Package 'espadon'

April 4, 2023

Type Package

Title Easy Study of Patient DICOM Data in Oncology

Version 1.3.2

Description Exploitation, processing and 2D-3D visualization of DICOM-RT files (structures, dosimetry, imagery) for medical physics and clinical research, in a patient-oriented perspective.

License GPL-3

URL https://espadon.cnrs.fr

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.0

Imports colorspace,
DT,
graphics,
grDevices,
igraph,
js,
mathjaxr,
Matrix,
methods,
misc3d,
openxlsx,
progress,
qs,
Rcpp (>= 1.0.8.3),
Rdpack,
rgl (>= 0.108.3.2),
Rvcg (>= 0.21),
shiny,
shinyWidgets,
sp,
stats

LinkingTo Rcpp

RdMacros Rdpack,
mathjaxr

Depends R (>= 4.1.0)

Suggests knitr,
rmarkdown

VignetteBuilder knitr

R topics documented:

`add.margin` *Adding or removing a margin to a volume*

Description

The `add.margin` function adds or subtracts a margin of the rectangular parallelepiped circumscribed by a volume.

Usage

```
add.margin(vol, xyz.margin, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object.
`xyz.margin`	Vector of the 3 positive or negative x, y and z margins in mm, in the frame of reference of volume cut planes.
`alias`	Character string, `$alias` of the created object
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `vol$description`

Value

Returns a "volume" class object (see espadon.class for class definitions), in which 3D volume is restricted or increased by the requested margins. If the created volume exceeds the initial volume, new voxels are set to `NA`.

See Also

nesting.cube, nesting.roi and nesting.bin.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "ct", roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]

# Calculation of new volumes decreased by 10 mm in all directions.
new.CT <- add.margin (CT, xyz.margin = c (-10, -10, 10), alias = "new CT")
# display of the CT before and after, in the middle plane
z.mid <- apply (get.extreme.pt (CT), 1, mean)[3]
display.plane (bottom = CT, view.coord = z.mid, bottom.col = pal.RVV(1000),
               bottom.breaks = seq(-1000, 1000, length.out = 1001),
               bg = "#00ffff", interpolate = FALSE)
display.plane (bottom = new.CT, view.coord = z.mid, bottom.col = pal.RVV(1000),
               bottom.breaks = seq(-1000, 1000, length.out = 1001),
               bg = "#00ffff", interpolate = FALSE)
```

`bin.closing` *Binary volume closing*

Description

The `bin.closing` function performs a morphological operation of closing, using a sphere, on a "volume" class object of "binary" modality. Closing is useful for :

- filling holes that are smaller than the `radius`,
- merging two shapes close to each other.

Usage

```
bin.closing(vol, radius = 10, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object, of "binary" modality
`radius`	Positive number, in millimeters. By default, radius = 10.
`alias`	Character string, `$object.alias` of the created object.
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `paste (vol$object.alias, "closing r =", radius)`.

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as `vol`, in which `$vol3D.data` has been transformed by the closing operation.

Note

Closing can be time consuming, try to reduce the `binary` volume to the strict minimum, before any operations.

See Also

bin.dilation, bin.erosion, bin.opening, add.margin, nesting.cube.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "mr", roi.name = "",
                             dxyz = rep (step, 3))
MR <- patient$mr[[1]]

# generation of a binary volume
b <- bin.from.vol(MR, min = 15, max = 30)

b.closing <- bin.closing (b, radius = step)
display.plane (bottom = MR, top = b, main = "Before closing",
               view.coord = -20, interpolate = FALSE)
display.plane (bottom = MR, top = b.closing, main = "After closing",
               view.coord = -20, interpolate = FALSE)
```

---

bin.clustering *Binary volume clustering*

Description

The `bin.clustering` function groups and labels TRUE voxels that have a 6-connectivity (i.e. sharing a common side).

Usage

```
bin.clustering(vol, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object, of `"binary"` modality
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `paste (vol$object.alias,"clustering")`

Value

Returns a "volume" class object (see espadon.class for class definitions), of `"cluster"` modality. This object contains the `$cluster.info` field, detailing the label and volumes in cm^3 of the different clusters. Note that the label "0" is used for the background.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "ct",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]

# generation of a binary volume
b <- bin.from.vol(CT, min = -80, max = 20)

# Display of the n = 3 largest volumes
n <- 3
cluster.b <- bin.clustering (b)

col <- c ("#00000000", rainbow (n))
breaks <- seq (0, n, length.out = n+2)
display.plane (CT, top = b, main = "Before clustering",
               view.coord = 50, top.col = col, top.breaks = breaks,
               interpolate = FALSE)
display.plane (CT, top = cluster.b, main = "After clustering",
               view.coord = 50, top.col = col, top.breaks = breaks,
               interpolate = FALSE)
```

`bin.dilation` *Binary volume dilation*

Description

The `bin.dilation` function enlarges a "volume" class object, of `"binary"` modality, by means of convolution with a sphere. Dilation is useful for :

- filling holes that are smaller than the `radius`,
- enlarging capes,
- filling narrow channels,
- merging two shapes close to each other.

Usage

```
bin.dilation(vol, radius = 10, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object, of `"binary"` modality
`radius`	Positive number, in millimeters. By default, radius = 10.
`alias`	Character string, `$object.alias` of the created object.
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `paste (vol$object.alias, "dilataion r =", radius)`.

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as vol, in which the selected volume has been enlarged by the radius.

Note

Dilation can be time consuming, try to reduce the binary volume to the strict minimum, before any operations.

See Also

bin.erosion, bin.opening, bin.closing, add.margin, nesting.cube.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "mr", roi.name = "",
                             dxyz = rep (step, 3))
MR <- patient$mr[[1]]

# generation of a binary volume
b <- bin.from.vol(MR, min = 15,max = 30)

b.dilation <- bin.dilation (b, radius = step)
display.plane (bottom = MR, top = b, main = "Before dilation",
               view.coord = -20, interpolate = FALSE)
display.plane (bottom = MR, top = b.dilation, main = "After dilation",
               view.coord = -20,interpolate = FALSE)
```

---

bin.erosion *Binary volume erosion*

Description

The bin.erosion function decreases a "volume" class object, of "binary" modality, by means of convolution with a sphere. Erosion is useful for :

- removing volumes that are smaller than the radius,
- eliminating narrow capes,
- enlarging channels,
- turning peninsulas into islands.

Usage

```
bin.erosion(vol, radius = 10, alias = "", description = NULL)
```

Arguments

vol	"volume" class object, of "binary" modality
radius	Positive number, in millimeters. By default, radius = 10.
alias	Character string, $object.alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be set to paste (vol$object.alias, "erosion r =", radius).

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as vol, in which the selected volume has been reduced by the radius.

Note

Erosion can be time consuming, try to reduce the binary volume to the strict minimum, before any operations.

See Also

bin.dilation, bin.opening, bin.closing, add.margin, nesting.cube.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "mr", roi.name = "",
                             dxyz = rep (step, 3))
MR <- patient$mr[[1]]

# generation of a binary volume
b <- bin.from.vol(MR, min = 15,max = 30)

b.erosion <- bin.erosion (b, radius = step)
display.plane (bottom = MR, top = b, main = "Before erosion",
               view.coord = -20, interpolate = FALSE)
display.plane (bottom = MR, top = b.erosion, main = "After erosion",
               view.coord = -20, interpolate = FALSE)
```

---

bin.from.roi *Creation of a binary volume according to RoI*

Description

The bin.from.roi function creates a "volume" class object, of "binary" modality, in which all the voxels of a RoI are set to TRUE.

Usage

```
bin.from.roi(
  vol,
  struct,
  roi.name = NULL,
  roi.sname = NULL,
  roi.idx = NULL,
  T.MAT = NULL,
  within = TRUE,
  alias = "",
  description = NULL
)
```

Arguments

vol	"volume" class object.
struct	"struct" class object.
roi.name	Vector of exact names of the RoI in the struct object. By default roi.name = NULL. See Details.
roi.sname	Vector of names or parts of names of the RoI in the struct object. By default roi.sname = NULL. See Details.
roi.idx	Vector of indices of the RoI that belong to the struct object. By default roi.idx = NULL. See Details.
T.MAT	"t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If T.MAT = NULL, struct$ref.pseudo must be equal to vol$ref.pseudo or set to NULL.
within	Boolean, defaults to TRUE. If within = TRUE, the contours included in a RoI are managed, depending on their $level field. If within = FALSE, only the $level = 0 fields of the RoI are used (i.e. only the external outlines).
alias	Character string, $alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be set to struct$roi.info$roi.pseudo[roi.idx].

Details

roi.name, roi.sname, and roi.idx must select only one RoI.

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as vol, in which the voxels in the RoI are set to TRUE.

See Also

bin.from.vol.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 3
patient <- toy.load.patient (modality = c("ct", "rtstruct"),
                             roi.name =  c("eye", "optical nerve", "brain"),
```

```
                              dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

# "optical nerve" binary without inclusions management
bin <- bin.from.roi (CT, struct = S, roi.sname = "left optical",
                     alias = "left_optical_nerve")
display.plane (CT, top = bin, struct = S,
               view.coord = S$roi.info[S$roi.info$roi.pseudo == "leftopticalnerve",]$Gz,
               legend.shift = -80, interpolate = FALSE, main = "Left nerve selection")

## Not run:
# with a smaller step
step <- 1
patient <- toy.load.patient (modality = c("ct", "rtstruct"),
                             roi.name =  c("eye", "optical nerve", "brain"),
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

# "optical nerve" binary without inclusions management
bin <- bin.from.roi (CT, struct = S, roi.sname = "left optical",
                     alias = "left_optical_nerve", within = FALSE)
display.plane (CT, top = bin, struct = S,
               view.coord = S$roi.info[S$roi.info$roi.pseudo == "leftopticalnerve",]$Gz,
               legend.shift = -80, interpolate = FALSE, main = "Left nerve selection")

# "optical nerve" binary with inclusions management
bin <- bin.from.roi (CT, struct = S, roi.sname = "left optical",
                     alias = "left_optical_nerve", within = TRUE)
display.plane (CT, top = bin, struct = S,
               view.coord = S$roi.info[S$roi.info$roi.pseudo == "leftopticalnerve",]$Gz,
               legend.shift = -80, interpolate = FALSE, main = "Left nerve selection")

## End(Not run)
```

`bin.from.vol` *Creation of a binary volume according to the voxel values of a volume*

Description

The `bin.from.vol` function creates a "volume" class object, of "binary" modality, in which the voxels fulfilling a condition on their value are selected.

Usage

```
bin.from.vol(
  vol,
  min = -Inf,
  max = Inf,
  in.selection = TRUE,
  alias = "",
  description = NULL
)
```

Arguments

vol	"volume" class object.
min	Minimum value of the selected voxel. Default to -Inf.
max	Maximum value of the selected voxel. Default to +Inf.
in.selection	Boolean, defaults to TRUE. If in.selection = FALSE, the selected pixels are those whose value is not between min and max.
alias	Character string, $alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be set to paste (min, vol$object.alias, max, sep = "<=") or if in.selection = FALSE, paste ("!(", description, ")").

Value

Returns a "volume" class object of "binary" modality, with the same grid as vol, in which the selected voxels (i.e. set to TRUE) are those satisfying the following conditions :

- If in.selection = TRUE, then min <= vol$vol3D.data <= max.
- If in.selection = FALSE, then vol$vol3D.data < min or max < vol$vol3D.data

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 3
patient <- toy.load.patient (modality = "ct", roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]

bin.bone <- bin.from.vol (CT, min = 300, max = 3000, alias = "bone")
display.plane (CT, top = bin.bone, interpolate = FALSE)
```

---

bin.intersection *Intersection of two binaries*

Description

The bin.intersection function creates a "volume" class object, of "binary" modality, representing the intersection (logical AND) of two binary objects.

Usage

```
bin.intersection(vol1, vol2, alias = "", description = NULL)
```

Arguments

vol1, vol2	"volume" class objects, of "binary" modality.
alias	Character string, $alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be set to paste (vol1$object.alias, "&", vol2$object.alias).

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as `vol1` and `vol2`, intersection of `vol1` and `vol2`.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 3
patient <- toy.load.patient (modality = c("mr", "rtstruct"),
                             roi.name = c("brain", "labyrinth processing unit"),
                             dxyz = rep (step, 3))
MR <- patient$mr[[1]]
S <- patient$rtstruct[[1]]

z.brain <- S$roi.info$Gz[S$roi.info$roi.pseudo == "brain"]

# Try to discriminate the processing unit with binary selections
bin.brain <- bin.from.roi (MR, struct = S, roi.name = "brain",
                           alias = "brain", T.MAT = patient$T.MAT)
bin.pu.density <- bin.from.vol (MR, min = 160)

display.plane (MR, top = bin.pu.density, display.ref = S$ref.pseudo,
               view.coord = z.brain, T.MAT = patient$T.MAT,
               interpolate = FALSE, main = "before brain intersection")
bin.pu <- bin.intersection (vol1 = bin.pu.density, vol2 = bin.brain,
                            alias = "processing unit")
display.plane (MR, top = bin.pu, display.ref = S$ref.pseudo,
               view.coord = z.brain, T.MAT = patient$T.MAT,
               interpolate = FALSE, main = "after brain intersection")
```

`bin.inversion` *Inversion of a binary*

Description

The `bin.inversion` function creates a "volume" class object, of "binary" modality, representing the inverse (logical `NOT`) of another binary object.

Usage

```
bin.inversion(vol, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object, of "binary" modality
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `paste ("!", vol$object.alias, sep = "")`.

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as `vol`, inverse of `vol`.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct"), roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

bin.patient <- bin.from.roi (CT, struct = S, roi.name = c ("patient"),
                             alias = "patient")
inverse.patient <- bin.inversion (bin.patient, alias = "inv (patient)")

display.plane(CT, top = inverse.patient, interpolate = FALSE)
```

`bin.opening` *Binary volume opening*

Description

The `bin.opening` function performs a morphological operation of opening, using a sphere, on a "volume" class object of "binary" modality. Opening is useful for :

- removing volumes that are smaller than the `radius`,
- smoothing shapes.

Usage

```
bin.opening(vol, radius = 10, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object, of "binary" modality.
`radius`	Positive number, in millimeters. By default, radius = 10.
`alias`	Character string, `$object.alias` of the created object.
`description`	Character string, describing the created object. If `description` = `NULL` (default value), it will be set to `paste (vol$object.alias, "opening r =", radius)`.

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as `vol`, in which `$vol3D.data` has been transformed by the opening operation.

Note

Opening can be time consuming, try to reduce the `binary` volume to the strict minimum, before any operations.

See Also

bin.dilation, bin.erosion, bin.closing, add.margin, nesting.cube.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "mr", roi.name = "",
                             dxyz = rep (step, 3))
MR <- patient$mr[[1]]

# generation of a binary volume
b <- bin.from.vol(MR, min = 15,max = 30)

b.opening <- bin.opening (b, radius = step)
display.plane (bottom = MR, top = b, main = "Before opening",
               view.coord = -20, interpolate = FALSE)
display.plane (bottom = MR, top = b.opening, main = "After opening",
               view.coord = -20, interpolate = FALSE)
```

`bin.subtraction` *Subtraction of two binaries*

Description

The `bin.subtraction` function creates a "volume" class object of "binary" modality, representing the subtraction of two binary objects.

Usage

```
bin.subtraction(vol1, vol2, alias = "", description = NULL)
```

Arguments

`vol1, vol2`	"volume" class objects of "binary" modality.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `paste (vol1$object.alias, "-", vol2$object.alias)`.

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as `vol1` and `vol2`, in which `vol2` is subtracted from `vol1`.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("mr", "rtstruct"), roi.name = "",
                             dxyz = rep (step, 3))
MR <- patient$mr[[1]]
S <- patient$rtstruct[[1]]

z.ptv <- S$roi.info$Gz[S$roi.info$roi.pseudo == "ptv"]

# binaries
```

```
bin.patient <- bin.from.roi (MR, struct = S, roi.name = "patient",
                             alias = "patient", T.MAT = patient$T.MAT)
bin.ptv <- bin.from.roi (MR, struct = S, roi.name = "ptv",
                             alias = "ptv", T.MAT = patient$T.MAT)

#' calculation of the 'patient - ptv' binary
bin <- bin.subtraction (bin.patient, bin.ptv, alias = "patient - ptv")
display.plane (MR, top = bin, view.coord = z.ptv,
               display.ref = S$ref.pseudo, T.MAT = patient$T.MAT,
               interpolate = FALSE)
```

`bin.sum` *Sum of two binaries*

Description

The `bin.sum` function creates a "volume" class object of "binary" modality, representing the sum (logical `OR`) of two binary objects.

Usage

```
bin.sum(vol1, vol2, alias = "", description = NULL)
```

Arguments

`vol1, vol2`	"volume" class objects of "binary" modality.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `paste (vol1$object.alias, "+", vol2$object.alias)`.

Value

Returns a "volume" class object of "binary" modality (see espadon.class for class definitions), with the same grid as `vol1` and `vol2`, sum of `vol1` and `vol2`.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct"), roi.name = "eye",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

z.leye<- S$roi.info$Gz[S$roi.info$roi.pseudo == "lefteye"]


# 'left eye' et 'right eye' binaries
bin.left.eye <- bin.from.roi (CT, struct = S, roi.sname = "lefteye",
                              alias = "left eye")
bin.right.eye <- bin.from.roi (CT, struct = S, roi.name = "righteye",
                               alias = "right eye")
bin.eyes <- bin.sum (bin.left.eye, bin.right.eye, alias = "eyes")
```

```
display.plane (CT, top = bin.eyes, struct = S, roi.sname = "eye",
               view.coord = z.leye, legend.shift = -90 ,
               interpolate = FALSE)
```

`castlow.str` *Cast of a character string*

Description

The `castlow.str` function converts a word to lowercase, without accents and spaces.

Usage

```
castlow.str(st)
```

Arguments

`st`	character string

Value

Returns the ASCII//TRANSLIT transcription of the word `st`, without accents, spaces and in lowercase letters.

See Also

castup.str.

Examples

```
castlow.str (st = c("Right eye", "Left_Lung", "Right-Lung"))
```

`castup.str` *Cast of a character string*

Description

The `castup.str` function converts a word to upper case, without accents and spaces.

Usage

```
castup.str(st)
```

Arguments

`st`	character string

Value

Returns the ASCII//TRANSLIT transcription of the word `st`, without accents, spaces and in capitals.

See Also

castlow.str.

Examples

```
castup.str (st = c("Right eye", "Left_Lung", "Right-Lung"))
```

`dicom.browser` *DICOM raw data browser*

Description

the `dicom.browser` function creates a dataframe describing the tags contained in the raw data of a DICOM file, as well as the information to access them.

Usage

```
dicom.browser(
  dicom.raw.data,
  nbTAG = 0,
  stop.tag = "",
  stop.level = 0,
  full.info = FALSE,
  tag.dictionary = dicom.tag.dictionary()
)
```

Arguments

`dicom.raw.data`	Raw vector, representing the binary extraction of the DICOM file.
`nbTAG`	Integer. If `nbTAG = 0` (default), and `stop.tag = ""`, all the DICOM raw data is browsed. Otherwise, the function only browses the first `nbTAG` tags.
`stop.tag`	Character string, representing the tag that stops the browse of the `dicom.raw.data`.
`stop.level`	Positive integer, specifying the encapsulation level of the `stop.tag` in `dicom.raw.data`.
`full.info`	Boolean. If `TRUE`, more information about the DICOM data is returned.
`tag.dictionary`	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.

Value

Returns a dataframe if `dicom.raw.data` is DICOM raw data, `NULL` otherwise.

If `full.info = FALSE`, dataframe columns are

- tag : the tags contained in `dicom.raw.data`,
- VR : value representation of the content of the tag,
- endian : the endianness of the tag content,
- start : the start address in `dicom.raw.data` of the tag content.
- stop : the stop address in `dicom.raw.data` of the tag content.

If `full.info = TRUE`, the following columns are added :

- encaps.load : If the tag contains nested data, this column gives the number of bytes remaining until the end of the nesting. If there are several levels of nesting, these numbers are collapsed and separated by a space.
- load.start : the start address in `dicom.raw.data` of the tag load size.
- load.stop : the stop address in `dicom.raw.data`of the tag load size.
- tag.start : the start address in `dicom.raw.data` of the tag.

See Also

dicom.raw.data.loader, dicom.tag.parser

Examples

```
# DICOM information dataframe of the dummy raw data toy.dicom.raw ()
df <- dicom.browser (toy.dicom.raw (), full.info = TRUE)
str (df)
```

`dicom.parser` *Conversion of DICOM raw data into a dataframe or a list of DICOM TAG information*

Description

The `dicom.parser` function creates a dataframe or a list from DICOM raw data. The created dataframe or list provides information about the content of the DICOM TAGs included in the raw data.

Usage

```
dicom.parser(
  dcm,
  as.txt = TRUE,
  nested.list = FALSE,
  try.parse = FALSE,
  txt.sep = "\\",
  txt.length = 100,
  tag.dictionary = dicom.tag.dictionary(),
  ...
)
```

Arguments

`dcm`	espadon object of class "volume", "rtplan", "struct" provided by DICOM files, or DICOM filename, or Rdcm filename, or raw vector representing the binary extraction of the DICOM file.
`as.txt`	Boolean. If `as.txt` = `TRUE`, the function returns a dataframe, a list otherwise.
`nested.list`	Boolean. Only used if `as.txt` = `FALSE`. If `nested.list` = `FALSE`, the returned list consists of nested lists.
`try.parse`	Boolean. If `TRUE`, the tag with unknown DICOM VR (value representation) is converted into string if possible.

txt.sep	String. Used if as.txt = TRUE. Separator of the tag value elements.
txt.length	Positive integer. Used if as.txt = TRUE. Maximum number of letters in the representation of the TAG value.
tag.dictionary	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.
...	Additional argument dicom.browser when previously calculated by dicom.browser. Argument dicom.raw.data (deprecated) replaced by dcm argument. Argument nb or dicom.nb representing the number of DICOM file, when dcm contains multiple DICOM files.

Value

Returns a list of elements or a dataframe, depending on as.list.

If it returns a dataframe, the columns are names TAG, VR (value representation), VM (value multiplicity), loadsize and Value. The field $Value is a string representation of the true value.

If it returns a list, each of its elements, named by a TAG, is either a vector or a string, depending of the TAG included in dicom.raw.data.

See Also

dicom.raw.data.loader, dicom.tag.parser, dicom.viewer,xlsx.from.dcm,xlsx.from.Rdcm

Examples

```
# content of the dummy raw data toy.dicom.raw (), as a list.
L <- dicom.parser (toy.dicom.raw (), as.txt = FALSE)
str(L[40:57])

L <- dicom.parser (toy.dicom.raw (), as.txt = FALSE, nested.list = TRUE)
str(L[40:45])

# content of the dummy raw data toy.dicom.raw (), as a dataframe.
L <- dicom.parser (toy.dicom.raw (), as.txt = TRUE)
str (L)
```

---

dicom.raw.data.anonymizer

DICOM anonymizer

Description

the dicom.raw.data.anonymizer function anonymizes dicom.raw.data.

Usage

```
dicom.raw.data.anonymizer(
  dicom.raw.data,
  offset = 0,
  new.PIN = "Anonymous ",
  reset.private.tag = FALSE,
  tag.dictionary = dicom.tag.dictionary()
)
```

Arguments

`dicom.raw.data`	Raw vector, representing the binary extraction of the DICOM file.
`offset`	Integer, default to 0. Each date of the DICOM will be shifted by this offset expressed in days.
`new.PIN`	Character string, representing the PIN remplacing the old one.
`reset.private.tag`	Boolean, if `TRUE`, the value of tags that are not in the `tag.dictionary` is removed.
`tag.dictionary`	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.

Value

Returns an anonymyzed raw vector. See Note.

Note

The raw data is anonymized as follows:

- Each date of the DICOM file will be shifted by `offset` expressed in days.
- Each patient's name, and patient'ID are remplaced by `new.PIN`
- All other patient data are deleted, except age, weight, height, gender and shifted birthday.
- All address, phone, physician, operator, author, reviewer, service.
- If `reset.private.tag` = `TRUE`, the values of the tags not contained in the `tag.dictionary` are deleted.

Examples

```
# pseudomization of the dummy raw data toy.dicom.raw ()
an.raw.data <- dicom.raw.data.anonymizer (toy.dicom.raw (), offset = -2)
data <- dicom.parser (toy.dicom.raw ())
an.data <- dicom.parser (an.raw.data)

# Checking for differences
flag.dif <- data$Value !=  an.data$Value
df <- cbind (data[flag.dif, c ("VM","Value")], an.data[flag.dif, "Value"])
colnames (df) <-   c ("VM","old Value","new Value")
df

# save data in a the new file
#############################
# new.file.name <- "an.dcm"
# zz <- file (new.file.name, "wb")
# writeBin (an.raw.data, zz, size = 1)
# close (zz)
```

`dicom.raw.data.loader` *DICOM file loading in raw data*

Description

the `dicom.raw.data.loader` function loads a DICOM file as raw data.

Usage

```
dicom.raw.data.loader(dcm.filename)
```

Arguments

`dcm.filename`	Character string, representing the full name of a DICOM file.

Value

Returns a vector of raw data from `dcm.filename`.

See Also

dicom.browser, dicom.tag.parser

Examples

```
# First, save toy.dicom.raw () raw data to a temporary file for testing.
pat.src.dir <- file.path (tempdir(), "toy_dccm")
dir.create (pat.src.dir, recursive = TRUE)
dcm.filename <- tempfile (pattern = "toyrtplan", tmpdir = pat.src.dir,
                          fileext = ".dcm")
zz <- file (dcm.filename, "wb")
writeBin (toy.dicom.raw (), zz, size = 1)
close (zz)

# loading of file
dicom.raw.data <- dicom.raw.data.loader (dcm.filename)

# checks if it is consistent with the original raw data
all ( dicom.raw.data == toy.dicom.raw () )

# Cleaning  temporary directory
unlink (pat.src.dir, recursive = TRUE)
```

`dicom.set.tag.value` *Change TAG value in DICOM raw data*

Description

The `dicom.set.tag.value` function changes, in the DICOM raw data, the values of the TAG whose VR is a string of characters.

Usage

```
dicom.set.tag.value(
  dicom.raw.data,
  tag,
  tag.value,
  tag.dictionary = dicom.tag.dictionary(),
  ...
)
```

Arguments

`dicom.raw.data`	Raw vector, representing the binary extraction of the DICOM file.
`tag`	String vector, representing the list of tags whose value is to be changed. See note 1.
`tag.value`	String vector,representing the list of new tag values.
`tag.dictionary`	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.
`...`	Additional arguments `dicom.browser` when previously calculated by dicom.browser with argument `full.info = TRUE`.

Value

Returns a raw vector, with new tag values.

Note

1- The list of tags included in the DICOM file are given by the first columns of the dataframe provided by the functions dicom.browser and dicom.parser.

2- The `dicom.set.tag.value` function may take some processing time. To minimize this time, it is recommended to prepare in advance all the tags to be modified, and use the `dicom.set.tag.value` function only once, as shown in the example.

Examples

```
# change the value of tags "(0010,0010)" and "(0010,0020)" in the
# dummy raw data toy.dicom.raw ()
new.raw.data <- dicom.set.tag.value (toy.dicom.raw (),
                                     tag =  c ("(0010,0010)", "(0010,0020)"),
                                     tag.value = c ("unknown", "000001"))
# change control
data <- dicom.parser (new.raw.data)
data[data$TAG %in% c ("(0010,0010)", "(0010,0020)"), ]
```

```
# save data in a the new file
############################
# new.file.name <- "new.dcm"
# zz <- file (new.file.name, "wb")
# writeBin (new.raw.data  , zz, size = 1)
# close (zz)
```

`dicom.tag.dictionary` *DICOM TAG dictionary*

Description

The `dicom.tag.dictionary` function gives the dictionary of tags used by default in the **espadon** package.

Usage

```
dicom.tag.dictionary(add.dict = c("raysearch.tag"))
```

Arguments

`add.dict`	Vector of the list of additional dictionaries. Put to NULL, if no additional dictionary is requested.

Value

Returns a 3-column dataframe, describing the VR (value representation) and the name of each DICOM TAG.

This dataframe is the fusion of the "nema.tag" dictionary, provided by *nema* **[1]**, with the dictionaries defined in the `add.dict` vector:

- "raysearch.tag" dictionary is provided by *RaySearch laboratories* **[2]**

References

[1] DICOM nema (Online; accessed 2022-02-16). "Current Edition." https://www.dicomstandard.org/current.

[2] Raysearch Laboratories (Online; accessed 2022-04-25). "RAYPLAN 11A, DICOM Conformance Statement." https://www.raysearchlabs.com/4aaf2e/siteassets/raystation-landing-page/dicom-conformance-statements/raystation-pdfs/rsl-d-rs-11a-dcs-en-1.0-2021-05-07-raystation-11a-pdf.

Examples

```
str (dicom.tag.dictionary ())
str (dicom.tag.dictionary (NULL))
```

dicom.tag.parser *DICOM TAG parser*

Description

the dicom.tag.parser function decodes the content between two DICOM raw data addresses.

Usage

```
dicom.tag.parser(start, stop, VR, endian, dicom.raw.data, try.parse = FALSE)
```

Arguments

start	Positive integer. Index of the first raw data to parse in the dicom.raw.data.
stop	Positive integer. Index of the last raw data to parse in the dicom.raw.data.
VR	Character string, representing the value representation of DICOM data. See DICOM standard.
endian	Character string, equal to "little" or "big".
dicom.raw.data	Raw vector, representing the binary extraction of the DICOM file.
try.parse	Boolean. If TRUE, the value, with an undocumented VR, is considered, as far as possible, as a string.

Value

Returns the dicom.raw.data content between the addresses start and stop. Depending on the representation of the value (VR), it can be a character string or a numerical vector.

