Open PaperOpt

A Monte Carlo simulation tool for simulating light scattering in paper and prints

Technical Description

Version 1 – 2010/01/11

Table of Contents

1 GNU General Public License (GPL)

Open PaperOpt is a free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; version 3 of the License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with *Open PaperOpt*. If not, see http://www.gnu.org/licenses

2 Scope

Open PaperOpt is a C++ Open Source Monte Carlo simulation program designed for calculating light scattering in paper and board. It describes the simulation volume as a three-dimensional layered structure that includes rough surfaces, homogeneous scattering layers, and structured layers. The structured layers describe composite layers and simulate light scattering from a system of geometric components such as hollow flattened cylinders and ellipsoids representing e.g. fibres, pigments, or pores. The program treats the incident light as indivisible wave packets and calculates their path according to physical rules and some semi-empirical approximations. Setting the distribution of the initial position, polarisation, and direction of the wave packets allows the simulation of virtually any light source. In the same way, detectors with defined geometry and response characteristics can collect the reflected and transmitted wave packets.

This document describes the program structure and coding conventions, external libraries, and file formats used by the program. It is intended for developers who wish to implement new objects or physical models, as a complement to the *Doxygen* (see *Error: Reference source not found*) generated documentation, which describes classes and methods in detail. The input specifications and output results are saved in binary, HDF5, and XML files. The user of the program can jump to the section 5 on file formats. This document does not however describe the physics involved in the different processes.

3 Program overview and main classes

Light is represented as indivisible wave packets in the **WavePacket** class. The **Main** program starts by initialising the random generator and calling the **Input** class for parsing the inputs and generating the simulated structure, light sources and detectors as a **SimVolume** class. The program goes then through three main loops simulating light scattering for each wavelength, light source, and wave packet. The initial state of the wave packet is generated by the **LightSource** class. The wave packet is then sent onto the simulated structure in the ***Process*** method of the **SimVolume** class and returned as absorbed, transmitted, reflected, of in error state. The main function updates then the simulated reflectance and transmittance and sends the wave packet to the different **Detectors** defined in **SimVolume**. The main program can also be called to generate static layers (see 3.3.4).

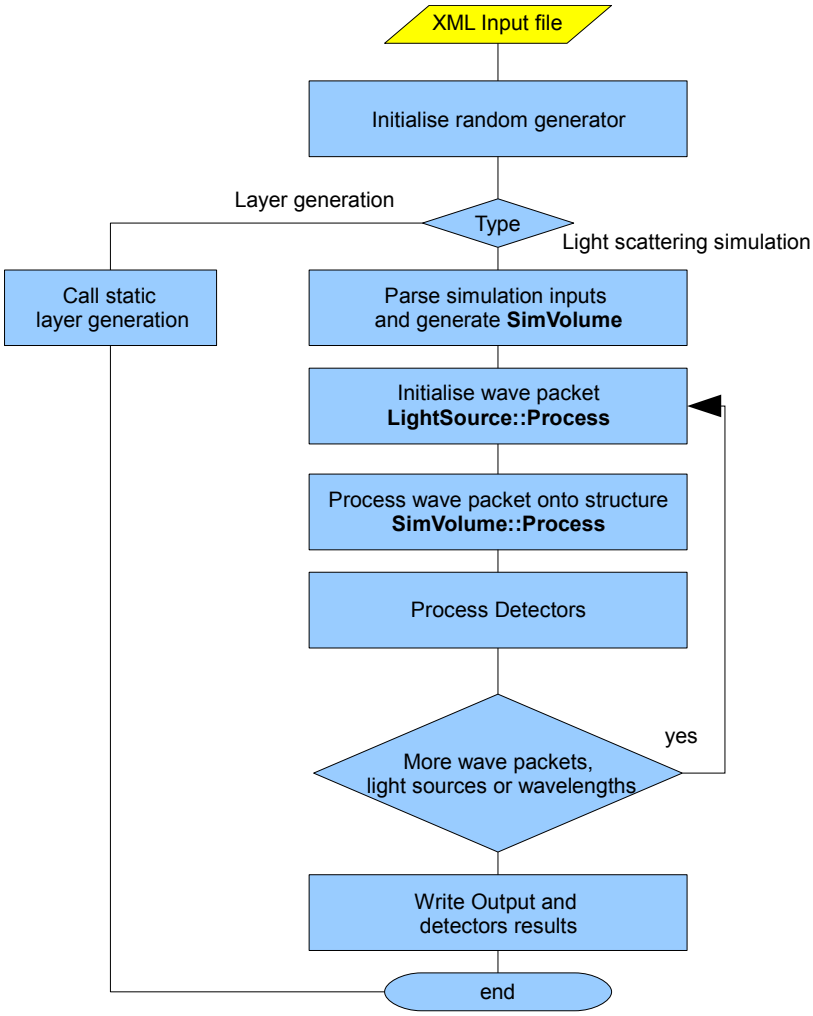


Figure 1: Flow chart of the Main function.

3.1 WavePacket and LightSource

Light is represented by indivisible wave packets that interact with the simulated structure. A **WavePacket** is defined by its position, direction, and polarisation. Only linearly polarised light is currently implemented. A single wave packet is initialised by a **LightSource** each time a new wave packet is sent onto the simulated structure, and is finally processed by the detectors. Light sources initialise the direction, polarisation, and the target point on the plane with z=0 (not necessarily the first hit point on the top surface that might fluctuates around z=0). For spectrally resolved simulations, the wave packet is also affected a weight according to the spectral distribution of the light source. This weighting is important when fluorescence occurs.

The wave packet's optical path is incremented and its wavelength updated in case of fluorescence. Since the wave packet is sent to nearly all functions, it can also carry information necessary at different simulation stage or layers, such as last position on a surface, or the last computation error.

3.2 SimVolume

SimVolume organises the different objects in the simulated structure. It contains the different light sources, objects (surfaces and layers), and detectors in the simulation. It points to the first object,

which is surrounding medium above the simulated structure to get the incoming refractive index, and to the first detector. Each object points to its previous and next object and the same applies to detectors.

The ***Process*** method called after light source generation in the main program is the scattering simulation main function. It sends the wave packet to the top surface and then to the other layers and surfaces until the wave packet has reached the first object (top surrounding), the last object (bottom surrounding), or has been absorbed.

The simulated structure is made of **StructureObject**, which are layers or surfaces delimiting the layers.

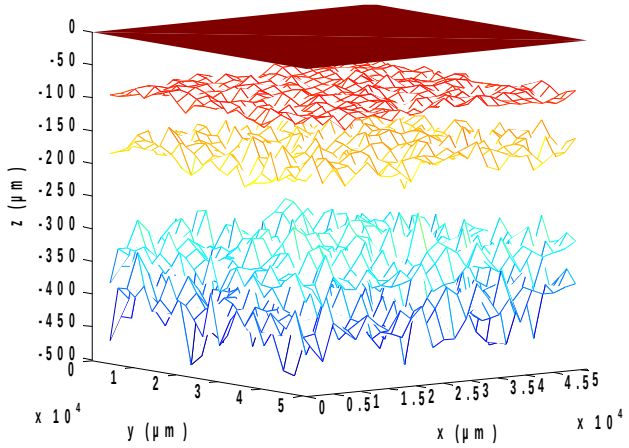


Figure 2: Example of simulated layered structure. Each layer is contained by two surfaces that define the layer thickness variation.

3.3 StructureObject (surfaces and layers)

StructureObject (here called object) is an abstract class from which all **surfaces** and **layers** are derived. Each object points to its previous and next object so that the wave packet can be sent to the next object for processing, when it leaves the current object. The next and previous object of a surface is always a layer, and vice-versa.

