# SIMPA

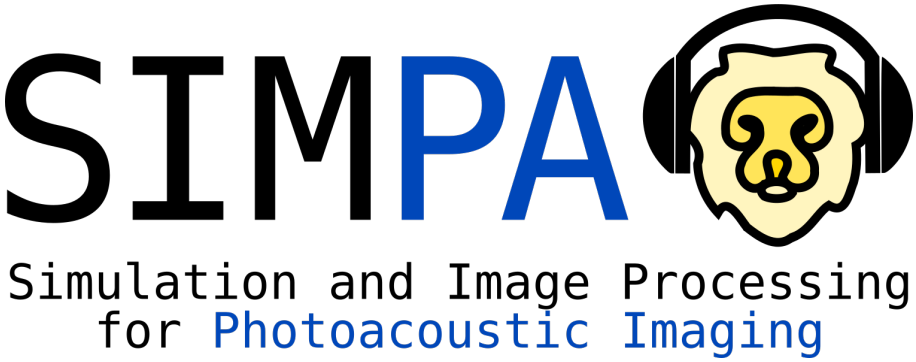
## version 0.1.0

**CAMI (Computer Assisted Medical Interventions), DKFZ, Heidelberg**

January 21, 2021

# Contents

# Welcome to the SIMPA documentation!




# README

The Simulation and Image Processing for Photoacoustic Imaging (SIMPA) toolkit.

## SIMPA Install Instructions

These install instructions are made under the assumption that you have access to the phabricator simpa project. When you are reading these instructions there is a 99% chance that is the case (or someone send these instructions to you).

So, for the 1% of you: Please also follow steps 1 - 3:

1. `git clone https://phabricator.mitk.org/source/simpa.git`
2. `git checkout master`
3. `git pull`

Now open a python instance in the 'simpa' folder that you have just downloaded. Make sure that you have your preferred virtual environment activated

1. `cd simpa`
2. `python -m setup.py build install`
3. Test if the installation worked by using `python` followed by `import simpa` then `exit()`

If no error messages arise, you are now setup to use simpa in your project.

## Building the documentation

When the installation went fine and you want to make sure that you have the latest documentation you should do the following steps in a command line:

1. Navigate to the `simpa` source directory (same level where the setup.py is in)
2. Execute the command `sphinx-build -b pdf -a simpa_documentation/src simpa_documentation`
3. Find the PDF file in `simpa_documentation/simpa_documantation.pdf`

## External Tools installation instructions

### *mcx (Optical Forward Model)*

Either download suitable executables or build yourself from the following sources: http://mcx.space/

### *k-Wave (Acoustic Forward Model)*

Please follow the following steps and use the k-Wave install instructions for further (and much better) guidance under http://www.k-wave.org/!

1. Install MATLAB with the core and parallel computing toolboxes activated at the minimum.
2. Download the kWave toolbox
3. Add the kWave toolbox base bath to the toolbox paths in MATLAB
4. If wanted: Download the CPP and CUDA binary files and place them inthe k-Wave/binaries folder
5. Note down the system path to the `matlab` executable file.

On MATLAB r2020a or newer there is a bug when using the GPU binaries with kWave. Please follow these instructions http://www.k-wave.org/forum/topic/error-reading-h5-files-when-using-binaries to fix this bug.

### *MITK*

## Overview

The main use case for the simpa framework is the simulation of photoacoustic images. However, it can also be used for image processing.

### *Simulating photoacoustic images*

A basic example on how to use simpa in you project to run an optical forward simulation is given in the samples/minimal_optical_simulation.py file.

## Performance profiling

Do you wish to know which parts of the simulation pipeline cost the most amount of time? If that is the case then you can use the following commands to profile the execution of your simulation script. You simply need to replace the `myscript` name with your script name.

```
python -m cProfile -o myscript.cprof myscript.py
```

```
pyprof2calltree -k -i myscript.cprof
```

# Developer Guide

Dear SIMPA developers, Dear person who wants to contribute to the SIMPA toolkit,

First of all: Thank you for your participation and help! It is much appreciated! This Guide is meant to be used as a collection of How-To's to contribute to the framework. In case you have any questions, do not hesitate to get in touch with the members of the core development team:

Kris K. Dreher (k.dreher@dkfz-heidelberg.de)

Janek M. Groehl (janek.grohl@cruk.cam.ac.uk)

## How to contribute

The SIMPA code is written and maintained on a closed repository that is hosted on a server of the German Cancer Research Center. The current master branch of the repository is open source and mirrored on github.

To contribute to SIMPA, please fork the SIMPA github repository and create a pull request with a branch containing your suggested changes. The core team developers will then review the suggested changes and integrate these into the code base.

