SDSIA
Synthetic Data Set for Image Analysis

P. Baillehache

April 14, 2019

Contents

Introduction

SDSIA is a Python module to generate synthetic data sets for image analysis.

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 one or several 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.

If the output format is supported by cv2.imread, the relative coordinates (as expected by Yolo) of the bounding box for each mask are also generated.

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 generated data sets are compatible with the C library GDataSet (type GDataSetType_GenBrushPair).

1 Usage

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.

1.2 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": "1",
  "desc": "unitTest",
  "dim": {
    "_dim":"2",
    "_val":["10", "20"]
  },
  "format": "tga",
  "nbSample": "3",
  "nbMask": "2"
}
```

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 }
}

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

// 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 A definition
#declare TargetA = 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
    #if (Mask = 1)
      texture {_texMaskTarget}
    #else
      #if (Mask = 2)
        texture {_texMaskNonTarget}
      #end
    #end
  #end

}

// Target B definition
#declare TargetB = 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
    #if (Mask = 1)
      texture {_texMaskNonTarget}
    #else
      #if (Mask = 2)
        texture {_texMaskTarget}
      #end
    #end
  #end

}

// Create the scene
object { TargetA }
object { TargetB }
```

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
}
```

1.3 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": "1",
  "desc": "unitTest",
  "dim": {
    "_dim": "2",
    "_val": [
      "10",
      "20"
    ]
  },
  "format": "png",
  "nbMask": "1",
  "nbSample": "3",
  "samples": [
    {
      "bounding": [
        [
          "0.45",
          "0.475",
          "0.4",
          "0.3"
        ]
      ],
      "img": "img000.png",
      "mask": [
        "mask000-000.png"
      ]
    },
    {
      "bounding": [
        [
          "0.5",
          "0.475",
          "0.3",
          "0.3"
        ]
      ],
      "img": "img001.png",
      "mask": [
        "mask001-000.png"
      ]
    },
    {
      "bounding": [
        [
          "0.55",
          "0.475",
          "0.4",
          "0.3"
        ]
      ],
      "img": "img002.png",
      "mask": [
        "mask002-000.png"
      ]
    }
  ]
}
```

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.

1.4 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.

