

# **Growth RTV**

# **Python API Reference**

V2.2.1J

2022/01/06

e-Growth Co., Ltd.

## 目次

<b>1. 概要</b> .....	<b>1</b>
<b>2. はじめに</b> .....	<b>2</b>
<b>3. API リファレンス(基本関数)</b> .....	<b>3</b>
3.1. readRTDir .....	3
3.2. getImgSize .....	3
3.3. getImgSpacing .....	3
3.4. getImageDcmTags .....	4
3.5. getImageParam .....	4
3.6. readGRTVRaw .....	5
3.7. getROIList .....	5
<b>4. API リファレンス(関心領域関連)</b> .....	<b>6</b>
4.1. setNewROI .....	6
4.2. predNewROI .....	7
4.3. exportROIMesh .....	8
4.4. interpROI .....	8
4.5. ROIDilation .....	9
4.6. ROIErosion .....	9
<b>5. API リファレンス(線量分布関連)</b> .....	<b>10</b>
5.1. setDose .....	10
5.2. exportDVHCsv .....	10
<b>6. API リファレンス(3D 画像処理関連)</b> .....	<b>11</b>
6.1. replaceCurrentImage .....	11
6.2. augmentImg3D .....	11
6.3. resizeImage3D .....	12
6.4. RegionGrow .....	13
6.5. deformCurrentImageByMesh .....	13
<b>7. API リファレンス(DICOM 処理関連)</b> .....	<b>14</b>
7.1. exportAsNewDcm .....	14
7.2. exportRT .....	15
7.3. imageRawToDicom .....	15
<b>8. API リファレンス(2D 画像関連)</b> .....	<b>16</b>
8.1. getModality2DNum .....	16
8.2. getModality2DList .....	16
8.3. getModality2DPixel .....	16

<b>9. API リファレンス(ユーティリティ).....</b>	<b>17</b>
9.1. readRaw .....	17
9.2. writeRaw .....	17
9.3. writeNrrd.....	18
9.4. closeGRTVTFSession.....	18
<b>10. サンプルコード .....</b>	<b>20</b>
10.1. サンプル 1 – 様々な API 呼び出し .....	20
10.2. サンプル 2 (ROI 定義および RT 化の一括処理) .....	22
10.3. サンプル 3(深層学習用 3D Raw の一括抽出).....	24
10.4. サンプル 4(深層学習用 3D ボリュームデータの拡張) .....	27
10.5. サンプル 5(DICOM IMAGE のピクセル置換および出力).....	28
10.6. サンプル 6(3D Raw の DICOM 変換および出力).....	29

# 1. 概要

Growth RTV では Python 連携プラットフォームを搭載し、ユーザは Growth RTV にバンドルされている連携用モジュール(`grtv`)をインポートするだけで、`pyGRTV` で提供されている様々な API を利用してデータ読み込みから、3D データの抽出・加工を簡便に行えます。本仕様書は、`pyGRTV` の API 仕様についてまとめたものです。

## 2. はじめに

まず、Growth RTV のデータへアクセスを行うためには、まず `grtv` モジュールをインポートし、`pyGRTV` クラスのインスタンス生成を行います。通常、`grtv` モジュールは「C:\Program Files\GrowthRTV\pyGRTV\」以下にインストールされています。インスタンス生成と同時に、`pyGRTV` は Growth RTV から接続情報を受取、初期化を行います。

例:

```
import sys
```

```
import numpy as np
```

```
sys.path.append("C:/Program Files/GrowthRTV/pyGRTV")
```

```
from grtv import *
```

```
# GRTV インスタンス生成
```

```
grtv = pyGRTV()
```

インスタンス生成時に接続エラーメッセージが表示されなければ、インスタンス生成処理は成功です。Growth RTV プロセスが起動されていない、またはすでに他のプロセスによって `pyGRTV` のインスタンスが生成されている場合、エラーメッセージが表示され、Python プロセスは終了します。

## 3. API リファレンス(基本関数)

### 3.1. readRTDir

readRTDir( rtDirPath, UseCache=True )

DICOM(CT / MRI /RT)データの読み込みを行います。

引数

- rtDirPath : DICOM ファイル群が直接保存されているフォルダのパス
- UseCache: ユーザの編集状態を読み込むかを指定。False に指定した場合、ユーザが前回までに編集した情報をすべて破棄し、新規データとして読み込みます。

戻り値

- status
  - GRTV\_STATUS\_OK : 読み込み成功
  - GRTV\_STATUS\_FALIED : 読み込み失敗

### 3.2. getImgSize

getImgSize()

読み込んだ 3D イメージのサイズ情報を取得します。

引数

なし

戻り値

- (size\_x, size\_y, size\_z): X, Y, Z の各 3 次元方向のボクセル数

### 3.3. getImgSpacing

getImgSpacing ()