Please see the github guidelines for creating pull requests: https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

# Coding style

When writing code for SIMPA, please use the PEP 8 python coding conventions (https://www.python.org/dev/peps/pep-0008/) and consider to use the following structures in your code in order to make a new developer or someone external always know exactly what to expect.

- Classnames are written in camel-case notation `ClassName`
- Function names are written in small letter with _ as the delimiter `function_name`
- Function parameters are always annotated with their type `arg1: type = default`
- Only use primitive types as defaults. If a non-primitive type is used, then the default should be `None` and the parameter should be initialized in the beginning of a function.
- A single line of code should not be longer than 120 characters.
- Functions should follow the following simple structure:
    1. Input validation (arguments all not `None`, correct type, and acceptable value ranges?)
    2. Processing (clean handling of errors that might occur)
    3. Output generation (sanity checking of the output before handing it off to the caller)

# Documenting your code

Only documented code will appear in the sphinx generated documentation.

A class should be documented using the following syntax:

```
class ClassName(Superclass):
    """
    Explain how the class is used and what it does.
    """
```

For functions, a lot of extra attributes can be added to the documentation:

```
def function_name(self, arg1:type = default, arg2:type = default) -> return_type:
    """
    Explain how the function is used and what it does.

    :param arg1: type, value range, Null acceptable?
    :param arg2: type, value range, Null acceptable?
    :returns: type, value range, does it return Null?
    :raises ExceptionType: explain when and why this exception is raised
    """
```

# Adding literature absorption spectra

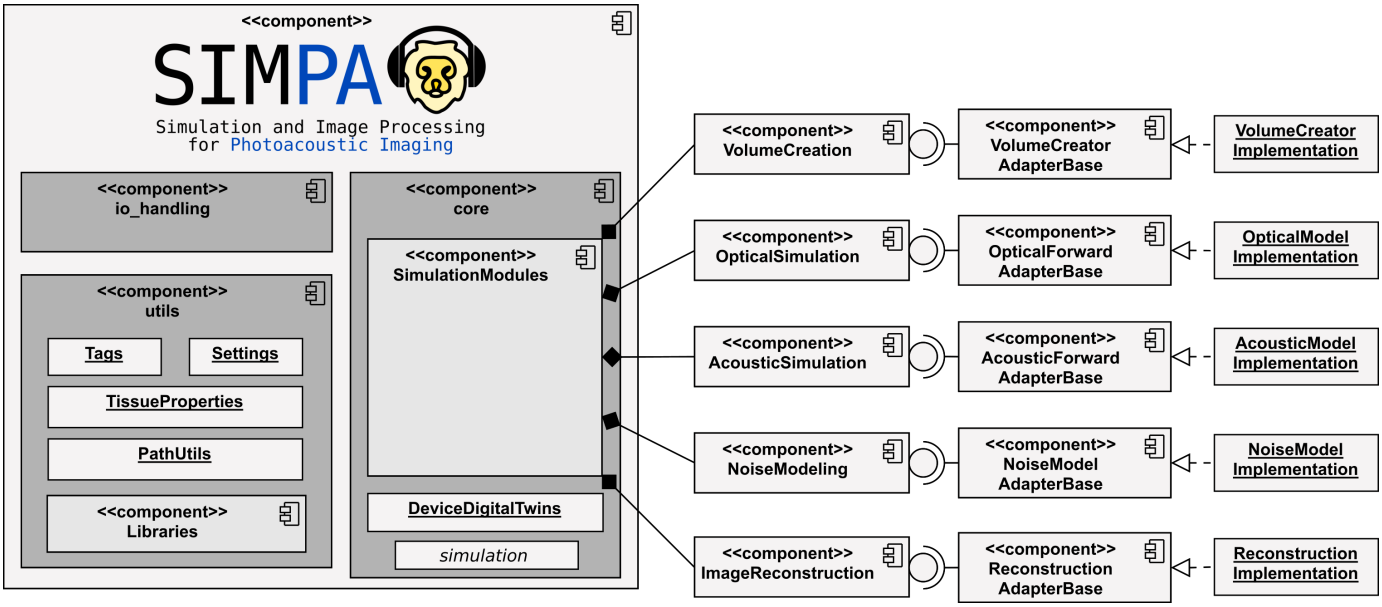
The central point, where absorption spectra are collected and handled is in `simpa.utils.libraries.spectra_library.py`. The file comprises the class `AbsorptionSpectrumLibrary`, in which the new absorption spectra can be added using the following two steps:

1. In the beginning of the class, there is a bunch of constants that define spectra using the `AbsorptionSpectrum` class. Add a new constant here: `NEW_SPECTRUM = AbsorptionSpectrum(absorber_name, wavelengths, absorptions)`. By convention, the naming of the constant should be the same as the `absorber_name` field. The `wavelengths` and `absorptions` arrays must be of the same length and contain corresponding values.
2. In the `__init__` method of the `AbsorptionSpectrumLibrary` class, the class constants are added to an internal list. This has the benefit of enabling the Library class to be iterable. Add your newly added constant field to the list here.