2 Code

2.1 generateDataSet.py

```
# Import necessary modules
import os, sys, json, glob, subprocess, re, shutil, cv2, numpy, platform

# Check the version of Python
version = sys.version_info
if str(version.major) + "." + str(version.minor) < "3.3":
  print("Sorry, require python version >= 3.3")
  quit()

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

# Command line for the POV-Ray executable
if platform.system() == "Linux":
  POVRAY_EXE = "povray"
elif platform.system() == "Windows":
  POVRAY_EXE = "pvengine.exe"
else:
  print("Sorry, unsupported system (" + str(platform.system()) + \
    ")")
  quit()

# Check that the POV-Ray executable is correctly found
if not shutil.which(POVRAY_EXE):
  print("Sorry, the command '" + POVRAY_EXE + \
    "' couldn't be understood. Check your installation of POV-Ray " + \
    "and/or update the value of POVRAY_EXE in " + __file__)
  quit()

# 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
_nbSample = "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._nbSample = dataSetDesc["nbSample"]
    self._dim = dataSetDesc["dim"]
    self._format = dataSetDesc["format"]
    self._nbMask = dataSetDesc["nbMask"]

  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["nbSample"] = self._nbSample
    content["dim"] = self._dim
    content["format"] = self._format
    content["samples"] = self._images
    content["nbMask"] = self._nbMask

    # 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:

    # Flush the result list
    self._images = []

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

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

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

      # 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_EXE)
      cmd.append("+O" + imgFileName)
      cmd.append("-W" + self._dim["_val"][0])
      cmd.append("-H" + self._dim["_val"][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)
# Avoid statying blocked on the terminal displaying info
# when running on windows
if platform.system() == "Windows":
  cmd.append("/EXIT")

# 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._nbSample.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

# Loop on masks
maskFileNames = []
maskBoundingBoxes = []
for iMask in range(int(self._nbMask)):
  iMaskPadded = str(iMask).zfill(3)
  maskFileName = "mask" + iRenderPadded + "-" + iMaskPadded + \
    "." + self._format
  maskFilePath = os.path.join(outFolder, maskFileName)

  # Update the command and the ini file to render the mask
  cmd[1] = "+O" + maskFileName
  cmd[6] = "-Q0"

  with open(iniFilePath, "w") as fp:
    fp.write("Declare=Mask=" + str(iMask + 1))

  # Render the mask silently
  print("        Rendering mask " + maskFilePath + " ...");
  subprocess.call(cmd, stderr = FNULL, cwd = outFolder)

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

  # Load the mask
  imgMask = cv2.imread(maskFilePath, cv2.IMREAD_GRAYSCALE)

  # If we could load the mask
  if not imgMask is None:

    # Get the bounding coordinates of the mask
    yMax, xMax = numpy.max(numpy.where(imgMask != 255), 1)
    yMin, xMin = numpy.min(numpy.where(imgMask != 255), 1)

    # Convert the bounding coordinates into yolo format
    xRelCenter = \
      0.5 * (xMax + xMin) / float(self._dim["_val"][0])
    yRelCenter = \
      0.5 * (yMax + yMin) / float(self._dim["_val"][1])
    widthRel = (xMax - xMin + 1) / float(self._dim["_val"][0])
```

```
            heightRel = (yMax - yMin + 1) / float(self._dim["_val"][1])
            maskBoundingBox = \
              [str(xRelCenter), str(yRelCenter), \
              str(widthRel), str(heightRel)]

            # Add the bounding box
            maskBoundingBoxes.append(maskBoundingBox)

          # Add the mask's name to the array of names
          maskFileNames.append(maskFileName)

        # Remove the ini file
        os.remove(iniFilePath)

        # Add the image and mask to the list
        self._images.append( \
          {"img":imgFileName, "mask":maskFileNames, \
          "bounding":maskBoundingBoxes})

      # 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("generateDataSet.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:
    prefix = "[ ]  "
  else:
    prefix = "[*]  "

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

  # Print the set name and descrition
  print(prefix + 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("dataset2.json",
      os.path.join("UnitTestIn", "dataset-002-001.json"))

    # Test listing
    cmd = []
    cmd.append("python3")
    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("python3")
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.png ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'mask000-000.png ...\n',
  '001/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'img001.png ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'mask001-000.png ...\n',
  '002/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'img002.png ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "001", "") +
  'mask002-000.png ...\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.png ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'mask000-000.png ...\n',
  '001/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'img001.png ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'mask001-000.png ...\n',
  '002/003 Rendering image ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'img002.png ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "001", "002", "") +
  'mask002-000.png ...\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", "") +
  'mask000-000.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'mask000-001.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", "") +
  'mask001-000.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'mask001-001.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", "") +
  'mask002-000.tga ...\n',
  '        Rendering mask ' +
  os.path.join(BASE_DIR, "UnitTestOut", "002", "001", "") +
  'mask002-001.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.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "img001.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "img002.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "mask000-000.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "mask001-000.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "001",
    "mask002-000.png")) 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.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "img001.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "img002.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "mask000-000.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "mask001-000.png")) or \
    not os.path.exists(os.path.join("UnitTestOut", "001", "002",
    "mask002-000.png")) 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",
    "mask000-000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "mask001-000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "mask002-000.tga")) or \
    not os.path.exists(os.path.join("UnitTestOut", "002", "001",
    "mask002-001.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.png ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, \
    "UnitTestOut", "001", "001", "") + 'mask000-000.png ...\n',
  '001/003 Rendering image ' + os.path.join(BASE_DIR, \
    "UnitTestOut", "001", "001", "") + 'img001.png ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, \
    "UnitTestOut", "001", "001", "") + 'mask001-000.png ...\n',
  '002/003 Rendering image ' + os.path.join(BASE_DIR, \
    "UnitTestOut", "001", "001", "") + 'img002.png ...\n',
  '        Rendering mask ' + os.path.join(BASE_DIR, \
    "UnitTestOut", "001", "001", "") + 'mask002-000.png ...\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.png ...\n',
'        Rendering mask ' + os.path.join(BASE_DIR, \
  "UnitTestOut", "001", "002", "") + 'mask000-000.png ...\n',
'001/003 Rendering image ' + os.path.join(BASE_DIR, \
  "UnitTestOut", "001", "002", "") + 'img001.png ...\n',
'        Rendering mask ' + os.path.join(BASE_DIR, \
  "UnitTestOut", "001", "002", "") + 'mask001-000.png ...\n',
'002/003 Rendering image ' + os.path.join(BASE_DIR, \
  "UnitTestOut", "001", "002", "") + 'img002.png ...\n',
'        Rendering mask ' + os.path.join(BASE_DIR, \
  "UnitTestOut", "001", "002", "") + 'mask002-000.png ...\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", "") + 'mask000-000.tga ...\n',
'        Rendering mask ' + os.path.join(BASE_DIR, \
  "UnitTestOut", "002", "001", "") + 'mask000-001.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", "") + 'mask001-000.tga ...\n',
'        Rendering mask ' + os.path.join(BASE_DIR, \
  "UnitTestOut", "002", "001", "") + 'mask001-001.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", "") + 'mask002-000.tga ...\n',
'        Rendering mask ' + os.path.join(BASE_DIR, \
  "UnitTestOut", "002", "001", "") + 'mask002-001.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()
```