Examples

```
# creation of the toy.dicom.raw () addresses dataframe:
df <- dicom.browser (toy.dicom.raw ())

# search for modality of toy.dicom.raw ()
idx <- grep ("^[(]0008,0060[)]$", df$tag)
modality  <- dicom.tag.parser (df$start[idx], df$stop[idx], df$VR[idx],
                               df$endian[idx], toy.dicom.raw ())
modality
```

dicom.to.Rdcm.converter

Conversion of DICOM object into files that can be interpreted by the **espadon** *package*

Description

The dicom.to.Rdcm.converter function creates, for each DICOM object, a *.Rdcm file usefull for using **espadon** package. Each Rdcm file created is referenced by the date of acquisition of the object (if it is not available, its creation date), the patient's PIN, a reference number, an object number in this reference system, and the object modality (mr, ct, rtstruct...).

Usage

```
dicom.to.Rdcm.converter(
  dcm.files,
  pat.dest.dir,
  update = TRUE,
  ignore.duplicates = FALSE,
  tag.dictionary = dicom.tag.dictionary(),
  verbose = TRUE
)
```

Arguments

`dcm.files`	String vector, representing the list of the full names of the DICOM files of the same patient, or its directory.
`pat.dest.dir`	Character string, representing the full name of patient directory, which will contain files converted **espadon**.
`update`	Boolean. If set to `TRUE`, and if `pat.dest.dir` contains previously converted files, these files are updated,even if they are duplicated. They retain the same **espadon** reference frame assignment.
`ignore.duplicates`	Boolean. If `TRUE`, the function ignores duplicated objects.
`tag.dictionary`	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.
`verbose`	Boolean. If `TRUE`, a progress bar indicates the progress of the conversion.

Value

Returns the list of basenames of the created files.

Returns `NULL` if there are no DICOM files in `dcm.files`

Note

For each DICOM object, `dicom.to.Rdcm.converter` creates a *.Rdcm file whose basename is made up of the date of the acquisition (or creation date if previous not found), the patient's PIN, the pseudonym of the frame of reference ("ref1", "ref2"...), the number of the volume object in the directory in this frame of reference ("do1", "do2"...), and the object modality ("mr", "ct", "rtdose", "rtstruct"...).

For example: `BASE = "20160514_a008e9ac_ref2_do1_mr"`

Examples

```
# First, save toy.dicom.raw () raw data to a temporary file for testing.
pat.src.dir <- file.path (tempdir(), "PM_dcm")
dir.create (pat.src.dir, recursive = TRUE)
dcm.filename <- tempfile (pattern = "PM_rtplan", tmpdir = pat.src.dir,
                          fileext = ".dcm")
zz <- file (dcm.filename, "wb")
writeBin (toy.dicom.raw (), zz, size = 1)
close (zz)

# Create a temporary destination directory where the *.Rdcm file will be saved
pat.dest.dir <- file.path (tempdir(), "PM_Rdcm")
```

```
dicom.to.Rdcm.converter (pat.src.dir, pat.dest.dir, update = TRUE)
# or
dicom.to.Rdcm.converter (dcm.filename, pat.dest.dir, update = TRUE)

list.files (pat.dest.dir)

# Cleaning  temporary directories
unlink (pat.src.dir, recursive = TRUE)
unlink (pat.dest.dir, recursive = TRUE)
```

`dicom.viewer` *DICOM content viewer*

Description

the `dicom.viewer` function displays the data of a DICOM file.

Usage

```
dicom.viewer(
  dcm,
  txt.sep = "\\",
  txt.length = 100,
  tag.dictionary = dicom.tag.dictionary(),
  height = 600,
  width = 900,
  ...
)
```

Arguments

`dcm`	espadon object of class "volume", "rtplan", "struct" provided by DICOM files, or DICOM filename, or Rdcm filename, or raw vector representing the binary extraction of the DICOM file.
`txt.sep`	String. Used if `as.txt = TRUE`. Separator of the tag value elements.
`txt.length`	Positive integer. Used if `as.txt = TRUE`. Maximum number of letters in the representation of the TAG value.
`tag.dictionary`	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.
`height, width`	Height and width in pixel of the DICOM table.
`...`	Additional argument `dicom.browser` when previously calculated by dicom.browser. Argument `nb` or `dicom.nb` representing the number of DICOM file, when `dcm` contains multiple DICOM files.

Value

Returns the DICOM file description in a browser window.

See Also

xlsx.from.dcm, xlsx.from.Rdcm, dicom.parser

Examples

```
if (interactive ()) dicom.viewer (toy.dicom.raw ())
```

`display.2D.histo` *Display of a 2D histogram*

Description

The `display.2D.histo` function displays the density map of a "histo2D" class object.

Usage

```
display.2D.histo(
  histo.2D,
  add = TRUE,
  main = NULL,
  x.lab = NULL,
  y.lab = NULL,
  x.lim = NULL,
  y.lim = NULL,
  bg = "#000000",
  i.rng = NULL,
  display.mode = c("mono.color", "rainbow.color", "line"),
  col = "#ffffff",
  alpha = 1,
  line.pc.levels = c(1, 100),
  line.lwd = 2,
  line.lty = 1
)
```

Arguments

`histo.2D`	"histo2D" class object.
`add`	Boolean indicating whether to display the background image.
`main`	Title of the background image. If `main = NULL`, the title indicates "2D histogram".
`x.lab`	Label of the x-axis of the background image. If `x.lab = NULL`, this label is `histo.2D$x.file.src`
`y.lab`	Label of the y-axis of the background image. If `y.lab = NULL`, this label is `histo.2D$y.file.src`.
`x.lim`	Vector, representing the range of the x-axis.
`y.lim`	Vector, representing the range of the y-axis.
`bg`	Background color of the image. By default, this color is black.
`i.rng`	Vector of 2 elements giving the minimum and the maximum intensity of the image. If `i.rng = NULL`, then the minimum is 0 and the maximum the maximum density.
`display.mode`	function display mode. See Details.

col	Color of the color gradient in display.mode = "mono.color", or of the level lines in display.mode = "line". By default, this color is white.
alpha	A number from 0 to 1, indicating the opacity of the image in "rainbow.color" mode. Default alpha = 1.
line.pc.levels	Vector of level lines in percent of maximum density in display.mode = "line". By default lines 1% and 100% are displayed.
line.lwd	Line thickness of the level lines in display.mode = "line".
line.lty	Type of lines for level lines in display.mode = "line".

Details

The display.mode argument can be set to three values: "mono.color", "rainbow.color", or "line". The 2D histogram graph is displayed by default in "mono.color" mode.

- The "mono.color" mode displays a gradient of the color defined by the col argument, depending on the intensity of $density.map 2-dimensional array.
- The "rainbow.color" mode makes a display according to the "rainbow" palette, while managing the opacity of the colors.
- The "line" mode draws level lines defined in percent by the line.pc.levels argument.

Value

Returns a display of the density map of histo.2D. This one must be an object of class "histo2D". See espadon.class for class definitions.

See Also

histo.2D.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "mr", "rtstruct"),
                             roi.name =  "brain",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
MR <- patient$mr[[1]]
S <- patient$rtstruct[[1]]
T.MAT <- patient$T.MAT

# restriction of the volume around the RoI
CT.on.roi <- nesting.roi (CT, S, roi.name = "brain", vol.restrict = TRUE,
                          xyz.margin = c (1, 1, 1), alias = CT$description)
MR.on.CT <- vol.regrid (vol = MR, back.vol = CT.on.roi, interpolate = TRUE,
                        T.MAT = T.MAT, alias = CT$description,
                        description = NULL)
# selection of voxels included in the RoI.
roi.bin <- bin.from.roi (vol = CT.on.roi, struct = S, roi.sname = "brain")
MR.select <- vol.from.bin (MR.on.CT, roi.bin, alias = MR$description)
CT.select <- vol.from.bin (CT.on.roi, roi.bin, alias = CT$description)
# 2D histogram
H2D <- histo.2D (MR.select, CT.select, x.breaks = seq (50, 400, 10),
  y.breaks = seq (50, 400, 10), alias = "H2D MR1 MR2")
```

```
display.2D.histo (H2D, display.mode = "mono.color", col = "#ffff00",
                  main ="mono color mode")
display.2D.histo (H2D, display.mode = "rainbow.color", main ="rainbow mode")
display.2D.histo (H2D, display.mode = "line", main ="level lines mode",
                  line.pc.levels = c (0, 25, 50, 75, 100), col = "#ff0000")
```

`display.3D.contour` *Display the 3D contours of the RoI*

Description

The `display.3D.contour` function performs a 3D display of the selected RoI in the chosen coordinate system.

Usage

```
display.3D.contour(
  struct,
  roi.name = NULL,
  roi.sname = NULL,
  roi.idx = NULL,
  roi.col = NULL,
  roi.print = FALSE,
  roi.lwd = 1,
  roi.cex = 1,
  display.ref = struct$ref.pseudo,
  T.MAT = NULL,
  FoR.axis = FALSE,
  FoR.col = "black"
)
```

Arguments

`struct`	"struct" class object. See espadon.class for class definitions.
`roi.name`	Vector of exact names of the RoI in the `struct` object. By default `roi.name = NULL`. See Details.
`roi.sname`	Vector of names or parts of names of the RoI in the `struct` object. By default `roi.sname = NULL`. See Details.
`roi.idx`	Vector of indices of the RoI that belong to the `struct` object. By default `roi.idx = NULL`. See Details.
`roi.col`	Color of the RoI. If `roi.col = NULL` (default),the RoI colors are specified in the `struct$roi.info`.
`roi.print`	Boolean vector indicating whether to display the pseudonym of the RoI.
`roi.lwd`	Line width of the RoI, by default at 1.
`roi.cex`	Numeric character expansion factor of RoI name if `roi.print = TRUE`, defaults to 1.
`display.ref`	Pseudonym of frame of reference of the display.
`T.MAT`	"t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If `T.MAT = NULL`, `display.ref` must be equal to `NULL` or to `struct$ref.pseudo`.

FoR.axis	Boolean or numeric, by default set to FALSE. If FoR.axis = TRUE, the function displays 200 mm length director vectors of the frame of reference. If FoR.axis is numeric, it represent the length in mm of the director vectors.
FoR.col	Color of the frame of reference.

Details

If roi.name, roi.sname, and roi.idx are all NULL, then all of the RoI are selected.

Value

If the concerned regions of interest (RoI) struct exist, it displays the 3D contours of these RoI in the current **RGL** window if it exists, in a new window otherwise.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "rtstruct", roi.name = "eye",
                             dxyz = rep (step, 3))
library (rgl)
open3d()
bg3d ("black")
display.3D.contour (struct = patient$rtstruct[[1]], roi.print = TRUE)
```

---

display.3D.mesh *3D display of a mesh*

Description

The display.3D.mesh function performs a 3D display of a mesh.

Usage

```
display.3D.mesh(mesh, display.ref = mesh$ref.pseudo, T.MAT = NULL, ...)
```

Arguments

mesh	"mesh" class object, created by the mesh.from.bin function. See espadon.class for class definitions.
display.ref	Character string. Pseudonym of the frame of reference used for display.
T.MAT	"t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If T.MAT is NULL, mesh must be displayed in display.ref = mesh$ref.pseudo.
...	Additional arguments passed to shade3d as color, specular, alpha...

Value

Returns a display of mesh in the current **RGL** window if it exists, in a new window otherwise.

See Also

mesh.from.bin.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct"), roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

# creation of the patient mesh
bin <- bin.from.roi (CT, struct = S, roi.name = "patient")
mesh.patient <- mesh.from.bin (bin, alias = "patient", verbose = FALSE)

# display of the patient mesh, with transparency
library (rgl)
open3d()
display.3D.mesh (mesh.patient, color = "burlywood2", specular = "#404040")
```

`display.3D.sections` *Display 3D sections of a patient*

Description

The `display.3D.sections` function displays transverse, sagittal and frontal views at a point in 3D.

Usage

```
display.3D.sections(
  vol,
  cross.pt = c(0, 0, 0),
  display.ref = vol$ref.pseudo,
  T.MAT = NULL,
  col = grey.colors(10, start = 0, end = 1, alpha = c(rep(0, 1), rep(1, 9))),
  breaks = NULL,
  trans = TRUE,
  sagi = TRUE,
  front = TRUE,
  border = TRUE,
  border.col = "#379DA2"
)
```

Arguments

`vol`	"volume" class object to display. See espadon.class for class definitions.
`cross.pt`	Vector of x, y, z coordinates, representing the cross point of the 3 planes.
`display.ref`	Character string. Pseudonym of the frame of reference used for display.
`T.MAT`	"t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If `T.MAT` is `NULL`, `vol` must be displayed in `display.ref` = `vol$ref.pseudo`.
`col`	Vector, representing the color palette of the image. Transparent colors are not represented.
`breaks`	One of :

- NULL : the minimum and the maximum value of the vol define the range.
- Vector giving the breakpoints of each color.

trans	Boolean. If TRUE (default), the transverse view is displayed.
sagi	Boolean. If TRUE (default), the sagittal view is displayed.
front	Boolean. If TRUE (default), the frontal view is displayed.
border	Boolean. If TRUE (default), the borders of the planes are displayed
border.col	Color of planes borders

Value

Returns a display of transverse, sagittal and frontal views of vol at cross.pt in the current **RGL** window if it exists, in a new window otherwise. Palette colors are managed by col and breaks.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "ct", dxyz = rep (step, 3))
CT <- patient$ct[[1]]

library (rgl)
open3d()
display.3D.sections(CT, cross.pt= c(0, 50, 80),
                    col= pal.RVV(200, alpha= c(rep(0,90), rep(1,110))),
                    breaks = seq(-1000, 1000, length.out = 201))
```

---

display.3D.stack *Display in 3D the selected planes of an* **espadon** *class volume*

Description

The display.3D.stack function displays in 3D the requested planes of a "volume" class object.

Usage

```
display.3D.stack(
  vol,
  k.idx = unique(vol$k.idx[seq(1, vol$n.ijk[3], length.out = 10)]),
  display.ref = vol$ref.pseudo,
  T.MAT = NULL,
  col = grey.colors(10, start = 0, end = 1, alpha = c(rep(0, 1), rep(1, 9))),
  breaks = NULL,
  cube = TRUE,
  border = TRUE,
  ktext = TRUE,
  line.col = "#379DA2",
  line.lwd = 1,
  cex = 1
)
```

Arguments

vol	"volume" class object to display.
k.idx	vector of plane numbers to be displayed, to be chosen in vol$k.idx. By default k.idx is a vector of 10 uniformly distributed planes in the volume.
display.ref	Character string. Pseudonym of the frame of reference used for display.
T.MAT	"t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If T.MAT is NULL, vol must be displayed in display.ref = vol$ref.pseudo.
col	Vector, representing the color palette of the image. Transparent colors are not represented.
breaks	One of : • NULL : The minimum and the maximum value of the vol define the range. • Vector giving the breakpoints of each color.
cube	Boolean. If TRUE (default), the "volume" edges are displayed.
border	Boolean. If TRUE (default), the borders of the planes defined in k.idx are displayed.
ktext	Boolean. If TRUE (default), the selected plane numbers are displayed.
line.col	Color of cube, planes and texts displayed.
line.lwd	Line width of the border and cube, by default at 1.
cex	Numeric character expansion factor of displayed plan numbers.

Value

Returns a display of the k.idx cutting planes of vol, in the current **RGL** window if it exists, in a new window otherwise. The colors of the palettes are managed by col and breaks.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "ct", dxyz = rep (step, 3))

# display o 3 planes
library (rgl)
open3d()
display.3D.stack (patient$ct[[1]],
                  col = pal.RVV (200, alpha = c(rep(0,90), rep (1, 110))),
                  breaks = seq (-1000, 1000, length.out = 201))
```

---

display.DVH *Display of a DVH*

Description

The display.DVH function displays the Dose Volume Histogram of a "dvh" class object. Y-units are cm^3.

Usage

```
display.DVH(
  dvh,
  add = FALSE,
  xgrid = TRUE,
  ygrid = TRUE,
  MC.plot = FALSE,
  MC.col = grey.colors(4, rev = TRUE),
  ...
)
```

Arguments

dvh	"dvh" class object.
add	Boolean indicating whether to display the background image.
xgrid	Boolean indicating the display of the x grid.
ygrid	Boolean indicating the display of the y grid.
MC.plot	Boolean. If MC.plot = TRUE, then display.DVH displays, if they exist, the quantile zones (Prob = 0, .025, .25, .5, .75, .975, 1) of MC DVH variations.
MC.col	Character string, a valid palette with 4 colours corresponding to 100%, 95%, 50% and median of MC data.
...	Additional arguments xlab, ylab, xlim, ylim, main, type, col, lwd, lty and log managed by the plot function.

Value

Returns a plot of the cumulative histogram included in dvh, with its median, and the quantile areas (0%-100%), (2.5%-97.5%) and (25%-75%) of the dvh$vol variations, if they exist.

See Also

display.DVH.pc

Examples

```
# DVH without MCMC
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"), roi.name = "",
                             dxyz = rep (step, 3), beam.nb = 3)
H <- histo.from.roi (patient$rtdose[[1]], patient$rtstruct[[1]], roi.name = "ptv",
                     breaks = seq (0, 60, by = 2))
DVH <- histo.DVH (H)
display.DVH (DVH)

## Not run:
# loading of toy-patient objects
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"),
                             roi.name = "gizzard",
                             dxyz = c (2, 2, 2), beam.nb = 3)
```

```
# Calculation of the histogram
H <- histo.from.roi (patient$rtdose[[1]], patient$rtstruct[[1]],
                     roi.name = "gizzard",
                     breaks = seq (0, 60, by = 1), MC = 100)

# DVH
DVH <- histo.DVH (H)
display.DVH (DVH, MC.plot = TRUE, ylim = c (0, 40))

## End(Not run)
```

`display.DVH.pc` *Display of a cumulative DVH in percent of total volume*

Description

The `display.DVH.pc` function displays the Dose Volume Histogram of "dvh" class object. Y-units are percents of total volume.

Usage

```
display.DVH.pc(
  dvh,
  add = FALSE,
  xgrid = TRUE,
  ygrid = TRUE,
  MC.plot = FALSE,
  MC.col = grey.colors(4, rev = TRUE),
  ...
)
```

Arguments

`dvh`	"dvh" class object. See espadon.class for class definitions.
`add`	Boolean indicating whether to display the background image.
`xgrid`	Boolean indicating the display of the x grid.
`ygrid`	Boolean indicating the display of the y grid.
`MC.plot`	Boolean. If `MC.plot = TRUE`, then `display.DVH.pc` displays, if they exist, the quantile zones (Prob = 0, .025, .25, .5, .75, .975, 1) of MC DVH variations.
`MC.col`	Character string, a valid palette with 4 colours corresponding to 100%, 95%, 50% and median of MC data.
`...`	Arguments xlab, ylab, xlim, ylim, main, type, col, lwd, lty and log managed by the plot function.

Value

Returns a plot in percent of total volume of the cumulative histogram included in `dvh`, with its median, and the quantile areas (0%-100%), (2.5%-97.5%) and (25%-75%) of the `dvh$pcv` variations, if they exist.

See Also

display.DVH

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"),
                             roi.name = "gizzard", dxyz = rep (step, 3),
                             beam.nb = 3)

# Calculation of the histogram
H <- histo.from.roi (patient$rtdose[[1]], patient$rtstruct[[1]],
                     roi.name = "gizzard",
                     breaks = seq (0, 60, by = 1))

# DVH
DVH <- histo.DVH (H)
display.DVH.pc (DVH)
```

---

display.dV_dx *Display of the volume density of a histogram*

Description

The display.dV_dx function displays the volume density of a "histo" class object. Y-units are $cm^3.Gy^{-1}$.

Usage

```
display.dV_dx(
  histo,
  add = FALSE,
  xgrid = TRUE,
  ygrid = TRUE,
  MC.plot = FALSE,
  MC.col = grey.colors(4, rev = TRUE),
  ...
)
```

Arguments

histo	"histo" class object. See espadon.class for class definitions.
add	Boolean indicating whether to display the background image.
xgrid	Boolean indicating the display of the x grid.
ygrid	Boolean indicating the display of the y grid.
MC.plot	Boolean. If MC.plot = TRUE, then display.dV_dx displays, if they exist, the quantile zones (Prob = 0, .025, .25, .5, .75, .975, 1) of variations in volume density.

MC.col	Character string, a valid palette with 4 colours corresponding to 100%, 95%, 50% and median of MC data.
...	Additional arguments xlab, ylab, xlim, ylim, main, type, col, lwd, lty and log managed by the plot function.

Value

Returns a plot of the differential histogram included in histo, with its median, and the quantile areas (0%-100%), (2.5%-97.5%) and (25%-75%) of the histo$dv_dx variations, if they exist.

See Also

display.histo.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"),
                             roi.name = "gizzard", dxyz = rep (step, 3),
                             beam.nb = 3)

# Calculation of the differential histogram
H <- histo.from.roi (patient$rtdose[[1]], patient$rtstruct[[1]],
                     roi.name = "gizzard", breaks = seq (0, 60, by = 2))
display.dV_dx (H, lwd = 2, col = '#00ff00', ylim = c (0,10))
```

---

display.histo *Display of the counts of a histogram*

Description

The display.histo function displays the counts of "histo" class object.

Usage

```
display.histo(
  histo,
  add = FALSE,
  xgrid = TRUE,
  ygrid = TRUE,
  MC.plot = FALSE,
  MC.col = grey.colors(4, rev = TRUE),
  ...
)
```

Arguments

histo	"histo" class object.
add	Boolean indicating whether to display the background image.
xgrid	Boolean indicating the display of the x grid.
ygrid	Boolean indicating the display of the y grid.
MC.plot	Boolean. If MC.plot = TRUE, then display.histo displays, if they exist, the quantile zones (Prob = 0, .05, .25, .5, .75, .95, 1) of variations in counts.
MC.col	Character string, a valid palette with 4 colours corresponding to 100%, 95%, 50% and median of MC data.
...	Additional arguments xlab, ylab, xlim, ylim, main, type, col, lwd, lty and log managed by the plot function.

Value

Returns a plot of the counts included in histo, with its median, and the quantile areas (0%-100%), (2.5%-97.5%) and (25%-75%) of the histo$counts variations, if they exist.

See Also

display.dV_dx.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 3
patient <- toy.load.patient (modality = "ct", dxyz = rep (step, 3))
CT <- patient$ct[[1]]

# histogram and display
H <- histo.vol (CT, breaks = seq (3, ceiling (CT$max.pixel), 1),
                alias = "CT_hist")
display.histo (H, log = "y", lwd = 2)
```

---

display.kplane *Display of a plane of a volume*

Description

The display.kplane function displays the requested plane of a "volume" class object. This function is low-level, used for example in the function display.plane with more intuitive arguments.

Usage

```
display.kplane(
  vol,
  k = vol$k.idx[ceiling(length(vol$k.idx)/2)],
  pt00 = c(0, 0),
  dxy = c(1, 1),
  col = grey.colors(255, start = 0, end = 1),
  breaks = NULL,
```

```
  sat.transp = FALSE,
  add = FALSE,
  main = NULL,
  abs.lab = "i",
  ord.lab = "j",
  abs.flip = FALSE,
  ord.flip = FALSE,
  bg = "#000000",
  abs.rng = NULL,
  ord.rng = NULL,
  interpolate = FALSE
)
```

Arguments

vol	"volume" class object to display. See espadon.class for class definitions.
k	Number of the plane to display. By default k is located at mid-plane of the volume.
pt00	Origin point of the displayed plane. By default pt00 = c (0, 0), corresponding to the bottom left of the displayed non-flipped image.
dxy	width and height of a pixel in the plane. If dxy = c (1, 1) (default) abcissa and ordinate correspond to pixel number in the plane.
col	Vector, representing the color palette of the image.
breaks	One of : • NULL : the minimum and the maximum value of the vol define the range. • Vector giving the breakpoints of each color. Outside values are transparent, leaving the background visible, depending on sat.transp.
sat.transp	Boolean. If TRUE, outside values are transparent, else set to breaks limits colors.
add	Boolean indicating whether to display the background image.
main	Title of the background image. If main = NULL, the title just indicates the value of k.
abs.lab	Label of the image abcissa. By default abs.lab = 'i'.
ord.lab	Label of the image ordinate. By default ord.lab = 'j'.
abs.flip	Boolean defaults to FALSE flipping the horizontal axis of the background image.
ord.flip	Boolean defaults to FALSE flipping the vertical axis of the background image.
bg	Background color of the image. By default, this color is black.
abs.rng	Vector of 2 elements indicating the minimum and maximum background image abscissa to display.
ord.rng	Vector of 2 elements indicating the minimum and maximum background image ordinate to display.
interpolate	Boolean, indicating whether to apply linear interpolation to the image.

Value

Returns a display of the k^{th} image plane of vol.

See Also

display.plane.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
patient <- toy.load.patient (modality = c("ct","mr", "rtdose"),
                             dxyz = rep (step, 3), beam.nb = 3)
MR <- patient$mr[[1]]
CT <- patient$ct[[1]]
D <- patient$rtdose[[1]]

# display

display.kplane (CT)

display.kplane (MR, k = floor (length(MR$k.idx)*5/8),
                col = grey.colors (256, start = 0, end = 1),
                breaks = seq (0, 500, length.out = 257), bg = "darkblue")


display.kplane (D, k = floor (length(D$k.idx)*3/8),
                col = rainbow (256, s = seq (1, 0, length.out = 256),
                               start = 0, end = 4/6,
                               alpha = seq (0.8, 0, length.out=256),
                               rev = TRUE),
                bg = "darkblue", ord.flip = TRUE, sat.transp = FALSE,
                interpolate = FALSE)

display.kplane (CT, k = floor (length(CT$k.idx)/3), col = pal.RVV (1000),
                breaks = seq(-1000, 1000, length.out = 1001),
                bg = "darkblue", ord.flip = TRUE, interpolate = FALSE)
```

`display.legend` *Display of the RoI legend*

Description

The `display.legend` function displays in an image the list of requested RoI and their associated color.

Usage

```
display.legend(
  struct = NULL,
  roi.name = NULL,
  roi.sname = NULL,
  roi.idx = NULL,
  lwd = 1,
  cex = 1,
  displayed.roi.name = NULL,
  bg = "black",
  text.col = "white"
)
```

Arguments

struct	"struct" class object.
roi.name	Vector of exact names of the RoI in the struct object. By default roi.name = NULL. See Details.
roi.sname	Vector of names or parts of names of the RoI in the struct object. By default roi.sname = NULL. See Details.
roi.idx	Vector of indices of the RoI that belong to the struct object. By default roi.idx = NULL. See Details.
lwd	Line thickness, defaults to 1
cex	Font size, default to 1.
displayed.roi.name	Vector. If different from NULL, it represents the replacement names of selected RoI if needed.
bg	color of the background.
text.col	color of the legend text.

Details

roi.name, roi.sname, and roi.idx indicates the RoI to display. If all three are set to NULL, all RoI are selected.

Value

Returns display of the RoI names and their associated color in the active graphics window.

Examples

```
# loading of toy-patient objects
patient <- toy.load.patient (modality = c("rtstruct"), dxyz = c (5, 5, 5))
S <- patient$rtstruct[[1]]

display.legend (struct = S, roi.idx = 2:10, lwd = 2)
```

---

display.obj.links *Display patient objects links*

Description

The display.obj.links function displays a graph of connections between objects of a patient. The name of the objects corresponds to their modality (ct, mr, rtdose...) followed by their position in their respective lists in the patient list objects. Connected objects are linked by arrows. Objects sharing the same frame of reference have the same color except for objects with warnings, errors or missing planes which are all in grey. Approved objects are circled in green. By default, objects shapes are circles, except rtdose represented as squares.

Usage

```
display.obj.links(
  pat,
  obj.selected = NULL,
  exclusion = NULL,
  square = "rtdose",
  group.by.connected.FoR = TRUE,
  interactive = FALSE,
  random.seed = 314
)
```

Arguments

pat	"patient" class object, as loaded using load.patient.from.dicom, load.patient.from.Rdcm or toy.load.patient.
obj.selected	Dataframe (default to NULL) containing the objects already selected, created by a previous call of display.obj.links for example.
exclusion	Vector of patient file modalities that should not be displayed, as for instance "mr"...
square	Vector of patient file modalities that should be enclosed by a square, as for instance c ("ct", "mr")... If NULL no object name is squared.
group.by.connected.FoR	Boolean. If TRUE (default), all objects sharing the same frame of reference or connected by a registration matrix have the same color. If group.by.connected.FoR =FALSE, only objects sharing the same FoR have the same color.
interactive	Boolean. If interactive = TRUE, buttons are available on the graph to get information about the objects and select or remove them from the data frame of the selected objects. Then simply click on the name of the object on which to apply the chosen action. If interactive = FALSE no interaction possible with the plot.
random.seed	Positive Integer or NULL. If random.seed = NULL, the objects are laid out randomly. The layout is otherwise fixed.

Value

The function displays all patient objects, linked by an arrow when they are connected or a line when they belongs to the same DICOM object, and with a color and a shape depending on square, group.by.connected.FoR.

When interactive = TRUE, it returns a dataframe of the selected objects, or NULL if no object is selected.

Items are circled in green when the DICOM file has been approved. They are circled in red, when the DICOM series is incomplete (e.g. missing plan).

See Also

get.obj.connection

Examples

```
# loading of toy-patient objects
patient <- toy.load.patient (dxyz = c (5, 5, 5), beam.nb = 1)
display.obj.links (patient, group.by.connected.FoR = FALSE)
display.obj.links (patient, group.by.connected.FoR = TRUE)
display.obj.links (patient, group.by.connected.FoR = TRUE, random.seed=NULL)
```

---

display.palette *Display of the color scale of a color palette*

Description

The `Display.palette` function displays the color scale as it is used for representations in espadon functions

Usage

