
lines3d (c (0, 0), c (0, 60), c (0, 0), col="red")
text3d (0, 65, 0, "Y")
lines3d (c (0, 0), c (0, 0), c (0, 60), col="red")
text3d (0, 0, 65, "Z")
text3d (0,0,0, "O")
wire3d (m$mesh, col="blue")

open3d ()
lines3d (c (0, 60), c (0, 0), c (0, 0), col="red")
text3d (65, 0, 0, "x")
lines3d (c (0, 0), c (0, 60), c (0, 0), col="red")
text3d (0, 65, 0, "Y")
lines3d (c (0, 0), c (0, 0), c (0, 60), col="red")
text3d (0, 0, 65, "Z")
text3d (0,0,0, "O")
display.3D.contour(S)
```



Mesh and contours of the mysteious object

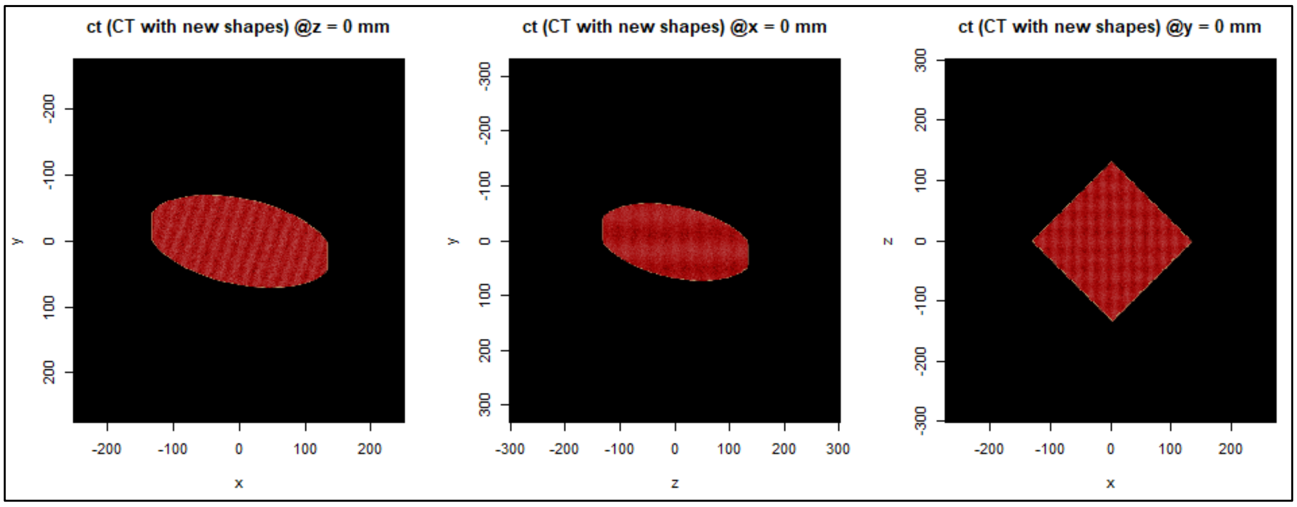
Well… The mysterious object was simply an object that DIYers know well, a screwdriver head!

4.4 Intensity modulation and noise

We saw that the add.shape() function allows us to add our binary or weight object in my_CT scanner object, with a gaussian noise.

Here we'll explain how to simulate other realistic defects that we can control. For example, we'll create a 'weight' modality cylinder, which we'll include in the my_CT scanner, initialized to -1000 HU. The intensity of the voxels specified by the cylinder is then modulated and added in proportion to the weight of the cylinder's voxels.

```
my_CT$vol3D.data[] <- -1000
bcylinder <- bin.cylinder (my_CT, center = c(0,0,0), radius = c(100,70),
                           height = 200, orientation= c(1,1,1,-1,1,-1) / sqrt(3))

idx <- which (bcylinder$vol3D.data > 0)
xyz.c <- get.xyz.from.index (idx, bcylinder)
my_CT$vol3D.data[idx] <- (100 + 20 * sin (2*pi / 20 * (xyz.c[, 1] + .3 * xyz.c[, 2])) +
  10 * cos (2*pi / 30 * xyz.c[, 3]) + 5 * rnorm (nrow (xyz.c), 0, 4)) * bcylinder$vol3D.data[idx] +
  (1-bcylinder$vol3D.data[idx]) * my_CT$vol3D.data[idx]