All object must have the same size. Each time the wave packet is translated within a layer, the ***Process*** method calls the ***PropInLayer*** method to check for interception with the upper or lower surface, or with the lateral boundaries defined by the size of the the layer. ***PropInLayer*** calls the ***Intercept*** method of the next surface to check for interception and implements the periodic boundary conditions if the wave packet hits a lateral boundary.

Process returns the number of objects the wave packets will pass through until the next active object at its current position. The value is positive if the wave packet is travelling upwards, and negative if the wave packet is travelling downwards. In case of absorption within the object, it returns 0. A wave packet leaving a layer will most often be sent to the next surface in the **SimVolume::*Process*** method. However, when leaving a surface, the wave packet can be sent to a layer further away, if two or more surfaces are in contact, meaning delimited layers with zero thickness, or if the layer controls the surface scattering itself.

A layer contains **Components**, which are geometric representations of the particles that build up the layer. All the internal scattering process within a layer is perform by the components. Light can be scattered at the component surface because of refractive index mismatch between two

components. It can also be scattered, absorbed, or fluoresced within the component. The component members are assigned in the derived **StructureObject** classes. The abstract class itself does not have any component.

Thus, the layer's methods control that the wave packet remains within the layer and send the wave packet to its different component according to the layer's component distribution, whereas the components' methods control the scattering process. Each time the wave packet is translated within a component, it must remain within the layer. Components point therefore to the layer they belong to for access to the layer's methods.

3.3.1 Surface

Surface is the interface separating two layers in the simulated structure. The surface affects the thickness variation of the surrounding layers and controls surface scattering between two layers with refractive index mismatch. The ***Process*** method calculates the interception of the wave packet with the surface (***Intercept***) and perform surface scattering. The Intercept method is also used in the ***PropInLayer*** method of layers to check for surface interception when translating the wave packet within the layer.

The parent class implements an ideally flat surface with constant microroughness. Currently implemented derived class share the same ***Process*** method but differs in their topographical representation in the ***Intercept*** method. The topography representation is based on a surface height map with a defined interpolation procedure attached to each derived class. Some derived class can also map spatially-resolved microroughness and surface normals.

The position of the surface in the simulated volume is defined by the position of the mean surface height map. The mean of the surface height map is not necessary at zero. The way the surface height map is defined will impact on the surrounding layers' thickness variation as shown in Figure 3. Surfaces may coincide at some points, generating a zero thickness layer, but they cannot cross each other.

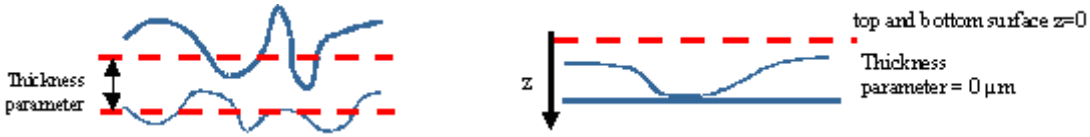


Figure 3: Thickness definition. Two examples on how to define surfaces and bulk thickness. Left: the bulk thickness is the distance between the bounding surfaces whose heights values are oscillating around their mean plane. Right: The surrounding surfaces have the same zero-level. The bulk thickness is then defined by the surface profiles, which in that example only have negative values.

3.3.2 HomogenousLayer

HomogeneousLayer is a simple non-structural layer that models a turbid media with a scattering and absorption coefficient, and a phase function that controls the direction of the wave packet upon scattering. It is implemented as having one single component of no particular shape, **HomogeneousScatteringMaterial**, in order to access the method from the **Component** class.

3.3.3 StatisticalLayer

In a statistical layer (**BaseSheet2** in current version), the components are generated dynamically under the simulation. The thickness variation is static and defined by the delimiting surfaces, but the

inner structure of the layer is described statistically by means of component position- and size distributions. The simulated reflectance is thus an average of the statistic representation and single fibres will for instance not be visible to an image detector.

A **Component** is created upon intersection with a wave packet and is deleted after the wave packet has left the component. The component size and shape is controlled by the component type and generated according to statistical distribution. A statistical layer makes use of **ComponentGenerator** derived classes to control the component generation. These classes can have several member **Distributions** that describe each varying parameter of the component. A statistical layer has accordingly one member **ComponentGenerator** for each component type in the layer.

Which type of component to be generated is controlled by the **ComponentDistribution**. The currently implemented component distribution is only depth dependent, i.e. the components can be heterogeneously distributed along the thickness direction but homogeneously distributed in the plane of the layer.

The scattering at a StatisticalLayer interface can be controlled by the layer components, instead of using a separate **Surface** object. This enables representing the surface of the layer by single particles. In that case, the ***Process*** method in **SimVolume** will jump other the surface process and send the wave packet directly to the layer. Since the same thing will happen when leaving the layer, layer interface scattering must in this case be implemented in the layer ***Process***. Surface scattering at the component outer surface occurs also when the wave packet is coming from a non scattering component, such a void, into the component.

The flow chart of the **StatisticalLayer** process is shown in Figure 4. The **ComponentDistribution** generates first which component will be hit at layer entrance. The component size and shape are generated according to its distributions. If layer surface scattering is controlled by the component at the surface, scattering will occur at the component outer surface based on the refractive index of the component and on the refractive index of the layer the wave packet comes from. If surface scattering was instead previously simulated in a **Surface** object, the wave packet will start within the first component. The component returns then if the wave packet is absorbed, which next component it will enter, or if the wave packet hits a layer surface boundary. If layer surface scattering is to be controlled by the layer components, the layer calls surface scattering and send the wave packet to a new component if it is reflected back to the layer. Otherwise, the wave packet is sent to the next surface object for further processing, unless it was absorbed.