```
display.palette(
  col,
  breaks = NULL,
  factors = NULL,
  override.breaks = FALSE,
  bg = "black",
  new.window = TRUE,
  ylab = ""
)
```

Arguments

`col`	Vector of colors like the ones generated by `rainbow`, `heat.colors`, etc.
`breaks`	Vector of breaks for the color palette. It is the usual option for images or dose, for instance. Its length must be one unit more than `col` length.
`factors`	Vector containing the labels associated to each `col`. It should be used for tissue identification or image segment labelling. Its length must be `col` length.
`override.breaks`	Boolean. When `FALSE` (by default) ordinates are set to breaks. when `TRUE` colors are uniformely displayed, and associated breaks set to the correct ordinates for the given colors.
`bg`	Color of the background, seen by transparency for palette having alpha channel.
`new.window`	Boolean. If `TRUE`, it opens a new window for displaying the palette.
`ylab`	character string. Label of ordinates.

Value

Returns in a new device (if `new.window = TRUE`), or in the active graphics window (if `new.window = FALSE`), the palette color defined by `col` and `breaks` in priority, or by `col` and `factors`.

Note

the breaks are not necessarily evenly spaced. In this case, the colour palette can be represented as the breaks are defined (default option) or by choosing a constant spacing for each colour and displaying the associated abscissa calculated from the breaks (override.breaks = TRUE).

Examples

```
## Not run:
# simple example for breaks and factors

display.palette (c ("red", "green", "blue"), breaks = c(0, 1, 3, 7),
                 ylab = "a simple color palette")
display.palette (c ("red", "green", "blue"), breaks = c(0, 1, 3, 7),
                 override.breaks = TRUE)
display.palette (c ("red", "green", "blue"), factors = c ("red", "green", "blue"))
display.palette (c ("gray", "green", "blue"), factors = c (NA, 1, 2))

# for RVV palette, HU range must be [-1000, 1000]
display.palette (pal.RVV (255), breaks = seq (-1000, 1000, length.out=256))

# a palette for dose, for instance
display.palette (rainbow (255, start = 0, end = 4/6, rev = TRUE),
                 breaks = seq (0, 60, length.out = 256))

# black & white palette for CTs or MRs
display.palette (grey.colors (255, start = 0, end = 1),
                 breaks = seq (0, 60, length.out = 256))

# transparency affects colors depending on background (black in first exemple,
# white in the second one)
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),
                          breaks = seq (0, 60, length.out=256))
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),
                          breaks = seq (0, 60, length.out=256), bg = "white")

## End(Not run)
# colors contracted range using non uniform breaks in the plot window
display.palette (rainbow(255, s = seq(1, 0.8, length.out = 255),
                         start = 0, end = 4/6,
                         alpha = seq(0.8, 0.6, length.out = 255), rev = TRUE),
                 breaks = seq (0, 1, length.out = 256)^0.25 * 60, bg="grey",
                 new.window = FALSE)

# the same using breaks override
display.palette (rainbow(255, s = seq(1, 0.8, length.out = 255),
                         start = 0, end = 4/6,
                         alpha = seq(0.8, 0.6, length.out = 255), rev = TRUE),
                 breaks = seq (0, 1, length.out = 256)^0.25 * 60, bg="grey",
                 override.breaks = TRUE, new.window = FALSE)
```

`display.plane` *Display the transverse frontal or sagittal view in the patient reference system*

Description

The `display.plane` function displays an overlay of images and RoI closed planar contours on a plane defined by the equations x = constant (sagittal view), or y = constant (frontal view) or z = constant (transverse view) in a frame of reference chosen by the user.

Usage

```
display.plane(
  bottom = NULL,
  top = NULL,
  struct = NULL,
  roi.name = NULL,
  roi.sname = NULL,
  roi.idx = NULL,
  struct.dxyz = c(1, 1, 1),
  display.ref = NULL,
  T.MAT = NULL,
  interpolate = TRUE,
  view.type = c("trans", "front", "sagi"),
  view.coord = 0,
  bg = "#000000",
  abs.rng = NULL,
  ord.rng = NULL,
  bottom.col = grey.colors(255, start = 0, end = 1),
  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),
  bottom.breaks = NULL,
  top.breaks = NULL,
  sat.transp = FALSE,
  struct.lwd = 2,
  main = NULL,
  legend.plot = TRUE,
  legend.shift = 0
)
```

Arguments

`bottom`	"volume" class object, displayed using `bottom.col` palette. If `bottom = NULL`, no bottom image is displayed.
`top`	"volume" class object, displayed as an overlay, using `top.col` palette. If `top = NULL`, no overlay image is displayed.
`struct`	"struct" class object. If `NULL`, no RoI is displayed. Only RoI of closed planar or point type are displayed.
`roi.name`	Vector of exact names of the RoI in the `struct` object. By default `roi.name = NULL`. See Details.

`roi.sname` Vector of names or parts of names of the RoI in the `struct` object. By default `roi.sname = NULL`. See Details.

`roi.idx` Vector of indices of the RoI that belong to the `struct` object. By default `roi.idx = NULL`. See Details.

`struct.dxyz` Vector of 3 numbers. Used in case of `bottom` and `top` are set to `NULL`. It represents the virtual steps of a temporary volume created in the `display.ref` frame of reference, initialized at 1 mm in the 3 directions x, y and z.

`display.ref` Character string. Pseudonym of the frame of reference used for display. If `NULL` (default), the bottom image FoR, or top image FoR (when no bottom image), or struct FoR (when no volume displayed).

`T.MAT` "t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If `T.MAT` is `NULL`, `bottom`, `top` and `struct` must have the same frame of reference.

`interpolate` Boolean, indicating whether to apply trilinear interpolation to the `bottom` and `top` volumes. If `interpolate = FALSE`, the values of the nearest voxels are used. When `TRUE` (by delfault), trilinear interpolation is used.

`view.type` Character string, defining the view to display. It must be set to

- `"trans"` for a transverse view,
- `"front"` for a frontal view or,
- `"sagi"` for a sagittal view.

`view.coord` Numeric vector of the coordinates along the normal vector of the selected view.

`bg` Background color of the image. By default, this color is black.

`abs.rng` Vector of 2 elements indicating the minimum and maximum abscissa to display on the background image.

`ord.rng` Vector of 2 elements indicating the minimum and maximum ordinate to display on the background image.

`bottom.col, top.col`
Vectors, representing the palette color of `bottom` and `top`.

`bottom.breaks, top.breaks`
One of :

- `NULL` : the minimum and the maximum value of `bottom` or `top` define the range.
- Vector giving the breakpoints of each color. Outside values are transparent, leaving the background visible, depending on `sat.transp`.

When breaks are specified, the number of breaks must be one unit more then the number of colors.

`sat.transp` Boolean. If `TRUE`, outside values are transparent, else set to `bottom.breaks` or `top.breaks` limits.

`struct.lwd` Line thickness of the RoI contours.

`main` Character string. When `main` different from `NULL`, it replaces the title, and removes the subtitle and the maximum dose indication if `top` is of modality rtdose.

`legend.plot` Boolean, that indicates whether the RoI legend should be displayed on the image. It is displayed by default.

`legend.shift` Numeric. It shifts (in mm) the display of the RoI legend on x-axis.

Details

If `roi.name`, `roi.sname`, and `roi.idx` are all set to `NULL`, all closed planar or point RoI are selected. If a RoI is not present in the requested plane, the RoI legend won't mention it.

Value

Returns a display of the transverse, sagittal or frontal plane. This plane has the coordinate z = view.coord (transverse), y = view.coord (sagittal) pr x = view.coord (frontal). The display is an overlay of:

- a background image of uniform color bg
- the bottom image if it exists
- the top image if it exists
- the contours of the regions of interest if they exist in the plane considered.

Note

1- The main title is given by `bottom`, the subtitle by `top`.

2- When `top` is in the "rtdose" modality, the maximum dose is written on the image.

See Also

display.kplane.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "mr", "rtstruct", "rtdose"),
                             roi.name  = "",
                             dxyz = rep (step, 3), beam.nb = 3)
CT <- patient$ct[[1]]
MR <- patient$mr[[1]]
D <- patient$rtdose[[1]]
S <- patient$rtstruct[[1]]

display.plane (bottom = CT, top = D, struct = S, view.coord = -30,
               interpolate = FALSE, legend.shift = -80)
# Display of CT in reference frame "ref1" and  MR in "ref2"
display.plane (bottom = CT, top = MR, interpolate = FALSE)

# Display of CT and MR in reference frame "ref2"
display.plane (bottom = CT, top = MR, interpolate = FALSE, display.ref ="ref2",
               T.MAT = patient$T.MAT)
```

`espadon.class` *ESPADON class*

Description

ESPADON class

Usage

```
espadon.class()
```

Value

Returns a vector of **espadon** class names.

Note

Each object of a class has specific features that are used to display or process that object.

- the "patient" class includes :

- `$patient` : dataframe providing patient's information as PIN, birth date and gender.
- `$pat.pseudo` : patient's pseudonym, initialized to the patient's PIN of `$patient` dataframe.
- `$description` : dataframe describing the patient's DICOM objects: their modality (rtstruct, ct, mr, rtplan ...), the base name of the relevant source file in the patient's directory, the pseudonym of their frame of reference (ref1, ref2 ...), their number of sub-objects, their description if any, their numbers of slices/RoI for all sub-objects, their maximum voxels (for volume sub-objects), and finally the aliases of the sub-objects.
- `$description.by.reg`: list of DICOM objects descriptions that are linked by a transfer matrix.
- `$T.MAT` : list of class "t.mat" containing the information of the transfer matrices to move from one frame of reference to another. See load.T.MAT.
- `$ct` : list of CT, if any. They are named by their `$object.alias` See load.obj.from.Rdcm.
- `$mr` : list of MRI, if any. They are formatted like the `$ct`.
- `$rtdose` : list of dose matrices. They are formatted like the `$ct`.
- `$rtstruct`: list of struct objects.
- ...any DICOM objects other than the reg files, and those previously mentioned, or any modalities created by **espadon**.
- `$dicom.dvh`: if any, list of DVH computed in rt-dose DICOM files.

- the "t.mat" class includes :

- `$ref.info`: dataframe giving the correspondence between the frame of reference (column `$ref`) of the DICOM object (TAG (0020,0052)) and its pseudonym (column `$ref_pseudo`).
- `$reg.info`: list of dataframes : the first one gives the PID, birthday, and sex of the patient, the second one gives the name of the source file of transfer matrices.
- `$matrix.description`: dataframe giving the transfer matrix names (column `$t`), its source frame of reference (column `$src`), the destination frame of reference (column `$dest`), and its type (`$type`). Note that only the RIGID type is supported.
- `$matrix.list`: list of 4X4 transfer matrices. This list contains at least as many Identity matrices as there are `ref.pseudo`.

A **espadon** object of class "dvh", "fan", "histo","histo2D","mesh", "reg", "struct", "t.mat","undef","volume" is a list containing at least:

- `$patient`: patient's PIN.
- `$patient.name`: patient's name.
- `$patient.bd`: patient's birthday.
- `$patient.sex`: patient's sex
- `$file.basename`: vector of .Rdcm or .dcm file basenames of the object, if it exists
- `$file.dirname` : directory including the .Rdcm or .dcm file, if it exists

- `$object.name`: name of the object.
- `$object.alias`: alias of the object.
- `$frame.of.reference`: value of TAG (0020,0052).
- `$ref.pseudo`: pseudonym of the `$frame.of.reference`
- `$modality`: modality of the object (e.g. ct, mr, bin, rtplan..)
- `$description`: description of the object.
- `$acq.date` : date of the content (TAG (0008,0023) for ct and mr and rtimage, TAG (300A,0006) for rtplan, TAG (3006,0008) for rtstruct)
- `$creation.date` : creation date of the object.

If the object was generated from a DICOM file, the list also contains:

- `$object.info`: Information of the object. It includes:
  - the SOP ID (value of TAG (0008,0016)),
  - the transfer syntax UID (value of TAG (0002,0010)),
  - the SOP implementation ID (value of TAG (0002,0012)),
  - the SOP type (value of TAG (0008,0008)),
  - the study ID (value of TAG (0020,0010)),
  - the study UID (value of TAG (0020,000D)),
  - the serie UID (value of TAG (0020,000E)),
  - the scanning sequence (value of TAG (0018,0020)),
  - the list of SOP labels (values of TAG (0008,0018)),
  - the dicom source files,
  - the encoding of the content of text tags (values of TAG (0008,0005)) and
  - the number of sub-objects.

if the object is linked to another DICOM object, the list also contains:

- `$ref.object.alias`: Alias of the reference object.
- `$ref.object.info`: Information of the reference object (not available for mr and ct). It includes:
  - the SOP ID of the reference object,
  - the list of SOP names of the reference object.

- the "dvh" class also includes :

- `$nb.MC`: set to `histo$nb.MC`.
- `$breaks`: vector breakpoints.
- `$mids`: vector of cell centers.
- `$mids.unit`: Character string, representing the unit of the abcissa of the DVH. For instance, "Gy".
- `$vol`: cumulative volume receiving at least the doses defined by `$mids`.
- `$pcv`: percentage of the total volume receiving at least the doses defined by `$mids`.
- if `$nb.MC` is different from 0, the arrays `MC.vol`, `MC.pcv` and `MC.dxyz` are added. See histo.DVH.

- the "fan" class also includes :

- `$origin`: the xyz-coordinates of the source point.

- `$direction`: the xyz-coordinates of the main direction of the fan.
- `$orientation`: the xyz-coordinates of the two unit vectors of the plane orthogonal to the $direction.
- `$xyz`: the xyz-coordinates of the unit vectors of the fan rays
- `$local`: depending on the generation of the fan rays, it can be the spherical coordinates, the deflection angles, the voxel coordinates...

- the "histo" class also includes :

- `$nb.MC`: number of Monte-Carlo simulations
- `$breaks`: vector breakpoints
- `$mids`: vector of cell centers.
- `$mids.unit`: Character string, representing the unit of the abcissa of the histogram. For instance, "Gy".
- `counts`: count of voxels whose value is included in the limits defined by `$breaks`.
- `dV_dx`: differential histogram, expressed in cm^3 by voxel units, at each `$mids`.
- if `$nb.MC` is different from 0, the arrays `MC.counts`, `MC.dV_dx` and `MC.dxyz` are added. See histo.from.roi.

- the "histo2D" class also includes :

- `$nb.pixels`: number of elements in the `density.map`.
- `$x.file.src`: x label. See histo.2D.
- `$y.file.src`: y label. See histo.2D.
- `x.breaks`: vector of x-axis breakpoints.
- `y.breaks`: vector of y-axis breakpoints.
- `x.mids`: vector of x-axis cell centers.
- `y.mids`: vector of y-axis cell centers.
- `density.map`: array of densities.
- `total.counts`: number of counted voxels.

- the "mesh" class also includes :

- `$nb.faces`: set to the number of faces of the mesh.
- `$mesh`: list of 3 elements defining the mesh :`$vb`, `$it` and `$normals`. See mesh.from.bin.

- the "reg" class also includes :

- `$nb.of.ref`: number of transfer matrices.
- `$ref.data`: list including the lists of information on transfer matrices, namely: the source frame of reference (`$src`), the matrix type (`$type`, for example 'RIGID') and the transfer matrix (`$matrix`).

- the "rtplan" class also includes :

- `$approval.status`: value of TAG (300E,0002).
- `$number`: sub-object number.

- $plan.info: dataframe containing, if they exist,

- $label the label for the treatment plan,
- $plan.name the name for the treatment plan,
- $plan.description description of treatment plan,
- $tt.protocol the treatment protocol,
- $plan.intent the intent of this plan,
- $tt.site describing the anatomical treatment site,
- $geometry describing whether RT Plan is based on patient or treatment device geometry.

- $presc.dose: dataframe containing, if they exist,

  - $ref.roi.nb value of TAG (3006,0084),
  - $dose.ref.nb value of TAG (300A,0012),
  - $dose.ref.id value of TAG (300A,0013),
  - $struct.type value of TAG (300A,0014),
  - $description value of TAG (300A,0016),
  - $pt.coord value of TAG (300A,0018),
  - $nominal.prior.dose value of TAG (300A,001A),
  - $dose.type value of TAG (300A,0020),
  - $constraint.weight value of TAG (300A,0021),
  - $deliv.warn.dose value of TAG (300A,0022),
  - $deliv.max.dose value of TAG (300A,0023),
  - $targ.min.dose value of TAG (300A,0025),
  - $targ.presc.dose value of TAG (300A,0026),
  - $targ.max.dose value of TAG (300A,0027),
  - $targ.underdose.vol.frac value of TAG (300A,0028),
  - $org.risk.full.vol.dose value of TAG (300A,002A),
  - $org.risk.lim.dose value of TAG (300A,002B),
  - $org.risk.max.dose value of TAG (300A,002C),
  - $org.risk.overdose.vol.frac value of TAG (300A,002D)

- $fraction.info: dataframe containing, if they exist,

- $fraction.id the id of the fraction group,
- $description its description,
- $planned.frac.nb the total number of treatments (Fractions) prescribed for current fraction group,
- $frac.pattern.digit.per.day.nb the number of digits in $frac.pattern used to represent one day,
- $repeat.frac.cycle.le the number of weeks needed to describe treatment pattern,
- $frac.pattern the value of TAG (300A,007B) describing treatment pattern every day,
- $nb.of.beam the number of beams in current fraction group,
- $beam.dose.meaning the value of TAG (300A,008B) indicating the meaning of Beam Dose,
- $nb.of.brachy.app the number of brachy application setups in current fraction group.

- $fraction.beam (in case of beam treatment): dataframe containing, if they exist,

  - $fraction.id,
  - $nb.of.frac.planned,
  - $beam.dose the value of TAG (00A,0084),
  - $beam.specif.pt the value of TAG (300A,0082),
  - $beam.meterset the value of TAG (300A,0086),
  - $beam.type the value of TAG (300A,0090,

- $alt.dose the value of TAG (300A,0091),
- $alt.type the value of TAG (300A,0092,
- $duration.lim the value of TAG (300A,00C5),
- $beam.nb the value of TAG (300C,0006) or (300A,00C0),

- $beam.info (in case of beam treatment): dataframe containing, if they exist,

  - $beam.nb the value of TAG (300C,0006) or (300A,00C0),
  - $beam.name the value of TAG (300A,00C2),
  - $beam.description the value of TAG (300A,00C3),
  - $beam.type the value of TAG (300A,00C4),
  - $radiation.type the value of TAG (300A,00C6),
  - $high.dose.technique.type the value of TAG (300A,00C7),
  - $treatment.machine.name the value of TAG (300A,00B2),
  - $device.serial.nb the value of TAG (0018,1000),
  - $primary.dosimeter.unit the value of TAG (300A,00B3),
  - $referenced.tolerance.table.nb the value of TAG (300C,00A0),
  - $src.axis.dist the value of TAG (300A,00B4),
  - $referenced.patient.setup.nb the value of TAG (300C,006A),
  - $treatment.delivery.type the value of TAG (300A,00CE),
  - $wedges.nb the value of TAG (300A,00D0),
  - $compensators.nb the value of TAG (300A,00E0),
  - $total.compensator.tray.factor the value of TAG (300A,00E2),
  - $boli.nb the value of TAG (300A,00ED),
  - $blocks.nb the value of TAG (300A,00F0),
  - $total.block.tray.factor the value of TAG (300A,00F2),
  - $final.cumul.meterset.weight the value of TAG (300A,010E),
  - $ctl.pts.nb the value of TAG (300A,0110),
  - $radiation.mass.nb the value of TAG (300A,0302),
  - $radiation.atomic.nb the value of TAG (300A,0304),
  - $radiation.charge.state the value of TAG (300A,0306),
  - $scan.mode the value of TAG (300A,0308),
  - $modulated.scan.mode.type the value of TAG (300A,0309),
  - $virtual.src.axis.dist the value of TAG (300A,030A),
  - $total.wedge.tray.water.equ.thickness the value of TAG (300A,00D7),
  - $total.compensator.tray.water.equ.thickness the value of TAG (300A,02E3),
  - $total.block.tray.water.equ.thickness the value of TAG (300A,00F3),
  - $range.shifters.nb the value of TAG (300A,0312),
  - $lateral.spreading.devices.nb the value of TAG (300A,0330),
  - $range.modulators.nb the value of TAG (300A,0340),
  - $fixation.light.azimuthal.angle the value of TAG (300A,0356),
  - $fixation.light.polar.angle the value of TAG (300A,0358).

- $beam.ctl.pt (in case of beam treatment): list containing, for each beam,

- $info a data.frame of control points information from DICOM
- $beam.source the coordinates of the source in the patient frame of reference
- $beam.direction the coordinates of the beam direction in the patient frame of reference
- $beam.direction the coordinates of the beam orientation in the patient frame of reference
- $beam.isocenter the coordinates of the isocenter in the patient frame of reference
- $spot.map, if they exist,the coordinates of the spots in the patient frame of reference

For the moment, only the rotations of the gantry and the patient support, and the position of the isocenter are taken into account in the calculation of these coordinates.

- `$brachy.info` (in case of brachy treatment): dataframe containing, if they exist,

  - `$fraction.id`
  - `$nb.of.frac.planned`,
  - `$brachy.dose` the value of TAG (300A,00A4),
  - `$brachy.nb` the value of TAG (300C,000C),
  - `$brachy.specif.pt` the value of TAG (300A,00A).

- the "struct" class also includes :

- `$nb.of.roi`: number of regions of interest (RoI).
- `$thickness`: thickness between two consecutive planes of a contour.
- `$ref.from.contour`: reference frame change matrix, from the contour reference frame to the ref.pseudo reference frame
- `$roi.info`: dataframe. Information on RoI contours. It includes the followings columns:

  -`$number`: value of TAG (3006,0084) for the concerned RoI.
  -`$name`: value of TAG (3006,0026) for the concerned RoI.
  -`$description`: value of TAG (3006,0028) for the concerned RoI.
  -`$generation.algorithm`: value of TAG (3006,0036) for the concerned RoI.
  -`$color`: value of TAG (3006,002A) for the concerned RoI.
  -`$dz`: z step between planes for the concerned RoI.
  -`$roi.pseudo`: pseudonym of the RoI `$name`. It can be changed by the user.
  -`$min.x`: minimum value x in mm of the RoI. Absent when `data = FALSE`.
  -`$max.x`: maximum value x in mm of the RoI. Absent when `data = FALSE`.
  -`$min.y`: minimum value y in mm of the RoI. Absent when `data = FALSE`.
  -`$max.y`: maximum value y in mm of the RoI. Absent when `data = FALSE`.
  -`$min.z`: minimum value z in mm of the RoI. Absent when `data = FALSE`.
  -`$max.z`: maximum value z in mm of the RoI. Absent when `data = FALSE`.
  -`$vol`: volume in mm^3 of the RoI. Absent when `data = FALSE`.
  -`$Gx`: position x in mm of the RoI center of gravity. Absent when `data = FALSE`.
  - -`$Gy`: position y in mm of the RoI center of gravity. Absent when `data = FALSE`.
  - -`$Gz`: position z in mm of the RoI center of gravity. Absent when `data = FALSE`.
  - -`$continue`: boolean, indicating whether the contours are on adjacent planes.

- `$roi.obs`: dataframe. RT RoI observations (sequence TAG (3006,0080)). It includes the followings columns :

  -`$nb`: value of TAG (3006,0082) for the concerned RoI.
  -`$roi.nb`: value of TAG (3006,0084) for the concerned RoI.
  -`$label`: value of TAG (3006,0085) for the concerned RoI.
  -`$code.value`: value of TAG (0008,0100) in the Identification code sequence.
  -`$code.scheme`: value of TAG (0008,0102) in the Identification code sequence.
  -`$code.scheme.v`: value of TAG (0008,0103) in the Identification code sequence.
  -`$code.meaning`: value of TAG (0008,0104) in the Identification code sequence.
  -`$type`: value of TAG (3006,00A4) for the concerned RoI.
  -`$interpreter`:value of TAG (3006,00A6) for the concerned RoI.

- `$roi.data`: exists only if the data is loaded. Contains the list of contour coordinates. The RoI of list number i is that of line i of roi.info. Each element of the list is a list giving the contour information for each plane, namely:

  - `$type`: value of TAG (3006,0042).
  - `$pt`: dataframe of the coordinates of the contour points.
  If the contour is closed (i.e.`$type = "CLOSED_PLANAR"`),
  then the first point is repeated at the end.
  - `$level`: contour inclusion level. If this number is even,
  the inside of the closed contour belongs to the RoI.
  Otherwise, if odd, the inside of the closed contour is excluded from the RoI.

- the "undef" class : is used for DICOM objects that will not be processed further by **espadon** functions. It can also include what the user wants.

- the "volume" class also includes :

- `$number`: sub-object number.
- `$n.ijk`: vector defining the number of indices i, j, k. The product `prod(...$n.ijk)` represents the number of voxels in the 3D volume.
- `$slice.thickness`: thickness in mm of a plane.
- `$min.pixel`: minimum value of voxels in the volume.
- `$max pixel`: maximum value of voxels in the volume.
- `$dxyz`: x, y, z steps in mm.
- `$orientation`: value of TAG (0020,0037). Vector, comprising the vectors i and j defining the orientation of the patient with respect to the volume planes.
- `$xyz0`: in the patient frame of reference, position of the first voxel of each plane.
- `$xyz.from.ijk`: transfer matrix of the voxels i, j, k indices to the position x, y, z in mm in the patient's frame of reference.
- `$k.idx`: index of planes in the 3D volume.
- `$missing.k.idx`: Boolean, indicating if k is a continuous sequence of integers.
- `$cube.idx`: 3D volume vertices indices.
- `$vol3D.data`: exists only if the data is loaded. 3D array of the voxel values of the 3D volume.

See Also

toy.load.patient, load.patient.from.dicom, load.patient.from.Rdcm, load.T.MAT histo.DVH, histo.vol, histo.from.roi, histo.from.bin, histo.2D, mesh.from.bin, load.obj.from.Rdcm

Examples

```
cat ("espadon class names are:", paste (espadon.class(), collapse = ", "))
```

`fan.beam` *Creation of pyramid fan object with constant angle step.*

Description

The `fan.beam` function creates a "fan" class object containing, among others, the coordinates of the unit director vectors of the rays of rectangular pyramid fan. Rays are uniformly distributed by angle.

Usage

```
fan.beam(
  alpha,
  dalpha,
  orientation = c(0, 0, 1, 1, 0, 0),
  origin = c(0, 0, 0),
  ref.pseudo = "",
  frame.of.reference = "",
  alias = "",
  description = "beam fan"
)
```

Arguments

`alpha`	Positive number specifying the half-angle of the conical beam.
`dalpha`	Positive number specifying the step of the angle between the rays of the cone beam.
`orientation`	Vector orientation of the pyramid base composed by the 2 orthonormal vectors coordinates.
`origin`	Numeric vector, giving the xyz coordinates of the fan origin. By default `c(0, 0, 0)`.
`ref.pseudo`	Character string, frame of reference pseudonym of the created object.
`frame.of.reference`	Character string, frame of reference of the created object.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object.

Value

Returns a "fan" class object (see espadon.class for class definitions) containing, among others,

- `$xyz` : a matrix of 3 columns giving the xyz coordinates of the fan rays.
- `$local` : a matrix of 2 columns indicating the deflection angle (in rad) in the main directions defined by `orientation`.

See Also

fan.planar, fan.sphere, fan.to.voxel.

Examples

```
fan <- fan.beam (alpha = 30, dalpha = 1)
head (fan$xyz)
library (rgl)
open3d ()
points3d (fan$xyz)
```

`fan.planar` *Creation of pyramid fan object passing through pixels of a plane.*

Description

The `fan.planar` function creates a "fan" class object containing, among others, the coordinates of the unit director vectors of the rays of rectangular pyramid fan. Rays are passing through all pixels of a plane, which represent the pyramid basis. It is for instance useful to compute rt-image.

Usage

```
fan.planar(
  vol,
  k = vol$k.idx[ceiling(length(vol$k.idx)/2)],
  origin = c(0, 0, 0),
  alias = "",
  description = "planar fan"
)
```

Arguments

`vol`	"volume" class object.
`k`	Positive number specifying the plane index that the rays of the fan must cross. By default, k is the central plane.
`origin`	Numeric vector, giving the xyz coordinates of the fan origin. By default `c(0, 0, 0)`.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object.

Value

Returns a "fan" class object (see espadon.class for class definitions) containing, among others,

- `$xyz` : a matrix of 3 columns giving the xyz coordinates of the fan rays.
- `$local.coords` : a list of the ijkt DICOM coordinates of the crossed plane, and the transfer matrix to xyz.from.ijk to compute xyz coordinates in `$ref.pseudo`.