3. Your absorption spectrum is now usable throughout all of simpa and is accessible using the SPECTRAL_LIBRARY sngleton that can be imported using `from simpa.utils import SPECTRAL_LIBRARY`.

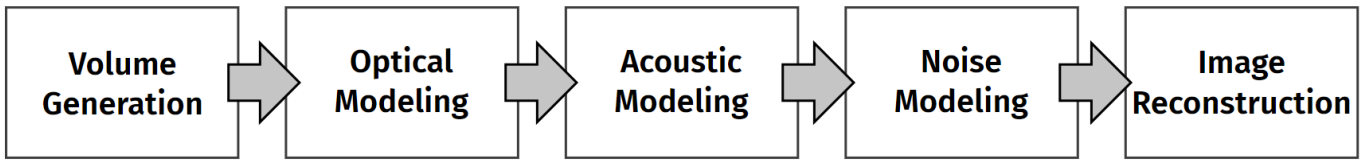
# Class references

This component diagram shows the three principle modules of the SIMPA toolkit and gives an insight into their constituents. The core is concerned with providing interfaces for the simulation tools, while the utils module contains many scripts and classes to facilitate the use of the simulation pipeline.




## Module: core

The purpose of the core module is to provide interfaces that facilitate the integration of toolboxes and code for photoacoustic modeling into a single continuous pipeline:




### *Volume creation*

*class* `simpa.core.volume_creation.`**`VolumeCreatorBase`**
Use this class to define your own volume creation adapter.

***abstract*** **`create_simulation_volume`** **(**`settings: simpa.utils.settings_generator.Settings`**)** → dict
This method will be called to create a simulation volume. @param settings:

### *Optical forward modeling*

*class* `simpa.core.optical_simulation.`**`OpticalForwardAdapterBase`**
Use this class as a base for implementations of optical forward models.

***abstract*** **`forward_model`** **(**`absorption_cm`**,** `scattering_cm`**,** `anisotropy`**,** `settings`**)**
A deriving class needs to implement this method according to its model.

**Parameters:**
- **absorption_cm** – Absorption in units of per centimeter
- **scattering_cm** – Scattering in units of per centimeter
- **anisotropy** – Dimensionless scattering anisotropy
- **settings** – Setting dictionary

**Returns:** Fluence in units of J/cm^2

**simulate**(optical_properties_path, settings)

Call this method to invoke the simulation process.
A adapter that implements the forward_model method, will take optical properties of absorption, scattering, and scattering anisotropy as input and return the light fluence as output.

**Parameters:**
- **optical_properties_path** – path to a *.npz file that contains the following tags: Tags.PROPERTY_ABSORPTION_PER_CM -> contains the optical absorptions in units of one per centimeter Tags.PROPERTY_SCATTERING_PER_CM -> contains the optical scattering in units of one per centimeter Tags.PROPERTY_ANISOTROPY -> contains the dimensionless optical scattering anisotropy
- **settings** –

**Returns:**

## Acoustic forward modeling

*class* simpa.core.acoustic_simulation.**AcousticForwardAdapterBase**

Use this class as a base for implementations of optical forward models.

***abstract*** **forward_model**(settings)

A deriving class needs to implement this method according to its model.

**Parameters:** **settings** – Setting dictionary

**Returns:** Fluence in units of J/cm^2

**simulate**(optical_properties_path, settings)

Call this method to invoke the simulation process. TODO
A adapter that implements the forward_model method, will take acoustic properties as input and return the time series pressure data as output.

**Parameters:**
- **optical_properties_path** – path to a *.npz file that contains the following tags: Tags.PROPERTY_ABSORPTION_PER_CM -> contains the optical absorptions in units of one per centimeter Tags.PROPERTY_SCATTERING_PER_CM -> contains the optical scattering in units of one per centimeter Tags.PROPERTY_ANISOTROPY -> contains the dimensionless optical scattering anisotropy
- **settings** –

**Returns:**

## Noise modeling

*class* simpa.core.noise_simulation.**GaussianNoiseModel**

This class is reponsible to apply an additive gaussian noise to the input data.

**apply_noise_model**(data, settings)

**Parameters:**
- **data** –
- **settings** –

**Returns:**

## Image reconstruction

## Digital device twins

At every step along the forward simulation, knowledge of the photoacoustic device that is used for the measurements is needed. This is important to reflect characteristic artefacts and challenges for the respective device.

To this end, we have included digital twins of commonly used devices into the SIMPA core.