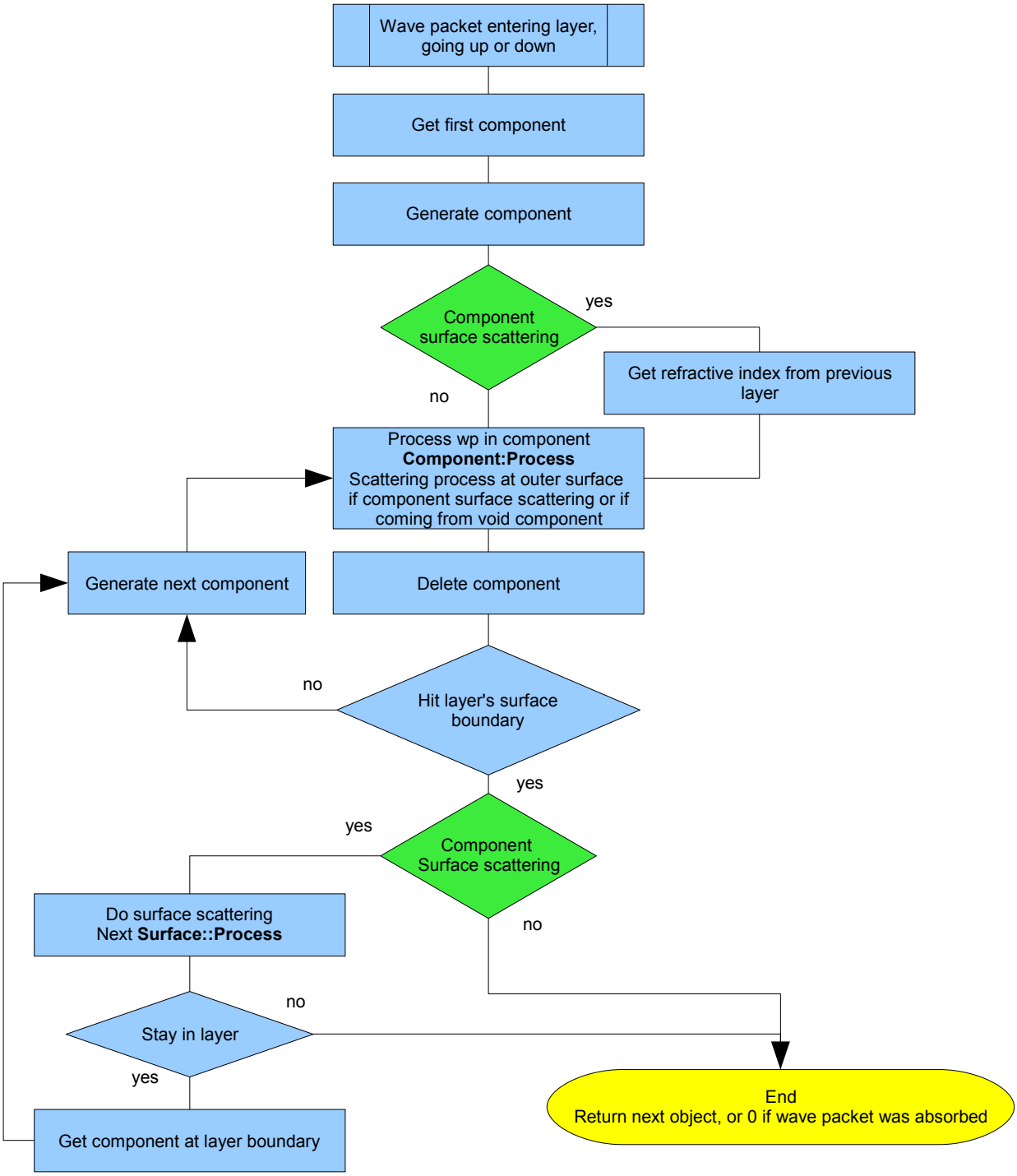


Figure 4: Flow chart of the StatisticalLayer process.

3.3.4 StaticSheet

StaticSheet is a prototype for the simulation of light scattering from a generated static 3D network of fibres. Based on the P3D model, it is yet only working with an own component **Fiber**, defined

under the **p3d** namespace. The fibre network generation process is illustrated in Figure 5.

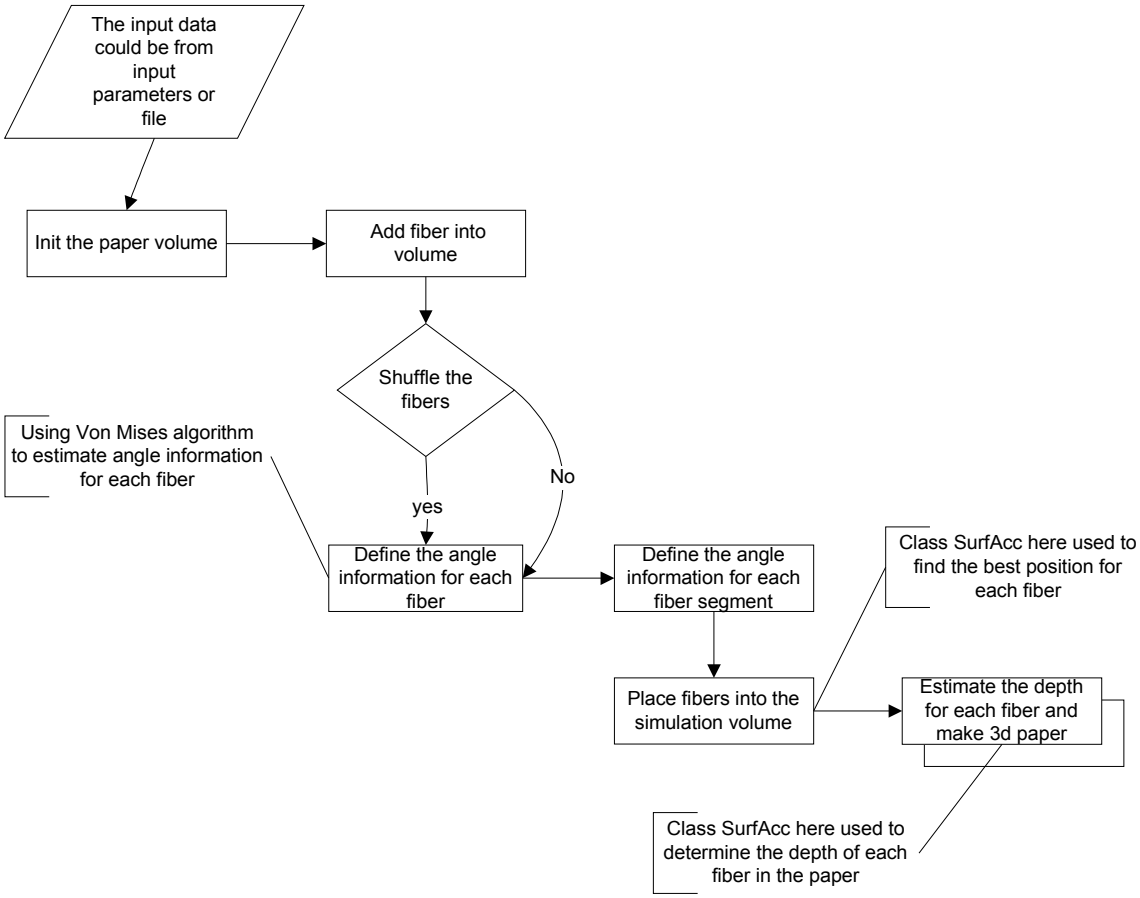


Figure 5: Generating a 3D paper structure.

For light scattering simulation, the network is converted into a more suitable format. During this procedure, two strategies are used to accelerate the light scattering.

1. ***Bounding volume***
   This volume contains a given object and permits a simpler ray intersection check than the object itself.
2. ***Spatial Subdivision***
   Spatial subdivision is processed top-down, partitioning a volume bounding the environment into smaller pieces. The smaller volumes thus formed are assigned collections of objects which are totally or partially contained with them.

Figure 6 shows the steps used to convert the data for faster ray-tracing.

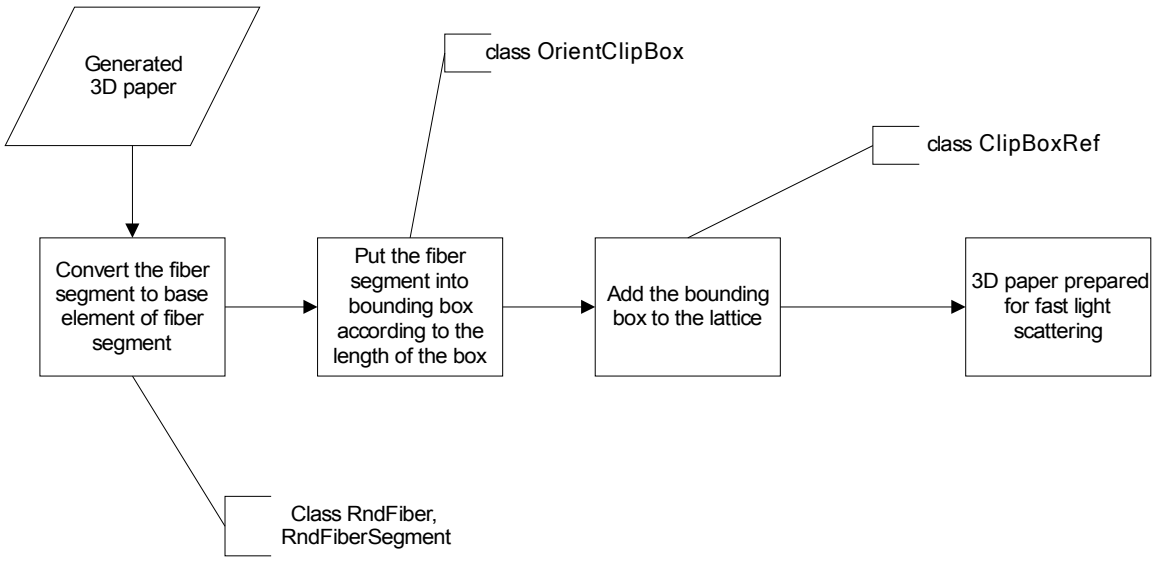


Figure 6: Converting the generated network for light scattering simulation.