See Also

fan.sphere, fan.beam, fan.to.voxel

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
patient <- toy.load.patient (modality = c("ct"), dxyz = rep (step, 3))
fan <- fan.planar (patient$ct[[1]], origin = patient$ct[[1]]$xyz0[1,])
head (fan$xyz)
library (rgl)
open3d ()
points3d (fan$xyz)
```

`fan.sphere` *Creation of spherical fan object.*

Description

The `fan.sphere` function creates a "fan" class object containing, among others, the coordinates of the unit director vectors of the rays of a spherical fan.

Usage

```
fan.sphere(
  angle = 1,
  method = c("regular", "random"),
  origin = c(0, 0, 0),
  ref.pseudo = "",
  frame.of.reference = "",
  alias = "",
  description = "spherical fan"
)
```

Arguments

`angle`	Positive number specifying the angle (or mean angle in case of `method = "random"`) between two nearest vectors.
`method`	Requested method of fan calculation from among 'regular' and 'random'. By default, `method = regular`. See details.
`origin`	Numeric vector, giving the xyz coordinates of the fan origin. By default `c (0, 0, 0)`.
`ref.pseudo`	Character string, frame of reference pseudonym of the created object.
`frame.of.reference`	Character string, frame of reference of the created object.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object.

Details

The "regular" and "random" method are explained by *Deserno* **[1]**.

- If `method = "regular"`, the returned vectors composing `$xyz` matrix are regularly equidistributed at the specified angle.
- If `method = "random"`, the returned vectors composing `$xyz` matrix are randomly equidistributed at the specified angle.

Value

Returns a "fan" class object (see espadon.class for class definitions) containing, among others,

- `$xyz` : a matrix of 3 columns giving the xyz coordinates of the fan rays.
- `$local` : a matrix of 2 columns indicating the polar angle `theta` (rad) and the azimuthal angle `phi` (rad) of each ray are added.

References

[1] Deserno, Markus (Online; accessed 2022-08-24). "How to generate equidistributed points on the surface of a sphere." https://www.cmu.edu/biolphys/deserno/pdf/sphere_equi.pdf.

See Also

fan.beam, fan.planar, fan.to.voxel

Examples

```
regular.fan <- fan.sphere (angle = 30)
head (regular.fan$xyz)
random.fan <- fan.sphere (angle = 30, method = "random")
head (random.fan$xyz)
library (rgl)
open3d ()
points3d (regular.fan$xyz)
open3d ()
points3d (random.fan$xyz)
```

`fan.to.voxel` *Indices of voxels crossed by a fan*

Description

The `fan.to.voxel` function computes the indices of voxels crossed by a fan. It is useful for retrieving voxel values and voxel indices of a volume (dose or ct) along the fan rays.

Usage

```
fan.to.voxel(vol, fan, restrict = FALSE, vol.value = 1)
```

Arguments

vol	"volume" class object.
fan	"fan" class object created by fan.sphere for example.
restrict	Boolean. If TRUE, only the voxels with a value equal to vol.value are taken into account.
vol.value	Value of the voxels taken into account, in case of restrict = TRUE

Value

Returns a dataframe of 4 columns. Each line gives:

- column "ray.index": the index (i.e. the row number) of the ray concerned in fan$dxyz,
- column "vol.index": the index of the voxel crossed in vol$vol.3Ddata,
- column "l.in": the distance between fan source (i.e. fan$origin) and the first face of the voxel crossed by the ray,
- column "dl": the distance crossed by the ray in the voxel.

If the rays do not cross any voxel, the dataframe has no row.

See Also

fan.beam, fan.planar, fan.sphere.

Examples

```
vol <- vol.create (pt000 = c(1,10,10), dxyz = c (1 , 1, 1),
                   n.ijk = c(100, 100, 100))
fan.origin <- c (50,50,50)
fan <- fan.sphere (angle = 10, origin = fan.origin)
fan.voxel <- fan.to.voxel (vol = vol, fan = fan)
head (fan.voxel)

# Use of the 2nd column of fan.voxel to select voxels
bin <- vol.copy (vol, modality = "binary")
bin$vol3D.data[] <- FALSE
bin$vol3D.data[fan.voxel[,2]] <- TRUE
bin$max.pixel <- TRUE
bin$min.pixel <- FALSE
display.kplane(bin, k=10)
```

---

get.extreme.pt *Coordinates of the extreme points*

Description

The get.extreme.pt function returns the x, y, z coordinates of the 2 extreme voxels of the rectangular parallelepiped, containing the objet obj of class volume, struct or mesh. These coordinates are given in the ref.pseudo frame of reference.

Usage

```
get.extreme.pt(obj, ref.pseudo = obj$ref.pseudo, T.MAT = NULL, ...)
```

Arguments

obj	object of class volume or struct or mesh.
ref.pseudo	Pseudonym of the frame of reference in which you want the coordinates.
T.MAT	"t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If T.MAT = NULL, ref.pseudo must be equal to obj$ref.pseudo.
...	Additional arguments min, max if obj is of class 'volume'. Arguments roi.name, roi.sname, roi.idx if obj is of class 'struct'. Arguments vol (depracated), replaced by obj.

Value

Returns a dataframe of min and max columns, and x, y and z rows.

- If obj is a member of the class volume: the returned dataframe represents the coordinates of the 2 extreme points of the rectangle parallelepiped including all the voxels such as min <= obj$vol3D.data <= max, if the arguments min or max exist, or including all the voxels otherwise.
- If obj is a member of the class struct: the returned dataframe represents the coordinates of the 2 extreme points of the rectangular parallelepiped including all the selected RoI.
- if obj is a member of the class mesh: the returned dataframe represents the coordinates of the 2 extreme points of the rectangular parallelepiped including all the mesh.

Examples

```
# loading of toy-patient objects
patient <- toy.load.patient (modality = "ct", roi.name = "", dxyz = c (5, 5, 5))
CT <- patient$ct[[1]]

# xyz extreme coordinate
get.extreme.pt (CT)
get.extreme.pt (CT, min = 0)
```

---

get.ijk.from.index *Conversion of the indices of a point into ijk vector*

Description

The get.ijk.from.index function converts the voxel indices of vol$vol3D.data (for example, obtained with the function which) into a vector or matrix of DICOM indices i, j, k.

Usage

```
get.ijk.from.index(idx, vol)
```

Arguments

idx	Index, or matrix of voxel indices of the array vol$vol3D.data.
vol	"volume" class object.

Value

Returns an i, j, k column matrix of the DICOM indices of the points of `vol$vol3D.data`.

See Also

get.value.from.ijk, display.kplane

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 4
patient <- toy.load.patient (modality = "rtdose", roi.name = "",
                             dxyz = rep (step, 3), beam.nb = 3)
D <- patient$rtdose[[1]]

# voxels location where the dose is greater than 99.9% Dmax
Dmax <- max (D$vol3D.data, na.rm = TRUE) # D$max.pixel
get.ijk.from.index (which (D$vol3D.data >= 0.999 * Dmax), D)
# or
get.ijk.from.index (which (D$vol3D.data >= 0.999 * Dmax, arr.ind = TRUE), D)

ijk <- as.numeric (get.ijk.from.index (which.max (D$vol3D.data), D))
display.kplane (D, k = ijk[3])
```

`get.ijk.from.xyz` *Indices relating to the coordinates of the points*

Description

The `get.ijk.from.xyz` function calculates the i, j, k DICOM indices of the points given in the patient's reference frame.

Usage

```
get.ijk.from.xyz(xyz = matrix(c(0, 0, 0), ncol = 3), vol, verbose = FALSE)
```

Arguments

`xyz`	Vector of length 3, corresponding to the x, y, z coordinates (in mm) of a point in the patient's frame of reference, or 3-column matrix of x, y, z coordinates of several points.
`vol`	"volume" class object.
`verbose`	Boolean, default to `FALSE`. If `verbose = TRUE`, then the `xyz` coordinates are printed.

Value

Returns a vector or a matrix of the i, j, k DICOM indices of the x, y, z coordinate points in the patient's frame of reference.

Note

The vector or matrix is made up of real numbers. It is up to the user to make the indices as integer.

The indices of the first voxel `vol` are 0, 0, 0. WARNING: As i,j,k are DICOM indices, they are not directly related to array indices. To get the value of the `vol$vol3D.data`, use the function get.value.from.ijk.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "ct", roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]

get.ijk.from.xyz (xyz = CT$xyz0[1,], vol = CT, verbose = TRUE)
get.ijk.from.xyz (xyz = c (1,1,1), vol = CT, verbose = TRUE)

index <- get.ijk.from.xyz (xyz = c (1,1,1), vol = CT)
floor (index)

index <- get.ijk.from.xyz (xyz = matrix (c (0,0,0,1,1,1), ncol = 3, byrow = TRUE),
                           vol = CT)
floor (index)
```

---

get.line *Image value along an axis*

Description

The `get.line` function calculates the value of the points of a volume `vol` along an axis in any direction.

Usage

```
get.line(
  vol,
  origin = c(0, 0, 0),
  orientation = c(1, 0, 0),
  grid = seq(-100, 100, 1),
  interpolate = TRUE
)
```

Arguments

`vol`	"volume" class object.
`origin`	Vector of x, y, z coordinates belonging to the line to extract. If `interpolate = FALSE`, these coordinates are replaced by the coordinates of the voxel closest to `origin`.
`orientation`	Directing vector of the line in the `vol` frame of reference. This vector is internally normalized.
`grid`	Vector, representing the curvilinear coordinates on the line to extract.

`interpolate` Boolean, default to `TRUE`. If `interpolate = TRUE`, a trilinear interpolation of the value of the voxels, relative to the values of adjacent voxels, is performed.

Value

Returns a dataframe, composed of the columns `$x`, `$y`, `$z`, representing the coordinates of the points where the values are taken in `vol` volume, the column `$s` representing the curvilinear abcissa, and the column `$value` representing values along `$s`.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 4
patient <- toy.load.patient (modality = "rtdose", roi.name = "",
                             dxyz = rep (step, 3), beam.nb = 3)
D <- patient$rtdose[[1]]

# Dose at maximum dose
origin <- get.xyz.from.index (which.max (D$vol3D.data), D)
display.plane (bottom = D, view.coord =  origin[3],
               bg = "#0000ff")

# Dose profile at x = origin[1] and z = origin[3].
l <- get.line (vol = D, origin = origin,
               orientation = c (0, 1, 0), interpolate = FALSE)
plot (l$y, l$value, type = "l")
grid ()

# Dose profile at y = origin[2] and z = origin[3].
l <- get.line (D, origin = origin,
               orientation = c (1, 0, 0), interpolate = FALSE)
plot (l$s, l$value, type = "l")
grid ()
```

---

get.obj.connection *List of connections between objects*

Description

The `get.obj.connection` function describes with 4 matrices the different connections between the DICOM objects of the patient.

Usage

```
get.obj.connection(pat)
```

Arguments

`pat` "patient" class object, as loaded using load.patient.from.dicom, load.patient.from.Rdcm or toy.load.patient.

Value

Returns a list of 4 named matrices:

- the `$adjacency` matrix matrix specifies the source objects that generated the destination objects: the column names correspond to the destinations, and the row names to the sources.
- the `$same.object` matrix specifies the elements belonging to the same DICOM object.
- the `$components` matrix specifies the objects belonging to the same study.
- the `$same.ref` matrix specifies the objects that share the same frame of reference, or with frames of reference linked in T.MAT (by a DICOM reg file for instance)

See Also

display.obj.links

Examples

```
# loading of toy-patient objects
patient <- toy.load.patient (dxyz = c (5, 5, 5), beam.nb = 1)
get.obj.connection (patient)
display.obj.links (patient)
```

`get.plane` *Extracting a plane from a volume*

Description

The `get.plane` function extracts a plane from a "volume" class object.

Usage

```
get.plane(
  vol,
  origin = c(0, 0, 0),
  plane.orientation = c(1, 0, 0, 0, 1, 0),
  alias = "plane.n",
  xgrid = NULL,
  ygrid = NULL,
  interpolate = TRUE
)
```

Arguments

`vol`	"volume" class object.
`origin`	Vector of x, y, z coordinates, representing the origin of the plane to extract. If `interpolate = FALSE`, these coordinates are replaced by the coordinates of the voxel closest to `origin`.
`plane.orientation`	Vector orientation of the plane in the `vol` frame of reference, composed by the 2 vectors coordinates of the orthonormal basis of the plane. First vector is x-axis, and second one is y-axis.

alias	$object.alias of the created object.
xgrid	Vector, representing the grid of the plane abscissa. See note.
ygrid	Vector, representing the grid of the plane ordinates. See note. If ygrid = NULL, the ordinate is the line intercepting the volume and the step is set to the projection of vol$dxyz onto the ordinate orientation.
interpolate	Boolean, default to TRUE. If interpolate = TRUE, a trilinear interpolation of the value of the voxels, relative to the values of adjacent voxels, is performed.

Value

Returns a "volume" class object, containing only a single plane, at k = 0, in the same frame of reference as vol. This returned object has 2 new fields local.xgrid, and local.ygrid, representing the local grids of the abscissa (columns) and ordinate (rows) of the plane.

Returns NULL if plane doesn't exist.

Note

Determination of axes :

- the x-axis has plane.orientation[1:3] as unit vector.
- the y-axis has plane.orientation[4:6] as unit vector.
- If xgrid is not NULL, origin + x.grid * plane.orientation[1:3] are the coordinates of the points on the x axis.
- If ygrid is not NULL, origin + y.grid * plane.orientation[4:6] are the coordinates of the points on the y axis.
- If xgrid or ygrid are NULL, they are determined to represent as closely as possible the initial volume in the required cut.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = "mr", dxyz = rep (step, 3))
MR <- patient$mr[[1]]

# mid-volume point
mid.point <- apply (get.extreme.pt (MR),1,mean)

plane <- get.plane (MR, origin = mid.point, interpolate = TRUE)
display.kplane (plane, interpolate = FALSE)

plane <- get.plane (MR, origin = mid.point, xgrid = seq (-50, 50, 1),
                    ygrid = seq (-50, 50, 1), interpolate = TRUE)
display.kplane (plane, interpolate = FALSE)

# 3 points on the inclined plane
pts <- t ((MR$xyz.from.ijk %*% MR$cube.idx) [1:3 , c (1, 2, 7)])
orientation <- orientation.create (A = pts[1,], B = pts[2,], C = pts[3,])
origin <- apply (pts, 2, mean)
plane <- get.plane (MR, origin = origin,
                    plane.orientation = orientation, interpolate = TRUE)
display.kplane (plane, interpolate = FALSE)
```

```
orientation <- orientation.create (A = c (0, 0, 0) , B = c (1, 1, 0),
                                   C = c (-1, 1, 0))
plane <- get.plane (MR, origin = origin,
                    plane.orientation = orientation, interpolate = TRUE)
display.kplane (plane, interpolate = FALSE)
```

---

get.rigid.M *Transfer matrix between two frames of reference*

Description

The function `get.rigid.M` provides, from the T.MAT list created by the functions load.patient.from.Rdcm, load.patient.from.dicom or load.T.MAT, the 4x4 transfer matrix from the FoR (frame o reference) pseudonym `src.ref` to the FoR pseudonym `dest.ref`.

Usage

```
get.rigid.M(T.MAT, src.ref, dest.ref)
```

Arguments

`T.MAT`	"t.mat" class object, created by the functions load.patient.from.Rdcm, load.patient.from.dicom or load.T.MAT
`src.ref`	Pseudonym of the source frame of reference
`dest.ref`	Pseudonym of the destination frame of reference

Value

Returns the 4x4 transfer matrix `dest.ref` from `src.ref`.

Examples

```
# loading of toy-patient objects
patient <- toy.load.patient (modality = c ("ct", "mr"), roi.name = "",
                             dxyz = c(5, 5, 5))
get.rigid.M (patient$T.MAT, "ref1", "ref2")
```

---

get.roi.connection *List of inter-connections between RoI*

Description

The `get.roi.connection` function describes the interconnections between Regions of Interest (RoI), from an imaging volume of "cluster" modality, created by struct.clustering.

Usage

```
get.roi.connection(vol)
```

Arguments

vol	"volume" class object of "cluster" modality, created by struct.clustering

Value

Returns the list of regions of interest (RoI), where each item in the list gives the inter-connections with other RoI.

See Also

struct.clustering

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 5
patient <- toy.load.patient (modality = c ("mr", "rtstruct"),
                             dxyz = rep (step, 3))
MR <- patient$mr[[1]]
S <- patient$rtstruct[[1]]
cluster.vol <- struct.clustering (MR, S, T.MAT = patient$T.MAT, verbose = FALSE)

get.roi.connection (cluster.vol)
```

---

get.value.from.ijk *Value of the volume at a selection of DICOM indices*

Description

The get.value.from.ijk function calculates the value of a "volume" class object at DICOM indices i, j, k, whether they are integers or not.

Usage

```
get.value.from.ijk(ijk, vol, interpolate = TRUE)
```

Arguments

ijk	Vector or 3-column matrix of DICOM indices.
vol	"volume" class object.
interpolate	Boolean, default to TRUE. If interpolate = TRUE, a trilinear interpolation of the value of the voxels, relative to the values of adjacent voxels, is performed.

Value

Returns a vector of the values of the volume at the requested DICOM indices.

See Also

get.ijk.from.index.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 4
patient <- toy.load.patient (modality = "rtdose", roi.name = "",
                             dxyz = rep (step, 3), beam.nb = 3)
D <- patient$rtdose[[1]]
# isodose
Dmax <- max (D$vol3D.data, na.rm = TRUE)
Dmax
idx <- which (D$vol3D.data >= (Dmax -1) & D$vol3D.data <= (Dmax - 0.2))
ijk <- get.ijk.from.index (idx, D)
get.value.from.ijk (ijk, vol = D, interpolate = FALSE)
```

---

get.value.from.mesh *Voxel value at a given depth of a mesh*

Description

The get.value.from.mesh function is used to retrieve the values of an object of class "volume" at the desired depth of a surface described by a mesh. If the mesh corresponds to the "patient" contour, the zero depth is the skin, positive depths enter the patient and negative depths exit to the outside.

Usage

```
get.value.from.mesh(
  mesh,
  vol,
  method = c("point", "disk", "sphere"),
  depth = 0,
  radius = 5,
  spacing = 1,
  T.MAT = NULL,
  FUN = median,
  ...
)
```

Arguments

mesh	espadon "mesh" class object, or rgl/Rvcg "mesh3d" class object. "mesh3d" class object shall an additional field "ref.pseudo" specifying the mesh frame of reference.
vol	"volume" class object.
method	string specifying the desired method for retrieving measurements in vol. by default "point". Other methods exist, for example "disk" or "sphere. See details.
depth	Numeric value, representing the depth, relative to the surface of the mesh, at which values are retrieved. 0 corresponds to the surface, positive values enter the volume used to define the mesh and negative values leave it.
radius	Positive number, defining the radius of the disk or sphere, depending on the desired method.

spacing	spacing of the measurement points on the disk or sphere.
T.MAT	"t.mat" class object, created by load.patient.from.Rdcm, load.patient.from.dicom, load.T.MAT or ref.add.
FUN	function to be applied to reduce the data ("disk" or "sphere" method) to a single value. Default, median value.
...	Additional arguments passed to FUN if needed.

Details

The get.value.from.mesh function works at each vertex of the mesh. It moves along the normal at that point to the desired depth.

- When the method is "point", it simply retrieves the value of the volume vol specified at that point.
- When the method is "disk", the values are retrieved on the disk orthogonal to the normal,with radius radius.
- When the method is "sphere", the values are retrieved inside the sphere of radius radius.

For "disk" or "sphere", the measurement points are spaced by spacing. For radius=5 and spacing=1, "disk" and "sphere" perform 78 and 523 measurements respectively. In both cases, the measured values must be reduced to a single result using the FUN function. By default, espadon uses the median, but it can be provided with more complex functions to filter the data efficiently (see example below).

Value

Returns a vector of values measured at the requested depth, with the desired method, filtered by FUN, at each vertex of the mesh.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct"), roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

# creation of the patient mesh
bin <- bin.from.roi (CT, struct = S, roi.name = "patient")
mesh.patient <- mesh.from.bin (bin, alias = "patient", verbose = FALSE)

# density value on the skin contour, extracted from CT
density <- get.value.from.mesh (mesh.patient, CT ,depth = 0)

# Display of mesh, with RVV pal
density[density < -1000] <- -1000
density[density > 1000] <- 1000
col <- pal.RVV(255)[cut (density, seq (-1000, 1000, length.out = 256),
                         include.lowest=TRUE)]
library (rgl)
open3d ()
display.3D.mesh (mesh.patient, col = col)
```

`get.value.from.xyz` *Voxel values on a selection of points*

Description

The `get.value.from.xyz` function calculates the voxel values at the x, y, z coordinate points in the chosen frame of reference.

Usage

```
get.value.from.xyz(
  xyz,
  vol,
  xyz.ref.pseudo = NULL,
  T.MAT = NULL,
  interpolate = TRUE,
  verbose = FALSE
)
```

Arguments

`xyz`	Vector of length 3, corresponding to the x, y, z coordinates (in mm) of a point in `xyz.ref.pseudo` frame of reference, or 3-column matrix or dataframe of x, y, z coordinates of several points.
`vol`	"volume" class object.
`xyz.ref.pseudo`	`ref.pseudo` in which the `xyz` coordinate points are given. This `ref.pseudo` must exist in the `T.MAT` list. If `ref.pseudo` is `NULL` then the point with coordinates xyz is considered to be in the patient frame of reference `vol$ref.pseudo`.
`T.MAT`	"t.mat" class object, created by load.patient.from.Rdcm, load.patient.from.dicom or load.T.MAT. If `T.MAT = NULL`, `xyz.ref.pseudo` must be equal to `vol$ref.pseudo` or `NULL`.
`interpolate`	Boolean, default to `FALSE`. If `interpolate = TRUE`, a trilinear interpolation of the value of the voxels, relative to the values of adjacent voxels, is performed.
`verbose`	Boolean, default to `FALSE`. If `verbose = TRUE`, then the xyz coordinates are printed.

Value

Returns a vector of the voxel values at the requested coordinates.

See Also

get.xyz.from.index

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 4
patient <- toy.load.patient (modality = "rtdose", roi.name = "",
```

```
                              dxyz = rep (step, 3), beam.nb = 3)
D <- patient$rtdose[[1]]
get.value.from.xyz (xyz = matrix (c (0, 0, 0, 10, 10, 10),
                    ncol = 3, byrow = TRUE), vol = D)

# isodose
Dmax <- max (D$vol3D.data, na.rm = TRUE)
idx <- which (D$vol3D.data >= (Dmax -1) & D$vol3D.data <= (Dmax - 0.3))
pt <- get.xyz.from.index (idx, D)
get.value.from.xyz (pt, vol = D, interpolate = FALSE, verbose = TRUE)
```

`get.volume.from.bin` *Volume selected by binary volume*

Description

The `get.volume.from.bin` function calculates the volume in cm^3 of the selection specified by a "volume" class object of `"binary"` modality.

Usage

```
get.volume.from.bin(bin)
```

Arguments

`bin`	"volume" class object, of "binary" modality.

Value

Returns the volume of the binary selection, in cm^3.

See Also

get.volume.from.roi

Examples

```
# loading of toy-patient objects
step <- 4
patient <- toy.load.patient (modality = c ("ct", "rtstruct"), roi.name = "brain",
                              dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

# creation of a binary object
bin.brain <- bin.from.roi (vol = CT, struct = S, roi.sname = "bra")
# Volume calculation
get.volume.from.bin (bin.brain)
```

`get.volume.from.roi` *Volume of a region of interest (RoI)*

Description

The `get.volume.from.roi` function extracts the volume incm^3 of one or more RoI, from the `$roi.info` of the "struct" class object.

Usage

```
get.volume.from.roi(struct, roi.name = NULL, roi.sname = NULL, roi.idx = NULL)
```

Arguments

`struct`	"struct" class object.
`roi.name`	Vector of exact names of the RoI in the `struct` object. By default `roi.name = NULL`. See Details.
`roi.sname`	Vector of names or parts of names of the RoI in the `struct` object. By default `roi.sname = NULL`. See Details.
`roi.idx`	Vector of indices of the RoI that belong to the `struct` object. By default `roi.idx = NULL`. See Details.

Details

If `roi.name`, `roi.sname`, and `roi.idx` are all set to NULL, all RoI are selected.

Value

Returns a vector of the volumes in cm^3 of the requested RoI.

See Also

get.volume.from.bin, select.names

Examples

```
# loading of toy-patient objects
step <- 4
patient <- toy.load.patient (modality = c ("rtstruct"),
                             dxyz = rep (step, 3))
S <- patient$rtstruct[[1]]

# Volume extraction
vol <- get.volume.from.roi (S, roi.sname = "bra", roi.idx = c (1, 3))
names (vol)
vol
```

get.xyz.from.index *Conversion of the indices of a point, into xyz coordinate vector in the patient's frame of reference*

Description

The get.xyz.from.index function converts the indices of a voxel of vol$vol3D.data (for example, obtained with the function which) into a vector or matrix of x, y, z coordinates in the patient's frame of reference.

Usage

```
get.xyz.from.index(idx, vol)
```

Arguments

idx	Index, or matrix of voxel indices in the array vol$vol3D.data. The first index of the array is 1.
vol	"volume" class object.

Value

Returns a column-matrix of coordinates in the patient's reference frame, corresponding to the indices idx.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for better
# result)
step <- 4
patient <- toy.load.patient (modality = "rtdose", roi.name = "",
                             dxyz = rep (step, 3), beam.nb = 3)
D <- patient$rtdose[[1]]

# voxels location where the dose is greater than 99.9% Dmax
Dmax <- max (D$vol3D.data, na.rm = TRUE) # D$max.pixel
get.xyz.from.index (which (D$vol3D.data >= 0.99 * Dmax), D)
# or
get.xyz.from.index (which (D$vol3D.data >= 0.99 * Dmax, arr.ind = TRUE), D)
```

grid.equal *Comparison of the grids of two volume objects*

Description

The grid.equal function checks that two volumes share the same grid, i.e. the same frame of reference, the same origin point, and the same dx, dy, dz steps.

Usage

```
grid.equal(vol1, vol2)
```

Arguments

vol1, vol2	"volume" class objects

Value

Returns TRUE if the 2 volumes share the same grid.

Examples

```
# loading of toy-patient objects
patient <- toy.load.patient (modality = c ("ct","mr","rtdose"), roi.name = "",
                             dxyz = c (4, 4, 4), beam.nb = 1)

# Comparison of the grids
grid.equal (patient$rtdose[[1]], patient$ct[[1]])
grid.equal (patient$mr[[1]], patient$ct[[1]])
```

---

histo.2D *2D histograms of 2 volumes*

Description

The histo.2D function creates a "histo2D" class object, containing the two-dimensional array of histograms of two "volume" class objects that have the same grid.

Usage

```
histo.2D(
  x.vol,
  y.vol,
  x.breaks = NULL,
  y.breaks = NULL,
  include.outer = TRUE,
  alias = "",
  description = ""
)
```

Arguments

x.vol, y.vol	"volume" class objects. The 2 volumes must have the grid (i.e. share the same voxels location).
x.breaks, y.breaks	Vectors giving the breakpoints of x and y axes. See Details.
include.outer	Boolean. If include.outer = TRUE, the values out the x.breaks and y.breaks of each volume are counted in the first and the last cell of the histograms. They are not taken into account otherwise.
alias	Character string, $alias of the created object
description	Character string, describing the created object.

Details

The arguments x.breaks and y.breaks represent the scales of the x and y axes of 2D-histogram graph. If they are NULL, the histo.2D function defaults to 256 cells between the values vol$min.pixel and vol$max.pixel.

Value

Returns a "histo2D" class object. This is a list including:

- $patient: set to x.vol$patient.
- $patient.name: set to x.vol$patient.name.
- $patient.bd: set to x.vol$patient.bd.
- $patient.sex: set to x.vol$patient.sex.
- $file.basename: set to "".
- $file.dirname: set to "".
- $object.name: set to "".
- $object.alias: alias of the histo2D object.
- $frame.of.reference: set to x.vol$frame.of.reference.
- $ref.pseudo : set to x.vol$ref.pseudo.
- $modality : set to "histo2D".
- $description: description of the histo2D object.
- $creation.date: set to Sys.Date.
- $nb.pixels: number of elements in the density.map.
- $x.file.src: set to x.vol$object.alias
- $y.file.src: set to y.vol$object.alias
- x.breaks: vector of x-axis breakpoints.
- y.breaks: vector of y-axis breakpoints.
- x.mids: vector of x-axis cell centers.
- y.mids: vector of y-axis cell centers.
- density.map: array of densities.
- total.counts: number of counted voxels.

See Also

display.2D.histo.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "mr", "rtstruct"),
                             roi.name =  "brain",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
MR <- patient$mr[[1]]
S <- patient$rtstruct[[1]]
T.MAT <- patient$T.MAT
```

```
# restriction of the volume around the RoI
CT.on.roi <- nesting.roi (CT, S, roi.name = "brain", vol.restrict = TRUE,
                          xyz.margin = c (1, 1, 1), alias = CT$description)
MR.on.CT <- vol.regrid (vol = MR, back.vol = CT.on.roi, interpolate = TRUE,
                        T.MAT = T.MAT, alias = CT$description,
                        description = NULL)
# selection of voxels included in the RoI.
roi.bin <- bin.from.roi (vol = CT.on.roi, struct = S, roi.sname = "brain")
MR.select <- vol.from.bin (MR.on.CT, roi.bin, alias = MR$description)
CT.select <- vol.from.bin (CT.on.roi, roi.bin, alias = CT$description)
# 2D histogram
H2D <- histo.2D (MR.select, CT.select, x.breaks = seq (50, 400, 10),
  y.breaks = seq (50, 400, 10), alias = "H2D MR CT")
str (H2D)
```

`histo.DVH` *Cumulative Dose Volume Histogram*

Description

The `histo.DVH` function calculates, for each dose, the volume receiving at least this dose.

Usage

```
histo.DVH(histo, alias = "", description = histo$description)
```

Arguments

`histo`	"histo" class object.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object. If the `description = NULL` (default value), it will be set to `histo$description`.

Value

Returns a "dvh" class object. This is a list including:

- `$patient`: set to `histo$patient`.
- `$patient.name`: set to `histo$patient.name`.
- `$patient.bd`: set to `histo$patient.bd`.
- `$patient.sex`: set to `histo$patient.sex`.
- `$file.basename`: set to "".
- `$file.dirname`: set to "".
- `$object.name`: set to "".
- `$object.alias`: alias of the dvh object..
- `$frame.of.reference`: set to `histo$frame.of.reference`.
- `$ref.pseudo` : set to `histo$ref.pseudo`.
- `$modality` : set to "dvh".

- $description: description of the dvh object. By default, set to histo$description.
- $creation.date: set to Sys.Date.
- $nb.MC: set to histo$nb.MC.
- $breaks: vector breakpoints.
- $mids: vector of cell centers.
- $mids.unit: Character string, representing the unit of the abcissa of the DVH. For instance, "Gy", when vol is a rtdose.
- $vol: cumulative volume receiving at least the doses defined by $mids.
- $pcv: percentage of the total volume receiving at least the doses defined by $mids.
- $MC.vol: cumulative volume associated with histo$MC.dV_dx, if it exists.
- $MC.pcv: percentage of the total volume associated with histo$MC.dV_dx, if it exists.
- $MC.dxyz: set to histo$MC.dxyz, if it exists.

See Also

histo.from.roi, histo.from.bin, histo.vol, display.DVH, display.DVH.pc

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"),
                             roi.name = "gizzard", dxyz = rep (step, 3),
                             beam.nb = 3)

# Calculation of the histogram
H <- histo.from.roi (patient$rtdose[[1]], patient$rtstruct[[1]],
                     roi.name = "gizzard",
                     breaks = seq (0, 60, by = 1))

# DVH
DVH <- histo.DVH (H)
str (DVH)
```

---

histo.from.bin *Histogram according to a binary*

Description

The histo.from.bin function computes the voxels histogram of the selection defined by the binary object sel.bin of a "volume" class object.

