SDSIA
Synthetic Data Set for Image analysis

P. Baillehache

January 27, 2019

Contents

1 Python module 3

1.1 Usage . . . 3

1.1.1 Unit test . . . 3

1.1.2 Create a new data set . . . 3

1.1.3 Generate one or several data sets . . . 6

1.1.4 Listing of the data sets . . . 7

1.2 Code . . . 7

1.2.1 generateDataSet.py . . . 7

1.3 exampleUse.py . . . 23

2 C Library 24

2.1 Interface . . . 24

2.2 Code . . . 28

2.2.1 sdsia.c . . . 28

2.2.2 sdsia-inline.c . . . 33

2.3 Unit test . . . 39

3 Makefile 39

4 Data sets 40

4.1 dataset-001-001 . . . 40

4.2 dataset-001-002 . . . 41

4.3 dataset-001-003 . . . 42

4.4 dataset-001-004 . . . 44

4.5 dataset-002-001 . . . 45

4.6 dataset-002-002 . . . 47

4.7 dataset-002-003 . . . 49

Introduction

SDSIA is a Python module to generate synthetic data sets for image analysis, and a C library to manipulate these data sets.

The goal of the Python module is to provide a way to easily generate data sets of images used by image analysis softwares or algorithms, in a context of learning or development.

The generation of the images is made using the 3D rendering software POV-Ray. This software uses text scripts to defined the scene to render, so it's very easy to parameterize a script to create variations of a given scene. The Python module then render as many samples as needed in the set, each time using different parameters' value.

By creating its own POV-Ray script, the user can create any needed data set, selecting precisely which aspect of the scene is constant and which is variable. The level of difficulty of the data set, from the point of view of the software or algorithm used for analysis, can then be controlled at will. By choosing the appropriate variable in the scene, the user could also study a particular property of a given software or algorithm (e.g, is it more sensitive to variation in shape or variation in color).

Once the user has written the script to render images, the python module takes care of the tedious tasks of rendering (only needed data sets), files and folders management, creation of a description file in JSON format for later use. It also generates perfect masks (black and white image) by modifying the texture properties of the objects in the scene.

The number of images per data set, the dimensions and format of each image are also defined by the user. Then, one can create sets corresponding to its needs, in particular memory, disk storage, processing time limits.

The current version of SDSIA is designed for image segmentation (localization of pixels corresponding to an object in a scene). However it has been developped with the view to be extended to other kind of data sets.

The C library offers the following functionalities: loading a data set from its description file, spitting the data set into user defined categories (e.g. training, validation, test), shuffling the data set, looping through the samples of the data set. It provides the samples of the data set as GenBrush objects to manipulate them easily through this library's functionalities.

The C library uses the `PBErr`, `GSet`, `PBJson`, `GenBrush` and `PBFileSys` libraries.

1 Python module

1.1 Usage

1.1.1 Unit test

One can run the unit test with the command python generateDataSet.py. Upon success the following messages will be displayed:

```
python generateDataSet.py -unitTest
[-list] OK
[-simul] OK
Generation OK
[-force] OK
UnitTest of generateDataSet.py succeeded
```

If the unit test fails, make sure POV-Ray is correctly installed by running, for example, the comand `./install test` in the folder where POV-Ray has been installed (Refer to the POV-Ray documentation for more details). If POV-Ray is correctly installed and the unit test still fails, please contact the developper.

1.1.2 Create a new data set

To create a new data set, one has to create the description file and the POV-Ray script for this new data set. Templates for each of them are given below.

Template for the description file:

```
{
  "dataSetType": "0",
  "desc": "unitTest",
  "dim": [
    "10",
    "20"
  ],
  "format": "tga",
  "nbImg": "3"
}
```

DataSetType is a place holder for future version and should always be set to 0. Desc is the description of the data set. Dim are the dimensions of the images (width, height). Supported formats are currently ”tga” and ”png”. NbImg is the number of images to be generated.

Template for the POV-Ray script:

```
#include "colors.inc"
#include "textures.inc"

// Black texture used to create the mask of the target
#declare _texMaskTarget = texture {
  pigment { color Black }
  finish { ambient 0 }
}

// White texture used to create the mask of the non-target
#declare _texMaskNonTarget = texture {
  pigment { color White }
  finish { ambient 1 diffuse 100 }
}

// Random generator seed, based on the clock variable set by the
// SDSIA generator
#declare RndSeed = seed(clock);

// Macro to get a random value in [A, B]
#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

// Camera
camera {
  location    <0.0,10.0,0.0>
  look_at     <0.0,0.0,0.0>
  right x
}

// Light source
light_source {
  <0.0,10.0,0.0>
}

// Background
background { color rgb <1.0, 1.0, 1.0> }

// Target definition
#declare Target = box {
  -1, 1

  // Transformation of the target to create various samples
  translate x * rnd(0, 1)

  // Apply the real texture or the mask texture according to the
  // Mask variable set by the SDSIA generator
  #if (Mask = 0)
    pigment { color Red }
  #else
    texture {_texMaskTarget}
  #end

}

// Non-target definition
#declare NonTarget = cylinder {
  -y, y, 0.5

  // Transformation of the non-target to create various samples
  translate x * rnd(0, 1)

  // Apply the real texture or the mask texture according to the
  // Mask variable set by the SDSIA generator
  #if (Mask = 0)
    pigment { color Blue }
  #else
    texture {_texMaskNonTarget}
  #end

}

// Create the scene
object { Target }
object { NonTarget }
```

These files must be saved with names, respectively, `dataset-XXX-YYY.json` and `dataset-XXX-YYY.pov`, where `XXX` is the data set group and `YYY` is the data set subgroup. One is free to save them wherever she wants but they must be in the same directory. Furthermore, if saved in the POV folders of the SDSIA repository, the module will detect them automatically and one won't need to specify the location of these files with the `-in` option.

About the POV-Ray script: one must be careful to craft it in such a way that the sequence of random values is the same in both version (image and mask) of the rendering. For example,

```
#if (Mask = 0)
  background { color rgb <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)> }
#else
  background { color White }
#end

#declare Target = box {
  -1, 1
  scale rnd(0.5, 1.5)
  #if (Mask = 0)
    pigment { color rgb <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)> }
  #else
    texture {_texMaskTarget}
  #end
}
```

won't render the correct mask. The call to the random generator in the background color being skipped when rendering the mask, the call to the random generator for the scale of the box will return different values. A correct script would be:

```
#declare bgColor = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
#if (Mask = 0)
  background { color rgb bgColor }
#else
  background { color White }
#end

#declare Target = box {
  -1, 1
  scale rnd(0.5, 1.5)
  #if (Mask = 0)
    pigment { color rgb <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)> }
  #else
    texture {_texMaskTarget}
  #end
}
```

1.1.3 Generate one or several data sets

One can generate the data sets in the default `POV` folder of the SDSIA repository with the command `python generateDataSet.py`. One can also specify another folder or one single data set with the `-in` argument: `python generateDataSet.py -in <path>` where path is the path to the folder containing the data sets' description file and POV-Ray script, or the path to the data set's POV-Ray script.

The data sets are generated by default into the DataSets folder of the SDSIA repository, but one can specify another folder with the `-out` argument: `python generateDataSet.py -out <path>` . Inside the DataSets folder each data set is generated in its own folder as follow: `DataSets/XXX/YYY/` where `XXX` is the data set group and `YYY` is the data set subgroup. Each data set's folder contains one description file (`dataset.json`) and the images and masks of the data set. Example of description file:

```
{
  "dataSet": "dataset-001-001",
  "dataSetType": "0",
  "desc": "unitTest",
  "dim": [
    "10",
    "20"
  ],
  "format": "tga",
  "images": [
    {
      "img": "img000.tga",
      "mask": "mask000.tga"
    },
    {
      "img": "img001.tga",
      "mask": "mask001.tga"
    },
    {
      "img": "img002.tga",
      "mask": "mask002.tga"
    }
  ],
  "nbImg": "3"
}
```

By default, data sets are only generated if necessary, i.e. the description file and POV-Ray scripts in the input folder are older than the description file in the output folder. One can override this with the argument `-force`.

One can use the `-simul` argument to check what will be generated without actually rendering the images and masks, which may be useful to check that everything will be as expected before running a time consuming generation.

1.1.4 Listing of the data sets

One can get a listing of the data sets with the command `python generateDataSet.py -list`. Example of output:

```
[*]  dataset-001-001: unitTest
[ ]  dataset-001-002: unitTest
[*]  dataset-002-001: unitTest
```

The listing is generated according to the arguments `-in` and `-out`.

The mark `[*]` means the data sets has been generated, and the mark `[ ]` means it is not yet generated.