3.4 Detectors

All detectors are derived from the abstract class **Detector**, whose member ***mNext*** points to the next detector so that all detectors are processed sequentially in the main function.

3.5 IO

All input and output to *Open PaperOpt* are handled by the classes located in the IO namespace. The underlying philosophy is that each XML element has a function that parses it and creates the object it is associated with.

3.5.1 Input

This class is the starting point for reading of input XML files. The constructor takes the path to the XML file from which reading should be done. After the Input class has been instantiated the root element of the XML file is indicated by the public `mRoot enum`. Depending on what type of XML file it is (simulation specification or staticSheet definition), different methods are called to parse the file.

3.5.2 XML parser

This class steps through the XML file one element at a time. To get the next element, the ***readNext*** function is used. This class also contains methods for parsing common XML types such as floatArray, float, tableDistribution, etc. An object of the **XMLParser** class is created by the **Input** class, and then passed around to the different parsing functions that then read from it.

3.5.3 Output

This class contains methods for writing output files. There are three methods for each detector type, one public and two private functions. The public function is called from the **writeResults** method in the **Detector** classes and writes the XML code that describes the detector result to the XML results file. It then calls one of the two private methods depending on if the program is compiled with HDF support or not.

One private function is used to write binary output and HDF output. If the `ENABLEHDF` macro is defined when the program is compiled the HDF output methods will be called, if `ENABLEHDF` is not defined the binary output methods will be called instead.

4 Used libraries and setting

Open PaperOpt uses the irrXML reader for reading the input XML files. It is a small, fast C++ XML parser that is platform independent. It consists of one .cpp file and seven .h files that need to be placed in a subfolder called /irrXML/ of the directory where the source code for *Open PaperOpt* is located. It is available at http://www.ambiera.com/irrxml/ under the zlib license. Once compiled with *Open PaperOpt* no other external files are needed.
For output, binary and HDF5 formats are supported. If *Open PaperOpt* is compiled with the preprocessor definition ENABLEHDF the HDF5 library will be linked into *Open PaperOpt* and output will be in HDF5 format.
The HDF5 library can be found at http://www.hdfgroup.org/HDF5/. The HDF5 library uses two further libraries for compression, ZLIB and SZIP. These libraries are also needed for compiling *Open PaperOpt*.

5 Data format

Open PaperOpt uses XML for simulation specifications and results. One XML file is used as input and one as output. For large volumes of data such as topographies, fluorescence matrices and table distributions a binary (or HDF5) format is used. These external files are described and referenced from the input XML file.

The binary files do not contain any metadata describing the contents but just the raw data, instead the contents of the binary files are described in XML. For matrices this includes information such as the number of rows and columns and what datatype the elements have.

5.1 Specification files

The basic structure of the input XML file when running a light scattering simulation is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<simulation>
 <description> </description>
 <date></date>
 <resultsDirectory></resultsDirectory>
 <binaryDirectory></binaryDirectory>
 <simulationVolume>
  <size x="" y=""/>
  <refIndexTop real="" imaginary=""/>
  <refIndexBottom real="" imaginary=""/>
  <lightSource nrOfWP="" type=""/>
  <surface z="" type=""/>
  <bulk type=""/>
  <surface z="" type=""/>
 </simulationVolume>
```

```
 <detector reflTrans="" type=""/>
</simulation>
```

The description element is optional and can be used to describe the simulation for later reference.
All result files will be placed in the directory specified in the resultsDirectory element, the binaryDirectory element points to the directory where external binary files such as surface topographies are stored. All references to binary files are relative from this directory.
For more details regarding the structure of the specification files, see Section 7 and the specification examples in Section 9.

5.2 The XML Schema

An XML Schema describes the structure of XML documents, they are not strictly necessary but can be used to validate XML documents and provide strong typing when used with a parser.
For the input and output XML documents there is an XML Schema available split up into tree files that can be used. This schema specifies the general structure of the specifications file and all the parameter types that the objects in the program uses.
The file *simulationSchema.xsd* describes the overall structure of specification, result and bulkGenerate XML files. *commonTypes.xsd* describes the structure of the different parameters used by the objects, and *paperObjects.xsd* specifies which parameters are used by all the different types of objects.
The Schema can be used when writing specification files manually, for example using Visual Studio 2005, or some XML editing tool. VS2005, for example, will display a list of surface types and will also tell you what parameters each type uses.

5.3 External input files

In the specification file different external files are referenced that contain large amounts of array or tabular data. Currently all these external files are in a simple binary format or in ascii format.

The binary files used as input can be loaded in Matlab using the *fopen* and the *fread* commands. To use the *fread* command, the dimension of the array or table are needed, and these are defined in the XML file.

5.4 Height maps used in Surface classes

The light scattering model uses height maps to define surfaces and interfaces between different layers such as homogenous or basesheet layers. A height map is simply a matrix of deviations from the mean plane of the surface in micrometers. The XML describing a height map is shown below:

```
<heightMap name="printSurface">
  <size x="10000" y="10000"/>
  <floatArray>
    <dimensions nrColumns="50" nrRows="50"/>
    <binaryFile>
      <filePath>printSurface.bin</filePath>
    </binaryFile>
  </floatArray>
</heightMap>
```

The size element describes the size of the surface in micrometers, while the dimensions element specifies the resolution of the height map. The actual data of the topography is stored in the file

indicated by the `filePath` element. The matrix is stored row wise and the elements are 32 bit floating point numbers.
To load this height map in Matlab the following command can be used.

```
>> id = fopen('printSurface.bin')
>> arr = fread(id,[50 50], 'float');
```

5.5 Tables used in the DistributionTable class

Tables are used for the DistributionTable class, they consists of a sampling vector and a probability density function forming two columns of a table. As with matrices the data is stored row wise in a binary file while the contents are described in XML. Example of XML describing a table of fiber widths:

```
<table nrRows="51">
 <column name="width" dataType="double"/>
 <column name="freq" dataType="double"/>
 <binaryFile encoding="littleendian">
  <filePath>efibwidth.bin</filePath>
 </binaryFile>
</table>
```

Currently, only the double datatype is used, which is a 64 bit floating point number. In this example the file fillersize1.bin would contain 102 64 bit floating point numbers stored row wise, i.e. width freq width freq etc.

5.6 Tables used in the DistributionComponents class

The DistributionComponents class is used in statistical Basesheets to determine which component a wave packet will hit, this will depend on the depth the wave packet is currently at. This class needs a table of probabilities as input, the XML code for that table is shown below.

```
<table nrRows="799">
 <column name="depth" dataType="double"/>
 <column name="pore" dataType="double"/>
 <column name="fiber" dataType="double"/>
 <column name="filler" dataType="double"/>
 <binaryFile>
  <filePath>e60Comp.bin</filePath>
 </binaryFile>
</table>
```

Here the order of the column elements depends on how the contents of the binary file are laid out, also the names of the columns are important and must be one of the four shown above.
In this example the file e60Comp.bin would consist of 4*799 64 bit floating point numbers. The pore, fiber and filler columns contain the relative frequencies of these components at the depth indicated by the depth column. Note that the depth column and one of the other columns is required.

5.7 Static fiber networks

Output file is generated by the static fibre network process. It can then be used as input in a light scattering specification file. When a fibre network is generated, the program saves one file containing the network itself, and two files containing the bounding surfaces of the generated fibre

network. The fibre network file description is given below.

order	data contents		data type
1	the number of the fibers		int
the following will be the data for each fiber			
2	length of fiber		double
3	radius of fiber		double
4	Form factor of fiber		double
5	segment length		double
6	the number of segments		int
7	angle of the fiber		double
8	indicator of local angle for fiber segment		int
	1	the segments have their own angle information	
	0	the segments don't have angle information	
9	the segments angle information (data size=the number of segments)		double
10	indicator of fiber segment		int
	1	the segments exist	
	0	the segments don't exist	
11	the fiber segments information (data size=the number of segments + 2)		FiberSegment

6 List of input parameters

This section describes the input parameters used by the different types of surfaces, bulks, light sources and detectors. The tag name of a parameter indicates in what way it is used. The parameters may be of different types such as *floatType* or *boolType*. The structures of the different types of parameters are described in the XML Schema. Some examples are shown below.