### MSOT Acuity Echo

*class* simpa.core.device_digital_twins.msot_devices.**MSOTAcuityEcho**

TODO

**adjust_simulation_volume_and_settings**(global_settings: simpa.utils.settings_generator.Settings)

In case that the PAI device needs space for the arrangement of detectors or illuminators in the volume, this method will update the volume accordingly.

**check_settings_prerequisites**(global_settings: simpa.utils.settings_generator.Settings) → bool

It might be that certain device geometries need a certain dimensionality of the simulated PAI volume, or that it required the existence of certain Tags in the global global_settings. To this end, a PAI device should use this method to inform the user about a mismatch of the desired device and throw a ValueError if that is the case.

**Raises:** **ValueError** – raises a value error if the prerequisites are not matched.

:returns : True if the prerequisites are met.

**get_detector_element_orientations**(global_settings: simpa.utils.settings_generator.Settings)

TODO

**get_detector_element_positions_accounting_for_device_position_mm**(global_settings: simpa.utils.settings_generator.Settings)

TODO

**get_detector_element_positions_base_mm**()

TODO

**get_illuminator_definition**(global_settings: simpa.utils.settings_generator.Settings)

TODO

### RSOM Explorer P50

*class* simpa.core.device_digital_twins.rsom_device.**RSOMExplorerP50**(element_spacing_mm=0.02)

This class represents an approximation of the Raster-scanning Optoacoustic Mesoscopy (RSOM) device built by iThera Medical (Munich, Germany). Please refer to the companie's website for more information (https://www.ithera-medical.com/products/rsom-explorer-p50/). <br> <br> Since simulating thousands of individual forward modeling steps to obtain a single raster-scanned image is computationally not feasible, we approximate the process with a device design that has detection elements across the entire field of view. Because of this limitation we also need to approximate the light source with a homogeneous illumination across the field of view. <br> <br> The digital device is modeled based on the reported specifications of the RSOM Explorer P50 system.

Technical details of the system can be found in the dissertation of Mathias Schwarz (https://mediatum.ub.tum.de/doc/1324031/1324031.pdf) and you can find more details on use cases of the device in the following literature sources:
Yew, Yik Weng, et al. “Raster-scanning optoacoustic mesoscopy (RSOM) imaging as an objective disease severity tool in atopic dermatitis patients.” Journal of the American Academy of Dermatology (2020).
Hindelang, B., et al. “Non■invasive imaging in dermatology and the unique potential of raster■scan optoacoustic mesoscopy.” Journal of the European Academy of Dermatology and Venereology 33.6 (2019): 1051-1061.

**adjust_simulation_volume_and_settings** **(**`global_settings: simpa.utils.settings_generator.Settings`**)**
In case that the PAI device needs space for the arrangement of detectors or illuminators in the volume, this method will update the volume accordingly.

**check_settings_prerequisites** **(**`global_settings: simpa.utils.settings_generator.Settings`**)** → bool
It might be that certain device geometries need a certain dimensionality of the simulated PAI volume, or that it required the existence of certain Tags in the global global_settings. To this end, a PAI device should use this method to inform the user about a mismatch of the desired device and throw a ValueError if that is the case.

**Raises:** **ValueError** – raises a value error if the prerequisites are not matched.

:returns : True if the prerequisites are met.

**get_detector_element_orientations** **(**`global_settings: simpa.utils.settings_generator.Settings`**)**
TODO

**get_detector_element_positions_accounting_for_device_position_mm** **(**`global_settings: simpa.utils.settings_generator.Settings`**)**
TODO

**get_detector_element_positions_base_mm** **()**
TODO

**get_illuminator_definition** **(**`global_settings: simpa.utils.settings_generator.Settings`**)**
TODO

## Module: utils

*class* `simpa.utils.libraries.literature_values.`**MorphologicalTissueProperties**
This class contains a listing of morphological tissue parameters as reported in literature. The listing is not the result of a meta analysis, but rather uses the best fitting paper at the time pf implementation. Each of the fields is annotated with a literature reference or a descriptions of how the particular values were derived for tissue modelling.

*class* `simpa.utils.libraries.literature_values.`**OpticalTissueProperties**
This class contains a listing of optical tissue parameters as reported in literature. The listing is not the result of a meta analysis, but rather uses the best fitting paper at the time pf implementation. Each of the fields is annotated with a literature reference or a descriptions of how the particular values were derived for tissue modelling.

*class* `simpa.utils.libraries.literature_values.`**StandardProperties**
This class contains a listing of default parameters that can be used. These values are sensible default values but are generally not backed up by proper scientific references, or are rather specific for internal use cases.

*class* `simpa.utils.libraries.spectra_library.`**AbsorptionSpectrum** (`spectrum_name: str, wavelengths: numpy.ndarray, absorption_per_centimeter: numpy.ndarray`)
An instance of this class represents the absorption spectrum over wavelength for a particular

**get_absorption_for_wavelength** **(**`wavelength: int`**)** → float

**Parameters:** **wavelength** – the wavelength to retrieve a optical absorption value for [cm^{-1}]. Must be an integer value between the minimum and maximum wavelength.