1.2 Code

1.2.1 generateDataSet.py

```
# Import necessary modules
import os, sys, json, glob, subprocess, re, shutil

# Base directory
BASE_DIR = os.path.dirname(os.path.abspath(__file__))

# Function to print exceptions
def PrintExc(exc):
  exc_type, exc_obj, exc_tb = sys.exc_info()
  fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
  print(exc_type, fname, exc_tb.tb_lineno, str(exc))

class DataSet:
  '''
  Class containing the information about a dataSet
  '''
```

```
# Name of the data set
_name = ""
# Description of the data set
_desc = ""
# Type of the data set (place holder for future extension)
_type = ""
# Number of images in the data set
_nbImg = "0"
# Dimensions of the images in the data set [width, height]
_dim = ["1", "1"]
# format of the images in the data set
_format = "tga"
# List of images and their masks
_images = []


def __init__(self, templateFilePath):
  '''
  Constructor
  Inputs:
    'templateFilePath': full path to the template of the description
      file for this data set
  '''
  try:

    # Load and decode the template file
    with open(templateFilePath, "r") as fp:
      dataSetDesc = json.load(fp)

    # Get the name of the data set
    dataSetName = os.path.basename(templateFilePath)[0:-5]

    # Set properties according to the template
    self._name = dataSetName
    self._desc = dataSetDesc["desc"]
    self._type = dataSetDesc["dataSetType"]
    self._nbImg = dataSetDesc["nbImg"]
    self._dim = dataSetDesc["dim"]
    self._format = dataSetDesc["format"]

  except Exception as exc:
    PrintExc(exc)

def GetDescFileContent(self):
  '''
  Create the content of the file describing a DataSet
  Input:
    'self': the dataSet for which we want to create the description
      file
  Output:
    Return the content of the file describing the DataSet as a string
    in JSON format
  '''
  try:

    # Init the return value
    ret = "{}"

    # Init a variable to memorize the content as an object
    content = {}

    # Add the info about the DataSet to the content
    content["dataSet"] = self._name
    content["desc"] = self._desc
    content["dataSetType"] = self._type
    content["nbImg"] = self._nbImg
    content["dim"] = self._dim
    content["format"] = self._format
    content["images"] = self._images

    # Encode the content to JSON format and return it
    ret = json.dumps(content, \
      sort_keys = True, indent = 2, separators = (',', ': '))

  except Exception as exc:
    PrintExc(exc)
  finally:
    return ret

def Render(self, inFolder, outFolder):
  '''
  Render (create images and masks) the DataSet
  Inputs:
    'inFolder': the full path to the folder where the pov file is
    'outFolder': the full path of the folder where images and masks
      are output
  '''
  try:

    # Loop on the number of images to be rendered
    for iRender in range(int(self._nbImg)):

      # Create the file name of the image and mask
      iRenderPadded = str(iRender).zfill(3)
      imgFileName = "img" + iRenderPadded + "." + self._format
      maskFileName = "mask" + iRenderPadded + "." + self._format

      # Create the full path of the image and mask
      imgFilePath = os.path.join(outFolder, imgFileName)
      maskFilePath = os.path.join(outFolder, maskFileName)

      # Create the full path to the pov file
      povFilePath = os.path.join(inFolder, self._name + ".pov")

      # Create the full path to the ini file used to render
      iniFilePath = os.path.join(outFolder, "pov.ini")

      # Create the command to render the image
      cmd = []
      cmd.append("povray")
      cmd.append("+Oimg" + iRenderPadded + "." + self._format)
      cmd.append("-W" + self._dim[0])
      cmd.append("-H" + self._dim[1])
      cmd.append("-D")
      cmd.append("-P")
      cmd.append("-Q9")
      cmd.append("+A")
      cmd.append("+k" + str(iRender))
      if self._format == "png":
        povFormat = "+FN"
      elif self._format == "tga":
        povFormat = "+FC"
      else:
        print("Unsupported format: " + self._format)
```

```
          return False
        cmd.append(povFormat)
        cmd.append("+I" + povFilePath)
        cmd.append(iniFilePath)

        # Create the ini file used to render
        with open(iniFilePath, "w") as fp:
          fp.write("Declare=Mask=0")

        # Render the image silently
        imgPath =  os.path.join(outFolder, \
          "img" + iRenderPadded + "." + self._format)
        print(iRenderPadded + "/" + self._nbImg.zfill(3) + \
          " Rendering image " + imgPath + " ...")
        FNULL = open(os.devnull, 'w')
        subprocess.call(cmd, stderr = FNULL, cwd = outFolder)

        # Check if the image has been created
        if not os.path.exists(imgPath):
          return False

        # Update the command and the ini file to render the mask
        cmd[1] = "+Omask" + iRenderPadded + "." + self._format
        with open(iniFilePath, "w") as fp:
          fp.write("Declare=Mask=1")

        # Render the mask silently
        maskPath = os.path.join(outFolder, \
          "mask" + iRenderPadded + "." + self._format)
        print("        Rendering mask " + maskPath + " ...");
        subprocess.call(cmd, stderr = FNULL, cwd = outFolder)

        # Check if the mask has been created
        if not os.path.exists(maskPath):
          return False

        # Remove the ini file
        os.remove(iniFilePath)

        # Add the image and mask to the lists
        self._images.append({"img":imgFileName, "mask":maskFileName})

      # Return the success flag
      return True

    except Exception as exc:
      PrintExc(exc)
      return False

class DataSetGenerator:
  '''
  Class to generate the data sets
  '''
  # Folder containing the POV files
  # Only files matching dataSet-[0-9][0-9][0-9]-[0-9][0-9][0-9].pov are
  # processed
  _povFolder = os.path.join(BASE_DIR, "POV")
  # Folder containing the data sets
  _dataSetFolder = os.path.join(BASE_DIR, "DataSets")
  # Flag to memorize if the data sets are generated even if their
  # description file is younger than their pov file
  _force = False
```

```
# Name of the description file
_descFileName = "dataset.json"
# Variables to memorize the successful generation
_successDataSets = []
# Variables to memorize the failed generation
_failedDataSets = []
# Variables to memorize the skipped generation
_skipDataSets = []
# Variable to memorize if we are in simulation mode
_simul = False
# Variable to memorize if we are in listing mode
_list = False

def __init__(self, args):
  '''
  Constructor
  Inputs:
    'args': the arguments passed to this script
  '''
  try:

    # Init a flag to memorize if the user requested unit tests
    flagUnitTest = False

    # Process arguments
    for iArg in range (len(args)):

      # Help
      if args[iArg] == "-help":
        print("server.py" + \
          " [-in <povFolder|povFile>] [-out <dataSetFolder>]" + \
          " [-force] [-simul] [-list] [-unitTest] [-help]")
        print("-in: folder containing the pov files, or one pov file")
        print("-out: folder where the data sets will be generated")
        print("-force: don't check time stamp and always generate" + \
          " all the data sets")
        print("-simul: don't actually generate the data sets")
        print("-list: display a list of the data sets")
        print("-unitTest: run the unit tests")
        quit()

      # POV files folder
      if args[iArg] == "-in":
        folder = args[iArg + 1]
        if os.path.exists(folder):
          self._povFolder = os.path.abspath(folder)
        else:
          print("The folder/file " + str(folder) + " doesn't exists.")
          quit()

      # DataSets folder
      if args[iArg] == "-out":
        folder = args[iArg + 1]
        if os.path.exists(folder):
          self._dataSetFolder = os.path.abspath(folder)
        else:
          print("The folder " + str(folder) + " doesn't exists.")
          quit()

      # Simulation mode, the sets are not actually rendered
      if args[iArg] == "-simul":
        self._simul = True
      # Force generation of all data sets even if their
      # description file is younger than their pov file
      if args[iArg] == "-force":
        self._force = True

      # Listing mode
      if args[iArg] == "-list":
        self._list = True

      # Unit tests
      if args[iArg] == "-unitTest":
        flagUnitTest = True

    # Run the unit tests if requested
    if flagUnitTest:
      ret = self.RunUnitTest()
      quit(ret)

  except Exception as exc:
    PrintExc(exc)

def Run(self):
  '''
  Generate the dataSets according to the properties of the
  DataSetGenerator 'self'
  '''
  try:

    # Reset the successful/failed/skipped dataSets lists
    self._successDataSets = []
    self._failedDataSets = []
    self._skippedDataSets = []

    # Get the list of POV files path
    # If the -in argument was a pov file, consider only this file,
    # else consider the pov files in the folder
    if self._povFolder[-4:] == ".pov":
      pattern = re.compile(
        "dataset-[0-9][0-9][0-9]-[0-9][0-9][0-9].pov")
      if pattern.match(os.path.basename(self._povFolder)):
        povFilePaths = [self._povFolder]
      else:
        print("\nThe pov file \n  " + self._povFolder + \
          "\ndoesn't match " + \
          "dataset-[0-9][0-9][0-9]-[0-9][0-9][0-9].pov\n")
        return None
    else:
      povFilePaths = glob.glob(os.path.join(self._povFolder, \
        "dataset-[0-9][0-9][0-9]-[0-9][0-9][0-9].pov"))

    # Sort the list of POV files path
    povFilePaths.sort()

    # If there are no Pov files in the input folder, inform the user
    # and stop
    if len(povFilePaths) == 0:
      print("\nNo Pov files in\n  " + self._povFolder + \
        "\nmatching dataset-[0-9][0-9][0-9]-[0-9][0-9][0-9].pov\n")
      return None

    # Loop on the POV file pathes
```

```
for povFilePath in povFilePaths:

  # Get the filename of the pov file
  povFileName = os.path.basename(povFilePath)

  # Get the path to the input folder
  inFolder = os.path.dirname(povFilePath)

  # Get the path to the template of the description file
  templateFilePath = povFilePath[0:-4] + ".json"

  # Get the dataSet group number from the file name
  groupNum = povFileName[8:11]

  # Get the dataSet subgroup number from the file name
  subGroupNum = povFileName[12:15]

  # Get the name of the corresponding output folder
  outFolder = os.path.join(self._dataSetFolder, \
    groupNum, subGroupNum)

  # Get the path to the description file for this data set
  descFilePath = os.path.join(outFolder, self._descFileName)

  # Init a flag to memorize if this data set must be generated
  isGenNecessary = False

  # If the output folder doesn't exist, the generation is necessary
  if not os.path.exists(outFolder):
    isGenNecessary = True

  # Else, the output folder exists
  else:

    # If the description file exists
    if os.path.exists(descFilePath):

      # Get the last modification time of the description file
      dateLastModifDesc = os.path.getmtime(descFilePath)

      # Get the last modification time of the pov file
      dateLastModifPov = os.path.getmtime(povFilePath)

      # Get the last modification time of the template for
      # the description file
      dateLastModifTemplate = os.path.getmtime(templateFilePath)

      # If the Pov or template file as been modified more
      # recently than the description file, the generation is
      # necessary
      if dateLastModifPov > dateLastModifDesc or \
        dateLastModifTemplate > dateLastModifDesc:
        isGenNecessary = True

    # Else, the description file doesn't exist, the generation is
    # necessary
    else:
      isGenNecessary = True

  # If we are in listing mode
  if self._list:
    # Print a mark to show if the set has been generated
    if isGenNecessary:
      print("[ ] "),
    else:
      print("[*] "),

    # Load the data set info
    dataSet = DataSet(templateFilePath)

    # Print the set name and descrition
    print(dataSet._name + ": " + dataSet._desc)


  # Else we are not in listing mode
  else:

    # If the generation is necessary or forced for this data set
    if isGenNecessary or self._force:

      # If we are not in simulation or listing mode
      if not self._simul and not self._list:

        # Ensure the output folder exists
        if not os.path.exists(outFolder):
          os.makedirs(outFolder)

        # Ensure the output folder is empty of the description file
        # and images
        try:
          os.remove(os.path.join(outFolder, self._descFileName))
        except:
          pass
        for f in glob.glob(os.path.join(outFolder, "img*.*")):
          try:
            os.remove(f)
          except:
            pass
        for f in glob.glob(os.path.join(outFolder, "mask*.*")):
          try:
            os.remove(f)
          except:
            pass

      # If we are not in listing mode
      if not self._list:

        # Generate this data set
        self.Generate(povFilePath, povFileName, groupNum, \
          subGroupNum, outFolder, descFilePath, templateFilePath, \
          inFolder)

    # Else, the generation of this data set is skipped
    else:

      # If we are not in listing mode
      if not self._list:

        # Append this data set to the list of skipped data sets
        self._skipDataSets.append(povFilePath)

# If we are not in listing mode
if not self._list:
      # If some generation were successful
      if len(self._successDataSets) > 0:

        # Inform the user
        print("\nThe following data sets were " + \
          "generated successfully:")
        for d in self._successDataSets:
          print("  " + d)

      # If some generation failed
      if len(self._failedDataSets) > 0:

        # Inform the user
        print("\nThe following data sets couldn't be " + \
          "generated successfully:")
        for d in self._failedDataSets:
          print("  " + d)