```
<wavelengths>0.3 0.32 0.34 0.36 0.38 0.52</wavelengths>
<distance>10000</distance>
<rmsMin>0</rmsMin>
<rmsMax>20</rmsMax>
```

The *distance*, *rmsMin* and *rmsMax* elements are all of *floatType* and thus have the same structure. The *wavelengths* element is of type *floatListType*.

6.1 Surfaces

All surface elements have the attributes *z* and *type*. *z* is the position of the surface in the simulation volume and type describes what kind of surface it is. Below is a list of currently implemented surface types and their parameters. *Element type* refers to the basic tags described above and *name* is the attribute of the tag that describes what the parameter is. Required tags are marked with a x,

and optional tags with -.

```
<surface z="0" type="flat">
 <rmsMin>0</rmsMin>
 <rmsMax>20</rmsMax>
</surface>
```

		Surface type				
Tag name	**Element type**	**flat**	**TopoBilinear**	**topoTriangulated**	**flatNormal**	**subFlat**
rmsMin	floatType	-	-	-	-	-
rmsMax	floatType	-	-	-	-	-
heightMap	heightMapType		x	x	x	x
roughMap	V2FloatArrayType		-	-	-	-
normalMap	V3FloatArrayType				-	-
inkLayer	inkLayerType				-	
normalInterpolation	boolType				x	x
normalFactor	integerType				x	x
normalInterpolatioon	boolType					-

Note that rms and a roughness map can not be used at the same time. The same goes for topograhies and normal maps. Normalfactor is only required when a topography is used.

An inkLayer is defined as

Tag name	**Element type**
xmin	floatType
xmax	floatType
ymin	floatType
ymax	floatType
screenAngle	floatType
dropletSpacing	floatType
dropletWidth	floatType
dropletHeight	floatType
scatteringProperties	scatteringProperties

6.2 Layers

All bulk layer elements have the *type* attribute. An example bulk definition is shown below.

```
<bulk type="homogeneouslayer">
 <scatteringProperties name="wall">
  <scatteringParameter lambda="0.2" s="0" a="0.0003" g="0" n_real="1.5" n_imag="0"/>
  <scatteringParameter lambda="0.7" s="0" a="0.0003" g="0" n_real="1.5" n_imag="0"/>
  </scatteringProperties>
</bulk>
```

6.2.1 HomogenenousLayer

Tag name	Element type	Use
fluorescence	fluorescenceType	optional
scatteringProperties	scatteringPropertiesType	required

6.2.2 StatisticalLayer

This is a statistic basesheet where pores are ellipsoidal.

Tag name	Element type	Use
componentsDistribution	componentsDistributionType	required
fiber	basesheetFiberType	required
pore	basesheetPoreType	optional
filler	basesheetFillerType	optional

Note that the fiber, pore and filler elements are the same as those used by basesheet2. So see section 7.2.3 for a description of basesheetFiberType, basesheetPoreType and basesheetFillerType.

6.2.3 basesheet2

In addition to the ordinary basic types, the basesheet also contains fibers, fillers and pores each having their own set of parameters described below.

Tag name	Element type	Use
topoScattering	boolType	required
componentsDistribution	componentsDistributionType	required
fiber	basesheetFiberType	required
pore	basesheetPoreType	optional
filler	basesheetFillerType	optional

basesheetFiberType:

Tag name	Element type	Use
rmsMin	floatType	optional
rmsMax	floatType	optional
contactReduction	floatType	required
wallScatParams	scatteringPropertiesType	required

lumenScatParams	scatteringPropertiesType	required
theta	constantDistributionType **or** tableDistributionType **or** ellipticDistributionType	required
phi	constantDistributionType **or** tableDistributionType **or** ellipticDistributionType	required
tilt	constantDistributionType **or** tableDistributionType	required
length	constantDistribution **or** tableDistributionType	required
width	constantDistributionType **or** tableDistributionType	required
wallThickness	constantDistributionType **or** tableDistributionType	required
ellipticity	constantDistributionType	required

basesheetFillerType:

Tag name	**Element type**	**Use**
rmsMin	floatType	optional
rmsMax	floatType	optional
contactReduction	floatType	required
fillerScat	scatteringPropertiesType	required
fluorescence	fluorescenceType	optional
theta	constantDistributionType **or** tableDistributionType **or** ellipticDistributionType	required
phi	constantDistributionType **or** tableDistributionType **or** ellipticDistributionType	required
aAxis	constantDistributionType **or** tableDistributionType	required
bAxis	constantDistributionType **or** tableDistributionType	optional
ellipticity	constantDistributionType **or** tableDistributionType	required

if bAxis is omitted it will use the same distribution as aAxis.

basesheetPoreType**:**

Tag name	**Element type**	**Use**
rmsMin	floatType	optional
rmsMax	floatType	optional
contactReduction	floatType	required
theta	constantDistributionType **or** tableDistributionType **or** ellipticDistributionType	Required
phi	constantDistributionType **or** tableDistributionType **or** ellipticDistributionType	required
tilt	constantDistributionType **or** tableDistributionType	required
aAxis	constantDistributionType **or** tableDistributionType	required
bAxis	constantDistributionType **or** tableDistributionType	optional
ellipticity	constantDistributionType	required

If bAxs is omitted, it will use the same distribution as aAxis.

6.2.4 Static basesheet

After a static basesheet has been generated it can be used like other layers/basesheets.

6.3 Light Sources

All lightSource elements have the *type* and *nrOfWP* attributes. Below is a list of the types currently implemented. If several light sources are present, they must have the same wavelengths. Required tags are marked with a x, and optional tags with -.

```
<lightSource xsi:type="lambert" nrOfWP="1000">
 <wavelengths>0.3 0.32 0.34 0.36 0.38 0.4 0.420.52</wavelengths>
 <distance>10000</distance>
</lightSource>
```

		Light source type		
Tag name	**Element type**	**beam**	**elrepho**	**lambert**
wavelengths	floatListType	x	x	x
weights	floatListType	-	-	-
polarizationValue	floatType	-	-	-
theta	floatType	x		

phi	floatType	x		
distance	floatType	x		
beamCentre	floatListType	-	-	-
centreDiameter	floatType	-	-	-
xmin	floatType	-	-	-
xmax	floatType	-	-	-
ymin	floatType	-	-	-
ymax	floatType	-	-	-

If beamCentre is present, the beam will have a pencil distribution on the surface, if centreDiameter is also present it will have a disc distribution. Otherwise it will use a uniform distribution.

6.4 Detectors

All the detectors have the *reflTrans* attribute which either has the value ”reflectance” or ”transmittance”. Example detector definition. Required tags are marked with a x, and optional tags with -.