読み込んだ 3D イメージのボクセル実寸情報(mm)を取得します。

引数

なし

戻り値

- (res\_x, res\_y, res\_z): X, Y, Z の各 3 次元方向のボクセル実寸(mm)

### 3.4. getImageDcmTags

getImageDcmTags( dcmTagsMap )

DICOM イメージデータのタグ(イメージボリュームを構成する 1 枚目のスライス)を読み込みます。

引数

- dcmTagsMap : map 型のコンテナ。取得した DICOM タグはこのコンテナ内に読み込まれます。

戻り値

- status
  - GRTV\_STATUS\_OK : 読み込み成功
  - GRTV\_STATUS\_FALIED : 読み込み失敗

### 3.5. getImageParam

getImageParam(paramID, paramMap)

※本関数は GetImgSize および GetImgSpacing の導入により、将来的に廃止予定です

現在の画像のサイズ情報を取得します。

引数:

- paramID : 取得したいパラメータの ID。
  - GRTV\_PARAM\_IMAGE\_SIZE : X, Y, Z の各 3 次元方向のボクセル数
  - GRTV\_PARAM\_IMAGE\_RES : X, Y, Z の各 3 次元方向のボクセル実寸(mm)
- paramMap : map 型のコンテナ。取得したパラメータ値が格納されます。

戻り値

- status
  - GRTV\_STATUS\_OK : 読み込み成功
  - GRTV\_STATUS\_FALIED : 読み込み失敗
- 

### 3.6. readGRTVRaw

readGRTVRaw( grtvDataType, dataID=-1 )

画像データの3次元ボリュームデータを3次元 numpy 型として取得します。

引数

- grtvDataType: 取得したい画像データタイプの ID。
  - GRTV\_DATA\_TYPE\_IMAGE : CT または MRI のピクセル(輝度値)データ
  - GRTV\_DATA\_TYPE\_DOSE : 線量分布のピクセル(輝度値)データ。Growth RTV では線量データを CT データまたは MRI データと同じ3次元サイズに線形補間したデータを返します。
  - GRTV\_DATA\_TYPE\_ROI : Structure(関心領域)の3D領域データ。Growth RTV では RT Structure のコンター情報から CT または MRI データと同じ3次元サイズのボリュームに変換して返します。3D領域データは0(領域外)と255(領域内の値を持つ。
- dataID: grtvDataTypeにGRTV\_DATA\_TYPE\_ROIを指定した場合のみこの引数を利用します。dataIDはStructureのインデックスを指定します。

戻り値

- status
  - GRTV\_STATUS\_OK : 読み込み成功
  - GRTV\_STATUS\_FALIED : 読み込み失敗
- rawData : 取得された3次元 numpy 型のボリュームデータ

### 3.7. getROIList

getROIList( roiList )

定義済み Structure をリスト型として取得します。

引数

- roiList: リスト型のコンテナ。定義済み Structure のリストが map 型として roiList 内に格納されます。各 Structure の map 内に下記の情報が格納されます。
  - roi\_name : Structure 名
  - roi\_index : Structure のインデックス。readGRTVRaw API などに利用します。

戻り値

- status
  - GRTV\_STATUS\_OK : 読み込み成功
  - GRTV\_STATUS\_FALIED : 読み込み失敗

## 4. API リファレンス(関心領域関連)

### 4.1. setNewROI

setNewROI( roiRaw, roiName, roiColor )

新たな Structure を現在の画像内に追加定義します。追加定義される Structure は領域情報 (roiRaw)に従い、体積、3D メッシュおよび 2D コンターが自動計算され、Dose(線量分布)がある場合は DVH も算出・表示されます。

引数

- roiRaw: 0~255 を持つ領域情報。CT または MRI の 3 次元データと同じサイズの 3 次元 Numpy 配列である必要があります。
- roiName: Structure 名
- roiColor: Structure の表示色。0~255 の整数型で。要素長が 3(R, G, B)のリスト型または Numpy 配列である必要があります。

戻り値

- status
  - GRTV\_STATUS\_OK : 読み込み成功
  - GRTV\_STATUS\_FALIED : 読み込み失敗

## 4.2. predNewROI

predNewROI( roiName, roiColor )

Growth RTV の臓器自動抽出機能を利用して Structure を追加定義します。追加定義される Structure は体積、3D メッシュおよび 2D コンターが自動計算され、Dose(線量分布)がある場合は DVH も算出・表示されます。なお、Growth RTV の AI Model Location を[Local]に設定してこの API を利用する場合、自動抽出時は GPU 版の TensorFlow が呼び出されるため、GPU メモリの競合が発生しないよう、ユーザ側の Python スクリプトでは GPU 版の TensorFlow を起動しないことを推奨します。ユーザ側で GPU メモリを利用する場合は、本 API の呼び出し後に closeGRTVTFSession 関数で Growth RTV 側の GPU セッションを終了してください。AI Model Location を[Cloud]に設定されている場合は特に制限はありません。