Usage

```
histo.from.bin(vol, sel.bin, breaks = NULL, alias = "", description = NULL)
```

Arguments

vol	"volume" class object
sel.bin	"volume" class object, of binary modality
breaks	Vector giving the breakpoints between histogram cells. If breaks = NULL, the chosen breakpoints are those used by the hist function by default. If breaks are specified, outside values of vol$vol3D.data are not taken into account.
alias	Character string, $alias of the created object
description	Character string, describing the the created object. If the description = NULL (default value),it will be set to vol$description

Value

Returns a "histo" class object. See histo.vol.

See Also

histo.from.roi, histo.vol, display.histo, display.dV_dx

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 5
patient <- toy.load.patient (modality = c("ct","rtstruct"), roi.name = "",
                             dxyz = rep (step, 3))
bin.patient <- bin.from.roi (patient$ct[[1]], struct = patient$rtstruct[[1]],
                             roi.name = "patient")
# ct histogram in patient  volume
H <- histo.from.bin (patient$ct[[1]], sel.bin = bin.patient, breaks = NULL,
                     alias = "patient_hist")
str(H)

## Not run:
# Skin dose histogram
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"), roi.name = "",
                             dxyz = c (2, 2, 2), beam.nb = 3)
D <- patient$rtdose[[1]]
S <- patient$rtstruct[[1]]

# Creation of the skin contour of 3 mm
bin.patient <- bin.from.roi (D, struct = S, roi.name = "patient",
                             alias = "patient")
inverse.patient <- bin.inversion (bin.patient, alias = "inv (patient)")
expansion <- bin.dilation (inverse.patient, radius = 3,
                           alias = "inv (patient) + 3")
contour.3mm <- bin.intersection (bin.patient, expansion,
                                 alias = "contour 3 mm")

# Dose histogram in this volume
H <- histo.from.bin (D, sel.bin = contour.3mm, breaks = NULL,
                     alias = "Skin dose")
str(H)

## End(Not run)
```

`histo.from.roi` *Histogram according to a RoI*

Description

The `histo.from.roi` function calculates the histogram of the volume voxels belonging to a RoI.

Usage

```
histo.from.roi(
  vol,
  struct,
  roi.name = NULL,
  roi.sname = NULL,
  roi.idx = NULL,
  T.MAT = NULL,
  breaks = NULL,
  MC = NULL,
  sd = c(1, 1, 1),
  offset = c(0, 0, 0),
  over.sampling.factor = 1,
  alias = "",
  description = NULL
)
```

Arguments

`vol`	"volume" class object
`struct`	"struct" class object.
`roi.name`	Exact name of a RoI in `struct` object. By default `roi.name = NULL`. See Details.
`roi.sname`	Name or part of name of a RoI in `struct` object. By default `roi.sname = NULL`. See Details.
`roi.idx`	Value of the index of a RoI that belong to the `struct` object. By default `roi.idx = NULL`. See Details.
`T.MAT`	"t.mat" class object, created by load.patient.from.Rdcm, load.patient.from.dicom or load.T.MAT. If `T.MAT = NULL`, `struct$ref.pseudo` must be equal to `vol$ref.pseudo`.
`breaks`	Vector giving the breakpoints between histogram cells. If `breaks = NULL`, the chosen breakpoints are those used by the hist function by default. If `breaks` are specified, outside values of `vol$vol3D.data` are not taken into account.
`MC`	If different from `NULL` (default value), number of calculations that will be performed, by Monte-Carlo, by randomly moving the chosen RoI over a random distance, generated according to a normal distribution with mean translation defined by `offset` and standard deviation `sd`.
`sd`	Vector representing the standard deviation of distances in the 3 directions x, y and z.
`offset`	Vector representing the translation of the RoI in the 3 directions x, y and z.
`over.sampling.factor`	Strictly positive integer, or a vector of 3 strictly positive integers, default to 1. Defined to oversample grids of `vol`. Oversampling can be very time consuming.

alias	Character string, $alias of the created object
description	Character string, describing the the created object. If the description = NULL (default value), it will be set to struct$roi.info$roi.pseudo[roi.idx]

Details

roi.name, roi.sname, and roi.idx must select only one RoI.

Value

Returns "histo" class object. This is a list including:

- $alias: alias of the histo object.
- $description: description of the histo object.
- $breaks: vector breakpoints
- $mids: vector of cell centers.
- $mids.unit: Character string, representing the unit of the abcissa of the histogram. For instance, "Gy", when vol is a rtdose.
- counts: count of voxels whose value is included in the limits defined by $breaks.
- dV_dx: differential histogram, expressed in cm3 by voxel units, at each $mids.
- MC.counts: array of MC rows. Each row i represents the histogram of the voxels contained in the RoI, whose points have been shifted by $MC.dxyz[i,].
- MC.dV_dx: array of MC rows. Each row i represents the differential histogram of the voxels contained in the RoI, the points of which have been shifted by $MC.dxyz[i,].
- MC.dxyz: array of MC rows, representing the offset applied to the RoI.

Note

Using Monte-Carlo can be time consuming for large RoI.

If you only want the result just for a translation, use the arguments MC = 1, sd = 0 and offset = desired translation vector.

See Also

histo.vol, histo.from.bin, display.histo, display.dV_dx

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for better
# result)
step <- 5
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"),
                             roi.name = "gizzard", dxyz = rep (step, 3),
                             beam.nb = 3)

# Calculation of the histogram
H <- histo.from.roi (patient$rtdose[[1]], patient$rtstruct[[1]],
                     roi.name = "gizzard",
                     breaks = seq (0, 60, by = 1))
str (H)
```

`histo.vol` *Histogram of a volume*

Description

The `histo.vol` function calculates the voxel values histogram of "volume" class object.

Usage

```
histo.vol(vol, breaks = NULL, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object.
`breaks`	Vector giving the breakpoints between histogram cells. If `breaks` = `NULL`, the chosen breakpoints are those used by the hist function by default. If `breaks` are specified, outside values of `vol$vol3D.data` are not taken into account.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object. If the `description` = `NULL` (default value), it will be set to `vol$description`.

Value

Returns a "histo" class object. This is a list including:

- `$patient`: set to `vol$patient`.
- `$patient.name`: set to `vol$patient.name`.
- `$patient.bd`: set to `vol$patient.bd`.
- `$patient.sex`: set to `vol$patient.sex`.
- `$file.basename`: set to "".
- `$file.dirname`: set to "".
- `$object.name`: set to "".
- `$object.alias`: alias of the histo object.
- `$frame.of.reference`: set to `vol$frame.of.reference`.
- `$ref.pseudo` : set to `vol$ref.pseudo`.
- `$modality` : set to `"histo"`.
- `$description`: description of the histo object.
- `$creation.date`: set to `Sys.Date`.
- `$nb.MC`: set to 0.
- `$breaks`: vector breakpoints
- `$mids`: vector of cell centers.
- `$mids.unit`: Character string, representing the unit of the abcissa of the histogram. For instance, "Gy", when `vol` is a rtdose.
- `counts`: count of voxels whose value is included in the limits defined by `$breaks`.
- `dV_dx`: differential histogram, expressed in cm^3 by voxel units, at each `$mids`.

See Also

histo.from.roi, histo.from.bin, display.histo, display.dV_dx

Examples

```
# loading of toy-patient objects
step <- 3
patient <- toy.load.patient (modality = "ct", dxyz = rep (step, 3))
CT <- patient$ct[[1]]

# histogram and display
H <- histo.vol (CT, breaks = seq (3, ceiling (CT$max.pixel), 1),
                alias = "CT_hist")
str (H)
```

---

load.obj.data *Load data of an* **espadon** *class object*

Description

The load.obj.data function loads all the data of an **espadon** object of class 'struct' or 'volume'.

Usage

```
load.obj.data(obj, tag.dictionary = dicom.tag.dictionary())
```

Arguments

obj	struct or "volume" class object
tag.dictionary	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files in case obj was extracted from DICOM files.

Value

Returns the **espadon** object with data $vol3D.data or $roi.data

See Also

load.obj.from.dicom and load.obj.from.Rdcm

Examples

```
# First, save toy patient objects to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dir.create (pat.dir, recursive = TRUE)
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = c (4, 4, 4))
save.to.Rdcm (patient$ct[[1]], dirname = pat.dir)
rm( patient)


patient <- load.patient.from.Rdcm (pat.dir, data = FALSE)
```

```
CT <- load.obj.data (patient$ct[[1]])
str (CT, max.level = 2)
```

`load.obj.from.dicom` *Loading an* **espadon** *object from DICOM files or folder*

Description

Loading an **espadon** object from DICOM files or folder.

Usage

```
load.obj.from.dicom(
  dcm.files,
  data = TRUE,
  ref.pseudo = "ref1",
  tag.dictionary = dicom.tag.dictionary(),
  verbose = TRUE
)
```

Arguments

`dcm.files`	String vector, representing the list of the full names of the DICOM files of the same DICOM object, or its directory.
`data`	Boolean. Only valid for objects usable by the **espadon** package, namely ct, mr, rtdose, rtstruct, pt... If `data = TRUE`, either the values of the voxels when modality is (ct, mr, rtdose, pt), or the coordinates of the RoI when modality is rtstruct, are loaded into memory.
`ref.pseudo`	String, `$ref.pseudo` (i.e. pseudonym of the frame of reference) to assign to the loaded object.
`tag.dictionary`	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.
`verbose`	Boolean. If `TRUE`, a progress bar indicates the progress of the conversion.

Value

Returns an **espadon** object of class "dvh","histo","histo2D","mesh", "rtplan","struct", "undef" or "volume" depending on the object modality. See espadon.class for class definitions.

See Also

load.obj.data and load.obj.from.Rdcm

Examples

```
# First, save toy.dicom.raw () raw data to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "PM_dcm")
dir.create (pat.dir, recursive = TRUE)
dcm.filename <- tempfile (pattern = "toyrtplan", tmpdir = pat.dir,
                          fileext = ".dcm")
zz <- file (dcm.filename, "wb")
```

```
writeBin (toy.dicom.raw (), zz, size = 1)
close (zz)

# loading of rt-plan object
RTplan <- load.obj.from.dicom (dcm.filename)
str (RTplan)
# Cleaning  temporary directory
unlink (pat.dir, recursive = TRUE)
```

`load.obj.from.Rdcm` *Loading an* **espadon** *object from *.Rdcm file*

Description

The `load.obj.from.Rdcm` function loads a DICOM object into memory, creating a list containing the information necessary for its subsequent use with the **espadon** package.

Usage

```
load.obj.from.Rdcm(
  Rdcm.filename,
  data = TRUE,
  nb = NULL,
  upgrade.to.latest.version = FALSE
)
```

Arguments

`Rdcm.filename`	Character string, representing the full name of a *.Rdcm file created by dicom.to.Rdcm.converter.
`data`	Boolean. Only works for objects usable by the **espadon** package, namely ct, mr, rtdose, rtstruct, pt... If `data = TRUE`, either the values of the voxels when modality is (ct, mr, rtdose), or the coordinates of the RoI when modality is rtstruct, are loaded into memory.
`nb`	Vector of integers, active only if `data = TRUE`, and only operating on rtstruct. If `nb = NULL`, all the RoI of rtstruct are loaded into memory. Otherwise only data of the RoI indices defined by the vector `nb` are loaded.
`upgrade.to.latest.version`	Boolean. If `TRUE`, the function attempts to upgrade to the latest version, parsing the DICOM data. It may take longer to load the data. Consider using the Rdcm.upgrade function.

Value

Returns an **espadon** object of class "dvh","histo","histo2D","mesh", "rtplan","struct", "undef" or "volume" depending on the object modality. See espadon.class for class definitions.

See Also

load.obj.data and load.obj.from.dicom

Examples

```
# First, save toy patient objects to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dir.create (pat.dir, recursive = TRUE)
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = c (4, 4, 4))
save.to.Rdcm (patient$ct[[1]], dirname = pat.dir)
save.to.Rdcm (patient$mr[[1]], dirname = pat.dir)
save.T.MAT (patient$T.MAT, dirname = pat.dir)
# Rdcm files in pat.dir
list.files(pat.dir)

CT <- load.obj.from.Rdcm (file.path (pat.dir,
                                     list.files(pat.dir, pattern="ct1[.]Rdcm")[1]),
                          data=TRUE)
MR <- load.obj.from.Rdcm (file.path (pat.dir,
                                     list.files(pat.dir, pattern="mr1[.]Rdcm")[1]),
                          data=TRUE)
Reg <-load.obj.from.Rdcm (file.path (pat.dir,"ref1_from_ref2.Rdcm"), data=TRUE)
str(Reg)

# Cleaning  temporary directory
unlink (pat.dir, recursive = TRUE)
```

`load.patient.from.dicom`

Loading patient data from DICOM files

Description

The `load.patient.from.dicom` function is used to load or pre-load in memory all patient objects from DICOM files.

Usage

```
load.patient.from.dicom(
  dcm.files,
  data = FALSE,
  dvh = FALSE,
  ignore.duplicates = FALSE,
  tag.dictionary = dicom.tag.dictionary(),
  verbose = TRUE
)
```

Arguments

`dcm.files`	String vector, representing the list of the full names of the DICOM files of the same patient, or its directories.
`data`	Boolean. If `data = TRUE`, the voxels value of the "volume" class objects, or the coordinates of the RoI (region of interest) of the `struct` class objects, are loaded into memory.

dvh	Boolean. if dvh = TRUE and if they exist, patient DVH are loaded, for convenience. They are not used as is in **espadon** package.
ignore.duplicates	Boolean. If TRUE, the function ignores duplicated objects.
tag.dictionary	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.
verbose	Boolean. If TRUE, a progress bar indicates the progress of the conversion.

Value

Returns an **espadon** object of class "patient", describing the information from dcm.files. See espadon.class for a description of the "patient" class.

See Also

dicom.to.Rdcm.converter, load.patient.from.Rdcm, load.obj.data, load.obj.from.dicom, load.obj.from.Rdcm and load.T.MAT.

Examples

```
# First, save toy.dicom.raw () raw data to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "toy_dccm")
dir.create (pat.dir, recursive = TRUE)
dcm.filename <- tempfile (pattern = "toyrtplan", tmpdir = pat.dir,
                          fileext = ".dcm")
zz <- file (dcm.filename, "wb")
writeBin (toy.dicom.raw (), zz, size = 1)
close (zz)

# loading patient. Here the toy patient ha only a unique rt-plan object
patient <- load.patient.from.dicom (pat.dir, data = FALSE)
str (patient, max = 2)
# description of object
patient$description
# transfer matrices :
patient$T.MAT
# rt-plan object
str (patient$rtplan[[1]])
# Cleaning  temporary directory
unlink (pat.dir, recursive = TRUE)
```

---

load.patient.from.Rdcm

*Loading patient data from *.Rdcm files*

Description

The load.patient.from.Rdcm function is used to load or pre-load in memory all patient objects converted in *.Rdcm files.

Usage

```
load.patient.from.Rdcm(
  dirname,
  data = FALSE,
  dvh = FALSE,
  upgrade.to.latest.version = FALSE,
  ignore.duplicates = FALSE
)
```

Arguments

dirname	Full paths of the directories of a single patient, or vector of full.path of Rdcm.files.
data	Boolean. If data = TRUE, the voxels value of the "volume" class objects, or the coordinates of the RoI (region of interest) of the struct class objects, are loaded into memory.
dvh	Boolean. if dvh = TRUE and if they exist, patient DVH are loaded, for convenience. They are not used as is in **espadon** package.
upgrade.to.latest.version	Boolean. If TRUE, the function attempts to upgrade to the latest version, parsing the DICOM data. It may take longer to load the data. Consider using theRdcm.upgrade function.
ignore.duplicates	Boolean. If TRUE, the function ignores duplicated objects.

Value

Returns an **espadon** object of class "patient", describing the information contained in dirname. See espadon.class for a description of the "patient" class.

See Also

dicom.to.Rdcm.converter, load.patient.from.dicom, load.obj.data, load.obj.from.dicom, load.obj.from.Rdcm and load.T.MAT.

Examples

```
# First, save toy patient objects to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dir.create (pat.dir, recursive = TRUE)
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = c (4, 4, 4))
save.to.Rdcm (patient$ct[[1]], dirname = pat.dir)
save.to.Rdcm (patient$mr[[1]], dirname = pat.dir)
save.T.MAT (patient$T.MAT, dirname = pat.dir)
# Rdcm files in pat.dir
list.files(pat.dir)

# loading patient from Rdcm files with data:
new.patient <- load.patient.from.Rdcm (pat.dir, data = TRUE)
str (new.patient, max.level = 2 )

# Cleaning  temporary directory
unlink (pat.dir, recursive = TRUE)
```

load.Rdcm.raw.data *Loading a *.Rdcm file*

Description

the load.Rdcm.raw.data function loads the content of a *.Rdcm file.

Usage

```
load.Rdcm.raw.data(
  Rdcm.filename,
  address = TRUE,
  data = TRUE,
  upgrade.to.latest.version = FALSE
)
```

Arguments

Rdcm.filename	Character string, representing the full name of a *.Rdcm file created by dicom.to.Rdcm.converter.
address	Boolean. If TRUE, a dataframe with the address of the tags in the raw DICOM data is returned.
data	Boolean. If TRUE, the DICOM information are returned as an R list.
upgrade.to.latest.version	Boolean. If TRUE, the function attempts to upgrade to the latest version, parsing the DICOM data. It may take longer to load the data. Consider using the Rdcm.upgrade function.

Value

Returns a list containing the information, converted by **espadon**, of a DICOM object..

See Also

dicom.to.Rdcm.converter, load.obj.from.Rdcm.

Examples

```
# For testing, save first toy.dicom.raw () raw data to a temporary file, and
# convert it in Rdcm fie
pat.src.dir <- file.path (tempdir(), "PM_dcm")
dir.create (pat.src.dir, recursive = TRUE)
dcm.filename <- tempfile (pattern = "PM_rtplan", tmpdir = pat.src.dir,
                          fileext = ".dcm")
zz <- file (dcm.filename, "wb")
writeBin (toy.dicom.raw (), zz, size = 1)
close (zz)
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dicom.to.Rdcm.converter (pat.src.dir, pat.dir, update = TRUE)
lf <- list.files (pat.dir, pattern = "[.]Rdcm$", full.names = TRUE)
lf
```

```
# Inspect Rdcm raw data
L <- load.Rdcm.raw.data (lf[1])
str (L, max.level =3)
```

`load.T.MAT` *Loading of information about transfer matrices between frames of reference of patient Rdcm objects.*

Description

The `load.T.MAT` function lists all the frames of reference of the objects included in the patient directory. It concatenates all the information of the reg matrices of a directory, creating, among other things, a list of 4x4 transfer matrices between frames of reference.

Usage

```
load.T.MAT(dirname, upgrade.to.latest.version = FALSE)
```

Arguments

`dirname`	Full paths of the directories of a single patient, or vector of full.path of Rdcm.files.
`upgrade.to.latest.version`	Boolean. If `TRUE`, the function attempts to upgrade to the latest version, parsing the DICOM data. It may take longer to load the data. Consider using the Rdcm.upgrade function.

Value

Returns a "t.mat" class object. It is a list that includes :

- `$ref.info`: dataframe giving the correspondence between the frame of reference (column `$ref`) of the DICOM object (TAG (0020,0052)) and its pseudonym (column `$ref_pseudo`).
- `$reg.info`:list of dataframes : the first one gives the PID, birthday, and sex of the patient, the second one gives the name of the source file of transfer matrices.
- `$matrix.description`: dataframe giving the transfer matrix names (column `$t`), its source frame of reference (column `$src`), the destination frame of reference (column `$dest`), and its type (`$type`). Note: only the RIGID type is supported.
- `$matrix.list`: list of 4X4 transfer matrices. This list contains at least as many Identity matrices as there are `ref.pseudo`.

Examples

```
# First, save toy patient objects to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dir.create (pat.dir, recursive = TRUE)
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = c (4, 4, 4))
save.to.Rdcm (patient$ct[[1]], dirname = pat.dir)
save.to.Rdcm (patient$mr[[1]], dirname = pat.dir)
save.T.MAT (patient$T.MAT, dirname = pat.dir)
# Rdcm files in pat.dir
list.files(pat.dir)
```

```
T.MAT <- load.T.MAT (pat.dir)
T.MAT

# Cleaning  temporary directory
unlink (pat.dir, recursive = TRUE)
```

`mesh.from.bin` *Creation of a mesh according to a binary volume*

Description

The `mesh.from.bin` function creates a `mesh` class object from a volume object of "binary" modality.

Usage

```
mesh.from.bin(
  bin,
  alias = "",
  tol = 1,
  smooth.iteration = 10,
  smooth.type = c("taubin", "laplace", "HClaplace", "fujiLaplace", "angWeight",
    "surfPreserveLaplace"),
  smooth.lambda = 0.5,
  smooth.mu = -0.53,
  smooth.delta = 0.1,
  verbose = FALSE
)
```

Arguments

`bin`	"volume" class object of "binary" modality.
`alias`	Character string, `$alias` of the mesh defining the `$alias` of the created object.
`tol`	Tolerance in mm, applied for mesh simplification. See vcgClean.
`smooth.iteration`	Number of iterations applied in the smoothing algorithm. See vcgSmooth.
`smooth.type`	character: select smoothing algorithm. Available are "taubin", "laplace", "HClaplace", "fujiLaplace", "angWeight" (and any sensible abbreviations). By default, set to "taubin". See vcgSmooth.
`smooth.lambda`	numeric: parameter for Taubin smooth. See vcgSmooth.
`smooth.mu`	numeric: parameter for Taubin smooth. See vcgSmooth.
`smooth.delta`	numeric: parameter for Scale dependent laplacian smoothing (see reference below).and maximum allowed angle (in radians) for deviation between normals Laplacian (surface preserving). See vcgSmooth.
`verbose`	Boolean, by default set to `FALSE`. Allows you to inhibit comments.

Value

Returns a "mesh" class object. This is a list including the following 6 elements:

- `$patient`: set to `bin$patient`.
- `$patient.bd`: set to `bin$patient.bd`.
- `$patient.name`: set to `bin$patient.name`.
- `$patient.sex`: set to `bin$patient.sex`.
- `$file.basename`: set to "".
- `$file.dirname`: set to "".
- `$object.name`: set to "".
- `$object.alias`: set to the `alias` argument of the function.
- `$frame.of.reference`: set to `bin$frame.of.reference`.
- `$ref.pseudo` : set to `bin$ref.pseudo`.
- `$modality` : set to `"mesh"`.
- `$description`: By default, set to `paste (bin$object.alias, "mesh")`.
- `$creation.date`: set to `Sys.Date`.
- `$nb.faces`: set to the number of faces of the mesh.
- `$mesh`: list of 3 elements defining the mesh:

- `$vb`: array made up of the generalized coordinates (x, y, z, 1) of the vertices of the triangles.
There are as many columns as there are vertices.
- `$it`: array of the 3 indices of the vertices forming a triangle, arranged by column.
There are as many columns as there are triangles in the mesh.
- `$normals`: array made up of the generalized coordinates (x, y, z, 1) of the normal vectors of each triangle.
There are as many columns as there are vertices.

Note

To compute the mesh, all NA voxels of the binary volume `bin` are set to FALSE. If all voxels are equal to FALSE, the function returns the code NULL.

See Also

vcgSmooth

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct"), roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

# creation of the patient mesh
bin <- bin.from.roi (CT, struct = S, roi.name = "patient")
mesh.patient <- mesh.from.bin (bin, alias = "patient", verbose = FALSE)
str (mesh.patient)
```

mesh.in.new.ref *Change of frame of reference of a mesh*

Description

The mesh.in.new.ref function allows you to change the frame of reference of a mesh.

Usage

```
mesh.in.new.ref(
  mesh,
  new.ref.pseudo,
  T.MAT = NULL,
  alias = "",
  description = NULL
)
```

Arguments

mesh	"volume" class object.
new.ref.pseudo	pseudonym of the frame of reference in which the mesh should be located. This new.ref.pseudo must exist in the T.MAT list.
T.MAT	"t.mat" class object, created by load.patient.from.Rdcm, load.patient.from.dicom, load.T.MAT or ref.add.
alias	Character string, $alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be that of the mesh.

Value

Returns "mesh" class object in the new frame of reference new.ref.pseudo.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct", "mr"),
                             roi.name = "", dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

#creation of the patient mesh
bin <- bin.from.roi (CT, struct = S, roi.name = "patient")
mesh.patient <- mesh.from.bin (bin, alias = "patient", verbose = FALSE)

# mesh in the MR frame of reference
new.mesh <- mesh.in.new.ref (mesh.patient, patient$mr[[1]]$ref.pseudo,
                             T.MAT = patient$T.MAT)

str (new.mesh, max.level = 2)
```

mesh.repair *Repair of a mesh*

Description

The mesh.repair function repairs holes in a mesh class object.

Usage

```
mesh.repair(mesh, verbose = TRUE)
```

Arguments

mesh	"mesh" class object.
verbose	Boolean, by default set to FALSE. Allows you to inhibit comments.

Value

Returns a mesh, repaired by removing degenerated triangles and filling holes.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct"),
                             roi.name = "gizzard", dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

# creation of the gizzard mesh
bin <- bin.from.roi (CT, struct = S, roi.name = "gizzard")
mesh.gizzard <- mesh.from.bin (bin, alias = "gizzard", verbose = FALSE)

repair.mesh.gizzard <- mesh.repair (mesh.gizzard, verbose = FALSE)
str (mesh.gizzard)
str (repair.mesh.gizzard)
```

mesh.spheric.proj *Adding spherical coordinates to a mesh*

Description

The mesh.spheric.proj function adds latitude and longitude coordinates to a mesh. These features map the mesh surface to a sphere. Latitude and longitude are computed using the heat diffusion approach explained by *Brechbühler and al* **[1]**.

Usage

```
mesh.spheric.proj(mesh, verbose = TRUE)
```

Arguments

mesh	"mesh" class object.
verbose	Boolean, by default set to FALSE. Allows you to inhibit comments.

Value

returns a "mesh" class object in which $mesh contains Lat and lon evaluated at vertices. The function allows to have a parameterized surface for later computations as curvature or shape index, hence, nor the surface, nor the angles are preserved. In the DICOM frame of reference, latitude goes along Z axis (from feet = -1 to head = +1) and longitude turns counter clockwise (from -1 to +1).

Note

This funtion is time consuming.

References

[1] Brechbuhler C, Gerig G, Kubler O (1995). "Parametrization of Closed Surfaces for 3-D Shape Description." *Computer Vision and Image Understanding*, **61**(2), 154-170. ISSN 1077-3142, doi:10.1006/cviu.1995.1013.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 5
patient <- toy.load.patient (modality = c("ct", "rtstruct"), roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

#creation of the patient mesh
bin <- bin.from.roi (CT, struct = S, roi.name = "patient")
m.patient <- mesh.from.bin (bin)
m.skin <- mesh.repair (m.patient, verbose = FALSE)

m.proj <- mesh.spheric.proj (m.skin, verbose = FALSE)

library (rgl)
col <- hcl.colors (12, "Blue-Red 3")
open3d()
shade3d (m.proj$mesh, meshColors = "vertices",
         color = col[round ((m.proj$mesh$Lat/2 + 0.5) * 11) + 1],
         specular = "#404040")
open3d()
shade3d (m.proj$mesh, meshColors = "vertices",
         color = col[round ((m.proj$mesh$Lon/2 + 0.5) * 11) + 1],
         specular = "#404040")
```

nesting.bin *Restrict volume to a binary selection*

Description

The nesting.bin function restricts a "volume" class object to the rectangular parallelepiped circumscribed to the selected voxels.

Usage

```
nesting.bin(
  vol,
  sel.bin,
  alias = "",
  description = NULL,
  xyz.margin = c(0, 0, 0),
  vol.restrict = FALSE
)
```

Arguments

vol	"volume" class object, containing data to restrict.
sel.bin	"volume" class object, of "binary" modality, specifying the selected voxels.
alias	Character string, $alias of the created object.
description	Character string, describing the the created object. If description = NULL, it will be paste (vol$description,"restricted to", sel.bin$description).
xyz.margin	Vector of length 3, representing the distances in mm to be added to the x, y and z directions of the rectangular parallelepiped circumscribed to the voxels selected in sel.bin, in the cutting planes frame of reference. By default xyz.margin = c (0, 0, 0).
vol.restrict	Boolean. If vol.restrict = TRUE, the rectangular parallelepiped circumscribed to the selected voxels, enlarged by xyz.margin cannot exceed the initial volume.

Value

Returns a "volume" class object, in which 3D volume is limited to the rectangular parallelepiped circumscribed to the voxels selected by sel.bin, increased by the requested margins.

See Also

add.margin, nesting.cube and nesting.roi.

Examples

```
# loading of toy-patient objects (decrease dxyz for  better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct"),
                             roi.name = "brain", dxyz = rep (step, 3))
CT <- patient$ct[[1]]
b <- bin.from.vol (CT, min = 0, max =200)
```

```
CT.restricted <- nesting.bin (CT, b, xyz.margin =  rep (step, 3))
display.plane (bottom = CT.restricted, top = b, view.type = "sagi",
             bottom.col = pal.RVV (1000),
             bottom.breaks = seq (-1000, 1000, length.out = 1001),
             bg = "#00ff00",  interpolate  = FALSE)
```

`nesting.cube` *Restriction of a volume to a rectangular parallelepiped*

Description

The `nesting.cube` function restricts or increases a volume to the rectangular parallelepiped defined by its 2 extreme vertices.

Usage

```
nesting.cube(obj, pt.min, pt.max, alias = "", description = NULL, ...)
```

Arguments

`obj`	object of class volume or mesh.
`pt.min`	minimum x, y, z coordinates of the rectangular parallelepiped vertex.
`pt.max`	maximum x, y, z coordinates of the rectangular parallelepiped vertex.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object. If the `description = NULL` (default value), it will be set to `obj$description`.
`...`	Additional arguments `vol` (depracated), replaced by `obj`.

Value

Returns a "volume" class object, in which 3D volume is restricted or increased to be circumscribed to the requested rectangular parallelepiped. If the created volume exceeds the initial volume, new voxels are set to `NA`.

See Also

add.margin, nesting.roi and nesting.bin.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 5
patient <- toy.load.patient (modality = "ct", roi.name = "",
                             dxyz = rep (step, 3))
CT <- patient$ct[[1]]
# Calculation of the new CT restricted to the parallelepiped reduced by 10 mm.
pt.CT <- get.extreme.pt (CT) # extreme points of CT
new.pt.CT <- pt.CT + matrix (rep (c (+ 12, -12), 3), ncol = 2, byrow = TRUE)
new.CT <- nesting.cube (CT, new.pt.CT$min, new.pt.CT$max, alias = "new CT")
## Not run:
# check for change
display.3D.stack (CT)
```

```
display.3D.stack (new.CT, line.col="red")

## End(Not run)
```

`nesting.roi` *Restrict volume to RoI*

Description

The `nesting.roi` function restricts a "volume" class object to the rectangular parallelepiped circumscribed to the chosen RoI.