**Returns:** the best matching linearly interpolated absorption value for the given wavelength.

**get_absorption_over_wavelength ()**

**Returns:** numpy array with the available wavelengths and the corresponding absorption properties

simpa.utils.libraries.spectra_library.**view_absorption_spectra** (save_path=None)
Opens a matplotlib plot and visualizes the available absorption spectra.

**Parameters:** **save_path** – If not None, then the figure will be saved as a png file to the destination.

*class* simpa.utils.libraries.tissue_library.**MolecularCompositionGenerator**
The MolecularCompositionGenerator is a helper class to facilitate the creation of a MolecularComposition instance.

*class* simpa.utils.libraries.tissue_library.**TissueLibrary**
TODO

**blood_arterial ()**

**Returns:** a settings dictionary containing all min and max parameters fitting for full blood.

**blood_generic (**oxygenation=None**)**

**Returns:** a settings dictionary containing all min and max parameters fitting for full blood.

**blood_venous ()**

**Returns:** a settings dictionary containing all min and max parameters fitting for full blood.

**bone ()**

**Returns:** a settings dictionary containing all min and max parameters fitting for full blood.

**constant (**mua**,** mus**,** g**)**
TODO

**dermis (**background_oxy=0.5**)**

**Returns:** a settings dictionary containing all min and max parameters fitting for dermis tissue.

**epidermis ()**

**Returns:** a settings dictionary containing all min and max parameters fitting for epidermis tissue.

**get_blood_volume_fractions (**total_blood_volume_fraction**,** oxygenation**)**
TODO

**muscle (**background_oxy=0.5**)**

**Returns:** a settings dictionary containing all min and max parameters fitting for generic background tissue.

**subcutaneous_fat (**background_oxy=0.5**)**

**Returns:** a settings dictionary containing all min and max parameters fitting for subcutaneous fat tissue.

## Module: io_handling

simpa.io_handling.io_hdf5.**load_hdf5**(file_path, file_dictionary_path='/')
Loads a dictionary from an hdf5 file.

**Parameters:**
- **file_path** – Path of the file to load the dictionary from.
- **file_dictionary_path** – Path in dictionary structure of hdf5 file to lo the dictionary in.

**Returns:** Dictionary

simpa.io_handling.io_hdf5.**save_hdf5**(dictionary: dict, file_path: str, file_dictionary_path: str = '/', file_compression: str = None)
Saves a dictionary with arbitrary content to an hdf5-file with given filepath.

**Parameters:**
- **dictionary** – Dictionary to save.
- **file_path** – Path of the file to save the dictionary in.
- **file_dictionary_path** – Path in dictionary structure of existing hdf5 file to store the dictionary in.
- **file_compression** – possible file compression for the hdf5 output file. Values are: gzip, lzf and szip.