      # If some generation were skipped
      if len(self._skipDataSets) > 0:

        # Inform the user
        print("\nThe following data sets were skipped:")
        for d in self._skipDataSets:
          print("  " + d)

      # Skip a line
      print("")

  except Exception as exc:
    PrintExc(exc)

def Generate(self, povFilePath, povFileName, groupNum, \
  subGroupNum, outFolder, descFilePath, templateFilePath, inFolder):
  '''
  Generate one dataSet
  Inputs:
    'povFilePath': the full path of the pov file
    'povFileName': the filename of the pov file
    'groupNum': the dataSet group number from the file name
    'subGroupNum': the dataSet subgroup number from the file name
    'outFolder': the output folder where the data set is generated
    'descFilePath': the full path to the output description file
    'templateFilePath': the full path to the template of the
      description file
    'inFolder': the input folder where the pov and template files are
  '''
  try:

    # Inform the user
    print("\n === Generate data set for\n  " + povFilePath + \
      "\nto\n  " + outFolder)

    # Skip a line
    print("")

    # If the template doesn't exist, add this dataSet to the failed
    # data sets, inform the user and give up
    if not os.path.exists(templateFilePath):
      print("The template file\n  " + templateFilePath + \
        "\ndoesn't exist. Give up.")
      self._failedDataSets.append(povFilePath)
      return None

    # Load the template file into a DataSet object
    dataSet = DataSet(templateFilePath)

    # If we are not in simulation mode
    if not self._simul:

      # Generate the images and masks
      if not dataSet.Render(inFolder, outFolder):

        # If the rendering of the data set has failed, inform the user
        # and give up
        print("The rendering of \n  " + povFilePath + \
          "\nhas failed. Give up.")
        self._failedDataSets.append(povFilePath)
        return None

      # Create the description file
      with open(descFilePath, "w") as fp:
        fp.write(dataSet.GetDescFileContent())

    # Append this data set to the list of successfull data sets
    self._successDataSets.append(povFilePath)

    # Inform the user
    print("\nGeneration of \n  " + outFolder + \
      "\ncompleted.")

  except Exception as exc:
    PrintExc(exc)

def RunUnitTest(self):
  '''
  Run the unit tests
  '''
  try:

    # Variable to memorize if the unit tests succeeded
    flagSuccess = True

    # Create temporary folders to run the test
    try:
      shutil.rmtree("UnitTestIn")
      shutil.rmtree("UnitTestOut")
    except:
      pass
    os.mkdir("UnitTestIn")
    os.mkdir("UnitTestOut")

    # Create fake data set
    shutil.copy("dataset.pov",
      os.path.join("UnitTestIn", "dataset-001-001.pov"))
    shutil.copy("dataset.pov",
      os.path.join("UnitTestIn", "dataset-001-002.pov"))
    shutil.copy("dataset.pov",
      os.path.join("UnitTestIn", "dataset-002-001.pov"))
    shutil.copy("dataset.json",
      os.path.join("UnitTestIn", "dataset-001-001.json"))
    shutil.copy("dataset.json",
      os.path.join("UnitTestIn", "dataset-001-002.json"))
```

```
shutil.copy("dataset.json",
  os.path.join("UnitTestIn", "dataset-002-001.json"))

# Test listing
cmd = []
cmd.append("python")
cmd.append("generateDataSet.py")
cmd.append("-in")
cmd.append("UnitTestIn")
cmd.append("-out")
cmd.append("UnitTestOut")
cmd.append("-list")
with open("out.txt", "w") as fp:
  subprocess.call(cmd, stdout = fp)
check = ["[ ]  dataset-001-001: unitTest\n",
  "[ ]  dataset-001-002: unitTest\n",
  "[ ]  dataset-002-001: unitTest\n"]
with open ("out.txt", "r") as fp:
  data = fp.readlines()
  if not data == check:
    print("[-list] NOK")
    flagSuccess = False
  else:
    print("[-list] OK")

# Test simulation
cmd = []
cmd.append("python")
cmd.append("generateDataSet.py")
cmd.append("-in")
cmd.append("UnitTestIn")
cmd.append("-out")
cmd.append("UnitTestOut")
cmd.append("-simul")
with open("out.txt", "w") as fp:
  subprocess.call(cmd, stdout = fp)
check = [
  '\n',
  ' === Generate data set for\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-001.pov") +
  '\n',
  'to\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001") +
  '\n',
  '\n',
  '\n',
  'Generation of \n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001") +
  '\n',
  'completed.\n',
  '\n',
  ' === Generate data set for\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-002.pov") +
  '\n',
  'to\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002") +
  '\n',
  '\n',
  '\n',
  'Generation of \n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002") +
  '\n',
  'completed.\n',
  '\n',
  ' === Generate data set for\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-002-001.pov") +
  '\n',
  'to\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001") +
  '\n',
  '\n',
  '\n',
  'Generation of \n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001") +
  '\n',
  'completed.\n',
  '\n',
  'The following data sets were generated successfully:\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-001.pov") +
  '\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-002.pov") +
  '\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-002-001.pov") +
  '\n',
  '\n']
with open ("out.txt", "r") as fp:
  data = fp.readlines()
  if not data == check or \
    os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "dataset.json")) or \
    os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "dataset.json")) or \
    os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "dataset.json")):
    print("[-simul] NOK")
    flagSuccess = False
  else:
    print("[-simul] OK")

# Test generation
cmd = cmd[:-1]
with open("out.txt", "w") as fp:
  subprocess.call(cmd, stdout = fp)
check = [
  '\n',
  ' === Generate data set for\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-001.pov") +
  '\n',
  'to\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001") + '\n',
  '\n',
  '000/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'img000.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'mask000.tga ...\n',
  '001/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'img001.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'mask001.tga ...\n',
  '002/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'img002.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'mask002.tga ...\n',
  '\n',
  'Generation of \n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001") +
  '\n', 'completed.\n',
  '\n',
  ' === Generate data set for\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-002.pov") +
  '\n',
  'to\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002") + '\n',
  '\n',
  '000/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'img000.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'mask000.tga ...\n',
  '001/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'img001.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'mask001.tga ...\n',
  '002/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'img002.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'mask002.tga ...\n',
  '\n',
  'Generation of \n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002") + '\n',
  'completed.\n',
  '\n',
  ' === Generate data set for\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-002-001.pov") +
  '\n',
  'to\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001") +
  '\n',
  '\n',
  '000/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'img000.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'mask000.tga ...\n',
  '001/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'img001.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'mask001.tga ...\n',
  '002/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'img002.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'mask002.tga ...\n',
  '\n',
  'Generation of \n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001") + '\n',
  'completed.\n',
  '\n',
  'The following data sets were generated successfully:\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-001.pov") +
  '\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-002.pov") +
  '\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-002-001.pov") +
  '\n',
  '\n']
with open ("out.txt", "r") as fp:
  data = fp.readlines()
  if not data == check or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "dataset.json")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "img000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "img001.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "img002.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "mask000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "mask001.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "mask002.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "dataset.json")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "img000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "img001.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "img002.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "mask000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "mask001.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "mask002.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "dataset.json")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "img000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "img001.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "img002.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "mask000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "mask001.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "mask002.tga")):
    flagSuccess = False
check = [
  '\n',
  'The following data sets were skipped:\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-001.pov") +
  '\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-002.pov") +
  '\n',
  '  ' +
  os.path.join(BASE_DIR, "UnitTestIn", "dataset-002-001.pov") +
  '\n',
  '\n']
if flagSuccess:
  with open("out.txt", "w") as fp:
    subprocess.call(cmd, stdout = fp)
  with open ("out.txt", "r") as fp:
    data = fp.readlines()
    if not data == check:
      flagSuccess = False
      print("Generation NOK")
    else:
      print("Generation OK")
else:
  print("Generation NOK")

# Test [-force]
dateTest = os.path.getmtime(os.path.join("UnitTestOut",
  "001", "001", "dataset.json"))
cmd.append("-force")
with open("out.txt", "w") as fp:
  subprocess.call(cmd, stdout = fp)
check = [
  '\n',
  ' === Generate data set for\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-001.pov") + '\n',
  'to\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001") + '\n',
  '\n',
  '000/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") + 'img000.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") + 'mask000.tga ...\n',
  '001/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") + 'img001.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") + 'mask001.tga ...\n',
  '002/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") + 'img002.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") + 'mask002.tga ...\n',
  '\n',
  'Generation of \n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "001") + '\n',
  'completed.\n',
  '\n',
  ' === Generate data set for\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-002.pov") + '\n',
  'to\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002") + '\n',
  '\n',
  '000/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") + 'img000.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") + 'mask000.tga ...\n',
  '001/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") + 'img001.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") + 'mask001.tga ...\n',
  '002/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") + 'img002.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") + 'mask002.tga ...\n',
  '\n',
  'Generation of \n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "001", "002") + '\n',
  'completed.\n',
  '\n',
  ' === Generate data set for\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestIn", "dataset-002-001.pov") + '\n',
  'to\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001") + '\n',
  '\n',
  '000/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") + 'img000.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") + 'mask000.tga ...\n',
  '001/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") + 'img001.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") + 'mask001.tga ...\n',
  '002/003 Rendering image ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") + 'img002.tga ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") + 'mask002.tga ...\n',
  '\n',
  'Generation of \n',
  '  ' + os.path.join(BASE_DIR, "UnitTestOut", "002", "001") + '\n',
  'completed.\n',
  '\n',
  'The following data sets were generated successfully:\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-001.pov") + '\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestIn", "dataset-001-002.pov") + '\n',
  '  ' + os.path.join(BASE_DIR, "UnitTestIn", "dataset-002-001.pov") + '\n',
  '\n']
with open ("out.txt", "r") as fp:
  data = fp.readlines()
  if not data == check or \
    dateTest == os.path.getmtime(os.path.join("UnitTestOut",
  "001", "001", "dataset.json")):
    flagSuccess = False
    print("[-force] NOK")
  else:
    print("[-force] OK")

# Delete the temporary file
os.remove("out.txt")

# Inform the user
if flagSuccess:
  print("UnitTest of generateDataSet.py succeeded")
  ret = 0
else:
  print("UnitTest of generateDataSet.py failed")
  ret = 1

return ret
```

```
    except Exception as exc:
      PrintExc(exc)

def Main():
  '''
  Main function
  '''
  try:

    # Create a DataSetGenerator
    generator = DataSetGenerator(sys.argv)

    # Generate the dataSets
    generator.Run()

  except Exception as exc:
    PrintExc(exc)

# Hook for the main function
if __name__ == '__main__':
  Main()
```