```
<detector type="image" reflTrans="reflectance">
 <numAperture>0.4</numAperture>
 <distance>10000</distance>
 <resX>512</resX>
 <resY>512</resY>
</detector>
```

		Detector type				
Tag name	**Element type**	**error**	**ars**	**image**	**arsGlobe**	**plr**
apertureDiameter	floatType		x			
distance	floatType		x	x	x	x
thetaMin	floatType		x			
thetaMax	floatType		x			
thetaStep	floatType		x			
phiMin	floatType		x			
phiMax	floatType		x			
phiStep	floatType		x			
numAperture				x		
theta				-		
phi				-		
resX				x		

resY				x		
nrTheta	integerType				x	
nrPhi	integerType				x	
accurate	boolType				x	
aperture	floatType					x
singleScattering	boolType					x
startingZ	floatType					x
lastZ	floatType					x
stepZ	floatType					x

7 Output

The output of *Open PaperOpt* consists of one XML file and several binary or HDF5 files. The XML file contain general simulation results and metadata describing the binary and HDF5 files.
If the program was compiled with HDF all detector results are placed in HDF5 files, if the program was not compiled with HDF the results are instead placed in binary files.
In either case, all output files will be placed in the folder indicated by the resultsDirectory element in the input specification file. If the element is not present output files will be placed in the same directory as the program executable.
The detector output files are given default names and are numbered if more than one detector of the same type is defined in the input specification file. E.g. the output of the first image detector defined will be placed in "imageResults1.h5", the second in "imageResults2.h5" etc.

8 Coding

8.1 Naming convention

As a long term running project, different programmers will join in the project in the future. In order to reduce the effort needed to read and understand source code and enhance source code appearance, the following set of rules will facilitate the coding work (adapted from http://www.possibility.com/Cpp/CppCodingStandard.html#names),

Type	**Convention**	**Example**
Class name	• User Upper case letters as word separators,lower case for the rest of a word • First character in a name is upper case • No underbars('_')	`class FiberSegment` `class Fiber`

File name and Method name	• File name should be the same as class name defined in the file
	• Use the same rules as for the class name
	• For header file we use .h extension
	• For implementation file we use .cpp
		`FiberStatic.cpp, FiberStatic.h`
		
		`public:`
		`int NumSeg();`
		`void GenerateSegments();`

	extension	

Class attribute name	• Attribute names should be prepended with the character 'm'.
	• after the 'm' use the same rules as for class names.'m' always precedes other name modifiers like 'p' for pointer.
		Prepending 'm' prevents any conflict with method names. Often your methods and attribute names will be similar, especially for accessors.
		`public:`
		`  double Length();`
		`  double Radius();`
		`private:`
		`  double mLength;`
		`  double mRadius;`

Method argument name	• The first character should be lower case.
	• All word beginnings after the first letter should be upper case as with class names.
		Always tell which variables are passed in variables.
		Using names similar to class names without conflicting with class names.
		`class Fiber`
		`{`
		`public:`
		`void GetSegmentCoordinates(int segNo,V2 <double>& rV1,V2 <double>& rV2) const ;`
		`}`

Variable name on the stack	• Use all lower case letters • Use '_' as the word separator.	The scope of the variable is clear in the code. Now all variables look different and are identifiable in the code. `double Fiber::Weight() { return mLength*mRadius*mRadius*4*gFiberRaw; }`
Pointer variable	• pointers should be prepended by a 'p' in most cases • place the * close to the pointer type not the variable name	`String* pName= new String;` `String* pName, name, address; // note, only pName is a pointer.`

Reference variable and function returning reference	• References should be prepended with 'r'.	`class Fiber`
		`{`
		`public:`
		`void GetSegmentCoordinates(int segNo,V2 <double>& rV1,V2 <double>& rV2) const ;`
		`private:`
		`FiberSegment& mrSegment;`
		`}`

Global variable	• Global variables should be prepended with a 'g'.	`double gFiberRaw;`
Global constant	• Global constants should be all caps with '_' separators.	`const int A_GLOBAL_CONSTANT= 5;`
Static variable	• Static variables may be prepended with 's'.	`static StatusInfo msStatus;`
Type name	• When possible for types based on native	`typedef uint16 ModuleType;`

	types make a typedef. • Typedef names should use the same naming policy as for a class with the word Type appended.	`typedef uint32  SystemType;`

Enum name	• Label All Upper Case with '_' Word Separators	`enum Occurrence`
		`{`
		` NOTHING,`
		` SCATTERING,`
		` ABSORPTION`
		`};`

#define and macro name	• Put #defines and macros in all upper using '_' separators.	`#define FIBER_C   (5.4e-8)`

C function name	• In a C++ project there should be very few C functions.
	• For C functions use the GNU convention of all lower case letters with '_' as the word delimiter.
		Makes C functions very different from any C++ related names.
		`static void add_heights(int& sp_added,int numCalc,SurfAcc& heights,Fiber* f,double rad)`
		`{`
		`}`

8.2 JavaDoc

JavaDoc convention is used to automatically generate the online documentation.

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
 public Image getImage(URL url, String name) {
                             try {
                                 return getImage(new URL(url, name));
                             } catch (MalformedURLException e) {
                                 return null;
                             }
 }
```

A more detail description of JavaDoc could be found here How to Write Doc Comments for the Javadoc Tool.

9 Examples

9.1 Light sources and detectors

This examples simulates four different light sources, an image detector, and a angle-resolved scatterometer (ARS) detector. The simulation volume consists of only one flat surface, required to define the size of the simulation volume, and the detectors detects the spatial- and angle-resolved distributions of the light sources. The light sources are defined at 4 different wavelengths but their weight is only different from 0 at one wavelength. Since every light source is simulated at a one own wavelength, their distributions can be distinguished in the detectors. The XML specification file is downloadable from SpecsFile.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<simulation xmlns="http://tempuri.org/simulationSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">