**Returns:** `Null`

# Examples

## Performing a complete forward simulation with acoustic modeling, optical modeling, as well as image reconstruction

The file can be found in simpa_examples/minimal_optical_simulation.py:

```
from simpa.utils import Tags, TISSUE_LIBRARY

from simpa.core.simulation import simulate
from simpa.utils.settings_generator import Settings
from simpa.utils.libraries.structure_library import HorizontalLayerStructure
import numpy as np

# TODO change these paths to the desired executable and save folder
SAVE_PATH = "D:/save/"
MCX_BINARY_PATH = "D:/bin/Release/mcx.exe"

VOLUME_TRANSDUCER_DIM_IN_MM = 75
VOLUME_PLANAR_DIM_IN_MM = 20
VOLUME_HEIGHT_IN_MM = 25
SPACING = 0.15
RANDOM_SEED = 4711


def create_example_tissue():
    """
    This is a very simple example script of how to create a tissue definition.
    It contains a muscular background, an epidermis layer on top of the muscles
    and a blood vessel.
    """
    background_dictionary = Settings()
    background_dictionary[Tags.MOLECULE_COMPOSITION] = TISSUE_LIBRARY.muscle()
    background_dictionary[Tags.STRUCTURE_TYPE] = Tags.BACKGROUND

    muscle_dictionary = Settings()
    muscle_dictionary[Tags.PRIORITY] = 1
```

```
    muscle_dictionary[Tags.STRUCTURE_START_MM] = [0, 0, 0]
    muscle_dictionary[Tags.STRUCTURE_END_MM] = [0, 0, 100]
    muscle_dictionary[Tags.MOLECULE_COMPOSITION] = TISSUE_LIBRARY.muscle()
    muscle_dictionary[Tags.CONSIDER_PARTIAL_VOLUME] = True
    muscle_dictionary[Tags.ADHERE_TO_DEFORMATION] = True
    muscle_dictionary[Tags.STRUCTURE_TYPE] = Tags.HORIZONTAL_LAYER_STRUCTURE

    vessel_1_dictionary = Settings()
    vessel_1_dictionary[Tags.PRIORITY] = 3
    vessel_1_dictionary[Tags.STRUCTURE_START_MM] = [VOLUME_TRANSDUCER_DIM_IN_MM/2,
                                                    0, 10]
    vessel_1_dictionary[Tags.STRUCTURE_END_MM] = [VOLUME_TRANSDUCER_DIM_IN_MM/2, VOLUME_PLAN
    vessel_1_dictionary[Tags.STRUCTURE_RADIUS_MM] = 3
    vessel_1_dictionary[Tags.MOLECULE_COMPOSITION] = TISSUE_LIBRARY.blood_generic()
    vessel_1_dictionary[Tags.CONSIDER_PARTIAL_VOLUME] = True
    vessel_1_dictionary[Tags.STRUCTURE_TYPE] = Tags.CIRCULAR_TUBULAR_STRUCTURE

    epidermis_dictionary = Settings()
    epidermis_dictionary[Tags.PRIORITY] = 8
    epidermis_dictionary[Tags.STRUCTURE_START_MM] = [0, 0, 0]
    epidermis_dictionary[Tags.STRUCTURE_END_MM] = [0, 0, 1]
    epidermis_dictionary[Tags.MOLECULE_COMPOSITION] = TISSUE_LIBRARY.epidermis()
    epidermis_dictionary[Tags.CONSIDER_PARTIAL_VOLUME] = True
    epidermis_dictionary[Tags.ADHERE_TO_DEFORMATION] = True
    epidermis_dictionary[Tags.STRUCTURE_TYPE] = Tags.HORIZONTAL_LAYER_STRUCTURE

    tissue_dict = Settings()
    tissue_dict[Tags.BACKGROUND] = background_dictionary
    tissue_dict["muscle"] = muscle_dictionary
    tissue_dict["epidermis"] = epidermis_dictionary
    tissue_dict["vessel_1"] = vessel_1_dictionary
    return tissue_dict

# Seed the numpy random configuration prior to creating the global_settings file in
# order to ensure that the same volume
# is generated with the same random seed every time.

np.random.seed(RANDOM_SEED)

settings = {
    # These parameters set the general propeties of the simulated volume
    Tags.RANDOM_SEED: RANDOM_SEED,
    Tags.VOLUME_NAME: "CompletePipelineTestMSOT_"+str(RANDOM_SEED),
    Tags.SIMULATION_PATH: SAVE_PATH,
    Tags.SPACING_MM: SPACING,
    Tags.DIM_VOLUME_Z_MM: VOLUME_HEIGHT_IN_MM,
    Tags.DIM_VOLUME_X_MM: VOLUME_TRANSDUCER_DIM_IN_MM,
    Tags.DIM_VOLUME_Y_MM: VOLUME_PLANAR_DIM_IN_MM,
    Tags.VOLUME_CREATOR: Tags.VOLUME_CREATOR_VERSATILE,
    Tags.SIMULATE_DEFORMED_LAYERS: True,
    # Tags.DEFORMED_LAYERS_SETTINGS: create_deformation_settings([[0, VOLUME_TRANSDUCER_DIM_
    #                                                             [0, VOLUME_PLANAR_DIM_IN_MM
    #                                                             maximum_z_elevation_mm=10,
    #                                                             filter_sigma=0,
    #                                                             cosine_scaling_factor=1),

    # Simulation Device
    Tags.DIGITAL_DEVICE: Tags.DIGITAL_DEVICE_MSOT,

    # The following parameters set the optical forward model
```

```
    Tags.RUN_OPTICAL_MODEL: True,
    Tags.WAVELENGTHS: [700],
    Tags.OPTICAL_MODEL_NUMBER_PHOTONS: 1e7,
    Tags.OPTICAL_MODEL_BINARY_PATH: MCX_BINARY_PATH,
    Tags.OPTICAL_MODEL: Tags.OPTICAL_MODEL_MCX,
    Tags.ILLUMINATION_TYPE: Tags.ILLUMINATION_TYPE_MSOT_ACUITY_ECHO,
    Tags.LASER_PULSE_ENERGY_IN_MILLIJOULE: 50,

    # The following parameters tell the script that we do not want any extra
    # modelling steps
    Tags.RUN_ACOUSTIC_MODEL: True,
    Tags.ACOUSTIC_SIMULATION_3D: False,
    Tags.ACOUSTIC_MODEL: Tags.ACOUSTIC_MODEL_K_WAVE,
    Tags.ACOUSTIC_MODEL_BINARY_PATH: "C:/Program Files/MATLAB/R2020b/bin/matlab.exe",
    Tags.ACOUSTIC_MODEL_SCRIPT_LOCATION: "C:/simpa/simpa/core/acoustic_simulation",
    Tags.GPU: True,

    Tags.MEDIUM_ALPHA_POWER: 1.05,

    Tags.SENSOR_RECORD: "p",
    # Tags.SENSOR_DIRECTIVITY_PATTERN: "pressure",

    Tags.PMLInside: False,
    Tags.PMLSize: [31, 32],
    Tags.PMLAlpha: 1.5,
    Tags.PlotPML: False,
    Tags.RECORDMOVIE: False,
    Tags.MOVIENAME: "visualization_log",
    Tags.ACOUSTIC_LOG_SCALE: True,

    Tags.APPLY_NOISE_MODEL: False,
    Tags.SIMULATION_EXTRACT_FIELD_OF_VIEW: True,

    Tags.PERFORM_IMAGE_RECONSTRUCTION: True,
    Tags.RECONSTRUCTION_ALGORITHM: Tags.RECONSTRUCTION_ALGORITHM_BACKPROJECTION
}
settings = Settings(settings)
# global_settings[Tags.SIMULATE_DEFORMED_LAYERS] = True
np.random.seed(RANDOM_SEED)

settings[Tags.STRUCTURES] = create_example_tissue()
print("Simulating ", RANDOM_SEED)
import time
timer = time.time()
simulate(settings)
print("Needed", time.time()-timer, "seconds")
print("Simulating ", RANDOM_SEED, "[Done]")
```