2.2 exampleUse.py

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

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

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


# Function to print exceptions
```

```
def print_exc(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(data_set_folder_path):
    """
    Main function
    Inputs:
      'data_set_folder_path': full path to the folder containing the data set
    """
    try:

        # Check if the folder exists
        if not os.path.exists(data_set_folder_path):
            print("The folder {} doesn't exists".format(data_set_folder_path))
            quit()

        # Check if the folder contains the data set description file
        desc_file_path = os.path.join(data_set_folder_path, "dataset.json")
        if not os.path.exists(desc_file_path):
            print("The description file {} doesn't exists".format(desc_file_path))
            quit()

        # Load and decode the content of the description file
        with open(desc_file_path, "r") as fp:
            data_set_desc = json.load(fp)

        # Display info about the data set
        print("Description: {}".format(data_set_desc["desc"]))
        print("Nb image: {}".format(data_set_desc["nbSample"]))
        print("Nb mask: {}".format(data_set_desc["nbMask"]))
        print("Dimension image (width, height): {}".format(data_set_desc["dim"]))
        print("Format image: {}".format(data_set_desc["format"]))

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

            # Get the path to the image and mask
            img_file_name = data_set_desc["samples"][iSample]["img"]
            mask_file_names = data_set_desc["samples"][iSample]["mask"]
            img_file_path = os.path.join(data_set_folder_path, img_file_name)

            # Check the full paths are valid
            if not os.path.exists(img_file_path):
                print("Description file corrupted. The image {} doesn't exists"
                      "".format(img_file_path))
                quit()

            # Loop on masks
            for maskFileName in mask_file_names:

                # Check for the existence of the mask
                mask_file_path = os.path.join(data_set_folder_path, maskFileName)
                if not os.path.exists(mask_file_path):
                    print("Description file corrupted. The mask {} doesn't exists"
                          "".format(mask_file_path))
                    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 ({}, {})".format(img_file_path, mask_file_path))

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

3 Unit test output

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

4 Makefile

```
help:
python3 generateDataSet.py -help

cleanDataSet:
rm -r DataSets/*

generate:
python3 generateDataSet.py

regenerate:
python3 generateDataSet.py -force

simul:
python3 generateDataSet.py -simul

listing:
python3 generateDataSet.py -list

unitTest:
python3 generateDataSet.py -unitTest

exampleUse:
python3 exampleUse.py UnitTestOut/001/001/
```

5 Data sets

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

5.1 dataset-001-001

dataset-001-001.json:

```
{
  "dataSetType": "1",
  "desc": "A red cube on a white background. Various position and size.",
  "dim": {
    "_dim":"2",
    "_val":["250", "250"]
  },
  "format": "tga",
  "nbSample": "100",
  "nbMask" : "1"
}
```

Example of image and its mask:

dataset-001-001.pov:

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

#declare _texMaskTarget = texture {
  pigment { color Black }
}

#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 }
```

5.2 dataset-001-002

dataset-001-002.json:

```
{
  "dataSetType": "1",
  "desc": "A red cube on a white background. Various position, size and rotation.",
  "dim": {
    "_dim":"2",
    "_val":["250", "250"]
  },
  "format": "tga",
  "nbSample": "100",
  "nbMask" : "1"
}
```

Example of image and its mask:

dataset-001-002.pov:

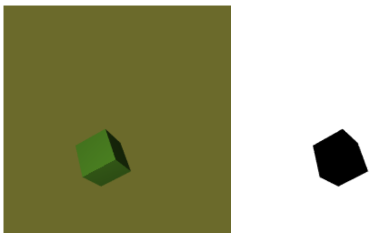
```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
}

#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 }
```

5.3 dataset-001-003

dataset-001-003.json:

```
{
  "dataSetType": "1",
  "desc": "A cube of various uniform color on a background of various uniform color. Various position, size and rotat
  "dim": {
    "_dim":"2",
    "_val":["250", "250"]
  },
  "format": "tga",
  "nbSample": "100",
  "nbMask" : "1"
}
```

Example of image and its mask:



dataset-001-003.pov:

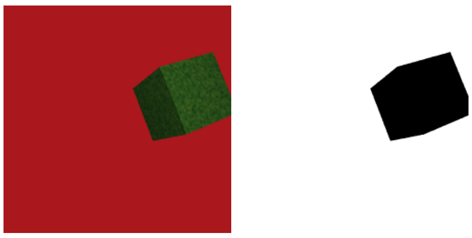
```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
}

#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 }
```

5.4 dataset-001-004

dataset-001-004.json:

```
{
  "dataSetType": "1",
  "desc": "A cube of various non-uniform color on a background of various uniform color. Various position, size and r
  "dim": {
    "_dim":"2",
    "_val":["250", "250"]
  },
  "format": "tga",
  "nbSample": "100",
  "nbMask" : "1"
}
```

Example of image and its mask:



dataset-001-004.pov:

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

#declare _texMaskTarget = texture {
  pigment { color Black }
}

#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)>;
```

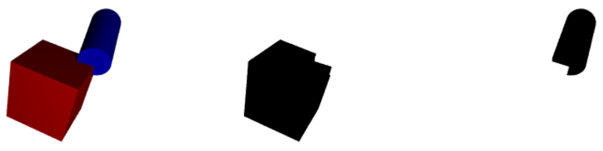
```
#if (Mask = 0)
  background { color rgb bgColor }
#else
  background { color White }
#end

#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 }
```

5.5 dataset-002-001

dataset-002-001.json:

```
{
  "dataSetType": "1",
  "desc": "A red cube and a blue cylinder on a white background. Various position, size and rotation. The cube is the
  "dim": {
    "_dim":"2",
    "_val":["250", "250"]
  },
  "format": "tga",
  "nbSample": "100",
  "nbMask" : "2"
}
```

Example of image and its mask:



dataset-002-001.pov:

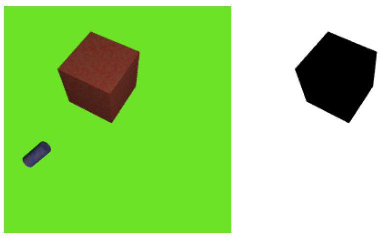
```
#include "colors.inc"
#include "textures.inc"

#declare _texMaskTarget = texture {
  pigment { color Black }
}

#declare _texMaskNonTarget = texture {
  pigment { color White }
}

#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 TargetA = 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
    #if (Mask = 1)
      texture {_texMaskTarget}
    #else
      texture {_texMaskNonTarget}
    #end
  #end
};

#declare TargetB = 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
    #if (Mask = 1)
      texture {_texMaskNonTarget}
    #else
      texture {_texMaskTarget}
    #end
  #end
}

object { TargetA }
object { TargetB }
```

5.6 dataset-002-002

dataset-002-002.json:

```
{
  "dataSetType": "1",
  "desc": "A cube of various non-uniform color and a cylinder of various non-uniform color on a background of variou
  "dim": {
    "_dim":"2",
    "_val":["250", "250"]
  },
  "format": "tga",
  "nbSample": "100",
  "nbMask" : "1"
}
```

Example of image and its mask:



dataset-002-002.pov:

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

#declare _texMaskTarget = texture {
  pigment { color Black }
}

#declare _texMaskNonTarget = texture {
  pigment { color White }
}

#declare RndSeed = seed(clock);

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

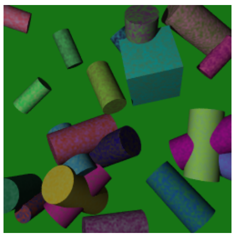
```
#macro rnd(A,B)
  (A+(B-A)*rand(RndSeed))
#end

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 }
```

5.7 dataset-002-003

dataset-002-003.json:

```
{
  "dataSetType": "1",
  "desc": "A cube of various non-uniform color and 20 cylinders of various non-uniform color on a background of vari
  "dim": {
    "_dim":"2",
    "_val":["250", "250"]
  },
  "format": "tga",
  "nbSample": "100",
  "nbMask" : "1"
}
```

Example of image and its mask:



dataset-002-003.pov:

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

#declare _texMaskTarget = texture {
  pigment { color Black }
}

#declare _texMaskNonTarget = texture {
  pigment { color White }
}

#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 }
```