my_CT$min.pixel <- min (my_CT$vol3D.data, na.rm = TRUE)
my_CT$max.pixel <- max (my_CT$vol3D.data, na.rm = TRUE)

par(mfrow=c(1,3))
plot (my_CT, view.type = "trans", view.coord = 0, cut.interpolate = TRUE, col = pal.RVV(256))
plot (my_CT, view.type = "sagi", view.coord = 0, cut.interpolate = TRUE, col = pal.RVV(256))
plot (my_CT, view.type = "front", view.coord = 0, cut.interpolate = TRUE, col = pal.RVV(256)
```


ct (CT with new shapes) @z = 0 mm
ct (CT with new shapes) @x = 0 mm
ct (CT with new shapes) @y = 0 mm
x
y
z

The three views of our "CT" with varying intensity and noise. Here, the 'Realistic Volume Vizualization' palette, developped by J.C. Silverstein and al, is used.

5 Working with surfaces

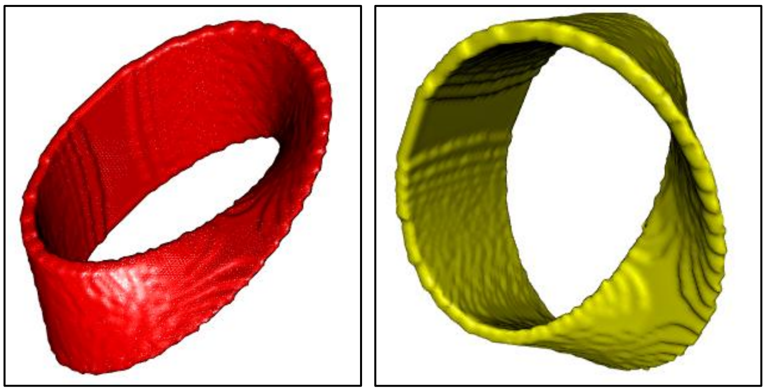
To create surface, it is necessary to use the voxel selection technique work. Let's create a Moebius strip.

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.

```
library(espadon)
moebius.strip <- function(rut){
  as.matrix(data.frame(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)))
}

obj <- vol.create (n.ijk = c(201, 201, 201), dxyz=c(1, 1, 1),
                   mid.pt = c(0,0,0),
                   default.value = FALSE, ref.pseudo = "moebius")

range.r <- c(29, 31)
range.u <- c(-10, 10)
range.tetha <- c(0, 2*pi)
```

```
step.u <- step.r <- min(obj$dxyz[1:3])/2.5/2
step.theta <- min(obj$dxyz[1:3])/(2*max(range.r) + max(range.u))/2

r <- seq(29,31, by=step.r)
u <- seq (-10, 10, by=step.u)
theta <- seq (0, 2*pi, by=step.theta)

r_u_theta <-  as.matrix (expand.grid (r, u, theta))
xyz <- moebius.strip(r_u_theta)

ijk <- round (get.ijk.from.xyz (xyz, obj))
obj$vol3D.data[ijk + 1] <- TRUE
obj$max.pixel <- TRUE
obj$min.pixel <- FALSE

m <- mesh.from.bin (obj)
rgl::wire3d (m$mesh, col="red")
rgl::open3d ()
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. It is important to choose the steps of these three parameters carefully to target all the voxels on the surface. 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. Then, we get the Moebius strip :



The Moebius strip as calculated above.

6 Exporting espadon objects to DICOM files

Now that we've created our new espadon object containing the desired shapes, it's time to export them in DICOM format.

We need to check that all items we want in the DICOM files appear in `my_CT`. We must to take care with patient information, and fill them if necessary.

```
# patient NIP
my_CT$patient
# patient name
my_CT$patient.name
# patient birthday
my_CT$patient.bd
# patient sex
my_CT$patient.sex
```

To export `my_CT` espadon object to DICOM files with the name “myCT” in the "CT_with_shape" directory, simply run :

```
export(my_CT,file.dirname="CT with shape", file.name ="myCT")

number of files created : 301
```

Note that if `frame.of.reference` item in my_CT object is not filled in, it will be automatically generated by the export() function. Similarly, DICOM file names will be automatically assigned if the `file.name` argument of the export() function remains `NULL`.