## Reading the HDF5 simulation output

The file can be found in simpa_examples/access_saved_PAI_data.py:

```
from simpa.io_handling import load_hdf5, save_hdf5
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
from simpa.utils import SegmentationClasses, Tags
from simpa.utils.settings_generator import Settings

values = []
```

```
names = []

for string in SegmentationClasses.__dict__:
    if string[0:2] != "__":
        values.append(SegmentationClasses.__dict__[string])
        names.append(string)

values = np.asarray(values)
names = np.asarray(names)
sort_indexes = np.argsort(values)
values = values[sort_indexes]
names = names[sort_indexes]

colors = [list(np.random.random(3)) for _ in range(len(names))]
cmap = mpl.colors.LinearSegmentedColormap.from_list(
    'Custom cmap', colors, len(names))

PATH = "D:/save/LNetOpticalForward_planar_0.hdf5"
WAVELENGTH = 532

file = load_hdf5(PATH)
settings = Settings(file["settings"])

fluence = (file['simulations']['original_data']['optical_forward_model_output']
           [str(WAVELENGTH)]['fluence'])
initial_pressure = (file['simulations']['original_data']
                    ['optical_forward_model_output']
                    [str(WAVELENGTH)]['initial_pressure'])
absorption = (file['simulations']['original_data']['simulation_properties']
              [str(WAVELENGTH)]['mua'])

segmentation = (file['simulations']['original_data']['simulation_properties']
              [str(WAVELENGTH)]['seg'])

reconstruction = None
speed_of_sound = None
if Tags.PERFORM_IMAGE_RECONSTRUCTION in settings and settings[Tags.PERFORM_IMAGE_RECONSTRUCT
    time_series = np.squeeze(
        file["simulations"]["original_data"]["time_series_data"][str(WAVELENGTH)]["time_seri
    reconstruction = np.squeeze(
            file["simulations"]["original_data"]["reconstructed_data"][str(WAVELENGTH)]["rec

    speed_of_sound = file['simulations']['original_data']['simulation_properties'][str(WAVEL

reconstruction = reconstruction.T

shape = np.shape(reconstruction)

x_pos = int(shape[0]/2)
y_pos = int(shape[1]/2)
z_pos = int(shape[2]/2)

plt.figure()
plt.subplot(161)
plt.imshow(np.fliplr(np.rot90(reconstruction[x_pos, :, :], -1)))
plt.subplot(162)
plt.imshow(np.rot90(np.log10(initial_pressure[x_pos, :, :]), -1))
plt.subplot(163)
plt.imshow(np.fliplr(np.rot90(reconstruction[:, y_pos, :], -1)))
plt.subplot(164)
```

```
plt.imshow(np.rot90(np.log10(initial_pressure[:, y_pos, :]), -1))
plt.subplot(165)
plt.imshow(np.fliplr(np.rot90(reconstruction[:, :, z_pos], -1)))
plt.subplot(166)
plt.imshow(np.rot90(np.log10(initial_pressure[:, :, z_pos]), -3))
plt.show()
exit()

if Tags.PERFORM_IMAGE_RECONSTRUCTION in settings and settings[Tags.PERFORM_IMAGE_RECONSTRUCT
    if len(shape) > 2:
        plt.figure()
        plt.subplot(141)
        plt.imshow(np.rot90(np.log10(np.log10(time_series[:, :]-np.min(time_series))), -1),
        plt.subplot(142)
        plt.imshow(np.rot90((reconstruction[:, y_pos, :]), -2))
        plt.subplot(143)
        plt.imshow(np.rot90(np.log10(initial_pressure[:, y_pos, :]), -1))
        plt.subplot(144)
        plt.imshow(np.rot90(segmentation[:, y_pos, :], -1), vmin=values[0], vmax=values[-1],
        plt.show()
    else:
        plt.figure()
        plt.subplot(141)
        plt.imshow(np.rot90((reconstruction[:, :]), -1))
        plt.subplot(142)
        plt.imshow(np.rot90((speed_of_sound), -1))
        plt.subplot(143)
        plt.imshow(np.rot90(np.log10(initial_pressure), -1))
        plt.subplot(144)
        plt.imshow(np.rot90(segmentation, -1), vmin=values[0], vmax=values[-1], cmap=cmap)
        plt.show()
else:
    if len(shape) > 2:
        plt.figure()
        plt.subplot(241)
        plt.title("Fluence")
        plt.imshow(np.rot90((fluence[x_pos, :, :]), -1))
        plt.subplot(242)
        plt.title("Absorption")
        plt.imshow(np.rot90(np.log10(absorption[x_pos, :, :]), -1))
        plt.subplot(243)
        plt.title("Initial Pressure")
        plt.imshow(np.rot90(np.log10(initial_pressure[x_pos, :, :]), -1))
        plt.subplot(244)
        plt.title("Segmentation")
        plt.imshow(np.rot90(segmentation[x_pos, :, :], -1), vmin=values[0], vmax=values[-1],
        cbar = plt.colorbar(ticks=values)
        cbar.ax.set_yticklabels(names)
        plt.subplot(245)
        plt.imshow(np.rot90(fluence[:, y_pos, :], -1))
        plt.subplot(246)
        plt.imshow(np.rot90(np.log10(absorption[:, y_pos, :]), -1))
        plt.subplot(247)
        plt.imshow(np.rot90(np.log10(initial_pressure[:, y_pos, :]), -1))
        plt.subplot(248)
        plt.imshow(np.rot90(segmentation[:, y_pos, :], -1), vmin=values[0], vmax=values[-1],
        cbar = plt.colorbar(ticks=values)
        cbar.ax.set_yticklabels(names)
        plt.show()
    else:
```

```
        plt.figure()
        plt.subplot(141)
        plt.imshow(np.rot90(np.log10(fluence), -1))
        plt.subplot(142)
        plt.imshow(np.rot90(np.log10(absorption), -1))
        plt.subplot(143)
        plt.imshow(np.rot90(np.log10(initial_pressure), -1))
        plt.subplot(144)
        plt.imshow(np.rot90(segmentation, -1))
        plt.show()
```