引数

- roiName: Growth RTV がサポートする臓器名を指定します。現在は以下になります。
  - body: 体表
  - Mandibular: 下顎骨
  - llung: 左肺野
  - rlung: 右肺野
  - spinal\_cord: 脊柱管
  - liver: 肝臓
  - lkidney: 左腎臓
  - rkidney: 右腎臓
  - stomach: 胃
- roiColor: Structure の表示色。0~255 の整数型で。要素長が 3(R, G, B)のリスト型または Numpy 配列であることが必要です。

戻り値

- status
  - GRTV\_STATUS\_OK: 読み込み成功
  - GRTV\_STATUS\_FALIED: 読み込み失敗

### 4.3. exportROIMesh

`exportROIMesh( dataID, meshPath, meshType=GRTV_MESH_TYPE_STL )`

Structure を STL または PLY 形式でファイル出力します。

引数

- `dataID` : Structure のインデックス
- `meshPath` : メッシュの出力ファイルパス
- `meshType` : 出力形式。
  - `GRTV_MESH_TYPE_STL` : STL 形式
  - `GRTV_MESH_TYPE_PLY` : PLY 形式

戻り値

- `status`
  - `GRTV_STATUS_OK` : 読み込み成功
  - `GRTV_STATUS_FALIED` : 読み込み失敗

### 4.4. interpROI

`interpROI(self, roiRaw)`

スライス間隔を空けて定義された領域情報(`roiRAW`)を線形補間した結果を取得します。

引数

- `roiRaw` : 0~255 を持つ領域情報。CT または MRI の 3 次元データと同じサイズの 3 次元 Numpy 配列であることが必要です。

戻り値

- `status`
  - `GRTV_STATUS_OK` : 読み込み成功
  - `GRTV_STATUS_FALIED` : 読み込み失敗
- `rawData` : 取得された 3 次元 numpy 型のボリュームデータ

## 4.5. ROI Dilation

ROIDilation(roiRaw, ope\_range, rangeType=GRTV\_RANGE\_TYPE\_VOXEL, spacing=(1.0, 1.0, 1.0))

3D 領域データに対して膨張処理を行います

引数

- roiRaw: 処理対象元の Numpy 型の 3D 領域データ。データタイプは Uint8 型が必須です。
- ope\_range: 膨張半径の値。
- rangeType: 膨張半径の単位
  - GRTV\_RANGE\_TYPE\_VOXEL: ボクセル数
  - GRTV\_RANGE\_TYPE\_MM: 実寸値(mm)
- spacing: 処理対象元の 3D 領域データのボクセル実寸値。(x, y, z)のタプル型。

戻り値

- status
  - GRTV\_STATUS\_OK: 処理成功
  - GRTV\_STATUS\_FALIED: 処理失敗
- rawData: 処理結果を示す numpy 型の 3D 領域データ。データ型は unsigned char(Uint8) 型であり、3D 領域データは 0(領域外)と 255(領域内の値を持つ)。

## 4.6. ROI Erosion

ROI Erosion (roiRaw, ope\_range, rangeType=GRTV\_RANGE\_TYPE\_VOXEL, spacing=(1.0, 1.0, 1.0))

3D 領域データに対して縮小処理を行います

引数

- roiRaw: 処理対象元の Numpy 型の 3D 領域データ。データタイプは Uint8 型が必須です。
- ope\_range: 縮小半径の値。
- rangeType: 縮小半径の単位
  - GRTV\_RANGE\_TYPE\_VOXEL: ボクセル数
  - GRTV\_RANGE\_TYPE\_MM: 実寸値(mm)
- spacing: 処理対象元の 3D 領域データのボクセル実寸値。(x, y, z)のタプル型。

戻り値

- status
  - GRTV\_STATUS\_OK : 処理成功
  - GRTV\_STATUS\_FALIED : 処理失敗
- rawData : 処理結果を示す numpy 型の 3D 領域データ。データ型は unsigned char(UInt8) 型であり、3D 領域データは 0(領域外)と 255(領域内の値を持つ)。

## 5. API リファレンス(線量分布関連)

### 5.1. setDose

setDose( doseRaw)

現在の画像内の Dose(線量分布)を変更します。変更された線量分布(doseRaw)に従い、DVH も再計算されます。

引数

- doseRaw: 0~65,535 を持つ cGY(センチグレイ)の領域情報。CT または MRI の 3 次元データと同じサイズの 3 次元 Numpy 配列であることが必要です。

戻り値

- status
  - GRTV\_STATUS\_OK : 設定成功
  - GRTV\_STATUS\_FALIED : 設定失敗

### 5.2. exportDVHCsv

exportDVHCsv(dvhCsvPath)