Usage

```
nesting.roi(
  obj,
  struct,
  roi.name = NULL,
  roi.sname = NULL,
  roi.idx = NULL,
  xyz.margin = c(0, 0, 0),
  vol.restrict = FALSE,
  T.MAT = NULL,
  alias = "",
  description = NULL,
  ...
)
```

Arguments

`obj`	object of class volume or mesh.
`struct`	"struct" class object.
`roi.name`	Vector of exact names of the RoI in the `struct` object. By default `roi.name = NULL`. See Details.
`roi.sname`	Names or parts of names of the RoI in the `struct` object. By default `roi.sname = NULL`. See Details.
`roi.idx`	Index of the RoI that belong to the `struct` object. By default `roi.idx = NULL`. See Details.
`xyz.margin`	Vector of length 3, representing the distances in mm to be added to the x, y and z directions of the rectangular parallelepiped circumscribed to the chosen RoI, in the cutting planes frame of reference. By default xyz.margin = c (0, 0, 0).
`vol.restrict`	Boolean. If `vol.restrict = TRUE`, the rectangular parallelepiped circumscribed to the chosen RoI, enlarged by xyz.margin cannot exceed the initial volume.
`T.MAT`	"t.mat" class object, created by load.patient.from.dicom, load.patient.from.Rdcm or load.T.MAT. If `T.MAT = NULL`, `struct$ref.pseudo` must be equal to `obj$ref.pseudo`.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object. If `description = NULL`, it will be that of the `obj`, plus "restricted to" the selected RoI.
`...`	Additional arguments `vol` (depracated), replaced by `obj`.

Details

If `roi.name`, `roi.sname`, and `roi.idx` are all set to NULL, all RoI are selected.

Value

Returns a "volume" class object, in which 3D volume is limited to the rectangular parallelepiped circumscribed to the chosen RoI, increased by the requested margins.

See Also

add.margin, nesting.cube and nesting.bin.

Examples

```
# loading of toy-patient objects (decrease dxyz for  better result)
step <- 4
patient <- toy.load.patient (modality = c("ct", "rtstruct"),
                             roi.name = "brain", dxyz = rep (step, 3))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

CT.brain <- nesting.roi (CT, S, roi.sname = "brain")
CT.brain.with.margin <- nesting.roi (CT, S, roi.sname = "brain",
                                     xyz.margin = c (10,10,10))

# display at the center of gravity of the cerebellum Gz
Gz <- S$roi.info [grep("^brain",S$roi.info$roi.pseudo),]$Gz
display.plane (bottom = CT.brain, view.coord = Gz,
               struct = S, bottom.col = pal.RVV (1000),
               bottom.breaks = seq (-1000, 1000, length.out = 1001),
               bg = "#00ff00",  interpolate  = FALSE, legend.shift = -20)
display.plane (bottom = CT.brain.with.margin,view.coord = Gz,
               struct = S,  bottom.col = pal.RVV (1000),
               bottom.breaks = seq(-1000, 1000, length.out = 1001),
               bg = "#00ff00", interpolate  = FALSE, legend.shift = -20)
```

`orientation.create` *Creation of orientation*

Description

The `orientation.create` function creates the orientation vectors of a plane:

- from 3 points A, B and C (see details),
- or from 2 vectors B and C, resp. defining x and y-axis (see details),
- or from 2 points A, B defining x-axis, and the normal vector to the plane (see details),
- or from a vector B defining x-axis, and the normal vector to the plane (see details).

Usage

```
orientation.create(A = c(0, 0, 0), B = NULL, C = NULL, normal = NULL)
```

Arguments

A	Vector of the x, y and z coordinates of point A, by default equal to c(0,0,0) in the case where B and C are vectors.
B	Vector of x, y and z coordinates of point or vector B.
C	Vector of x, y and z coordinates of point or vector C.
normal	Vector of x, y and z coordinates of normal vector.

Details

When using B and C, B-A define the x-axis unit vector. The unit vector of the y-axis is orthonormal to the x-axis, coplanar with A, B and C, and in the direction of A to C.

When using B and normal, the unit vector of the x-axis is orthonormal to the normal vector, in the direction of A to B. The unit vector of the y-axis is defined so as to constitute a direct orthonormal basis with the unit vector of the x-axis and the normal vector of the plane.

Value

Returns the orientation of the plane. That means the concatenation of 2 vectors, defining an orthonormal basis of the plane.

Examples

```
A <- c (-29.93, 18.85, 4.34)
B <- c (28.73, 15.36, 4.46)
C <- c (1.53, 75.21, 13.51)
orientation.create (A, B, C)
```

---

pal.RVV — *Conversion of Hounsfied Units to Realistic Volume Vizualization colors*

Description

The RVV.pal function produces a color palette where Hounsfield Units in the range -1000 HU to 1000 HU are converted into realistically colorized virtual anatomy (for use with CT), developped by *J.C. Silverstein and al* [1]

Usage

```
pal.RVV(n, alpha = NULL, min.col = "#000000", max.col = "#FFFFFF")
```

Arguments

n	Integer, number of colors to be in the palette
alpha	Vector representing the opacity, in the range of 0 (transparent) to 1 (opaque). If alpha = NULL (default), all colors are opaque, and no alpha channel is added to the colors.
min.col, max.col	respectively the color below -1000HU (by default, black, i.e. "#000000") and above +1000HU (by default, white, i.e. "#FFFFFF")

Value

Returns a vector of colors of size n.

References

[1] Silverstein JC, Parsad NM, Tsirline V (2008). "Automatic perceptual color map generation for realistic volume visualization." *Journal of Biomedical Informatics*, **41**(6), 927-935. ISSN 1532-0464, doi:10.1016/j.jbi.2008.02.008.

Examples

```
pal <- pal.RVV (256)

image (x = seq (-1000, 1000, length.out = 1024), y = 1,
       z = matrix (seq (-1100, 1100, length.out = 1024), ncol = 1),
       col = pal,
       main = "Realistic Volume Vizualization colors")
```

Rdcm.inventory *Inventory of* **espadon** *objects from Rdcm files*

Description

The Rdcm.inventory function creates, from Rdcm files in a patient's directory, a dataframe describing objects.

Usage

```
Rdcm.inventory(dirname, upgrade.to.latest.version = FALSE)
```

Arguments

dirname	Character string, representing the full name of patient directory, including Rdcm files.
upgrade.to.latest.version	Boolean. If TRUE, the function attempts to upgrade to the latest version, parsing the DICOM data. It may take longer to load the data. Consider using the Rdcm.upgrade function.

Value

Returns a dataframe, providing information of DICOM objects.

Examples

```
# First, save toy patient objects to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dir.create (pat.dir, recursive = TRUE)
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = c (4, 4, 4))
save.to.Rdcm (patient$ct[[1]], dirname = pat.dir)
save.to.Rdcm (patient$mr[[1]], dirname = pat.dir)
save.T.MAT (patient$T.MAT, dirname = pat.dir)
```

```
Rdcm.inventory (pat.dir)

# Cleaning  temporary directory
unlink (pat.dir, recursive = TRUE)
```

Rdcm.upgrade *Updating Rdcm files.*

Description

The Rdcm.upgrade function updates Rdcm files that were created with a previous version.

Usage

```
Rdcm.upgrade(Rdcm.files)
```

Arguments

Rdcm.files	String vector, representing the list of the full names of the Rdcm files, or its directories.

Value

Saves the updated Rdcm files. If the Rdcm files were generated from the dicom files, the data is updated from the DICOM fields.

Examples

```
# First, save toy patient objects to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dir.create (pat.dir, recursive = TRUE)
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = c (4, 4, 4))
save.to.Rdcm (patient$ct[[1]], dirname = pat.dir)
save.to.Rdcm (patient$mr[[1]], dirname = pat.dir)
save.T.MAT (patient$T.MAT, dirname = pat.dir)
# Rdcm files in pat.dir
list.files(pat.dir)

# test of Rdcm.upgrade

Rdcm.upgrade (pat.dir)
# or
Rdcm.upgrade (list.files (pat.dir, full.names = TRUE))

# Cleaning  temporary directories
unlink (pat.dir, recursive = TRUE)
```

`ref.add` *Adding a frame of reference in T.MAT*

Description

The `ref.add` function adds the transfer matrices from or to a new frame of reference defined from 2 unit vectors and an origin point.

Usage

```
ref.add(
  src.ref,
  orientation = c(1, 0, 0, 0, 1, 0),
  origin = c(0, 0, 0),
  new.ref.pseudo = "newref",
  T.MAT = NULL
)
```

Arguments

`src.ref`	Character string, pseudonym of the frame of reference in which the `orientation` vector and the origin point `origin` are defined.
`orientation`	Vector of 6 or 9 elements, composed of the coordinates of the 2 orthonormal vectors (i, j), or of the 3 orthonormal vectors (i, j, k) of the new coordinate system, in the `src.ref` frame of reference.
`origin`	Vector of the x, y, z coordinates of the origin point of the new frame of reference in the `src.ref` frame of reference. Default to c (0, 0, 0).
`new.ref.pseudo`	Character string, pseudonym of the new frame of reference to add.
`T.MAT`	"t.mat" class object created by load.patient.from.dicom, load.patient.from.Rdcm or load.T.MAT. If `T.MAT = NULL`, then only the link between `src.ref` and `new.ref.pseudo` is computed.

Value

Returns a "t.mat" class object, which contains the transfer matrices from or to `new.ref.pseudo` pseudonym of the new frame of reference. If the `T.MAT` is `NULL`, then the returned object will contain only 4 matrices: "src.ref<-src.ref", "src.ref<-new.ref.pseudo", "new.ref.pseudo<- new.ref.pseudo", "new.ref.pseudo<-src.ref".

Returns a `NULL` if `orientation` is not well defined.

See Also

ref.cutplane.add, ref.remove, ref.srctodest.add.

Examples

```
# Adding of the reference frame "ref1_60", which is a 60 degree rotation of
# reference frame "ref1".
orientation <- c (cos (pi / 3), sin (pi / 3), 0,
                  -sin (pi / 3), cos (pi / 3), 0)
```

```
local.Tmat <- ref.add (src.ref = "ref1", orientation = orientation,
                  new.ref.pseudo = "ref1_60")

str(local.Tmat)
```

`ref.cutplane.add` *Adding volume's cutting planes frame of reference in T.MAT*

Description

The `ref.cutplane.add` function adds in T.MAT the transfer matrices from or to volume's cutting planes frame of reference.

Usage

```
ref.cutplane.add(
  vol,
  origin = c(0, 0, 0),
  ref.cutplane = paste0(vol$ref.pseudo, "m"),
  T.MAT = NULL
)
```

Arguments

`vol`	"volume" class object.
`origin`	Vector of the x, y, z coordinates of the origin point of the cut planes frame of reference.
`ref.cutplane`	Name of the volume's cutting planes frame of reference. By default `ref.cutplane = paste0 (vol$ref.pseudo,".m")`.
`T.MAT`	"t.mat" class object created by load.patient.from.dicom, load.patient.from.Rdcm or load.T.MAT. If `T.MAT = NULL`, then only the link between `vol$ref.pseudo` and `ref.name` is established.

Value

Returns a "t.mat" class object, which contains the transfer matrices from or to volume's cutting planes frame of reference. If the `T.MAT` is `NULL`, then the returned object will contain only 4 matrices: "src.ref<-src.ref", "src.ref<-ref.cutplane", "ref.cutplane<-ref.cutplane", "ref.cutplane<-src.ref".

See Also

ref.add, ref.srctodest.add, ref.remove.

Examples

```
# loading of toy-patient objects
patient <- toy.load.patient (modality = "mr", roi.name = "", dxyz = c (4, 4, 4))
MR <- patient$mr[[1]]
MR$xyz.from.ijk

# creation of t.mat, containing the transfer matrix to the frame of reference
```

```
# of the MR cutting planes
t.mat <- ref.cutplane.add (MR)

# Change of frame of reference
MR.m <- vol.in.new.ref (MR, paste0 (MR$ref.pseudo, "m"), t.mat)

MR.m$xyz.from.ijk
```

`ref.remove` *Deletion of a frame of reference in T.MAT*

Description

The `ref.remove` function removes the management of a frame of reference in T.MAT.

Usage

```
ref.remove(ref.name, T.MAT)
```

Arguments

`ref.name`	Character string, pseudonym of the frame of reference to delete.
`T.MAT`	"t.mat" class object in which the `ref.name` frame of reference is to be deleted.

Value

Returns a "t.mat" class object, which no longer contains transfer matrices from or to the ref.pseudo `ref.name`. ref.cutplane.add.

Examples

```
# Adding of the reference frame "ref1_60", which is a 60 degree rotation of
# reference frame "ref1".
orientation <- c (cos (pi / 3), sin (pi / 3), 0,
                  -sin (pi / 3), cos (pi / 3), 0)

local.Tmat <- ref.add (src.ref = "ref1", orientation = orientation,
                  new.ref.pseudo = "ref1_60")
str(local.Tmat)

# Removal of  "ref1_60"
local.Tmat <- ref.remove (ref.name =  "ref1_60", T.MAT = local.Tmat)
str(local.Tmat)
```

`ref.srctodest.add` *Linking two existing frames of reference in T.MAT*

Description

The `ref.srctodest.add` function links the source frame of reference with the destination frame of reference.

Usage

```
ref.srctodest.add(src.ref, dest.ref, TM = diag(4), T.MAT = NULL)
```

Arguments

`src.ref`	Character string, pseudonym of the source frame of reference.
`dest.ref`	Character string, pseudonym of the destination frame of reference.
`TM`	4x4 tansfert matrix for moving from `src.ref` to dest.ref.
`T.MAT`	"t.mat" class object created by load.patient.from.dicom, load.patient.from.Rdcm or load.T.MAT. If `T.MAT = NULL`, then only the link between `src.ref` and `dest.ref` is established.

Value

Returns a "t.mat" class object, which contains the transfer matrices from or to `dest.ref` pseudonym of the new frame of reference. If the `T.MAT` is `NULL`, then the returned object will contain only 4 matrices: "src.ref<-src.ref", "src.ref<-dest.ref", "dest.ref<- dest.ref", "dest.ref<-src.ref".

See Also

ref.add, ref.cutplane.add, ref.remove.

Examples

```
local.Tmat <-  ref.srctodest.add ("ref1","ref2",
                                  TM = matrix(c (0.5, -sin (pi / 3), 0, 0,
                                                 sin (pi / 3), 0.5, 0, 0,
                                                 0, 0, 1, 0, 0, 0, 0, 1),
                                              ncol = 4))
str (local.Tmat)
```

`rt.chi.index` *Chi index 2D - 3D*

Description

The `rt.chi.index` function computes the local or global Chi index from a measurement and a reference. These latter are "volume" class objects containing one (2D) or several planes (3D).

Usage

```
rt.chi.index(
  vol,
  vol.ref,
  abs = TRUE,
  vol.max = vol.ref$max.pixel,
  dose.th = 0.02,
  delta.r = 3,
  analysis.th = 0.05,
  local = FALSE,
  local.th = 0.3,
  project.to.isocenter = TRUE,
  alias = "",
  description = NULL
)
```

Arguments

`vol`	"volume" class object, which represents the measured volume.
`vol.ref`	"volume" class object, which represents the reference volume.
`abs`	Boolean. If `TRUE` (default), the absolute value of Chi is computed.
`vol.max`	Positive number, by default equal to the maximum value of the reference volume. See Details.
`dose.th`	Positive number, in percent, used to determine the dose difference criterion. See Details.
`delta.r`	Positive number, in mm. Distance difference criterion.
`analysis.th`	Positive number, in percent. Only the voxels whose value are greater than or equal `analyse.th` * `vol.max` are processed.
`local`	Boolean. If `local` = `FALSE` (default), a global Chi index is computed, and a local Chi index otherwise.
`local.th`	Positive number, in percent. Local threshold, only used if `local` = `TRUE`. See Details.
`project.to.isocenter`	Boolean. If `TRUE`, and if `vol` and `vol.ref` are of modality "rtimage", the size of the pixels is corrected to correspond to that found if the sensor was at the isocenter.
`alias`	Character string, `$object.alias` of the created object.
`description`	Character string, describing the created object. If `description` = `NULL` (default value), it will be set to Chi index setup.

Details

The Chi index of a voxel n was defined by *Bakai and al* **[1]**. It is computed from the formulae:

$$\chi_n = \frac{D_i - Dref_n}{\sqrt{\Delta D^2 + \Delta r^2 \cdot \|\nabla Dref_n\|^2}}$$

If `abs = TRUE`, the used formulae is :

$$\chi_n = \frac{|D_i - Dref_n|}{\sqrt{\Delta D^2 + \Delta r^2 \cdot \|\nabla Dref_n\|^2}}$$

with D_i the measured dose at voxel i, $Dref_n$ the reference dose at voxel n, $\nabla Dref_n$ the gradient of reference dose at voxel n, Δr the distance difference criterion equal to `delta.r`, and ΔD the distance difference criterion at voxel n defined as follows:

- If `local = FALSE` a global Chi index is computed and $\Delta D = dose.th \cdot vol.max$.
- If `local = TRUE`, then $\Delta D = dose.th \cdot Dref_n$ when $Dref_n \geq local.th \cdot vol.max$, and $\Delta D = dose.th \cdot local.th \cdot vol.max$ otherwise.

Value

Returns a "volume" class object (see espadon.class for class definitions). The `$vol3D.data` field represents the Chi index. Two fields are added: the `$setup` field recalls the calculation setup, and the `$chi.info` field details the number of dose points, the number of evaluated dose points,the rate of evaluated dose points, the rate of absolute values of the Chi index below 1, above 1.2 and 1.5,the max and the mean Chi index.

References

[1] Bakai A, Alber A, Nüsslin F (2003). "A revision of the Gamma-evaluation concept for the comparison of dose distributions." *Physics in Medicine and Biologys*, **48**(21), 3543–3553.

See Also

rt.gamma.index

Examples

```
# Creation of a reference volume  and measured volume
# loading of toy-patient objects (decrease dxyz for  better result)
patient <- toy.load.patient (modality = c ("rtdose", "rtstruct"),
                             roi.name = "ptv", dxyz = c (3, 3, 3))
D.ref <- patient$rtdose[[1]]
# We will assume that the measured dose is equal to the reference dose shifted
# by 3 pixels on the x axis
D.meas <- vol.copy (D.ref, alias = "measured_dose")
D.meas$vol3D.data[1:(D.meas$n.ijk[1] - 3) ,,] <- D.ref$vol3D.data[4:D.ref$n.ijk[1],,]
D.max <- as.numeric(quantile(as.numeric(D.ref$vol3D.data),
                             probs = 99.99/100, na.rm = TRUE))
abs_chi <- rt.chi.index (D.meas, D.ref, vol.max = D.max, delta.r = 6)
abs_chi$chi.info

# Display chi index at isocenter
G.iso <- patient$rtstruct[[1]]$roi.info$Gz[
```

```
  patient$rtstruct[[1]]$roi.info$name == "ptv"]
display.plane(abs_chi, view.coord = G.iso,
              bottom.col = c ("#00FF00", "#007F00", "#FF8000", "#FF0000",
                              "#AF0000"),
              bottom.breaks = c (0, 0.8, 1, 1.2, 1.5, abs_chi$max.pixel),
              interpolate = FALSE, bg = "blue")
```

`rt.gamma.index` *Gamma index 2D - 3D*

Description

The `rt.gamma.index` function computes the local or global Gamma index from a measurement and a reference. These latter are "volume" class objects containing one (2D) or several planes (3D).

Usage

```
rt.gamma.index(
  vol,
  vol.ref,
  over.sampling.factor = 1,
  vol.max = vol.ref$max.pixel,
  dose.th = 0.02,
  delta.r = 3,
  analysis.th = 0.05,
  local = FALSE,
  local.th = 0.3,
  project.to.isocenter = TRUE,
  alias = "",
  description = NULL
)
```

Arguments

`vol`	"volume" class object, which represents the measured volume.
`vol.ref`	"volume" class object, which represents the reference volume.
`over.sampling.factor`	Strictly positive integer, or a vector of 3 strictly positive integers, default to 1. Defined to oversample grids of `vol` and `vol.ref`. Oversampling can be very time consuming.
`vol.max`	Positive number, by default equal to the maximum value of the reference volume. See Details.
`dose.th`	Positive number, in percent, used to determine the dose difference criterion. See Details.
`delta.r`	Positive number, in mm. Distance difference criterion.
`analysis.th`	Positive number, in percent. Only the voxels whose value is greater than or equal `analyse.th*vol.max` are processed.
`local`	Boolean. If `local = FALSE` (default), a global Gamma index is computed, and a local Gamma index otherwise.

`local.th` Positive number, in percent. Local threshold, only used if `local = TRUE`. See Details.

`project.to.isocenter` Boolean. If `TRUE`, and if `vol` and `vol.ref` are of modality "rtimage", the size of the pixels is corrected to correspond to that found if the sensor was at the isocenter.

`alias` Character string, `$object.alias` of the created object.

`description` Character string, describing the created object. If `description = NULL` (default value), it will be set to Gamma index setup.

Details

The Gamma index of a voxel n was defined by *Low and al* **[1]**. It is computed from the formulae:

$$\gamma_n = min\left(\sqrt{\frac{(D_i - Dref_n)^2}{\Delta D^2} + \frac{{r_i}^2}{\Delta {r}^2}}\right)$$

whith D_i the measured dose at voxel i, $Dref_n$ the reference dose at voxel n, r_i the distance between voxels i and n, Δr the distance difference criterion equal to `delta.r`, ΔD the distance difference criterion at voxel n defined as follows:

- If `local = FALSE` a global Gamma index is computed and $\Delta D = dose.th \cdot vol.max$.
- If `local = TRUE`, then $\Delta D = dose.th \cdot Dref_n$ when $Dref_n \geq local.th \cdot vol.max$, and $\Delta D = dose.th \cdot local.th \cdot vol.max$ otherwise.

Value

Returns a "volume" class object (see espadon.class for class definitions). The `$vol3D.data` field represents the Gamma index. Two fields are added: the `$setup` field recalls the calculation setup, and the `$gamma.info` field details the number of dose points, the number of evaluated dose points,the rate of evaluated dose points, the rate of Gamma indices below 1, above 1.2 and 1.5, the max and the mean Gamma index.

References

[1] Low DA, Harms WB, Mutic S, Purdy JA (1998). "A technique for the quantitative evaluation of dose distributions." *Medical Physics*, **25**(5), 656–661.

See Also

rt.chi.index

Examples

```
# Creation of a reference volume  and measured volume
# loading of toy-patient objects (decrease dxyz for  better result)
patient <- toy.load.patient (modality = c ("rtdose", "rtstruct"),
                             roi.name = "ptv", dxyz = c (3, 3, 3))
D.ref <- patient$rtdose[[1]]
# We will assume that the measured dose is equal to the reference dose shifted
# by 3 pixels on the x axis
D.meas <- vol.copy (D.ref, alias = "measured_dose")
D.meas$vol3D.data[1:(D.meas$n.ijk[1] - 3) ,,] <- D.ref$vol3D.data[4:D.ref$n.ijk[1],,]
D.max <- as.numeric(quantile(as.numeric(D.ref$vol3D.data),
```

```
                              probs = 99.99/100, na.rm = TRUE))
gamma <- rt.gamma.index (D.meas, D.ref, delta.r = 6, vol.max = D.max)
gamma$gamma.info

# Display gamma index at isocenter
G.iso <- patient$rtstruct[[1]]$roi.info$Gz[
  patient$rtstruct[[1]]$roi.info$name == "ptv"]
display.plane(gamma, view.coord = G.iso,
              bottom.col = c ("#00FF00", "#007F00", "#FF8000", "#FF0000",
                              "#AF0000"),
              bottom.breaks = c (0, 0.8, 1, 1.2, 1.5, gamma$max.pixel),
              bg = "blue", interpolate = FALSE)
```

`rt.indices.from.bin` *Dosimetry, volume, conformity, homogeneity indices from binary selection*

Description

The `rt.indices.from.bin` function calculates, from a "volume" class object of modality "rtdose", all the standard indicators of radiotherapy, as long as their options are transmitted, for the target and healthy "volume" object of modality "binary".

Usage

```
rt.indices.from.bin(
  vol,
  target.bin.list = NULL,
  healthy.bin.list = NULL,
  T.MAT = NULL,
  presc.dose = NA,
  healthy.tol.dose = NA,
  healthy.weight = 1,
  dosimetry = c("D.min", "D.max", "D.mean", "STD"),
  volume.indices = c("V.tot", "area", "V.prescdose"),
  conformity.indices = c("PITV", "CI.lomax2003", "CN", "NCI", "DSC", "CI.distance",
    "CI.abs_distance", "CDI", "CS3", "ULF", "OHTF", "gCI", "COIN", "G_COSI", "COSI"),
  homogeneity.indices = c("HI.RTOG.max_ref", "HI.RTOG.5_95", "HI.ICRU.max_min",
    "HI.ICRU.2.98_ref", "HI.ICRU.2.98_50", "HI.ICRU.5.95_ref", "HI.mayo2010",
    "HI.heufelder"),
  gradient.indices = c("GI.ratio.50"),
  D.xpc = NULL,
  D.xcc = NULL,
  V.xpc = NULL,
  V.xGy = NULL,
  verbose = TRUE
)
```

Arguments

`vol`	"volume" class object, of "rtdose" modality.

`target.bin.list`
: list of "volume" class objects, of "binary" modality. The `$object.alias` field of each `target.bin.list` object represents the name of the selected region of interest of the target volume.

`healthy.bin.list`
: list of "volume" class objects, of "binary" modality. The `$object.alias` field of each `healthy.bin.list` object represents the name of the selected region of interest of the healthy tissues.

`T.MAT`
: "t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If `T.MAT = NULL`, all `$ref.pseudo` of `bin.list` elements must be equal to `vol$ref.pseudo`.

`presc.dose`
: vector of prescription doses that serve as reference doses for the target RoI.

`healthy.tol.dose`
: vector of tolerance dose of each healthy RoI.

`healthy.weight`
: Vector of weight, indicating the importance of the healthy RoI.

`dosimetry`
: Vector indicating the requested dose indicators from among 'D.min', 'D.max', 'D.mean' and 'STD.' If `D.xpc` is different from `NULL`, it will be added.

`volume.indices`
: Vector indicating the requested volume indices from among 'V.tot', 'V.prescdose' (i.e. volume over `presc.dose`) and 'area'. If `V.xGy` is different from `NULL`, it will be added.

`conformity.indices`
: Vector. Requested conformity indices from among 'PITV', 'PDS', 'CI.lomax2003', 'CN', 'NCI', 'DSC', 'CI.distance', 'CI.abs_distance', 'CDI', 'CS3', 'ULF', 'OHTF', 'gCI', 'COIN', 'COSI' and 'G_COSI'.

`homogeneity.indices`
: Vector. Requested homogeneity indices from among 'HI.RTOG.max_ref', 'HI.RTOG.5_95', 'HI.ICRU.max_min', 'HI.ICRU.2.98_ref', 'HI.ICRU.2.98_50', 'HI.ICRU.5.95_ref', 'HI.mayo2010' and 'HI.heufelder.'

`gradient.indices`
: Vector. Requested gradient indices from among 'GI.ratio.50', 'mGI'.

`D.xpc`
: Vector of the percentage of the volume, for which the dose coverage is requested.

`D.xcc`
: Vector of the volume in cm^3, for which the dose coverage is requested.

`V.xpc`
: Vector of the percentage of the reference dose, received by the volume to be calculated.

`V.xGy`
: Vector of the minimum dose in Gy, received by the volume to be calculated.

`verbose`
: Boolean. if `TRUE` (default) a progress bar is displayed.

Value

Return a list of indices dataframe. For details, see rt.indices.from.roi.