1.3 exampleUse.py

A boilerplate to use the generated data sets is given below:

```
# Import necessary modules
import os, sys, json

# Base directory
BASE_DIR = os.path.dirname(os.path.abspath(__file__))

# Function to print exceptions
def PrintExc(exc):
  exc_type, exc_obj, exc_tb = sys.exc_info()
  fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
  print(exc_type, fname, exc_tb.tb_lineno, str(exc))

def Main(dataSetFolderPath):
  '''
  Main function
  Inputs:
    'dataSetFolderPath': full path to the folder containing the data set
  '''
  try:

    # Check if the folder exists
    if not os.path.exists(dataSetFolderPath):
      print("The folder " + dataSetFolderPath + " doesn't exists")
      quit()

    # Check if the folder contains the data set description file
    descFilePath = os.path.join(dataSetFolderPath, "dataset.json")
    if not os.path.exists(descFilePath):
      print("The description file " + descFilePath + " doesn't exists")
      quit()

    # Load and decode the content of the description file
    with open(descFilePath, "r") as fp:
      dataSetDesc = json.load(fp)
    # Display info about the data set
    print("Description: " + dataSetDesc["desc"])
    print("Nb image: " + dataSetDesc["nbImg"])
    print("Dimension image (width, height): " + str(dataSetDesc["dim"]))
    print("Format image: " + dataSetDesc["format"])

    # Loop on the data set
    for iSample in range(int(dataSetDesc["nbImg"])):

      # Get the path to the image and mask
      imgFileName = dataSetDesc["images"][iSample][0]
      maskFileName = dataSetDesc["images"][iSample][1]
      imgFilePath = os.path.join(dataSetFolderPath, imgFileName)
      maskFilePath = os.path.join(dataSetFolderPath, maskFileName)

      # Check the full paths are valid
      if not os.path.exists(imgFilePath):
        print("Description file corrupted. The image " + \
          imageFilePath + " doesn't exists")
        quit()
      if not os.path.exists(maskFilePath):
        print("Description file corrupted. The mask " + \
          maskFilePath + " doesn't exists")
        quit()

      # Train on the pair image-mask
      # In the mask, the black pixels match the target and the white
      # pixels match the non-target
      print("Train on (" + imgFilePath + ", " + maskFilePath + ")")

  except Exception as exc:
    PrintExc(exc)

# Hook for the main function
if __name__ == '__main__':
  try:

    # Get the data set folder path from the argument line
    if not len(sys.argv) == 2:
      print("Usage: python exampleUse.py <dataSetFolderPath>")
      quit()
    dataSetFolderPath = os.path.abspath(sys.argv[1])

    # Call the main function with the data set folder
    Main(dataSetFolderPath)

  except Exception as exc:
    PrintExc(exc)
```