現在の画像内の DVH 情報を CSV データとしてエクスポートします。

引数

- `dvhCsvPath`: CSV ファイルの出力ファイルパス

戻り値

- `status`
  - `GRTV_STATUS_OK`: 読み込み成功
  - `GRTV_STATUS_FALIED`: 読み込み失敗

## 6. API リファレンス(3D 画像処理関連)

### 6.1. `replaceCurrentImage`

```
replaceCurrentImage(img3d_in, exportDcmDir="")
```

現在の読み込まれている DICOM IMAGE のピクセルデータを置き換え、新たなシリーズの DICOM 形式としてのファイル出力を行います。

引数

- `img3d_in`: 置き換える Numpy 型の 3 次元配列。現在読み込まれている CT または MRI の 3 次元データと同じサイズの 3 次元 Numpy 配列であることが必要です。
- `exportDcmDir`: 出力先のディレクトリパス。指定しない場合はピクセル値の置き換え処理のみが行われます。指定されている場合、現在読み込まれている DICOM IMAGE に対し、新たなシリーズとして出力されます(新たな `Series Instance UID` が付与されます)。

戻り値

- `status`
  - `GRTV_STATUS_OK`: 処理成功
  - `GRTV_STATUS_FALIED`: 処理失敗

### 6.2. `augmentImg3D`

```
augmentImg3D(imageList, outputDirectory, augmentTimes=5, fileNameList=[],  
rotate=True, rotateAngRange=2.5, deform=False, deformRate=0.25)
```

3D ボリュームデータに対し、回転および局所変形によるランダムデータ拡張を行います。

## 引数

- **imageList**: 拡張元の Numpy 型の 3 次元配列のリスト。本リストに格納された 3 次元配列は整合性を維持しながらデータ拡張されます。  
リスト格納例(CT および臓器領域の同時拡張): [img\_CT, img\_ROI1, img\_ROI2]
- **outputDirectory**: 拡張データの出力フォルダ:
- **augmentTimes**: 拡張回数。
- **fileNameList**: imageList に格納された各 3 次元配列に対し、出力ファイル名を設定可能です。fileNameList を設定済みの場合、指定ファイル名をプレフィックスとして使用して拡張データを出力します。  
例: fileName.0001.raw, fileName1.0002.raw, ...  
fileNameList 未設定の場合、img\_000N (N は imageList 内の Index)が設定されます。  
例: img\_0000.0001.raw, img\_0000.0002.raw, ...
- **rotate**: データ拡張時の回転有無を指定します。本オプションは deform と併用可能です。
- **rotateAngRange**: 最大回転角を指定します。拡張元 3D ボリュームデータは指定回転角範囲内において、X、Y、Z 軸中心にランダム回転してデータ拡張されます。
- **deform**: データ拡張時の変形有無を指定します。本オプションは rotate と併用可能です。
- **deformRate**: 最大変形量を指定します。3D ボリュームデータは、-1.0~1.0 の範囲の 3 次元空間内において、上記最大変形量の範囲内でランダム変形してデータ拡張されます。

## 戻り値

- **status**
  - GRTV\_STATUS\_OK : 処理成功
  - GRTV\_STATUS\_FALIED : 処理失敗

## 6.3. resizeImage3D

resizeImage3D( img3d\_in, newShape, dataType=np.int16 )

3D ボリュームデータを線形補間します。

## 引数

- **img3d\_in**: 線形補間元の Numpy 型の 3 次元配列
- **newShape**: 線形補間先の 3 次元サイズ(タプル型)
- **dataType**: ボリュームデータの Numpy のデータ型。現在は uint8, int16, uint16 がサポートされています。

戻り値

- status
  - GRTV\_STATUS\_OK : 読み込み成功
  - GRTV\_STATUS\_FALIED : 読み込み失敗
- rawData : 線形補間後の 3 次元 numpy 型のボリュームデータ

## 6.4. RegionGrow

RegionGrow(img3d\_in, region\_th=(128,255), seedPos=(0,0,0))

領域拡張(リージョングロウイング)法によって指定点から拡張された 3 次元領域を取得します。

引数

- img3d\_in: 領域拡張対象元の Numpy 型の 3 次元配列
- region\_th: 領域拡張対象画素値範囲。(最小値, 最大値)のタプル型。
- seedPos: 領域拡張開始点のボクセル座標(z, y, x)。

戻り値

- status
  - GRTV\_STATUS\_OK : 処理成功
  - GRTV\_STATUS\_FALIED : 処理失敗
- rawData : 対象領域を示す numpy 型の 3D 領域データ。データ型は unsigned char(UInt8) 型であり、3D 領域データは 0(領域外)と 255(領域内の値を持つ)。

## 6.5. deformCurrentImageByMesh

deformCurrentImageByMesh( srcPlyList, dstPlyList, deform\_resolution )