See Also

rt.indices.from.roi.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for better
# result)
step <- 5
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"), roi.name = "eye",
```

```
                              dxyz = rep (step, 3), beam.nb = 3)
D <- patient$rtdose[[1]]
struct <- patient$rtstruct[[1]]
T.MAT <- patient$T.MAT

# creation of the list of target binary volumes
taget.roi.idx <- select.names (struct$roi.info$roi.pseudo, roi.sname = "ptv")
healthy.roi.idx <- select.names (struct$roi.info$roi.pseudo, roi.sname = "eye")
target.bin.list <- lapply (taget.roi.idx , function (idx) {
  vr <- nesting.roi (D, struct, roi.idx = idx, xyz.margin = c (5, 5, 5),
                     T.MAT = T.MAT, alias = struct$roi.info$name[idx])
  b <- bin.from.roi(vr, struct, roi.idx = idx, T.MAT = T.MAT,
                    alias = struct$roi.info$name[idx])
  })
names (target.bin.list) <- struct$roi.info$name[taget.roi.idx]

healthy.bin.list <- lapply (healthy.roi.idx , function (idx) {
  vr <- nesting.roi (D, struct, roi.idx = idx, xyz.margin = c (5, 5, 5),
                     T.MAT = T.MAT, alias = struct$roi.info$name[idx])
  b <- bin.from.roi(vr, struct, roi.idx = idx, T.MAT = T.MAT,
                    alias = struct$roi.info$name[idx])
})
names (healthy.bin.list) <- struct$roi.info$name[healthy.roi.idx]

indices <- rt.indices.from.bin (D, target.bin.list, healthy.bin.list,
                                presc.dose = 50,
                                conformity.indices = c("PITV", "PDS", "CI.lomax2003",
                                                       "CN", "NCI", "DSC","COIN"),
                                verbose = FALSE)
indices
```

`rt.indices.from.roi` *Dosimetry, volume, conformity, homogeneity indices from RoI*

Description

The `rt.indices.from.roi` function calculates, from a "volume" class object of modality "rtdose", standard indicators of radiotherapy in relation to the target and healthy RoI, as long as their options are transmitted.

Usage

```
rt.indices.from.roi(
  vol,
  struct = NULL,
  T.MAT = NULL,
  target.roi.name = NULL,
  target.roi.sname = NULL,
  target.roi.idx = NULL,
  healthy.roi.name = NULL,
  healthy.roi.sname = NULL,
  healthy.roi.idx = NULL,
  presc.dose = NA,
```

```
  healthy.tol.dose = NA,
  healthy.weight = 1,
  dosimetry = c("D.min", "D.max", "D.mean", "STD"),
  volume.indices = c("V.tot", "area", "V.prescdose"),
  conformity.indices = c("PITV", "PDS", "CI.lomax2003", "CN", "NCI", "DSC",
   "CI.distance", "CI.abs_distance", "CDI", "CS3", "ULF", "OHTF", "gCI", "COIN",
    "G_COSI", "COSI"),
  homogeneity.indices = c("HI.RTOG.max_ref", "HI.RTOG.5_95", "HI.ICRU.max_min",
    "HI.ICRU.2.98_ref", "HI.ICRU.2.98_50", "HI.ICRU.5.95_ref", "HI.mayo2010",
    "HI.heufelder"),
  gradient.indices = c("GI.ratio.50", "mGI"),
  D.xpc = NULL,
  D.xcc = NULL,
  V.xpc = NULL,
  V.xGy = NULL,
  verbose = TRUE
)
```

Arguments

`vol`	"volume" class object, of "rtdose" modality.
`struct`	"struct" class object.
`T.MAT`	"t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If `T.MAT = NULL`, `struct$ref.pseudo` must be equal to `vol$ref.pseudo`.
`target.roi.name`	Exact name of target RoI in `struct` object. By default `target.roi.name = NULL`. See Details.
`target.roi.sname`	Name or part of name of target RoI in `struct` object. By default `target.roi.sname = NULL`. See Details.
`target.roi.idx`	Value of the index of target RoI that belong to the `struct` object. By default `target.roi.idx = NULL`. See Details.
`healthy.roi.name`	Exact name of healthy RoI in `struct` object. By default `healthy.roi.name = NULL`.
`healthy.roi.sname`	Name or part of name of healthy RoI in `struct` object. By default `healthy.roi.sname = NULL`.
`healthy.roi.idx`	Value of the index of healthy RoI that belong to the `struct` object. By default `healthy.roi.idx = NULL`.
`presc.dose`	Vector of prescription doses that serve as reference doses for the target RoI.
`healthy.tol.dose`	Vector of tolerance doses of each healthy RoI.
`healthy.weight`	Vector of weights, indicating the importance of the healthy RoI.
`dosimetry`	Vector indicating the requested dose indicators from among 'D.min', 'D.max', 'D.mean' and 'STD.' If `D.xpc` is different from `NULL`, it will be added.
`volume.indices`	Vector indicating the requested volume indices from among 'V.tot', 'V.prescdose' (i.e. volume over `presc.dose`) and 'area'. If `V.xGy` is different from `NULL`, it will be added.

`conformity.indices`
: Vector. Requested conformity indices from among 'PITV', 'PDS', 'CI.lomax2003', 'CN', 'NCI', 'DSC', 'CI.distance', 'CI.abs_distance', 'CDI', 'CS3', 'ULF', 'OHTF', 'gCI', 'COIN', 'COSI' and 'G_COSI'.

`homogeneity.indices`
: Vector. Requested homogeneity indices from among 'HI.RTOG.max_ref', 'HI.RTOG.5_95', 'HI.ICRU.max_min', 'HI.ICRU.2.98_ref', 'HI.ICRU.2.98_50', 'HI.ICRU.5.95_ref', 'HI.mayo2010' and 'HI.heufelder.'

`gradient.indices`
: Vector. Requested gradient indices from among 'GI.ratio.50', 'mGI'.

`D.xpc`
: Vector of the percentage of the volume, for which the dose coverage is requested.

`D.xcc`
: Vector of the volume in cm^3, for which the dose coverage is requested.

`V.xpc`
: Vector of the percentage of the reference dose, received by the volume to be calculated.

`V.xGy`
: Vector of the minimum dose in Gy, received by the volume to be calculated.

`verbose`
: Boolean. if `TRUE` (default) a progress bar is displayed.

Details

If `target.roi.name`, `target.roi.sname`, and `target.roi.idx` are all set to `NULL`, all RoI containing 'ptv' (if they exist) are selected.

If `target.roi.name`, `target.roi.sname`, and `target.roi.idx` are all set to `NULL`,no target RoI are selected.

If `healthy.roi.name`, `healthy.roi.sname`, and `healthy.roi.idx` are all set to `NULL`, no healthy RoI are selected.

Value

Return a list containing (if requested)

$-$ $dosimetry$: dataframe containing, for all target and healthy structures:

- the requested `dosimetry` : `D.min` (Gy), `D.max` (Gy), `D.mean` (Gy) and `STD` (Gy), respectively the minimum, maximum, mean and standard deviation of the dose in the regions of interest.
- the requested `$D.x%` : (Gy) Dose covering x percent of structure volume.
- the requested `$D.xcc` : (Gy) Dose covering x (cm^3) of structure volume.

$-$ $volume$: dataframe containing, for all target and healthy structures, and isodoses:

- the requested `volume.indices` : `V_tot` (cm^3) (except for isodose) the total volume of the regions of interest, `area` (cm^2) (except for isodose) their surface areas, `V.prescdose` (cm^3) the volumes receiving at least `presc.dose` Gy,
- the requested `V.xGy` (cm^3): volumes receiving at least x Gy.
- the requested `V.xpc` (cm^3) Volume receiving at least x% of the reference dose.

$-$ $conformity$: dataframe containing, if requested,

- `PITV` : (1) Prescription Isodose Target Volume, or conformity index defined by *E.Shaw* **[1]**

$$PITV = \frac{V_{presc.dose}}{V_{target}}$$

- `PDS` : (1) Prescription Dose Spillage defined by *SABR UK Consortium 2019* **[2]**

$$PDS = \frac{V_{presc.dose}}{V_{target \geq presc.dose}} = \frac{V_{presc.dose}}{V_{target} \cap V_{presc.dose}}$$

- `CI.lomax2003` : (1) Conformity Index defined by *Lomax and al* **[3]**

$$CI_{lomax2003} = \frac{V_{target \geq presc.dose}}{V_{presc.dose}} = \frac{V_{target} \cap V_{presc.dose}}{V_{presc.dose}}$$

- `CN` : (1) Conformation Number defined by *Van't Riet and al* **[4]**. It corresponds to conformity index defined by *Paddick* **[5]**

$$CN = CI_{paddick2000} = \frac{V^2_{target \geq presc.dose}}{V_{target} \cdot V_{presc.dose}} = \frac{(V_{target} \cap V_{presc.dose})^2}{V_{target} \cdot V_{presc.dose}}$$

- `NCI` : (1) New conformity index, inverse of CN, defined by *Nakamura and al* **[6]**

$$NCI = \frac{1}{CN}$$

- `DSC` : (1) Dice Similarity Coefficient **[7]**

$$DSC = 2 \cdot \frac{V_{target \geq presc.dose}}{V_{target} + V_{presc.dose}} = 2 \cdot \frac{V_{target} \cap V_{presc.dose}}{V_{target} + V_{presc.dose}}$$

- `CI.distance` : (1) Conformity Index based on distance defined by *Park and al* **[8]**

$$CI.distance = \frac{100}{N} \sum^{N} \frac{dist_{S_{presc.dose} \to G_{target}} - dist_{S_{target} \to G_{target}}}{dist_{S_{target} \to G_{target}}}$$

where $dist_{S_{presc.dose} \to G_{target}}$ is the distance between the surface of the prescription dose volume and the centroid of the target, and $dist_{S_{target} \to G_{target}}$ the surface of the target volume and the centroid of the target. N is the number of directions where the distances are calculated. These directions are computed every $1°$. If the centroid is not within the target volume, then `CI.distance = NA`.

- `CI.abs_distance` : (1) Conformity Index based on distance defined by *Park and al* **[8]**

$$CI.abs_distance = \frac{100}{N} \sum^{N} \frac{|dist_{S_{presc.dose} \to G_{target}} - dist_{S_{target} \to G_{target}}|}{dist_{S_{target} \to G_{target}}}$$

- `CDI` : (1) Conformity Distance Index defined by *Wu and al* **[9]**

$$CDI = 2\frac{V_{presc.dose} + V_{target} - 2\,V_{target \geq presc.dose}}{S_{target} + S_{presc.dose}} = \frac{V_{presc.dose} + V_{target} - 2 \cdot V_{target} \cap V_{presc.dose}}{S_{target} + S_{presc.dose}}$$

where S_{target} is the surface of the target volume and $S_{presc.dose}$ is the surface of the prescription dose volume.

- `CS3` : (1) Triple Point Conformity Scale defined by *Ansari and al* **[10]**

$$CS3 = \frac{V_{0.95 \cdot presc.dose} + V_{presc.dose} + V_{1.05 \cdot presc.dose}}{3 \cdot V_{target}}$$

- `ULF` : (1) Underdosed lesion factor defined by *Lefkopoulos and al* **[11]**

$$ULF = \frac{V_{target < presc.dose}}{V_{target}} = \frac{V_{target} \cap \overline{V_{presc.dose}}}{V_{target}}$$

- `OHTF` :(1) Overdosed healthy tissues factor defined by *Lefkopoulos and al* **[11]**

$$OHTF = \frac{\sum V_{healthy \geq presc.dose}}{V_{target}} = \frac{\sum V_{healthy} \cap V_{presc.dose}}{V_{target}}$$

- `gCI` : (1) Geometric Conformity Index defined by *Lefkopoulos and al* **[11]**

$$gCI = ULF + OHTF$$

- `COIN` : Conformity Index defined by *Baltas and al* **[12]**

$$COIN = \frac{V^2_{target \geq presc.dose}}{V_{target} \cdot V_{presc.dose}} \cdot \prod^{N_{healthy}} \left(1 - \frac{V_{healthy \geq presc.dose}}{V_{healthy}}\right)$$

- `gCOSI` : generalized COSI, defined by *Menhel and al* **[13]**.

$$gCOSI = 1 - \sum^{N_{healthy}} healthy.weight \cdot \frac{\frac{V_{healthy \geq healthy.tol.dose}}{V_{healthy}}}{\frac{V_{target \geq presc.dose}}{V_{target}}}$$

$- \ COSI$: if "COSI" is requested in `conformity.indices`, it returns a dataframe of Critical Organ Scoring Index for each healthy organ, at each `presc.dose`, and for each target. COSI is defined by *Menhel and al* **[13]**

$$COSI = 1 - \frac{\frac{V_{healthy \geq healthy.tol.dose}}{V_{healthy}}}{\frac{V_{target \geq presc.dose}}{V_{target}}}$$

$- \ homogeneity$: dataframe containing

- `HI.RTOG.max_ref` : (1) Homogeneity Index from RTOG defined by *E.Shaw* **[1]**

$$HI.RTOG.max\_ref = \frac{D_{\ max}}{presc.dose}$$

  where D_{max} is the maximum dose in the target volume.

- `HI.RTOG.5_95` : (1) Homogeneity Index from RTOG **[1]**

$$HI.RTOG.5\_95 = \frac{D.5pc}{D.95pc}$$

  where $D.5pc$ and $D.95pc$ are respectively the doses at 5% and 95% of the target volume in cumulative dose-volume histogram.

- `HI.ICRU.max_min` : (1) Homogeneity Index from *ICRU report 62* **[14]**

$$HI.ICRU.max\_min = \frac{D_{\ max}}{D_{\ min}}$$

  where D_{max} and D_{min} are respectively the maximum and the minimum dose in the target volume.

- `HI.ICRU.2.98_ref` : (1) Homogeneity Index from *ICRU report 83* **[15]**

$$HI.ICRU.2.98\_ref = 100\frac{D.2pc - D.98pc}{presc.dose}$$

  where $D.2pc$ and $D.98pc$ are respectively the doses at 2% and 98% of the target volume in cumulative dose-volume histogram.

- `HI.ICRU.2.98_50` : (1) Homogeneity Index from *ICRU report 83* **[15]**

$$HI.ICRU.2.98\_50 = 100\frac{D.2pc - D.98pc}{D.50pc}$$

where $D.2pc$, $D.98pc$ and $D.50pc$ are respectively the doses at 2%, 98% and 50% of the target volume in cumulative dose-volume histogram.

- `HI.ICRU.5.95_ref` : (1) Homogeneity Index from *ICRU report 83* **[15]**

$$HI.ICRU.5.95\_ref = 100\frac{D.5pc - D.95pc}{presc.dose}$$

where $D.5pc$ and $D.95pc$ are respectively the doses at 5% and 95% of the target volume in cumulative dose-volume histogram.

- `HI.mayo2010` : (1) Homogeneity Index defined by *Mayo and al* **[16]**

$$HI.mayo2010 = \sqrt{\frac{D_{\,max}}{presc.dose} \cdot (1 + \frac{\sigma_D}{presc.dose})}$$

where D_{max} is the maximum dose in the target volume, and σ_D the standard deviation of the dose in the target volume.

- `HI.heufelder` : (1) Homogeneity Index defined by *Heufelder and al* **[17]**

$$HI.heufelder = e^{-0.01 \cdot (1-\frac{\mu_D}{presc.dose})^2} \cdot e^{-0.01 \cdot (\frac{\sigma_D}{presc.dose})^2}$$

where μ_D and σ_D are respectively the mean and the standard deviation of the dose in the target volume.

– $gradient$: dataframe containing

- `GI.ratio.50`: Gradient Index based on volumes ratio defined by *Paddick and Lippitz* **[18]**

$$GI.ratio.50 = \frac{V_{0.5 \cdot presc.dose}}{V_{presc.dose}}$$

- `mGI`: Modified Gradient Index defined by *SABR UK Consortium 2019* **[2]**

$$mGI = \frac{V_{0.5 \cdot presc.dose}}{V_{target \geq presc.dose}} = \frac{V_{0.5 \cdot presc.dose}}{V_{target} \cap V_{presc.dose}}$$

References

[1] Shaw E, Kline R, Gillin M, Souhami L, Hirschfeld A, Dinapoli R, Martin L (1993). "Radiation therapy oncology group: Radiosurgery quality assurance guidelines." *International journal of radiation oncology, biology, physics*, **27**(5), 1231-1239. ISSN 0360-3016, doi:10.1016/0360-3016(93)90548A.

[2] UK SABR Consortium (Online; accessed 2022-04-01). "Stereotactic Ablative Radiation Therapy (SABR): a resource. v6.1, January 2019." https://www.sabr.org.uk/wp-content/uploads/2019/04/SABRconsortium-guidelines-2019-v6.1.0.pdf.

[3] Lomax NJ, Scheib SG (2003). "Quantifying the degree of conformity in radiosurgery treatment planning." *International journal of radiation oncology, biology, physics*, **55**(5), 1409-1419. ISSN 0360-3016, doi:10.1016/S03603016(02)045996.

[4] Riet AV, Mak AC, Moerland MA, Elders LH, Van der Zee W (1997). "A conformation number to quantify the degree of conformality in brachytherapy and external beam irradiation: Application

to the prostate." *International journal of radiation oncology, biology, physics*, **37**(3), 731-736. ISSN 0360-3016, doi:10.1016/S03603016(96)006013.

[5] Paddick I (2000). "A simple scoring ratio to index the conformity of radiosurgical treatment plans. Technical note." *Journal of neurosurgery*, **93 Suppl 3**, 219-222.

[6] Nakamura J, Verhey L, Smith V, Petti P, Lamborn K, Larson D, Wara W, Mcdermott M, Sneed P (2002). "Dose conformity of Gamma Knife radiosurgery and risk factors for complications." *International journal of radiation oncology, biology, physics*, **51**, 1313-9. doi:10.1016/S0360-3016(01)017576.

[7] Dice LR (1945). "Measures of the Amount of Ecologic Association Between Species." *Ecology*, **26**(3), 297–302. ISSN 00129658, 19399170.

[8] Park JM, Park S, Ye S, Kim J, Carlson J, Wu H (2014). "New conformity indices based on the calculation of distances between the target volume and the volume of reference isodose." *The British journal of radiology*, **87**, 20140342. doi:10.1259/bjr.20140342.

[9] Wu Q, Wessels BW, Einstein DB, Maciunas RJ, Kim EY, Kinsella TJ (2003). "Quality of coverage: Conformity measures for stereotactic radiosurgery." *Journal of Applied Clinical Medical Physics*, **4**, 374-381.

[10] Ansari S, Satpathy S, Singh P, Lad S, Thappa N, Singh B (2018). "A new index: Triple Point Conformity Scale (CS3) and its implication in qualitative evaluation of radiotherapy plan." *Journal of Radiotherapy in Practice*, **17**, 1-4. doi:10.1017/S1460396917000772.

[11] Lefkopoulos D, Dejean C, balaa ZE, Platoni K, Grandjean P, Foulquier J, Schlienger M (2000). "Determination of dose-volumes parameters to characterise the conformity of stereotactic treatment plans." In chapter XIII, 356-358. Springer Berlin Heidelberg. ISBN 978-3-540-67176-3, doi:10.1007/9783642597589_135.

[12] Baltas D, Kolotas C, Geramani KN, Mould RF, Ioannidis G, Kekchidi M, Zamboglou N (1998). "A conformal index (COIN) to evaluate implant quality and dose specification in brachytherapy." *International journal of radiation oncology, biology, physics*, **40 2**, 515-24. doi:10.1016/s0360-3016(97)007323.

[13] Menhel J, Levin D, Alezra D, Symon Z, Pfeffer R (2006). "Assessing the quality of conformal treatment planning: a new tool for quantitative comparison." *Physics in Medicine and Biology*, **51**(20), 5363–5375.

[14] Landberg T, Chavaudra J, Dobbs J, Gerard J, Hanks G, Horiot J, Johansson K, Möller T, Purdy J, Suntharalingam N, Svensson H (1999). "ICRU Report 62: Prescribing, Recording and Reporting Photon Beam Therapy (Supplement to ICRU Report 50),3. Absorbed Doses." *Reports of the International Commission on Radiation Units and Measurements*, **os-32**(1), 21-25.

[15] ICRU (2010). "Report 83 : Prescribing, Recording, and Reporting Photon-Beam Intensity-Modulated Radiation Therapy (IMRT)." *Reports of the International Commission on Radiation Units and Measurements*, **10**(1), 1-3.

[16] Mayo CS, Ding L, Addesa A, Kadish S, Fitzgerald TJ, Moser R (2010). "Initial Experience With Volumetric IMRT (RapidArc) for Intracranial Stereotactic Radiosurgery." *International Journal of Radiation Oncology*Biology*Physics*, **78**(5), 1457-1466. ISSN 0360-3016, doi:10.1016/j.ijrobp.2009.10.005.

[17] Heufelder J, Zink K, Scholz M, Kramer K, Welker K (2003). "Eine Methode zur automatisierten Bewertung von CT-basierten Bestrahlungsplanen in der perkutanen Strahlentherapie." *Zeitschrift fur Medizinische Physik*, **13**(4), 231-239. ISSN 0939-3889, doi:10.1078/0939388900175.

[18] Paddick I, Lippitz BE (2006). "A simple dose gradient measurement tool to complement the conformity index." *Journal of neurosurgery*, **105 Suppl**, 194-201.

All this references are compiled by

- Kaplan LP, Korreman SS (2021). "A systematically compiled set of quantitative metrics to describe spatial characteristics of radiotherapy dose distributions and aid in treatment planning." *Physica Medica*, **90**, 164-175. ISSN 1120-1797, doi:10.1016/j.ejmp.2021.09.014. and
- Patel G, Mandal A, Choudhary S, Mishra R, Shende R (2020). "Plan evaluation indices: A journey of evolution." *Reports of Practical Oncology & Radiotherapy*, **25**. doi:10.1016/j.rpor.2020.03.002..

See Also

rt.indices.from.bin.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb
#  for better result)
step <- 5
patient <- toy.load.patient (modality = c("rtdose", "rtstruct"), roi.name = "eye",
                             dxyz = rep (step, 3), beam.nb = 3)
indices <- rt.indices.from.roi (patient$rtdose[[1]],  patient$rtstruct[[1]],
                                target.roi.sname = "ptv",
                                healthy.roi.sname = "eye", presc.dose = 50,
                                conformity.indices = c("PITV", "PDS", "CI.lomax2003",
                                                       "CN", "NCI", "DSC","COIN"),
                                verbose = FALSE)
indices
```

---

save.T.MAT *Save a T.MAT class object*

Description

The `save.T.MAT` function saves the data required by load.T.MAT, load.patient.from.dicom or load.patient.from.Rdcm to generate `T.MAT`, as pre-formatted Rdcm files.

Usage

```
save.T.MAT(T.MAT, dirname)
```

Arguments

`T.MAT`	"t.mat" class object to save.
`dirname`	Directory where new reg .Rdcm files will be saved.

Details

Reg files from DICOM files cannot be updated with the `save.T.MAT` function. Only transfer matrices added with ref.add or ref.cutplane.add will be saved.

Value

Returns `TRUE`, if all reg files generating T.MAT are saved.

Examples

```
# loading of toy-patient objects (decrease dxyz for  better result)
step <- 5
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = rep (step, 3))

# Save T.MAT to a temporary file pat.dir
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dir.create (pat.dir, recursive = TRUE)
save.T.MAT (patient$T.MAT, dirname = pat.dir)
list.files(pat.dir)

# Cleaning  temporary directory
unlink (pat.dir, recursive = TRUE)
```

---

save.to.Rdcm *Save a* **espadon** *object in a pre-formatted *.Rdcm file*

Description

The function `save.to.Rdcm` allows you to save an object created by **espadon** in a pre-formatted *.Rdcm file. This object will also be accessible by the `load.patient.from.Rdcm` function.

Usage

```
save.to.Rdcm(obj, object.name = obj$object.alias, dirname = obj$file.dirname)
```

Arguments

`obj`	**espadon** object of class `"volume"`, `"struct"`, `"mesh"`, `"histo"`, `"dvh"`, `"histo2D"`.
`object.name`	Character string, representing the name of the object, default to `obj$object.alias`.
`dirname`	Directory where new files from `obj` will be saved.

Value

Returns `TRUE`, if `paste0(object.name,".Rdcm")` exists in `dirname`.

Returns `FALSE`, if `object.name` is not a valid file name, or if the file that is created would replace a *.Rdcm file created by dicom.to.Rdcm.converter.

Note

`save.to.Rdcm` can not replace an *.Rdcm file created by dicom.to.Rdcm.converter.

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 5
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = rep (step, 3))

# Save T.MAT to a temporary file pat.dir
pat.dir <- file.path (tempdir(), "PM_Rdcm")
```

```
dir.create (pat.dir, recursive = TRUE)
save.to.Rdcm (patient$ct[[1]], dirname = pat.dir)
save.to.Rdcm (patient$mr[[1]], dirname = pat.dir)
list.files(pat.dir)

# Cleaning  temporary directory
unlink (pat.dir, recursive = TRUE)
```

`select.names` *Regions of Interest (RoI) indices*

Description

The select.names function allows you to select words from a vector of words, according to several criteria, eliminating spaces and case.

Usage

```
select.names(names, roi.name = NULL, roi.sname = NULL, roi.idx = NULL)
```

Arguments

`names`	Words vector
`roi.name`	Vector of words to compare to `names`. By default `roi.name` = `NULL`. See Details
`roi.sname`	Vector of words or parts of words to compare. By default `roi.sname` = `NULL`. See Details
`roi.idx`	Index vector. By default `roi.idx` = `NULL`. See Details.

Details

If `roi.name`, `roi.sname`, and `roi.idx` are all `NULL`, then all RoI are selected.

Value

Returns the indices of the elements of the word vector `names` satisfying one or more of the following conditions:

- ASCII // TRANSLIT transcriptions, without spaces, of `names` and `roi.name`, are identical.
- ASCII // TRANSLIT transcriptions, without spaces of `roi.sname` are identical to part of ASCII // TRANSLIT transcriptions, without spaces of `names`.
- `names` indices belong to the index vector `roi.idx`.

Examples

```
# loading patient objects
names <- c ("Eye left", "EyeR", "OPTICAL nerve L", "opical nervR", "chiasma")

# RoI selection.
select.names (names = names, roi.name = c("eye left", "eye right"))
select.names (names = names, roi.sname = c("eye", "ner"))
select.names (names = names, roi.idx = 4:9)
```

set.reference.obj *Set the reference objects of a espadon object*

Description

The function set.reference.obj adds to an espadon object the information identifying the espadon objects from which it derives.

Usage

```
set.reference.obj(obj, ref.obj, add = TRUE)
```

Arguments

obj	espadon object of class "dvh", "fan", "histo", "histo2D", "mesh", "rtplan", "struct", "undef" or "volume".
ref.obj	espadon object of class "dvh", "fan", "histo", "histo2D", "mesh", "rtplan", "struct", "undef" or "volume". List of espadon objects.
add	Boolean. If TRUE, the reference objects are added to those already contained by obj.

Value

Returns the espadon object obj, containing the ref.object.alias and ref.object.info fields identifying its reference objects

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 6
pat <- toy.load.patient (modality = c("ct", "rtdose", "rtstruct"),
                         roi.name = c("eye"), dxyz = rep (step, 3),
                         beam.nb = 3)
display.obj.links(pat)
pat$rtstruct[[1]] <- set.reference.obj(pat$rtstruct[[1]],pat$ct[[1]])
display.obj.links(pat)
```

struct.clustering *Clustering volumes by RoI*

Description

The struct.clustering function creates a new volume in which voxels are clustered and labeled by region of interest defined in an rt-struct.

Usage

```
struct.clustering(
  vol,
  struct,
  roi.name = NULL,
  roi.sname = NULL,
  roi.idx = NULL,
  T.MAT = NULL,
  alias = "",
  description = NULL,
  verbose = TRUE
)
```

Arguments

vol	"volume" class object.
struct	"struct" class object.
roi.name	Vector of exact names of the RoI in the struct object. By default roi.name = NULL. See Details.
roi.sname	Vector of names or parts of names of the RoI in the struct object. By default roi.sname = NULL. See Details.
roi.idx	Vector of indices of the RoI that belong to the struct object. By default roi.idx = NULL. See Details.
T.MAT	"t.mat" class object, created by load.patient.from.Rdcm or load.T.MAT. If T.MAT = NULL, struct$ref.pseudo must be equal to vol$ref.pseudo.
alias	Character string, $alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be set to paste (struct$object.alias,"clustering")
verbose	Boolean. if TRUE (default), the RoI studied are listed.

Details

If roi.name, roi.sname, and roi.idx are all set to NULL, all RoI are selected.

Value

Returns a "volume" class object (see espadon.class for class definitions), of "cluster" modality. This object contains the $cluster.info field, detailing the label and volumes in cm^3 of the different clusters. Note that the label NA or value 0 is used for the voxels which are not contained in any RoI (air for instance).