## Defining custom tissue structures and properties

The file can be found in simpa_examples/create_custom_tissues.py:

```
from simpa.utils import MolecularCompositionGenerator
from simpa.utils import MOLECULE_LIBRARY
from simpa.utils import Molecule
from simpa.utils import AbsorptionSpectrum
import numpy as np


def create_custom_absorber():
    wavelengths = np.linspace(200, 1500, 100)
    absorber = AbsorptionSpectrum(spectrum_name="random absorber",
                                  wavelengths=wavelengths,
                                  absorption_per_centimeter=np.random.random(
                                      np.shape(wavelengths)))
    return absorber


def create_custom_chromophore(volume_fraction: float = 1.0):
    chromophore = Molecule(
            spectrum=create_custom_absorber(),
            volume_fraction=volume_fraction,
            mus500=40.0,
            b_mie=1.1,
            f_ray=0.9,
            anisotropy=0.9
        )
    return chromophore


def create_custom_tissue_type():

    # First create an instance of a TissueSettingsGenerator
    tissue_settings_generator = MolecularCompositionGenerator()

    water_volume_fraction = 0.4
    bvf = 0.5
    oxy = 0.4

    # Then append chromophores that you want
    tissue_settings_generator.append(key="oxyhemoglobin", value=
                            MOLECULE_LIBRARY.oxyhemoglobin(oxy * bvf))
    tissue_settings_generator.append(key="deoxyhemoglobin", value=
                            MOLECULE_LIBRARY.deoxyhemoglobin(oxy * bvf))
    tissue_settings_generator.append(key="water", value=
                            MOLECULE_LIBRARY.water(water_volume_fraction))
    tissue_settings_generator.append(key="custom", value=
```

```
                            create_custom_chromophore(0.1))

    return tissue_settings_generator.get_settings()
```

# Index

# Python Module Index

## s