変形前後の表面メッシュ群を指定して、現在読み込みしている画像データを 3 次元変形します。

引数

- srcPlyList: 変形前の表面メッシュファイル(PLY 形式)のリスト。  
リスト格納例: ["D:/src\_body.ply", "D:/src\_lung.ply", "D:/src\_liver.ply"]
- dstPlyList: 変形後の表面メッシュファイル(PLY 形式)のリスト。srcPlyList とは PLY の順序が必要であり、かつ変形前後の各 PLY 同士の頂点对応が必要です。
- resolution: 変形計算精度を指定します。精度が高いほど、計算時間を要します。また、計算時間は PLY の頂点数および読み込み中の画像データのボクセル数に依存します。
  - GRTV\_DEFORM\_RESOLUTION\_HIGH: 最高精度
  - GRTV\_DEFORM\_RESOLUTION\_MIDDLE: 高精度
  - GRTV\_DEFORM\_RESOLUTION\_STANDARD: 標準精度
  - GRTV\_DEFORM\_RESOLUTION\_LOW: 低精度

戻り値

- status
  - GRTV\_STATUS\_OK: 処理成功
  - GRTV\_STATUS\_FALIED: 処理失敗

## 7. API リファレンス(DICOM 処理関連)

### 7.1. exportAsNewDcm

exportAsNewDcm (exportDcmDir)

現在の読み込まれている DICOM IMAGE を新たなシリーズの DICOM 形式としてのファイル出力を行います。

引数

- exportDcmDir: 出力先のディレクトリパス。現在読み込まれている DICOM IMAGE に対し、新たなシリーズとして出力されます(新たな Series Instance UID が付与されます)。

戻り値

- status
  - GRTV\_STATUS\_OK: 処理成功
  - GRTV\_STATUS\_FALIED: 処理失敗

## 7.2. exportRT

exportRT( exportDir )

現在のデータを新たな DICOM-RT 形式としてファイル出力します。

引数

- exportDir: 出力先のディレクトリパス
- meshPath: メッシュの出力ファイルパス
- meshType: 出力形式。
  - GRTV\_MESH\_TYPE\_STL: STL 形式
  - GRTV\_MESH\_TYPE\_PLY: PLY 形式

戻り値

- status
  - GRTV\_STATUS\_OK: 読み込み成功
  - GRTV\_STATUS\_FALIED: 読み込み失敗

## 7.3. imageRawToDicom

imageRawToDicom(img3d\_in, res\_z, res\_y, res\_x, exportDcmDir):

3D ボリュームデータを DICOM 形式へ変換出力を行います。

引数

- img3d\_in: 変換元の Numpy 型の 3 次元配列。int8, uint8, int16 の場合は 2 バイト長の CT データ, uint16 の場合は 2 バイト長の MRI データとして DICOM 形式に変換されます。
- res\_z: 3D ボリュームデータの Z 方向のスライス間隔
- res\_y: 3D ボリュームデータの Y 方向のピクセル間隔
- res\_x: 3D ボリュームデータの X 方向のピクセル間隔
- exportDcmDir: 出力先のディレクトリパス。現在読み込まれている DICOM IMAGE に対し、新たなシリーズとして出力されます(新たな Series Instance UID が付与されます)。

戻り値

- status
  - GRTV\_STATUS\_OK : 処理成功
  - GRTV\_STATUS\_FALIED : 処理失敗

## 8. API リファレンス(2D 画像関連)

### 8.1. getModality2DNum

getModality2DNum()

読み込んだデータ内に含まれる 2D 画像数を取得します

引数

- なし

戻り値

- imgNum: 2D 画像数。

### 8.2. getModality2DList

getModality2DList()

読み込んだデータ内に含まれる 2D 画像群の DICOM タグ情報を取得します。

引数

- なし

戻り値

- dcmList: 2D 画像群の各画像の DICOM タグが格納された map 型のコンテナリスト。

### 8.3. getModality2DPixel

getModality2DPixel(dcmList, iData)

getModality2DList で取得されたリスト内の任意の画像のピクセル情報を取得します。

引数

- `dcmList`: `getModality2DList` で取得した `map` 型のコンテナリスト
- `iData`: ピクセル情報を取得したい画像の(`dcmList` 内における)インデックス

戻り値

- `status`
  - `GRTV_STATUS_OK`: 処理成功
  - `GRTV_STATUS_FALIED`: 処理失敗
- `pixel`: 画像のピクセル値を示す 2 次元 `numpy` 配列。データ型は DICOM タグの定義 (Pixel Representation. 0028,0031)に従う。

## 9. API リファレンス(ユーティリティ)

### 9.1. readRaw

```
readRaw(self, raw_path, dataType=np.int16)
```

`raw` ファイルデータの読み込み

引数

- `raw_path`: 読み込み対象となる `raw` データのファイルパス
- `dataType`: 読み込み対象となる `raw` データのデータ型

戻り値

- `data`: `raw` ファイルデータを示す 1 次元の `numpy` 配列。

### 9.2. writeRaw

```
writeRaw(raw_path, data)
```

`numpy` 配列の `raw` ファイルデータ出力。

引数

- `raw_path`: `raw` データの出力先ファイルパス

- data: 出力対象となる numpy 配列

戻り値

- なし

### 9.3. writeNrrd

writeNrrd(raw\_path, img3D, spacing, contentType)