See Also

get.roi.connection

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 5
patient <- toy.load.patient (modality = c ("mr", "rtstruct"),
                             dxyz = rep (step, 3))
```

```
MR <- patient$mr[[1]]
S <- patient$rtstruct[[1]]
cluster.vol <- struct.clustering (MR, S, T.MAT = patient$T.MAT, verbose = FALSE)
head (cluster.vol$cluster.info)

# Display
n = nrow(cluster.vol$cluster.info)
col = c ("#00000000", rainbow (n))
breaks <- seq (0, n, length.out = n+2)

display.plane (cluster.vol, main = "RoI clustering", view.coord = 0,
               bottom.col = col, bottom.breaks = breaks, interpolate = FALSE)
```

`struct.from.bin` *Creation of struct class object from a binary volume*

Description

The `struct.from.bin` function creates a struct object with a unique RoI, defined by the contours of binary volume.

Usage

```
struct.from.bin(
  vol,
  roi.name = vol$description,
  roi.nb = 1,
  roi.color = "#379DA2",
  roi.type = c("", "EXTERNAL", "PTV", "CTV", "GTV", "TREATED_VOLUME", "IRRAD_VOLUME",
    "OAR", "BOLUS", "AVOIDANCE", "ORGAN", "MARKER", "REGISTRATION", "ISOCENTER",
    "CONTRAST_AGENT", "CAVITY", "BRACHY_CHANNEL", "BRACHY_ACCESSORY", "BRACHY_SRC_APP",
    "BRACHY_CHNL_SHLD", "SUPPORT", "FIXATION", "DOSE_REGION", "CONTROL",
    "DOSE_MEASUREMENT"),
  external.only = FALSE,
  alias = "",
  description = paste("RoI from", vol$object.alias)
)
```

Arguments

`vol`	"volume" class object, of binary modality.
`roi.name`	Character string, representing the name of created RoI.
`roi.nb`	Positive integer, representing the number of created RoI.
`roi.color`	Color of the created RoI, in hex code format ("#RRGGBB").
`roi.type`	Type of RoI, from among "", "EXTERNAL", "PTV", "CTV", "GTV", "TREATED_VOLUME", "IRRAD_VOLUME", "OAR", "BOLUS", "AVOIDANCE", "ORGAN", "MARKER", "REGISTRATION", "ISOCENTER", "CONTRAST_AGENT", "CAVITY", "BRACHY_CHANNEL "BRACHY_ACCESSORY", "BRACHY_SRC_APP", "BRACHY_CHNL_SHLD", "SUPPORT", "FIXATION", "DOSE_REGION","CONTROL" and "DOSE_MEASUREMENT"
`external.only`	Boolean. If `TRUE`, only external contours are kept.

alias	Character string, $alias of the created object.
description	Character string, describing the created object.

Value

Returns a "struct" class object (see espadon.class for class definition), including the unique roi.name as region of interest.

Examples

```
# Contours of a sphere of 10 mm radius
b.sphere <- vol.create (n.ijk = c (40, 40, 40), dxyz = c(1,1,1),
                        mid.pt = c (0, 0, 0), modality = "binary",
                        default.value = FALSE)
xyz <- expand.grid (-20:19, -20:19, -20:19)
R <- 10
Sphere.flag <- (xyz[, 1]^2 + xyz[, 2]^2 + xyz[, 3]^2) <= R^2
b.sphere$vol3D.data[Sphere.flag] <- TRUE
b.sphere$max.pixel <- TRUE
S.sphere <- struct.from.bin (b.sphere, roi.name = "sphere", external.only = TRUE)
str (S.sphere$roi.info)
```

---

struct.in.new.ref *Change of frame of reference of a "struct" class object.*

Description

The struct.in.new.ref function allows you to change the frame of reference of a struct.

Usage

```
struct.in.new.ref(struct, new.ref.pseudo, T.MAT, alias = "")
```

Arguments

struct	"struct" class object.
new.ref.pseudo	pseudonym of the frame of reference in which the struct should be located. This new.ref.pseudo must exist in the T.MAT list.
T.MAT	"t.mat" class object, created by load.patient.from.dicom, load.patient.from.Rdcm, load.T.MAT or ref.add.
alias	Character string, $alias of the created object.

Value

Returns "struct" class object in the new frame of reference new.ref.pseudo.

See Also

vol.in.new.ref

Examples

```
# loading of toy-patient objects (decrease dxyz for better result)
step <- 4
patient <- toy.load.patient (modality = c("mr", "rtstruct"), roi.name = "",
                             dxyz = rep (step, 3))
S <- patient$rtstruct[[1]]
S.in.new.ref <- struct.in.new.ref (S, patient$mr[[1]]$ref.pseudo, patient$T.MAT)
```

---

struct.merge *Merging of structures into a new structure*

Description

The struct.merge function merges two structures into a new one. It is useful for comparing contours, for example.

Usage

```
struct.merge(
  ref.struct,
  add.struct,
  roi.name = NULL,
  roi.sname = NULL,
  roi.idx = NULL,
  suffix = "",
  alias = "",
  description = ""
)
```

Arguments

ref.struct	struct class object. All RoI of this structure are kept.
add.struct	struct class object. Only the selected RoI are kept for merging.
roi.name	Vector of exact names of the RoI in the add.struct object. By default roi.name = NULL. See Details.
roi.sname	Vector of names or parts of names of the RoI in the add.struct object. By default roi.sname = NULL. See Details.
roi.idx	Vector of indices of the RoI that belong to the add.struct object. By default roi.idx = NULL. See Details.
suffix	Character string. '-suffix' is added to RoI name.
alias	Character string, $alias of the resulted object.
description	Character string, describing the the resulted object.

Details

If roi.name, roi.sname, and roi.idx are all NULL, then all RoI of add.struct are selected.

Value

Returns a struct class object. See espadon.class for class definitions.

Note

Beware that, when merging structures, some RoI may have same name or `roi.info$roi.pseudo`. In this case `struct.merge` prints a warning message. Consider changing `suffix` to avoid the ambiguity.

See Also

struct.from.bin.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
patient <- toy.load.patient (modality = c("rtdose"),
                             dxyz = rep (step, 3), beam.nb = 3)
D <- patient$rtdose[[1]]

# isodose 50% Dmax Gy and 90% Dmax
bin50 <- bin.from.vol (D, min = 0.5 * D$max.pixel)
bin90 <- bin.from.vol (D, min = 0.9 * D$max.pixel)
S.isodose50 <- struct.from.bin (bin50, roi.name = "50pc" ,
                                roi.color = "#00FFFF")
S.isodose90 <- struct.from.bin (bin90, roi.name = "90pc" ,
                                roi.color = "#FFFF00")
S <- struct.merge (S.isodose50, S.isodose90, alias = "isodose",
                   description = paste ("isodose of", D$object.alias))
# Dmax location :
z.dmax <- get.xyz.from.index(which (D$vol3D.data == D$max.pixel), D)[1,3]
display.plane(top = D, struct = S, view.coord = z.dmax, legend.shift = -50)
```

`study.deployment` *Deployment of DICOM files from multiple patients*

Description

The `study.deployment` function duplicates DICOM data from multiple patients, so that it becomes data independent of the original data. This function simplifies the analysis of multi-center or multi-expert studies in dosimetry challenges, contouring consensus searches, etc.

Usage

```
study.deployment(
  pats.dir,
  deploy.dir,
  design.matrix = matrix(TRUE, nrow = length(dir(pats.dir)), ncol = 1, dimnames =
    list(basename(dir(pats.dir)), "expert_1")),
  pid.prefix = "",
  white.list = c("instance", "reference"),
  black.list = c("frame of reference", "class"),
  tag.dictionary = dicom.tag.dictionary()
)
```

Arguments

pats.dir	Name of the directory in which all patient directories are stored, each containing the DICOM files to be duplicated.
deploy.dir	Name of the directory where all patient files will be duplicated.
design.matrix	Boolean matrix. See Details.
pid.prefix	string vector of length 1 or string vector of length ncol(design.matrix), representing the prefix added to the new unique identifier of the deployed patient (tag (0010,0020)).
white.list	Names vector, representing a part of the DICOM tag name UI value representation, other than those defined by the DICOM standard, which will be modified. By default, the UID name containing 'instance' or 'reference' will be modified.
black.list	Names vector, representing a part of the DICOM tag name UI value representation, other than those defined by the DICOM standard, which will not be modified. By default, the frame of reference UID will not be modified.
tag.dictionary	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.

Details

The design.matrix argument defines how patients DICOM files will be deployed. The names of the lines must match the names of the directories contained in pats.dir. The names of the columns are for example the different experts or hospitals who will study the patient files. These experts will only review the patients files defined by rownames(design.matrix)[design.matrix[,"expert"]].

Value

Creates the deploy.dir directory, containing the expert directories defined by the design.matrix column names. Each expert directory contains as many patient directories as defined by the design.matrix row names. All patients will be independent of each other. The new created patients have the pats.dir as name, and expert name as first name, and an independent patient ID, with prefix pid.prefix.

Examples

```
# First, save toy.dicom.raw () raw data to a temporary file/pats.dir/toy_PM
# for testing.
toy_PM.dir <- file.path (tempdir(), "pats.dir","toy_PM")
dir.create (toy_PM.dir, recursive = TRUE)
dcm.filename <- tempfile (pattern = "toyrtplan", tmpdir = toy_PM.dir,
                          fileext = ".dcm")
zz <- file (dcm.filename, "wb")
writeBin (toy.dicom.raw (), zz, size = 1)
close (zz)

# function test:
pats.dir <- dirname (toy_PM.dir)
deploy.dir <- file.path (tempdir(), "deploy.dir")
design.matrix <- matrix(TRUE, nrow = length (dir (pats.dir)), ncol=3,
                        dimnames = list (basename (dir (pats.dir)),
                                         c("Dr Quinn","Dr Who","Dr House")))
design.matrix
study.deployment (pats.dir, deploy.dir, design.matrix,
                  pid.prefix = c("zz_", "yy_", "xx_"))
```

```
# check result
list.files(deploy.dir, recursive = TRUE)
load.patient.from.dicom(deploy.dir)$patient
# Cleaning  temporary directory
unlink (pats.dir, recursive = TRUE)
unlink (deploy.dir, recursive = TRUE)
```

`toy.dicom.raw` *toy DICOM raw data*

Description

The `toy.dicom.raw` loads raw data from a dummy DICOM file. It is used for the test.

Usage

```
toy.dicom.raw()
```

Value

Returns the raw data of a dummy DICOM file of rtplan modality.

Examples

```
toy.dicom.raw ()
```

`toy.load.patient` *Load a toy patient for test*

Description

The `toy.load.patient` creates a dummy "patient" class object. It is used for the test.

Usage

```
toy.load.patient(
  modality = c("ct", "mr", "rtdose", "rtstruct"),
  roi.name = c("eye", "optical nerve", "brain", "labyrinth processing unit",
    "energy unit", "gizzard", "ghost container", "exhaust valve"),
  dxyz = c(1, 1, 1),
  beam.nb = 7
)
```

Arguments

modality	String vector, whose elements are chosen among the modalities "ct", "mr", "rtstruct" and "rtdose".
roi.name	String vector, whose elements are chosen among the regions of interest (RoI) "eye", "optical nerve", "brain", "labyrinth processing unit", "energy unit", "gizzard", "ghost container" and "exhaust valve". Note that the RoI "couch", "patient" and "ptv" are still present.
dxyz	Vector of length 3, representing the x, y, z steps in mm, between ct, mr and rtdose voxels.
beam.nb	Positive integer. Number of radiotherapy beams in rtdose modality.

Value

Returns an toy object of "patient" class, containing the modalities defined in modality. See espadon.class for class definitions.

Examples

```
# loading of toy-patient objects (decrease dxyz for  better result)
step <- 5
pat <- toy.load.patient (dxyz = rep (step, 3), beam.nb = 2)
str (pat, max.level = 2)
```

---

vector.product *Vector product of two vectors*

Description

Vector product of two vectors

Usage

```
vector.product(v1, v2)
```

Arguments

v1	Vector of x, y, z coordinates
v2	Vector of x, y, z coordinates

Value

Returns the x, y, z coordinates of the vector product of v1 and v2

Examples

```
vector.product(c (1, 0, 0), c (0, 1, 0))
```

vol.copy *Creating a volume from another one*

Description

The vol.copy function creates a "volume" class object, with the same grid as the vol volume object.

Usage

```
vol.copy(vol, alias = "", modality = NULL, description = NULL, number = NULL)
```

Arguments

vol	"volume" class object, template of the created object.
alias	Character string, $object.alias of the created object.
modality	Character string, modality of the created volume. If modality = NULL, then the created object will have the modality of vol.
description	Character string, description of the returned object. If descritption = NULL, then the created object will have the description of vol.
number	number of the returned volume. If number = NULL, then the returned object will have the number of vol.

Value

Returns a "volume" class object (see espadon.class for class definitions), with the same grid as vol, in which $vol3D.data is initialized to NA.

Examples

```
# loading of toy-patient objects (decrease dxyz for  better result)
patient <- toy.load.patient (modality = "ct", roi.name = "",dxyz = c (4, 4, 4))
CT <- patient$ct[[1]]

# creating a volume
vol.from.CT <- vol.copy (CT, alias = "ct reference")
str (vol.from.CT)
```

vol.create *Volume creating*

Description

The vol.create function creates a volume object from a user-defined grid.

Usage

```
vol.create(
  n.ijk,
  dxyz,
  mid.pt = NULL,
  pt000 = NULL,
  default.value = NA,
  ref.pseudo = "ref1",
  frame.of.reference = "",
  alias = "",
  modality = "",
  description = "",
  number = 0
)
```

Arguments

n.ijk	Vector of length 3, representing the number of elements on the i, j and k axes.
dxyz	Vector of length 3, representing the x, y, z steps in mm, between voxels. See details.
mid.pt	Vector of length 3, representing the x, y, z coordinates of the midpoint of the volume. See details.
pt000	Vector of length 3, representing the x, y, z coordinates of the first voxel of the first plane.
default.value	Numerical or boolean value, representing the default value of the voxels.
ref.pseudo	Character string, frame of reference pseudonym of the created object.By defaukt equal to "ref1"
frame.of.reference	Character string, frame of reference of the created object.
alias	Character string, $alias of the created object.
modality	Character string, $modality of the created object.
description	Character string, describing the the created object.
number	Integer, by default set to 0, number of the created object.

Details

If mid.pt and pt000 are both equal to NULL, then mid.pt = c (0, 0, 0) by default. If mid.pt and pt000 are both different from NULL, then only mid.pt is taken into account.

Value

Returns a "volume" class object (see espadon.class for class definitions), in which the grid is defined by pt000 or mid.pt, dxyz n.ijk. If default.value are initialized to FALSE, then modality = "binary". The orientation of the patient is orthonormal to the grid.

Examples

```
new.vol <- vol.create (pt000 = c(1,10,10), dxyz = c (1 , 1, 1),
                       n.ijk = c(100, 100, 100),
                       ref.pseudo = "ref1",
```

```
                        frame.of.reference = "toyref1",
                        alias = "new ct", modality = "ct",
                        description = "")
str (new.vol)
```

`vol.from.bin` *Volume class object according to binary selection*

Description

The `vol.from.bin` function selects a part of a "volume" class object of "binary" modality which has the same grid. It is especially useful to restrict voxel data in region of interest.

Usage

```
vol.from.bin(vol, sel.bin, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object, containing data to restrict.
`sel.bin`	"volume" class object, of "binary" modality. `vol` and `sel.bin` must have the same grid.
`alias`	Character string, `$alias` of the created object
`description`	Character string, describing the created object. If `description` = `NULL` (default value), it will be set to `paste (vol$object.alias, "from", sel.bin$object.alias)`

Value

Returns a "volume" class object (see espadon.class for class definitions), in which non-selected voxels have the value `NA`, and selected voxels have the original value of `vol`.

Examples

```
# loading of toy-patient objects (decrease dxyz for  better result)
patient <- toy.load.patient (modality = c ("ct", "rtstruct"),
                             roi.name = "brain", dxyz = c (4, 4, 4))
CT <- patient$ct[[1]]
S <- patient$rtstruct[[1]]

# select the brain in the volume
bin.brain <- bin.from.roi (vol = CT, struct = S, roi.name = "brain")
vol.brain <- vol.from.bin (CT, bin.brain)
# display at the center of gravity of the brain Gz
Gz <- S$roi.info [grep("^brain", S$roi.info$roi.pseudo),]$Gz
display.plane (bottom = vol.brain, view.coord = Gz, struct = S,
               roi.sname = "brain", bg = "#00ff00", interpolate = FALSE)
```

vol.gradient *Gradient of a volume*

Description

The vol.gradient function calculates the 3D gradient of a "volume" class object

Usage

```
vol.gradient(vol, alias = "", description = NULL)
```

Arguments

vol	"volume" class object.
alias	Character string, $alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be set to paste (vol$object.alias, "gradient").

Value

Returns a "volume" class object (see espadon.class for class definitions), with the same grid and modality as vol, gradient of vol.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 3
pat <- toy.load.patient (modality = c ("ct"), dxyz = rep (step, 3))
CT.gradient <- vol.gradient (pat$ct[[1]])

display.plane (CT.gradient, view.type = "sagi", view.coord = 61,
               interpolate = FALSE)
```

vol.in.new.ref *Change of frame of reference of a volume*

Description

The vol.in.new.ref function allows you to change the frame of reference of a volume.

Usage

```
vol.in.new.ref(vol, new.ref.pseudo, T.MAT, alias = "", description = NULL)
```

Arguments

vol	"volume" class object.
new.ref.pseudo	pseudonym of the frame of reference in which the volume should be located. This new.ref.pseudo must exist in the T.MAT list.
T.MAT	"t.mat" class object, created by load.patient.from.dicom, load.patient.from.Rdcm, load.T.MAT or ref.add.
alias	Character string, $alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be that of the vol volume.

Value

Returns "volume" class object in the new frame of reference new.ref.pseudo.

See Also

struct.in.new.ref

Examples

```
# loading of toy-patient objects (decrease dxyz for  better result)
patient <- toy.load.patient (modality = c("ct", "mr"), roi.name = "",
                             dxyz = c (4, 4, 4))
CT <- patient$ct[[1]]
CT.in.new.ref <- vol.in.new.ref (CT, patient$mr[[1]]$ref.pseudo, patient$T.MAT)
```

---

vol.median *Median filter on a volume*

Description

The vol.median function applies a 26-connectivity median filter on all the voxels of a "volume" class object.

Usage

```
vol.median(vol, alias = "", description = NULL)
```

Arguments

vol	"volume" class object.
alias	Character string, $alias of the created object.
description	Character string, describing the created object. If description = NULL (default value), it will be set to paste (vol$object.alias, "median").

Value

Returns a "volume" class object (see espadon.class for class definitions), with the same grid and modality as vol, in which voxels are filtered by a 26-connectivity median filter.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 3
pat <- toy.load.patient (modality = c ("ct"), dxyz = rep (step, 3))
CT.median <- vol.median (pat$ct[[1]])

display.plane (CT.median, view.type = "sagi", view.coord = 61,
               interpolate = FALSE)
```

`vol.oversampling` *Oversampling a volume*

Description

The `vol.oversampling` function oversamples the grid of a "volume" class object.

Usage

```
vol.oversampling(vol, fact.ijk = 2, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object.
`fact.ijk`	Strictly positive integer, or a vector of 3 strictly positive integers.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object. If `description = NULL`, it will be `paste ("oversampling" ,vol$description)`.

Value

Returns a "volume" class object, in which 3D volume grid is oversampled: the voxel size is divided by `fact.ijk`.

See Also

vol.subsampling.

Examples

```
vol <- vol.create(n.ijk = c(10,10,1),dxyz = c(2,2,2), ref.pseudo = "ref1",
                  modality ="test", pt000 = c(0,0,0))
vol$vol3D.data[] <- array(1:prod(vol$n.ijk), dim = vol$n.ijk)
vol$max.pixel <- prod(vol$n.ijk)
vol$min.pixel <- 1
mid <- as.numeric (apply (get.extreme.pt (vol), 1, mean))

vol_os <- vol.oversampling (vol, fact.ijk= 2)
mid_os <- as.numeric (apply (get.extreme.pt (vol_os), 1, mean))

display.plane(vol,interpolate = FALSE, view.coord = mid[3],
              abs.rng = c(-5,25), ord.rng = c(-5,25), bg="green")
```

```
points (mid[1], mid[2], pch=16, col="red")
display.plane(vol_os,interpolate = FALSE, view.coord = mid_os[3],
              abs.rng = c(-5,25), ord.rng = c(-5,25), bg="green")
points (mid_os[1], mid_os[2], pch=16, col="red")
```

`vol.regrid` *Transform the grid of a volume class object into the grid of another*

Description

The `vol.regrid` function transforms the grid of a volume according to the grid of another.

Usage

```
vol.regrid(
  vol,
  back.vol,
  T.MAT = NULL,
  interpolate = TRUE,
  alias = "",
  description = NULL,
  verbose = TRUE
)
```

Arguments

`vol`	"volume" class object to regrid.
`back.vol`	"volume" class object whose grid will be used for regriding. Its `$ref.pseudo` must exist in the `T.MAT` list.
`T.MAT`	"t.mat" class object, created by load.patient.from.dicom, load.patient.from.Rdcm, load.T.MAT or ref.add. If `T.MAT = NULL`, `back.vol$ref.pseudo` must be equal to `vol$ref.pseudo`.
`interpolate`	Boolean, default to `TRUE`. If `interpolate = TRUE`, a trilinear interpolation of the value of the voxels, relative to the values of adjacent voxels, is performed.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object. If `description = NULL`, it will be that of `vol`.
`verbose`	Boolean. if `TRUE` (default) a progress bar is displayed.

Value

Returns a copy of `vol`, in which grid is that of `back.vol`.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
patient <- toy.load.patient (modality = c ("mr", "rtdose"),
                             dxyz = rep (step, 3), beam.nb = 4)
```

```
MR <- patient$mr[[1]]
D <- patient$rtdose[[1]]

# change grid
D.on.MR <- vol.regrid (vol = D, back.vol = MR, interpolate = TRUE,
                       T.MAT = patient$T.MAT, alias = "",
                       description = NULL, verbose = FALSE)

# maximum dose location
max.dose.in.MR <- get.xyz.from.index (which.max (D.on.MR$vol3D.data), D.on.MR)
display.plane (bottom = MR, view.coord = max.dose.in.MR[3],
               top= D.on.MR, bottom.col = grey.colors(255, start = 0, end = 1),
               bottom.breaks = seq (0, 500, length.out = 256),
               bg = "#00ff00", interpolate = FALSE)
```

`vol.repair` *repairing missing planes of volumes*

Description

The `vol.repair` function repairs missing planes in volumes.

Usage

```
vol.repair(vol, alias = "", description = NULL)
```

Arguments

`vol`	"volume" class object.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `paste (vol$object.alias, "repair")`.

Details

Missing planes at download can generate errors or unpredictible results in espadon processing. The `vol.repair` function detects such missing planes and recreates their value by interpolation.

Value

Returns a "volume" class object (see espadon.class for class definitions), with no missing plane, if `vol` is to be repaired. Returns `vol` otherwise.

Examples

```
step <- 4
patient <- toy.load.patient (modality = c("ct", "mr", "rtstruct", "rtdose"),
                             roi.name  = "",
                             dxyz = rep (step, 3), beam.nb = 3)
CT <- patient$ct[[1]]

# this function removes a plane in a volume in order to simulate
# a dicom transfer issue
```

```
remove.plane <- function (vol, k) {
  idx <- which (vol$k.idx == k)
  vol$n.ijk[3] <- vol$n.ijk[3] - 1
  vol$xyz0 <- vol$xyz0[-idx, ]
  vol$k.idx <- vol$k.idx[-idx]
  vol$missing.k.idx <- TRUE
  vol$vol3D.data <- vol$vol3D.data[, , -idx]
  return (vol)
}

# Creation of CT.damaged without the 29th slice.
CT.damaged<- remove.plane (CT, 29)
CT.fix <- vol.repair (CT.damaged)

# Display
par (mfrow=c(3, 3))
for (k in 28:30) {
display.kplane (CT, k, main = paste("CT @ k =",k),interpolate = FALSE)
display.kplane (CT.damaged, k, main = "damaged CT",interpolate = FALSE)
display.kplane (CT.fix, k, main = "fixed CT", interpolate = FALSE)
}
```

`vol.subsampling` *Subsampling a volume*

Description

The `vol.subsampling` function sub-samples the grid of a "volume" class object.

Usage

```
vol.subsampling(
  vol,
  fact.ijk = 2,
  interpolate = TRUE,
  alias = "",
  description = NULL
)
```

Arguments

`vol`	"volume" class object.
`fact.ijk`	Strictly positive integer, or a vector of 3 strictly positive integers.
`interpolate`	Boolean, default to `TRUE`. If `interpolate = TRUE`, a trilinear interpolation of the value of the voxels, relative to the values of adjacent voxels, is performed.
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the the created object. If `description = NULL`, it will be `paste ("subsampling" ,vol$description)`.

Value

Returns a "volume" class object, in which 3D volume grid is subsampled: the voxel size is multiplied by `fact.ijk` and the center location of the volume is invariant.

See Also

vol.oversampling.

Examples

```
vol <- vol.create(n.ijk = c(10,10,1),dxyz = c(2,2,2), ref.pseudo = "ref1",
                  modality ="test", pt000 = c(0,0,0))
vol$vol3D.data[] <- array(1:prod(vol$n.ijk), dim = vol$n.ijk)
vol$max.pixel <- prod(vol$n.ijk)
vol$min.pixel <- 1
mid <- as.numeric (apply (get.extreme.pt (vol), 1, mean))
vol_ss <- vol.subsampling (vol, fact.ijk= 2)
mid_ss <- as.numeric (apply (get.extreme.pt (vol_ss), 1, mean))
display.plane(vol,interpolate = FALSE, view.coord = mid[3],
              abs.rng = c(-5,25), ord.rng = c(-5,25), bg="green")
points (mid[1], mid[2], pch=16, col="red")
display.plane(vol_ss,interpolate = FALSE, view.coord = mid_ss[3],
              abs.rng = c(-5,25), ord.rng = c(-5,25), bg="green")
points (mid_ss[1], mid_ss[2], pch=16, col="red")
```

`vol.sum` *Sum of 2 volumes*

Description

The `vol.sum` function adds two "volume" class objects of the same grid and of the same modality.

Usage

```
vol.sum(vol1, vol2, alias = "", description = NULL)
```

Arguments

`vol1, vol2`	"volume" class objects. The 2 volumes must have the same modality, and the same grid (i.e. share the same position of the voxels).
`alias`	Character string, `$alias` of the created object.
`description`	Character string, describing the created object. If `description = NULL` (default value), it will be set to `paste (vol1$object.alias, "+", vol2$object.alias)`.

Value

Returns a "volume" class object (see espadon.class for class definitions), with the same grid and modality as `vol1` and `vol2`, sum of `vol1` and `vol2`.

Examples

```
# loading of toy-patient objects (decrease dxyz and increase beam.nb for
# better result)
step <- 5
pat<- toy.load.patient (modality = c ( "rtdose"), dxyz = rep (step, 3),
                        beam.nb = 3)
```

```
# Double dose
D <- vol.sum (pat$rtdose[[1]], pat$rtdose[[1]])
pat$rtdose[[1]]$max.pixel
D$max.pixel
```

`xlsx.from.dcm` *Converting DICOM files to .xlsx files*

Description

The `xlsx.from.dcm` function creates an Excel file from DICOM files.

Usage

```
xlsx.from.dcm(
  dcm.filenames,
  xlsx.filenames,
  multipage = TRUE,
  txt.sep = "\\",
  txt.length = 100,
  tag.dictionary = dicom.tag.dictionary()
)
```

Arguments

`dcm.filenames`	String vector, representing the list of full names of DICOM files.
`xlsx.filenames`	String vector, representing the list of full names of created *.xlsx files. If `multipage = TRUE`, only the `xlsx.filenames[1]` is used.
`multipage`	Boolean. If TRUE, all `dcm.filenames` are converted into multiple pages of the same *.xlsx file.
`txt.sep`	String. Used if `as.txt = TRUE`. Separator of the tag value elements.
`txt.length`	Positive integer. Used if `as.txt = TRUE`. Maximum number of letters in the representation of the TAG value.
`tag.dictionary`	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.

Value

Returns a boolean vector, establishing the existence of the created Excel files.

Examples

```
# First, save toy.dicom.raw () raw data to a temporary file pat.dir for testing.
pat.dir <- file.path (tempdir(), "PM_dcm")
dir.create (pat.dir, recursive = TRUE)
dcm.filename <- tempfile (pattern = "PMrtplan", tmpdir = pat.dir, fileext = ".dcm")
zz <- file (dcm.filename, "wb")
writeBin (toy.dicom.raw (), zz, size = 1)
close (zz)
list.files (pat.dir)
```

```
# Creating an Excel file
xlsx.fnames <- file.path (pat.dir,
                          paste (basename (dcm.filename),"xlsx", sep = "."))
xlsx.from.dcm (dcm.filename, xlsx.fnames)
list.files (pat.dir)

# Cleaning temporary directory
unlink (pat.dir, recursive = TRUE)
```

`xlsx.from.Rdcm` *Converting .Rdcm files to .xlsx files*

Description

A *.Rdcm file contains the list of contents, in dataframe form, of the DICOM files of the same object. The `xlsx.from.Rdcm` function creates, from a *.Rdcm file, an Excel file, in which each page contains the dataframe representation of a DICOM file of the same object.

Usage

```
xlsx.from.Rdcm(
  Rdcm.filenames,
  dest.dirname = dirname(Rdcm.filenames),
  txt.sep = "\\",
  txt.length = 100,
  tag.dictionary = dicom.tag.dictionary()
)
```

Arguments

`Rdcm.filenames`	String vector, representing the *.Rdcm filenames to be converted.
`dest.dirname`	String vector of the same length as `Rdcm.filenames`, indicating the directory where the *.xlsx files will be created.
`txt.sep`	String. Used if `as.txt` = `TRUE`. Separator of the tag value elements.
`txt.length`	Positive integer. Used if `as.txt` = `TRUE`. Maximum number of letters in the representation of the TAG value.
`tag.dictionary`	Dataframe, by default equal to dicom.tag.dictionary, whose structure it must keep. This dataframe is used to parse DICOM files.

Value

Returns a boolean vector, establishing the existence of the created Excel files which have the same basenames as the *.Rdcm files.

Examples

```
# First, create a Rdcm file from toy.dicom.raw () to a temporary file for testing.
pat.dir <- file.path (tempdir(), "PM_Rdcm")
dir.create (pat.dir, recursive = TRUE)
dcm.filename <- tempfile (pattern = "PM_rtplan", tmpdir = pat.dir, fileext = ".dcm")
zz <- file (dcm.filename, "wb")
```

```
writeBin (toy.dicom.raw (), zz, size = 1)
close (zz)
dicom.to.Rdcm.converter (dcm.filename, pat.dir, update = TRUE)
file.remove (dcm.filename)
list.files (pat.dir)

# Creating an Excel file
Rdcm.filenames <- list.files (pat.dir, pattern = "[.]Rdcm$",
                              recursive = TRUE, full.names = TRUE)
xlsx.from.Rdcm (Rdcm.filenames)
list.files (pat.dir)

# Cleaning temporary directory
unlink (pat.dir, recursive = TRUE)
```

Index