2 C Library

2.1 Interface

```
// =========== SDSIA.H ===============

#ifndef SDSIA_H
#define SDSIA_H
```

```
// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <execinfo.h>
#include <errno.h>
#include <string.h>
#include "pberr.h"
#include "gset.h"
#include "pbjson.h"
#include "genbrush.h"
#include "pbfilesys.h"

// ================= Define ==================

#define SDSIA_DESCFILENAME "dataset.json"

// ================= Data structure ===================

typedef struct SDSIAPairName {
  // Names of img and mask
  char* _imgName;
  char* _maskName;
} SDSIAPairName;

typedef struct SDSIASample {
  // Image and mask
  GenBrush* _img;
  GenBrush* _mask;
} SDSIASample;

typedef struct SDSIA {
  // Path to the folder of the data set
  char* _folderPath;
  // Name of the data set
  char* _name;
  // Description of the data set
  char* _desc;
  // Format of images and masks
  char* _format;
  // Type of set
  int _type;
  // Dimensions of images and masks
  VecShort2D _dim;
  // Set of SDSIAPairName
  GSet _samples;
  // Splitting of samples
  VecLong* _split;
  // Sets of splitted samples
  GSet* _categories;
  // Iterators on the sets of splitted samples
  GSetIterForward* _iterators;
} SDSIA;

// ================ Functions declaration ====================

// Create a new SDSIA with the content of the directory 'folderPath'
// The random generator must have been initialized before calling
// this function
SDSIA SDSIACreateStatic(const char* const folderPath);
```

```
// Free the memory used by a SDSIA
void SDSIAFreeStatic(SDSIA* const that);

// Get the total number of images in the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
long SDSIAGetSize(const SDSIA* const that);

// Get the total number of images/masks in the SDSIA 'that' for the
// category 'iCat'. Return 0 if the category doesn't exists
#if BUILDMODE != 0
inline
#endif
long SDSIAGetSizeCat(const SDSIA* const that, const long iCat);

// Split the samples of the SDSIA 'that' into several categories
// defined by 'cat'. The dimension of 'cat' gives the number of
// categories and the value for each dimension of 'cat' gives the
// number of samples in the correpsonding category. For example <3,4>
// would mean 2 categories with 3 samples in the first one and 4
// samples in the second one. There must me at least as many samples
// in the data set as the sum of samples in 'cat'.
// Each category must have at least one sample. Samples are allocated // randomly to the categories.
// If 'that' was already splitted the previous splitting is discarded.
void SDSIASplit(SDSIA* const that, const VecLong* const cat);

// Unsplit the SDSIA 'that', i.e. after calling SDSIAUnsplit 'that'
// has only one category containing all the samples
#if BUILDMODE != 0
inline
#endif
void SDSIAUnsplit(SDSIA* const that);

// Shuffle the samples of the category 'iCat' of the SDSIA 'that'.
// Reset the iterator of the category
#if BUILDMODE != 0
inline
#endif
void SDSIAShuffle(SDSIA* const that, const long iCat);

// Shuffle the samples of all the categories of the SDSIA 'that'.
// Reset the iterator of the categories
#if BUILDMODE != 0
inline
#endif
void SDSIAShuffleAll(SDSIA* const that);

// Get the name of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
const char* SDSIAName(const SDSIA* const that);

// Get the description of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
const char* SDSIADesc(const SDSIA* const that);

// Get the format of the SDSIA 'that'
```

```
#if BUILDMODE != 0
inline
#endif
const char* SDSIAFormat(const SDSIA* const that);

// Get the path of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
const char* SDSIAFolderPath(const SDSIA* const that);

// Get the type of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
int SDSIAGetType(const SDSIA* const that);

// Get the number of categories of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
long SDSIAGetNbCat(const SDSIA* const that);

// Get the dimensions of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
VecShort2D SDSIAGetDim(const SDSIA* const that);
#if BUILDMODE != 0
inline
#endif
const VecShort2D* SDSIADim(const SDSIA* const that);

// Release the memory used by the SDSIASample 'that'
#if BUILDMODE != 0
inline
#endif
void SDSIASampleFreeStatic(SDSIASample* const that);

// Get the current sample in the category 'iCat' of the SDSIA 'that'
// The pointers in the returned SDSIASample may be null
// Example of correct usage to loop through all samples:
// SDSIA* sdsia = ...;
// do {
//   SDSIASample sample = SDSIAGetSample(sdsia, 0);
//   ...
//   SDSIASampleFreeStatic(&sample);
// } while(SDSIAStepSample(sdsia, 0));
#if BUILDMODE != 0
inline
#endif
SDSIASample SDSIAGetSample(const SDSIA* const that, const long iCat);

// If there is a next sample move to the next sample of the category
// 'iCat' and return true, else return false
#if BUILDMODE != 0
inline
#endif
bool SDSIAStepSample(const SDSIA* const that, const long iCat);

// Reset the iterator on category 'iCat' of the SDSIA 'that', i.e.
// the next call to SDSIAGetNextSample will give the first sample of
// the category 'iCat'
#if BUILDMODE != 0
inline
#endif
void SDSIAReset(SDSIA* const that, const long iCat);

// Reset the iterator on all categories of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
void SDSIAResetAll(SDSIA* const that);

// ================= Polymorphism ==================

// ================ Inliner ====================

#if BUILDMODE != 0
#include "sdsia-inline.c"
#endif

#endif
```