3次元の numpy 配列の raw ファイルデータ出力(NRRD v1 ヘッダー付き)。

引数

- raw\_path: raw データの出力先ファイルパス。当該パスと同じフォルダに nrrd ヘッダーファイルが出力されます。
- img3D: 出力対象となる numpy 配列
- spacing: img3D のボクセル実寸情報(mm)
- contentType: “CT”等のコンテンツの説明情報

戻り値

- nrrdPath: 出力された nrrd ファイルのパス

### 9.4. closeGRTVTFSession

closeGRTVTFSession()

※本 API は AI Location を Local に設定した場合にのみ必要です

Growth RTV 側の臓器の自動抽出用の TensorFlow セッションを終了します。ユーザ側で TensorFlow を実行したい場合、この API によって先に Growth RTV 側のセッションを終了し、GPU メモリを開放することを推奨します。

引数

- なし

戻り値

- status
  - GRTV\_STATUS\_OK : 読み込み成功
  - GRTV\_STATUS\_FALIED : 読み込み失敗

## 10. サンプルコード

### 10.1. サンプル 1 – 様々な API 呼び出し

このサンプルでは様々な API の呼び出し例を示します。

```
import numpy as np
from time import sleep
import zmq
import os, sys, glob, shutil
import psutil
import numpy as np
import re, gc
import cv2
sys.path.append("C:/Program Files/GrowthRTV/pyGRTV")
from grtv import *

if __name__ == "__main__":

    # GRTV インスタンス生成
    grtv = pyGRTV()
    print(grtv.pyIni)
    print(grtv.userPyCache)

    # DICOM-RT 読み込み
    rtDir = "c:/testdata/"
    ret = grtv.readRTDir(rtDir)

    # 画像 DICOM タグ取得サンプル
    dcmTagsMap = {}
    ret = grtv.getImageDcmTags(dcmTagsMap)
    print(dcmTagsMap)

    # 現在の画像のサイズ情報取得
    paramMap = {}
```

```

ret = grtv.getImageParam(grtv.GRTV_PARAM_IMAGE_SIZE, paramMap)
print(ret)

#現在の画像のボクセルサイズ取得
ret = grtv.getImageParam(grtv.GRTV_PARAM_IMAGE_RES, paramMap)
print(paramMap)

# CT RAW ボリューム取得サンプルコード
ret, img_short = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_IMAGE)

# Dose ボリューム取得サンプルコード
ret, dose_short = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_DOSE)

# ROI リスト取得
roiList = []
ret = grtv.getROIList(roiList)
print(roiList)

# 中間スライス表示サンプルコード
imgShape = img_short.shape
slicePos = int(imgShape[0] / 2)
slice = img_short[slicePos, :, :]
gray = grtv.getSliceByLut(slice, wl=0, ww=2000)
cv2.imshow("slice", gray)
cv2.waitKey(0)

### roi Mesh Export サンプル
if False:
    meshType = grtv.GRTV_MESH_TYPE_PLY
    extMap = {grtv.GRTV_MESH_TYPE_STL:".stl",
grtv.GRTV_MESH_TYPE_PLY:".ply"}
    meshDir = "d:/dltest/tmp/mesh"
    if not os.path.isdir(meshDir):
        os.makedirs(meshDir)
    for roi in roiList:
        meshPath = "%s/%s%s" % (meshDir, roi["roi_name"],

```

```
extMap[meshType])
    ret = grtv.exportROIMesh(dataID=int(roi["roi_index"]),
meshPath=meshPath, meshType=meshType)
    print(ret)
```