 <description>Example light sources and detectors 1</description>
<!--*************************************************************************
  * This example simulates light sources with different spatial and angular
  * distributions. No scattering nor absorption occur in simulated structure
  * so the light is propagates straight to the detectors, which then detect the
  * spatial and angular distributions of the light sources.
  *************************************************************************-->
 <date>2009-01-20</date>
 <simulationVolume>
<!--*************************************************************************
  * Simulation volume 1mm*1mm
  *************************************************************************-->
  <size x="1000" y="1000"/>
<!--*************************************************************************
  * The refractive index of the surrounding is set to 1
  * (over and below simulation strucutre
  *************************************************************************-->
 <refIndexTop real="1" imaginary="0"/>
 <refIndexBottom real="1" imaginary="0"/>
<!--*************************************************************************
  * Light sources definition:
  * Every light source must be defined for all wavelengths
  * Here the spectral weight is different from zero only at one wavelength,
  * i.e. each light source is monochromatic. This will enable visualising
  * the light source distributions individually in the detectors.
  *************************************************************************-->
<!--*************************************************************************
  * Lambertian illumination illuminating the surface homogenously.
  *************************************************************************-->
 <lightSource xsi:type="lambert" nrOfWP="000000">
  <wavelengths>0.3 0.4 0.5 0.6</wavelengths>
  <weights>1 0 0 0</weights>
 </lightSource>
<!--*************************************************************************
  * Elrepho like angular distribution illuminating the surface homogenously.
  *************************************************************************-->
 <lightSource xsi:type="elrephoLightSource" nrOfWP="0000">
```

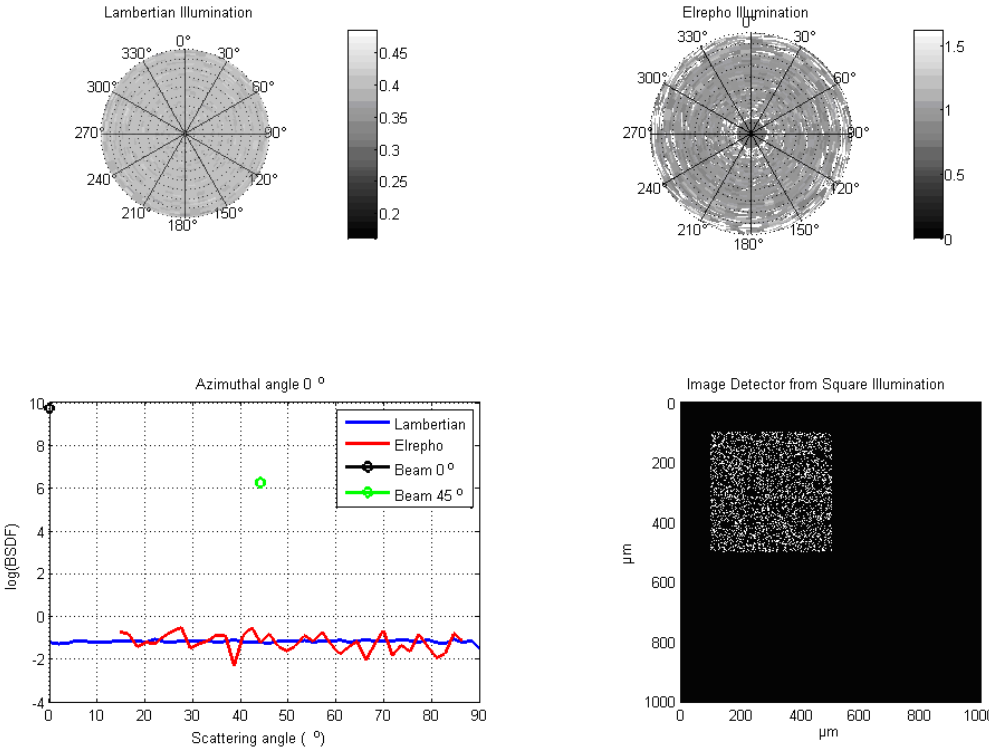
```
  <wavelengths>0.3 0.4 0.5 0.6</wavelengths>
  <weights>0 1 0 0</weights>
 </lightSource>
<!--**************************************************************************
  * Collimated beam at normal incidence illuminating the surface homogenously.
  **************************************************************************-->
 <lightSource xsi:type="beam" nrOfWP="0000">
  <wavelengths>0.3 0.4 0.5 0.6</wavelengths>
  <weights>0 0 1 0</weights>
  <polarizationValue>0</polarizationValue>
  <theta>0</theta>
  <phi>0</phi>
  <distance>10000</distance>
 </lightSource>
<!--**************************************************************************
  * Collimated beam at 45o incident angle illuminating the surface
  * on a square area.
  **************************************************************************-->
 <lightSource xsi:type="beam" nrOfWP="10000">
  <wavelengths>0.3 0.4 0.5 0.6</wavelengths>
  <weights>0 0 0 1</weights>
  <polarizationValue>0</polarizationValue>
  <theta>0</theta>
  <phi>0</phi>
  <distance>10000</distance>
  <!-- no effect since only angular distribution matters -->
  <xmin>100</xmin>
  <xmax>500</xmax>
  <ymin>100</ymin>
  <ymax>500</ymax>
 </lightSource>

<!--**************************************************************************
  * The program requires at least on paper object to be defined
  * A flat surface is used. Since refractive index above and below that
  * suface are equal, light will continue straight to the detectors
   **************************************************************************-->
 <surface z="0" xsi:type="flat"> </surface>
/simulationVolume>

<!--**************************************************************************
  * Detectors
  * An image detector and a goniophotometer (Angle resolved scatterometer
  * (ARS)). Detectors are set to "tranmsittance" since all light is
  * transmitted.
  **************************************************************************-->
 <detector xsi:type="image" reflTrans="transmittance">
  <numAperture>1</numAperture>
  <distance>100000</distance>
```

```
  <resX>512</resX>
  <resY>512</resY>
 </detector>
<!--************************************************************************
  *  This detector is commented because it is very slow
  ************************************************************************
 <detector  xsi:type="ARS" reflTrans="transmittance">
  <apertureDiameter>28000</apertureDiameter/>
  <distance>500000</distance
  <thetaMin>-50</thetaMin>
  <thetaMax>50</thetaMax>
  <thetaStep>1</thetaStep>
  <thetaMin>-50</thetaMin>
 </detector>   -->
 <detector xsi:type="ARSGlobe" reflTrans="transmittance">
  <distance>500000</distance>
  <nrTheta>50</nrTheta>
  <nrPhi>50</nrPhi>
  <accurate>false</accurate>
 </detector>
</simulation>
```

The plotresults.m Matlab® file visualises the detector results from this example:



9.2 Homogeneous layers

This example simulates the light scattering from a turbid layer at one wavelength. Since the refractive index of the layer is different from 1, light is reflected at the interfaces between the layer and air. The effect of surface microroughness and layer refractive index on the Elrepho detector reflectance factor can be simulated by varying the microroughness `<rmsMax>`, and the refractive index `<n_real>`. The XML specification file is downloadable from ExampleHomogeneousLayer.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<simulation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://tempuri.org/simulationSchema"
xsi:schemaLocation="http://tempuri.org/simulationSchema
C:/ScatSimMC/Examples/SimulationSchema/simulationSchema.xsd">
  <description>Test Saunderson</description>
  <date>2008-10-06</date>
  <!-- <resultsDirectory>C:\results</resultsDirectory>  -->
  <simulationVolume>
  <size x="10000" y="10000"/>
    <refIndexTop real="1" imaginary="0"/>
    <refIndexBottom real="1" imaginary="0"/>
    <lightSource xsi:type="lambert" nrOfWP="100000">
      <wavelengths>0.3</wavelengths>
      <weights>1</weights>
      <distance>10000</distance>
    </lightSource>
    <!-- Define upper surface -->
```

```
    <surface z="0" xsi:type="flat">
      <rmsMin>0</rmsMin>
      <rmsMax>0</rmsMax>
    </surface>
    <bulk xsi:type="homogeneouslayer">
      <!--This defines the optical properties of the layer  -->
      <scatteringProperties name="HomogeneousLayer1">
        <scatteringParameter lambda="0.3" s="0.01" a="0.0001" g="0"
n_real="1.6" n_imag="0"/>
        <scatteringParameter lambda="0.32" s="0.01" a="0.0001" g="0"
n_real="1.6" n_imag="0"/>
      </scatteringProperties>
    </bulk>
    <!-- define the lower surface-->
    <surface z="-93" xsi:type="flat">
      <rmsMin>0</rmsMin>
      <rmsMax>0</rmsMax>
    </surface>
  </simulationVolume>
  <!-- detectors>
  <detector xsi:type="elrephoDetector" reflTrans="reflectance"/>
</simulation>
```

9.3 Statistical structural layers

Examples of statistical structural layers are downloadable from ExampleStatisticalStructurealLayer . Note that all required input files (.txt and .bin) should be loaded into the same map as the xml specification file. These examples simulate the light scattering of real paper sheets of eucalyptus and pine pulps, from measured input parameters, such as fibre length distribution, thickness, etc.