2.2 Code

2.2.1 sdsia.c

```
// ============ SDSIA.C ================

// ================= Include =================

#include "sdsia.h"
#if BUILDMODE == 0
#include "sdsia-inline.c"
#endif

// ================= Define ==================

// ================ Functions declaration ====================


// ================ Functions implementation ====================

// Create a new SDSIA with the content of the directory 'folderPath'
// The random generator must have been initialized before calling
// this function
SDSIA SDSIACreateStatic(const char* const folderPath) {
#if BUILDMODE == 0
  if (folderPath == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'folderPath' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Declare the new SDSIA
  SDSIA that;
  // Get the name of the description file
  that._folderPath = PBFSJoinPath(folderPath);
  char* descFileName = PBFSJoinPath(folderPath, SDSIA_DESCFILENAME);
  // Open the description file
  FILE* descFile = fopen(descFileName, "r");
```

```
// If the description file doesn’t exist
if (descFile == NULL) {
  SDSIAErr->_type = PBErrTypeInvalidArg;
  sprintf(SDSIAErr->_msg, "Can’t open the description file");
  PBErrCatch(SDSIAErr);
}
// Declare a json to load the encoded data
JSONNode* json = JSONCreate();
// Load the whole encoded data
if (JSONLoad(json, descFile) == false) {
  printf("%s\n", SDSIAErr->_msg);
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg, "Can’t load the description file");
  PBErrCatch(SDSIAErr);
}
// Decode dataSet
JSONNode* prop = JSONProperty(json, "dataSet");
if (prop == NULL) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (dataSet missing)");
  PBErrCatch(SDSIAErr);
}
JSONNode* val = JSONValue(prop, 0);
that._name = PBErrMalloc(SDSIAErr,
  sizeof(char) * (strlen(JSONLabel(val)) + 1));
strcpy(that._name, JSONLabel(val));
// Decode desc
prop = JSONProperty(json, "desc");
if (prop == NULL) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (desc missing)");
  PBErrCatch(SDSIAErr);
}
val = JSONValue(prop, 0);
that._desc = PBErrMalloc(SDSIAErr,
  sizeof(char) * (strlen(JSONLabel(val)) + 1));
strcpy(that._desc, JSONLabel(val));
// Decode format
prop = JSONProperty(json, "format");
if (prop == NULL) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (format missing)");
  PBErrCatch(SDSIAErr);
}
val = JSONValue(prop, 0);
that._format = PBErrMalloc(SDSIAErr,
  sizeof(char) * (strlen(JSONLabel(val)) + 1));
strcpy(that._format, JSONLabel(val));
// Decode dataSetType
prop = JSONProperty(json, "dataSetType");
if (prop == NULL) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (dataSetType missing)");
  PBErrCatch(SDSIAErr);
}
val = JSONValue(prop, 0);
that._type = atoi(JSONLabel(val));
// Decode dim
```

```
prop = JSONProperty(json, "dim");
if (prop == NULL) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (dim missing)");
  PBErrCatch(SDSIAErr);
}
if (JSONGetNbValue(prop) != 2) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (dim's dimension != 2)");
  PBErrCatch(SDSIAErr);
}
that._dim = VecShortCreateStatic2D();
VecSet(&(that._dim), 0, atoi(JSONLabel(JSONValue(prop, 0))));
VecSet(&(that._dim), 1, atoi(JSONLabel(JSONValue(prop, 1))));
// Decode nbImg
prop = JSONProperty(json, "nbImg");
if (prop == NULL) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (nbImg missing)");
  PBErrCatch(SDSIAErr);
}
val = JSONValue(prop, 0);
int nbImg = atoi(JSONLabel(val));
// Decode images
prop = JSONProperty(json, "images");
if (prop == NULL) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (images missing)");
  PBErrCatch(SDSIAErr);
}
if (JSONGetNbValue(prop) != nbImg) {
  SDSIAErr->_type = PBErrTypeUnitTestFailed;
  sprintf(SDSIAErr->_msg,
    "Invalid description file (images's dimension != nbImg)");
  PBErrCatch(SDSIAErr);
}
that._samples = GSetCreateStatic();
for (int iImg = 0; iImg < nbImg; ++iImg) {
  val = JSONValue(prop, iImg);
  // Allocate memory for the pair image/mask
  SDSIAPairName* pair = PBErrMalloc(SDSIAErr, sizeof(SDSIAPairName));
  // Decode img
  JSONNode* subProp = JSONProperty(val, "img");
  if (subProp == NULL) {
    SDSIAErr->_type = PBErrTypeUnitTestFailed;
    sprintf(SDSIAErr->_msg,
      "Invalid description file (images.img missing)");
    PBErrCatch(SDSIAErr);
  }
  JSONNode* subVal = JSONValue(subProp, 0);
  pair->_imgName = PBErrMalloc(SDSIAErr,
    sizeof(char) * (strlen(JSONLabel(subVal)) + 1));
  strcpy(pair->_imgName, JSONLabel(subVal));
  // Decode mask
  subProp = JSONProperty(val, "mask");
  if (subProp == NULL) {
    SDSIAErr->_type = PBErrTypeUnitTestFailed;
    sprintf(SDSIAErr->_msg,
        "Invalid description file (images.mask missing)");
      PBErrCatch(SDSIAErr);
    }
    subVal = JSONValue(subProp, 0);
    pair->_maskName = PBErrMalloc(SDSIAErr,
      sizeof(char) * (strlen(JSONLabel(subVal)) + 1));
    strcpy(pair->_maskName, JSONLabel(subVal));
    GSetAppend(&(that._samples), pair);
  }
  that._split = NULL;
  that._categories = NULL;
  that._iterators = NULL;
  VecLong* split = VecLongCreate(1);
  VecSet(split, 0, nbImg);
  SDSIASplit(&that, split);
  VecFree(&split);

  // Free the memory used by the JSON
  JSONFree(&json);
  // Close the description file
  fclose(descFile);
  // Free memory
  free(descFileName);

  // Return the new SDSIA
  return that;
}

// Free the memory used by a SDSIA
void SDSIAFreeStatic(SDSIA* const that) {
  if (that == NULL)
    return;
  // Free memory
  free(that->_name);
  free(that->_desc);
  free(that->_format);
  free(that->_folderPath);
  for (int iCat = SDSIAGetNbCat(that); iCat--;) {
    GSetFlush(that->_categories + iCat);
  }
  free(that->_categories);
  free(that->_iterators);
  VecFree(&(that->_split));
  if (SDSIAGetSize(that) > 0) {
    do {
      SDSIAPairName* pair = GSetPop(&(that->_samples));
      if (pair->_imgName != NULL)
        free(pair->_imgName);
      if (pair->_maskName != NULL)
        free(pair->_maskName);
      free(pair);
    } while (SDSIAGetSize(that) > 0);
  }
}

// Split the samples of the SDSIA 'that' into several categories
// defined by 'cat'. The dimension of 'cat' gives the number of
// categories and the value for each dimension of 'cat' gives the
// number of samples in the correpsonding category. For example <3,4>
// would mean 2 categories with 3 samples in the first one and 4
// samples in the second one. There must me at least as many samples
// in the data set as the sum of samples in 'cat'.
// Each category must have at least one sample. Samples are allocated // randomly to the categories.
// If 'that' was already splitted the previous splitting is discarded.
void SDSIASplit(SDSIA* const that, const VecLong* const cat) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  long nb = 0;
  for (long iCat = VecGetDim(cat); iCat--;)
    nb += VecGet(cat, iCat);
  if (nb > SDSIAGetSize(that)) {
    SDSIAErr->_type = PBErrTypeInvalidArg;
    sprintf(PBImgAnalysisErr->_msg,
      "Not enough images for the requested splitting");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Free the current splitting if necessary
  if (that->_categories != NULL) {
    if (that->_split != NULL) {
      for (int iCat = SDSIAGetNbCat(that); iCat--;) {
        GSetFlush(that->_categories + iCat);
      }
    }
    free(that->_categories);
  }
  if (that->_iterators)
    free(that->_iterators);
  VecFree(&(that->_split));
  // Get the number of categories
  long nbCat = VecGetDim(cat);
  // Allocate memory for the categories
  that->_categories = PBErrMalloc(SDSIAErr, sizeof(GSet) * nbCat);
  for (long iCat = nbCat; iCat--;) {
    that->_categories[iCat] = GSetCreateStatic();
  }
  // Copy the splitting
  that->_split = VecClone(cat);
  // Shuffle the samples
  GSetShuffle(&(that->_samples));
  // Declare an iterator on the samples
  GSetIterForward iter = GSetIterForwardCreateStatic(&(that->_samples));
  // Loop on categories
  for (long iCat = nbCat; iCat--;) {
    // Get the nb of samples for this category
    long nbSample = VecGet(cat, iCat);
    // Loop on the sample
    for (long iSample = nbSample; iSample--; GSetIterStep(&iter)) {
      // Get the next sample
      SDSIAPairName* pair = GSetIterGet(&iter);
      // Add the sample to the category
      GSetAppend(that->_categories + iCat, pair);
    }
  }
  // Allocate memory for the iterators
  that->_iterators = PBErrMalloc(SDSIAErr,
    sizeof(GSetIterForward) * nbCat);
  for (long iCat = nbCat; iCat--;) {
    that->_iterators[iCat] =
      GSetIterForwardCreateStatic(that->_categories + iCat);
  }
}
```