## 10.2. サンプル 2 (ROI 定義および RT 化の一括処理)

このサンプルでは、複数の CT データに対し、臓器抽出から DICOM-RT 化してエクスポートを一括処理する例を示します。

```
import os, sys, glob, shutil
sys.path.append("C:/Program Files/GrowthRTV/pyGRTV")
from grtv import *

if __name__ == "__main__":

    # GRTV インスタンス生成
    grtv = pyGRTV()

    # 複数症例に対する臓器抽出～RT エクスポートまでのサンプルコード
    dataRoot = "c:/data/CTSets/"
    expDataRoot = "c:/data/export/"
    if not os.path.isdir(expDataRoot):
        os.makedirs(expDataRoot)

    dirList = os.listdir(dataRoot)
    exportedDirList = os.listdir(expDataRoot)
    for dirName in dirList:
        if dirName in exportedDirList or "export" == dirName:
            continue
        print("dirName", dirName)

    # データ読み込み
    dirPath = dataRoot + dirName
    ret = grtv.readRTDir(dirPath)
```

```

if ret == grtv.GRTV_STATUS_FAILED:
    continue

# ROI 自動抽出サンプルコード
roiList, definedRoiNames = [], []
ret = grtv.getROIList(roiList)
# for roi in roiList:
#     definedRoiNames.append(roi["roi_name"])
if ("rlung" not in definedRoiNames):
    ret = grtv.predNewROI("rlung", (0, 255, 0)) # right lung 自
動抽出
    print(ret)
if ("llung" not in definedRoiNames):
    ret = grtv.predNewROI("llung", (0, 0, 255)) # left lung 自
動抽出
    print(ret)

#### merge as rlung and llung as total lung
roiList = []
ret = grtv.getROIList(roiList)
print(roiList)
roiId1, roiId2 = -1, -1
for roi in roiList:
    if (roi["roi_name"] == "rlung"):
        roiId1 = int(roi["roi_index"])
    if (roi["roi_name"] == "llung"):
        roiId2 = int(roi["roi_index"])

if roiId1 < 0 or roiId2 < 0:
    continue
print("read rlung")
ret, roi_uint8_LLung = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_ROI,
dataID=roiId1) # 自動抽出された right lung の領域を取得
print("read llung")
ret, roi_uint8_Heart = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_ROI,
dataID=roiId2) # 自動抽出された left lung の領域を取得

```

```

    # merge
    roi_new = roi_uint8_LLung.astype("int") +
roi_uint8_Heart.astype("int") # 領域をマージ
    roi_new = np.clip(roi_new, 0, 255).astype("uint8")
    # Set to GRTV
    print("set merged roi")
    ret = grtv.setNewROI(roi_new, "Lung", (255, 255, 0)) # マージ
された領域を Lung として定義

    # 新たな RT データとしてエクスポート
    print(ret, "export rt")
    exportDir = expDataRoot + dirName
    if not os.path.isdir(exportDir):
        os.makedirs(exportDir)
    ret = grtv.exportRT(exportDir)

```

### 10.3. サンプル 3(深層学習用 3D Raw の一括抽出)

このサンプルでは、複数症例に対する深層学習用 CT および ROI(教師データ)の 3D Raw の抽出～3D 補間によるダウンサイジング処理～ファイル出力を一括処理する例を示します。

```

import os, sys, glob, shutil
sys.path.append("C:/Program Files/GrowthRTV/pyGRTV")
from grtv import *
import csv

if __name__ == "__main__":

    grtv = pyGRTV() # GRTV インスタンス生成

    # 複数症例に対する深層学習用 CT および ROI (教師データ) の 3D Raw 出力サ
    ンプル
    # 本サンプルでは 3D Raw の出力の際に、3次元補完処理によるダウンサイ
    ジング処理の例を含む

```

```

dataRoot = "c:/data/RTSets/"
expDataRoot = "c:/data/raw3d/"
if not os.path.isdir(expDataRoot):
    os.makedirs(expDataRoot)

# データフォルダ一覧取得
dirList = os.listdir(dataRoot)
dirList.sort()

# 各データについて出力処理
for iData, dirName in enumerate(dirList):
    print("dirName", dirName)

    dirPath = dataRoot + os.sep + dirName
    ret = grtv.readRTDir(dirPath) # データ読み込み
    if ret == grtv.GRTV_STATUS_FALIED:
        continue

    print("read ct")
    ret, img_short = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_IMAGE)
# CT データ読み込み
    newShape = (img_short.shape[0], 256, 256)

    fInfoCsv = "%s/%04d_info.txt" % (expDataRoot, iData)
    with open(fInfoCsv, 'w') as f:
        writer = csv.writer(f, lineterminator='¥n')
        writer.writerow(list(newShape)) # サイズ情報のファイル
保存
        if False:
            # ボクセル実寸情報をファイルに保存する場合
            paramMap = {}
            ret = grtv.getImageParam(grtv.GRTV_PARAM_IMAGE_RES,
paramMap)

            res_z = float(paramMap["res_z"])
            res_y = float(paramMap["res_y"]) * img_short.shape[1] /
256

```

```

    res_x = float(paramMap["res_x"]) * img_short.shape[2] /
256
    writer.writerow([res_z, res_y, res_x])

print("resize ct")
# CTの3Dデータを(originalZ, 256, 256)にリサイズ
ret, img_short_resize = grtv.resizeImage3D(img_short, newShape,
dataType=np.int16)
resizeCTPath = "%s/%04d_ct.raw" % (expDataRoot, iData)
print("write resized ct")
grtv.writeRaw(resizeCTPath, img_short_resize,
dataType=np.int16)    # リサイズしたCT 3D Rawをファイル保存

# 深層学習用に各ROIの3D Rawを書き出し
# (3D線形補間によるリサイズ処理のサンプル含む)
roiList = []
ret = grtv.getROIList(roiList)
print(roiList)
for roi in roiList:
    print(roi["roi_index"], roi["roi_name"])
    # ROIの3D Rawを取得
    ret, roi_uint8 = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_ROI,
dataID=int(roi["roi_index"]))

    # (originalZ, 256, 256)にリサイズ
    ret, roi_uint8_resize = grtv.resizeImage3D(roi_uint8,
newShape, dataType=np.uint8)

    # リサイズしたROI 3D Rawをファイル保存
    resizeRoiPath = "%s/%04d_%s.raw" % (expDataRoot, iData,
roi["roi_name"])
    grtv.writeRaw(resizeRoiPath, roi_uint8_resize,
dataType=np.uint8)