9.4 Static structural layers

This examples show how to generate a static fibre which has 50 different fibres. These fibres have different length and widths which are defined by the distribution function. The XML specification file is downloadable from StrucuralLayerExample.

```
<?xml version="1.0" encoding="utf-8"?>
<staticBulkGeneration xmlns="http://tempuri.org/simulationSchema">
 <!--************************************************************
  * This example illustrate generating a static fiber network
  * with size of 1000 * 1000, there are 50 fibers in the
  * fiber network. the length and radius of each fiber is
  * different and are generated by the given distribution function
  ************************************************************-->
 <bulk xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="genStaticBasesheet">
  <!--the size of the fiber network-->
  <sizeX>1000</sizeX>
  <sizeY>1000</sizeY>
  <!--the segment length-->
  <segmentLength>2</segmentLength>
  <!--add different kind of fibers-->
```

```
  <fiber>
   <!-- the number of fibers`-->
   <numberOfFibers>50</numberOfFibers>
   <!-- the radius of fibers`-->
   <width xsi:type="tableDistribution">
    <interval from="0.5" to="49.5"/>
    <table nrRows="51">
     <column name="width" dataType="double"/>
     <column name="length" dataType="double"/>
     <binaryFile encoding="littleendian">
      <filePath>efibwidth.bin</filePath>
     </binaryFile>
    </table>

   </width>
   <!-- the length of fibers`-->
   <length xsi:type="tableDistribution">
    <interval from="50" to="4950"/>
    <table nrRows="51">
     <column name="length" dataType="double"/>
     <column name="prob" dataType="double"/>
     <binaryFile encoding="littleendian">
      <filePath>efiblength.bin</filePath>
     </binaryFile>
    </table>
   </length>
   <!-- the Form Factor of fibers`-->
   <formFactor xsi:type="constantDistribution" value="0.95"/>

  </fiber>
  <fiber>
   <numberOfFibers>50</numberOfFibers>
   <width xsi:type="constantDistribution" value="10"/>
   <length xsi:type="constantDistribution" value="1000"/>
   <formFactor xsi:type="constantDistribution" value="0.95"/>
  </fiber>
  <!--*****************************************************
  * first ,please remove the heightmap tag to generate
  * a fiber network on flat surface second , if you want to
  * generate a fiber network on an uneven surface, use the heightmap and
  * set the filepath tag to the file which contain unflat surface information
   *********************************************************************-->
<!-
<heightMap name="upper">
 <size x="1000" y="1000" />
 <floatArray>
  <dimensions nrColumns="256" nrRows="256" />
```

```
    <binaryFile>
      <filePath>base_surf1.bin</filePath>
    </binaryFile>
  </floatArray>
</heightMap>
-->
  <!--*****************************************************************
  *
  * the generated fiber network will be stored in the following file
  *
  *****************************************************************-->
  <outputFilenameXML>bulk1.xml</outputFilenameXML>
  <!--*****************************************************************
  *
  * the surface on the top of the fiber network will be saved in the following file
  *
  *****************************************************************-->
  <topSurfaceFilenameXML>surf1.xml</topSurfaceFilenameXML>
 </bulk>
</staticBulkGeneration>
```

The following picture is generate from the above XML file and visualized by the Cosmo player.

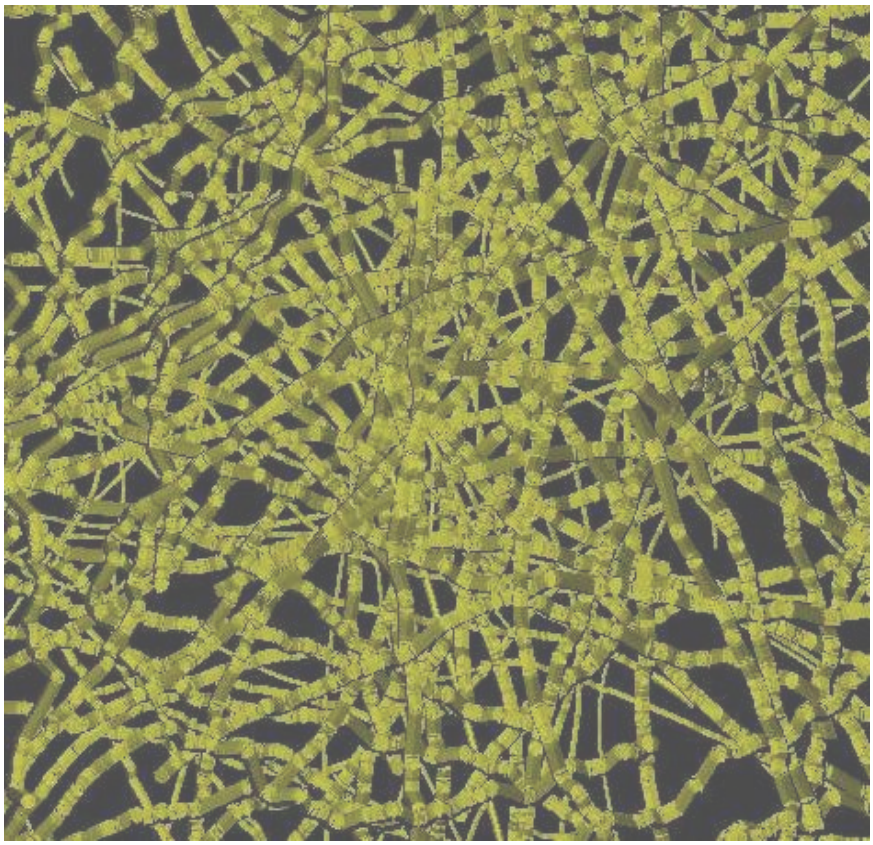


Figure 7: Fiber network used light scattering computation

This examples using a light source and an image detector to do the light scattering on the generated fibre network which is illustrated above, The XML specification file is downloadable from p3d_specs.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<simulation                                          xmlns="http://tempuri.org/simulationSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <description>Light scattering simulation no.1</description>
 <date>2008-10-06</date>
 <resultsDirectory>c:\results</resultsDirectory>
 <simulationVolume>
  <size x="1000" y="1000"/>
  <refIndexTop real="1" imaginary="0"/>
  <refIndexBottom real="1" imaginary="0"/>

  <lightSource xsi:type="beam" nrOfWP="10000000">
   <wavelengths>0.8</wavelengths>
   <weights>1</weights>
   <polarizationValue>0.5</polarizationValue>
   <theta>0.5</theta>
   <phi>0.4</phi>
   <distance>10000</distance>
   <!-- no effect since only angular distribution matters -->
   <xmin>0</xmin>
   <xmax>1000</xmax>
   <ymin>0</ymin>
   <ymax>1000</ymax>
  </lightSource>

  <surface z="0" xsi:type="topoBilinear">
   <heightMap name="upper">
    <size x="1000" y="1000" />
    <floatArray>
     <dimensions nrColumns="256" nrRows="256" />
     <binaryFile>
      <filePath>surf1_surface_upper.bin</filePath>
     </binaryFile>
    </floatArray>
   </heightMap>
  </surface>
  <!--------------------------------------------------------------------------->
  <bulk type="staticBasesheet">
   <sizeX>1000</sizeX>
   <sizeY>1000</sizeY>
   <numberOfFibers>100</numberOfFibers>
```

```
    <segmentLength>2</segmentLength>
    <totalWeight>7.84363e-006</totalWeight>
    <fiber>
     <binaryFile>
      <filePath>bulk1_fibers.bin</filePath>
     </binaryFile>
    </fiber>
    <surfaceStructure>
     <surfTop>58.79</surfTop>
     <surfMeanTop>12.7985</surfMeanTop>
     <surfBottom>-3.33067e-016</surfBottom>
     <surfMeanBottom>7.39313</surfMeanBottom>
    </surfaceStructure>
    <validity>
     <initFlag>13</initFlag>
     <segmentType>1</segmentType>
    </validity>
   </bulk>

   <!--------------------------------------------------------------------------->
   <surface z="-93" xsi:type="topoBilinear">
    <heightMap name="lower">
     <size x="1000" y="1000" />
     <floatArray>
      <dimensions nrColumns="256" nrRows="256" />
      <binaryFile>
       <filePath>surf1_surface_lower.bin</filePath>
      </binaryFile>
     </floatArray>
    </heightMap>
   </surface>
  </simulationVolume>

  <detector xsi:type="image" reflTrans="reflectance">
   <numAperture>1</numAperture>
   <distance>10000</distance>
   <resX>256</resX>
   <resY>256</resY>
  </detector>
 </simulation>
```