2.2.2 sdsia-inline.c

```
// ============ SDSIA_INLINE.C ================

// ================ Functions implementation ====================

// Shuffle the samples of the category 'iCat' of the SDSIA 'that'.
// Reset the iterator of the category
#if BUILDMODE != 0
inline
#endif
void SDSIAShuffle(SDSIA* const that, const long iCat) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (that->_categories == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that->_categories' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (iCat < 0 || iCat >= SDSIAGetNbCat(that)) {
    SDSIAErr->_type = PBErrTypeInvalidArg;
    sprintf(PBImgAnalysisErr->_msg, "'iCat' is invalid (0<=%ld<%ld)",
      iCat, SDSIAGetNbCat(that));
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Shuffle the GSet of the category
  GSetShuffle(that->_categories + iCat);
  // Reset the iterator
  SDSIAReset(that, iCat);
}

// Shuffle the samples of all the categories of the SDSIA 'that'.
// Reset the iterator of the categories
#if BUILDMODE != 0
inline
#endif
void SDSIAShuffleAll(SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Loop on categories
  for (int iCat = SDSIAGetNbCat(that); iCat--;)
    // Shuffle the category
    SDSIAShuffle(that, iCat);
}

// Unsplit the SDSIA 'that', i.e. after calling SDSIAUnsplit 'that'
// has only one category containing all the samples
```

```
#if BUILDMODE != 0
inline
#endif
void SDSIAUnsplit(SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Unsplitting is equivalent to splitting in one category with all the
  // samples
  VecLong* split = VecLongCreate(1);
  VecSet(split, 0, SDSIAGetSize(that));
  SDSIASplit(that, split);
  VecFree(&split);
}

// Get the total number of images in the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
long SDSIAGetSize(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return GSetNbElem(&(that->_samples));
}

// Get the total number of images/masks in the SDSIA 'that' for the
// category 'iCat'. Return 0 if the category doesn't exists
#if BUILDMODE != 0
inline
#endif
long SDSIAGetSizeCat(const SDSIA* const that, const long iCat) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (that->_split == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that->_split' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (iCat < 0 || iCat >= SDSIAGetNbCat(that)) {
    SDSIAErr->_type = PBErrTypeInvalidArg;
    sprintf(PBImgAnalysisErr->_msg, "'iCat' is invalid (0<=%ld<%ld)",
      iCat, SDSIAGetNbCat(that));
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return VecGet(that->_split, iCat);
}

// Get the name of the SDSIA 'that'
```

```
#if BUILDMODE != 0
inline
#endif
const char* SDSIAName(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return that->_name;
}

// Get the description of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
const char* SDSIADesc(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return that->_desc;
}

// Get the format of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
const char* SDSIAFormat(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return that->_format;
}

// Get the path of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
const char* SDSIAFolderPath(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return that->_folderPath;
}

// Get the type of the SDSIA 'that'
#if BUILDMODE != 0
inline
```

```
#endif
int SDSIAGetType(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return that->_type;
}

// Get the number of categories of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
long SDSIAGetNbCat(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return VecGetDim(that->_split);
}

// Get the dimensions of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
VecShort2D SDSIAGetDim(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return that->_dim;
}
#if BUILDMODE != 0
inline
#endif
const VecShort2D* SDSIADim(const SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return &(that->_dim);
}

// Get the current sample in the category 'iCat' of the SDSIA 'that'
// The pointers in the returned SDSIASample may be null
// Example of correct usage to loop through all samples:
// SDSIA* sdsia = ...;
// do {
//   SDSIASample sample = SDSIAGetSample(sdsia, 0);
//   ...
//   SDSIASampleFreeStatic(&sample);
// } while(SDSIAStepSample(sdsia, 0));
#if BUILDMODE != 0
inline
#endif
SDSIASample SDSIAGetSample(const SDSIA* const that, const long iCat) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (that->_iterators == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that->_iterators' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (iCat < 0 || iCat >= SDSIAGetNbCat(that)) {
    SDSIAErr->_type = PBErrTypeInvalidArg;
    sprintf(PBImgAnalysisErr->_msg, "'iCat' is invalid (0<=%ld<%ld)",
      iCat, SDSIAGetNbCat(that));
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  SDSIAPairName* pair = GSetIterGet(that->_iterators + iCat);
  SDSIASample sample;
  char *path = PBFSJoinPath(SDSIAFolderPath(that), pair->_imgName);
  sample._img = GBCreateFromFile(path);
  GBSetFileName(sample._img, path);
  free(path);
  path = PBFSJoinPath(SDSIAFolderPath(that), pair->_maskName);
  sample._mask = GBCreateFromFile(path);
  GBSetFileName(sample._mask, path);
  free(path);
  return sample;
}

// If there is a next sample move to the next sample of the category
// 'iCat' and return true, else return false
#if BUILDMODE != 0
inline
#endif
bool SDSIAStepSample(const SDSIA* const that, const long iCat) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (that->_iterators == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that->_iterators' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (iCat < 0 || iCat >= SDSIAGetNbCat(that)) {
    SDSIAErr->_type = PBErrTypeInvalidArg;
    sprintf(PBImgAnalysisErr->_msg, "'iCat' is invalid (0<=%ld<%ld)",
      iCat, SDSIAGetNbCat(that));
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  return GSetIterStep(that->_iterators + iCat);
```

```
}

// Reset the iterator on category 'iCat' of the SDSIA 'that', i.e.
// the next call to SDSIAGetNextSample will give the first sample of
// the category 'iCat'
#if BUILDMODE != 0
inline
#endif
void SDSIAReset(SDSIA* const that, const long iCat) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (that->_iterators == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that->_iterators' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (iCat < 0 || iCat >= SDSIAGetNbCat(that)) {
    SDSIAErr->_type = PBErrTypeInvalidArg;
    sprintf(PBImgAnalysisErr->_msg, "'iCat' is invalid (0<=%ld<%ld)",
      iCat, SDSIAGetNbCat(that));
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  GSetIterReset(that->_iterators + iCat);
}

// Reset the iterator on all categories of the SDSIA 'that'
#if BUILDMODE != 0
inline
#endif
void SDSIAResetAll(SDSIA* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Loop on categories
  for (int iCat = SDSIAGetNbCat(that); iCat--;)
    // Shuffle the category
    SDSIAReset(that, iCat);
}

// Release the memory used by the SDSIASample 'that'
#if BUILDMODE != 0
inline
#endif
void SDSIASampleFreeStatic(SDSIASample* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    SDSIAErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  GBFree(&(that->_img));
  GBFree(&(that->_mask));
```

```
}
```

2.3 Unit test

```
[-list] OK
[-simul] OK
Generation OK
[-force] OK
UnitTest of generateDataSet.py succeeded
UnitTestPython OK
UnitTestSDSIACreateFree OK
UnitTestSDSIAGet OK
UnitTestSDSIASplitUnsplitShuffle OK
UnitTestSDSIAGetSample OK
```

3 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=1

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=sdsia
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) ‘echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ’ ’ ’\n’ | sort -u‘ $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) ‘echo "$($(repo)_INC_DIR)" | tr ’ ’ ’\n’ | sort -u‘ -c $($(repo)_DIR)/

cleanDataSet:
rm -r DataSets/*

generate:
python generateDataSet.py

regenerate:
python generateDataSet.py -force

simul:
```

```
python generateDataSet.py -simul

help:
python generateDataSet.py -help

listing:
python generateDataSet.py -list

unitTest:
python generateDataSet.py -unitTest

exampleUse:
python exampleUse.py UnitTestOut/001/001/
```

4 Data sets

The repository contains several already generated data sets to be used immediately or as references and examples to make new ones.

4.1 dataset-001-001

dataset-001-001.json:

```
{
  "dataSetType": "0",
  "desc": "A red cube on a white background. Various position and size.",
  "dim": [
    "250",
    "250"
  ],
  "format": "tga",
  "nbImg": "100"
}
```

Example of image and its mask:

dataset-001-001.pov:

```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
  finish { ambient 0 }
}

#declare RndSeed = seed(clock);
#declare _posCamera = <0.0,10.0,0.0>;
#declare _lookAt = <0.0,0.0,0.0>;

camera {
  location    <0.0,10.0,0.0>
  look_at     <0.0,0.0,0.0>
  right x
}

#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

light_source {
  <rnd(-5.0, 5.0), 10.0, rnd(-5.0, 5.0)>
  color rgb 1.0
}

background { color rgb <1.0, 1.0, 1.0> }

#declare Target = box {
  -1, 1
  scale rnd(0.5, 1.5)
  translate x * rnd(-3, 3)
  translate z * rnd(-3, 3)
  #if (Mask = 0)
    pigment { color Red }
  #else
    texture {_texMaskTarget}
  #end
};

object { Target }
```

4.2 dataset-001-002

dataset-001-002.json:

```
{
  "dataSetType": "0",
  "desc": "A red cube on a white background. Various position, size and rotation.",
  "dim": [
    "250",
    "250"
  ],
  "format": "tga",
  "nbImg": "100"
}
```

Example of image and its mask:

dataset-001-002.pov:

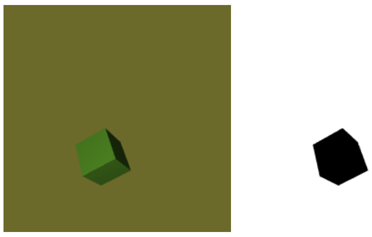
```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
  finish { ambient 0 }
}

#declare RndSeed = seed(clock);
#declare _posCamera = <0.0,10.0,0.0>;
#declare _lookAt = <0.0,0.0,0.0>;

camera {
  location    <0.0,10.0,0.0>
  look_at     <0.0,0.0,0.0>
  right x
}

#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

light_source {
  <rnd(-5.0, 5.0), 10.0, rnd(-5.0, 5.0)>
  color rgb 1.0
}

background { color rgb <1.0, 1.0, 1.0> }

#declare Target = box {
  -1, 1
  scale rnd(0.5, 1.5)
  rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
  translate x * rnd(-3, 3)
  translate z * rnd(-3, 3)
  #if (Mask = 0)
    pigment { color Red }
  #else
    texture {_texMaskTarget}
  #end
};

object { Target }
```

4.3 dataset-001-003

dataset-001-003.json:

```
{
  "dataSetType": "0",
  "desc": "A cube of various uniform color on a background of various uniform color. Various position, size and rotat
  "dim": [
    "250",
    "250"
  ],
  "format": "tga",
  "nbImg": "100"
}
```

Example of image and its mask:



dataset-001-003.pov:

```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
  finish { ambient 0 }
}