```

## 10.4. サンプル 4(深層学習用 3D ボリュームデータの拡張)

このサンプルでは、深層学習用 CT および ROI(教師データ)の 3D Raw データ拡張処理する例を示します。拡張されたデータ群は raw 形式で指定フォルダ内に出力されます。本サンプルで読み込んでいる DICOM-RT データでは関心領域(ROI)として 0 番目に Body、5 番目に LLung が定義されており、CT とこれらの ROI の整合性を維持した上でデータ拡張する例を示します。

```
import sys
sys.path.append("C:/Program Files/GrowthRTV/pyGRTV")
from grtv import *

if __name__ == "__main__":

    # GRTV インスタンス生成
    grtv = pyGRTV()

    # データ読み込み
    rtDir = "d:/testdata"
    ret = grtv.readRTDir(rtDir)

    # CT RAW ボリューム取得
    print("read CT")
    ret, ct_short = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_IMAGE)

    # 0 番目の ROI (Body) の領域を取得
    print("read Body")
    ret, roi_uint8_Body = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_ROI,
dataID=0)

    # 5 番目の ROI (LLung) の領域を取得
    print("read LLung")
    ret, roi_uint8_LLung = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_ROI,
dataID=5)
```

*# 拡張データリスト (本リストのデータ群は整合性を維持してデータ拡張  
されます)*

```
imageList = [ct_short, roi_uint8_Body, roi_uint8_LLung]
```

*# 各データ名リスト*

```
fileNameList = ["ct", "body", "llung"]
```

```
print("augmentImg3D")
```

```
ret = grtv.augmentImg3D(imageList=imageList,  
                        outputDirectory="d:/augImgs",  
                        augmentTimes=5,      #5回拡張を実施  
                        fileNameList=fileNameList,  
                        rotate=True,        #回転有り  
                        rotateAngRange=3.0,  #最大回転角は3度  
                        deform=True,        #変形有り  
                        deformRate=0.05     #最大変形量は0.05  
                        )
```

## 10.5. サンプル 5(DICOM IMAGE のピクセル置換および出力)

このサンプルでは、読み込まれた CT データに対し、ピクセルの置き換え処理および出力処理する例を示します。

```
import os, sys, glob, shutil
```

```
sys.path.append("C:/Program Files/GrowthRTV/pyGRTV")
```

```
from grtv import *
```

```
if __name__ == "__main__":
```

*# GRTV インスタンス生成*

```
grtv = pyGRTV()
```

*# DICOM 読み込み*

```
rtDir = "d:/convTest/dcm0"
```

```

ret = grtv.readRTDir(rtDir)
print("read dicom", ret)

# CT RAW ボリューム取得
ret, img_short = grtv.readGRTVRaw(grtv.GRTV_DATA_TYPE_IMAGE)
print("get 3d raw", ret)

orgShape = img_short.shape
img_short[int(orgShape[0]*3/4):] = 500 #上部 1/4 の CT 値を 500 に
設定

print("call replace image API")

# replace と export の同時実行
ret = grtv.replaceCurrentImage(img_short,
exportDcmDir="d:/convTest/repDcm1/")
print("replace result1", ret)

img_short[:int(orgShape[0] / 4)] = 200 # 更に下部 1/4 の CT 値を 200
に設定

# replace と export を分けて実行
ret = grtv.replaceCurrentImage(img_short)
print("replace result2", ret)
grtv.exportAsNewDcm(exportDcmDir="d:/convTest/repDcm2/")
print("export result", ret)

```

## 10.6. サンプル 6(3D Raw の DICOM 変換および出力)

このサンプルでは、指定した 3D ボリュームデータ(raw)を DICOM Image への変換処理する例を示します。

```

import os, sys, glob, shutil
sys.path.append("C:/Program Files/GrowthRTV/pyGRTV")
from grtv import *

```

```
if __name__ == "__main__":  
  
    # GRTV インスタンス生成  
    grtv = pyGRTV()  
  
    # 3D Raw 読み込み  
    img_short = grtv.readRaw("d:/convTest/test.raw", dataType=np.int16)  
  
    # 3次元配列(depth, height, width)に整形  
    img_short = img_short.reshape((512, 512, 512))  
  
    # 画素間隔  
    resZ, resY, resX = 1.5, 1.0, 1.0  
  
    # Dicomへ変換  
    grtv.imageRawToDicom(img_short, resZ, resY, resX,  
"d:/convTest/rawToDcm")
```