#declare RndSeed = seed(clock);
#declare _posCamera = <0.0,10.0,0.0>;
#declare _lookAt = <0.0,0.0,0.0>;

camera {
  location    <0.0,10.0,0.0>
  look_at     <0.0,0.0,0.0>
  right x
}

#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

light_source {
  <rnd(-5.0, 5.0), 10.0, rnd(-5.0, 5.0)>
  color rgb 1.0
}

#declare bgColor = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
#if (Mask = 0)
  background { color rgb bgColor }
#else
  background { color White }
```

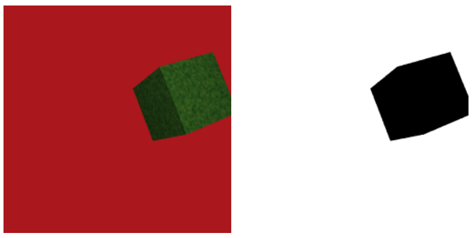
```
#end

#declare Target = box {
  -1, 1
  scale rnd(0.5, 1.5)
  rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
  translate x * rnd(-3, 3)
  translate z * rnd(-3, 3)
  #if (Mask = 0)
    pigment { color rgb <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)> }
  #else
    texture {_texMaskTarget}
  #end
};

object { Target }
```

4.4 dataset-001-004

dataset-001-004.json:

```
{
  "dataSetType": "0",
  "desc": "A cube of various non-uniform color on a background of various uniform color. Various position, size and
  "dim": [
    "250",
    "250"
  ],
  "format": "tga",
  "nbImg": "100"
}
```

Example of image and its mask:



dataset-001-004.pov:

```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
  finish { ambient 0 }
}

#declare RndSeed = seed(clock);
```

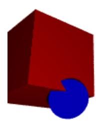
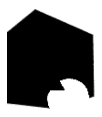
```
#declare _posCamera = <0.0,10.0,0.0>;
#declare _lookAt = <0.0,0.0,0.0>;

camera {
  location    <0.0,10.0,0.0>
  look_at     <0.0,0.0,0.0>
  right x
}

#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

light_source {
  <rnd(-5.0, 5.0), 10.0, rnd(-5.0, 5.0)>
  color rgb 1.0
}

#declare bgColor = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
#if (Mask = 0)
  background { color rgb bgColor }
#else
  background { color White }
#end

#declare Target = box {
  -1, 1
  scale rnd(0.5, 1.5)
  rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
  translate x * rnd(-3, 3)
  translate z * rnd(-3, 3)
  #if (Mask = 0)
    pigment {
      bozo
      scale 0.1
      color_map {
        [0.0 color rgb <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>]
        [1.0 color rgb <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>]
      }
    }
  #else
    texture {_texMaskTarget}
  #end
};

object { Target }
```

4.5 dataset-002-001

dataset-002-001.json:

```
{
  "dataSetType": "0",
  "desc": "A red cube and a blue cylinder on a white background. Various position, size and rotation. The cube is the
  "dim": [
    "250",
    "250"
  ],
  "format": "tga",
  "nbImg": "100"
}
```

Example of image and its mask:

dataset-002-001.pov:

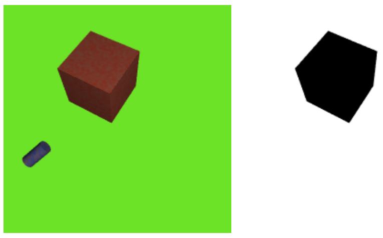
```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
  finish { ambient 0 }
}

#declare _texMaskNonTarget = texture {
  pigment { color White }
  finish { ambient 1 diffuse 100 }
}

#declare RndSeed = seed(clock);

camera {
  location <0.0,10.0,0.0>
  look_at <0.0,0.0,0.0>
  right x
}

#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

light_source {
  <rnd(-5.0, 5.0), 10.0, rnd(-5.0, 5.0)>
  color rgb 1.0
}

background { color rgb <1.0, 1.0, 1.0> }

#declare Target = box {
  -1, 1
  rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
  translate x * rnd(-3, 3)
  translate z * rnd(-3, 3)
  scale rnd(0.5, 1.5)
  #if (Mask = 0)
    pigment { color Red }
  #else
    texture {_texMaskTarget}
  #end
};

#declare NonTarget = cylinder {
  -y, y, 0.5
  scale rnd(0.5, 1.5)
  rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
  translate x * rnd(-5, 5)
  translate z * rnd(-5, 5)
  #if (Mask = 0)
    pigment { color Blue }
  #else
    texture {_texMaskNonTarget}
  #end
}

object { Target }
object { NonTarget }
```

4.6 dataset-002-002

dataset-002-002.json:

```
{
  "dataSetType": "0",
  "desc": "A cube of various non-uniform color and a cylinder of various non-uniform color on a background of variou
  "dim": [
    "250",
    "250"
  ],
  "format": "tga",
  "nbImg": "100"
}
```

Example of image and its mask:



dataset-002-002.pov:

```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
  finish { ambient 0 }
}
```

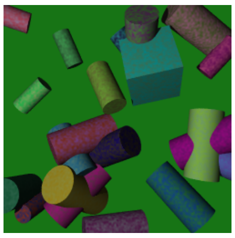
```
#declare _texMaskNonTarget = texture {
  pigment { color White }
  finish { ambient 1 diffuse 100 }
}

#declare RndSeed = seed(clock);

camera {
  location <0.0,10.0,0.0>
  look_at <0.0,0.0,0.0>
  right x
}

#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

light_source {
  <rnd(-5.0, 5.0), 10.0, rnd(-5.0, 5.0)>
  color rgb 1.0
}

#declare bgColor = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
#if (Mask = 0)
  background { color rgb bgColor }
#else
  background { color White }
#end

#declare Target = box {
  -1, 1
  rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
  translate x * rnd(-3, 3)
  translate z * rnd(-3, 3)
  scale rnd(0.5, 1.5)
  #declare colorMapStart = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
  #declare colorMapEnd = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
  #if (Mask = 0)
    pigment {
      bozo
      scale 0.1
      color_map {
        [0.0 color rgb colorMapStart]
        [1.0 color rgb colorMapEnd]
      }
    }
  #else
    texture {_texMaskTarget}
  #end
};

#declare NonTarget = cylinder {
  -y, y, 0.5
  scale rnd(0.5, 1.5)
  rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
  translate x * rnd(-5, 5)
  translate z * rnd(-5, 5)
  #declare colorMapStart = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
  #declare colorMapEnd = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
  #if (Mask = 0)
    pigment {
      bozo
      scale 0.1
      color_map {
        [0.0 color rgb colorMapStart]
        [1.0 color rgb colorMapEnd]
      }
    }
  #else
    texture {_texMaskNonTarget}
  #end
}

object { Target }
object { NonTarget }
```

4.7 dataset-002-003

dataset-002-003.json:

```
{
  "dataSetType": "0",
  "desc": "A cube of various non-uniform color and 20 cylinders of various non-uniform color on a background of vari
  "dim": [
    "250",
    "250"
  ],
  "format": "tga",
  "nbImg": "100"
}
```

Example of image and its mask:



dataset-002-003.pov:

```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
  finish { ambient 0 }
}

#declare _texMaskNonTarget = texture {
  pigment { color White }
  finish { ambient 1 diffuse 100 }
}
```

```
#declare RndSeed = seed(clock);

camera {
  location <0.0,10.0,0.0>
  look_at <0.0,0.0,0.0>
  right x
}

#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

light_source {
  <rnd(-5.0, 5.0), 10.0, rnd(-5.0, 5.0)>
  color rgb 1.0
}

#declare bgColor = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
#if (Mask = 0)
  background { color rgb bgColor }
#else
  background { color White }
#end

#declare Target = box {
  -1, 1
  rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
  translate x * rnd(-3, 3)
  translate z * rnd(-3, 3)
  scale rnd(0.5, 1.5)
  #declare colorMapStart = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
  #declare colorMapEnd = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
  #if (Mask = 0)
    pigment {
      bozo
      scale 0.1
      color_map {
        [0.0 color rgb colorMapStart]
        [1.0 color rgb colorMapEnd]
      }
    }
  #else
    texture {_texMaskTarget}
  #end
};

#declare NonTarget = union {
  #declare i = 0;
  #while (i < 20)
    cylinder {
      -y, y, 0.5
      scale rnd(0.5, 1.5)
      rotate <rnd(-90.0, 90.0), rnd(-90.0, 90.0), rnd(-90.0, 90.0)>
      translate x * rnd(-5, 5)
      translate z * rnd(-5, 5)
      #declare colorMapStart = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
      #declare colorMapEnd = <rnd(0.0, 1.0), rnd(0.0, 1.0), rnd(0.0, 1.0)>;
      #if (Mask = 0)
        pigment {
          bozo
          scale 0.1
          color_map {
            [0.0 color rgb colorMapStart]
            [1.0 color rgb colorMapEnd]
          }
        }
      #else
        texture {_texMaskNonTarget}
      #end
    }
    # declare i = i + 1;
  #end
}

object { Target }
object { NonTarget }
```