



SPRAY

© 2001 Wolfgang Theiss

Note:

To change the product logo for your own print manual or PDF, click "Tools > Manual Designer" and modify the print manual template.

Title page 1

Use this page to introduce the product

by Wolfgang Theiss

This is "Title Page 1" - you may use this page to introduce your product, show title, author, copyright, company logos, etc.

This page intentionally starts on an odd page, so that it is on the right half of an open book from the readers point of view. This is the reason why the previous page was blank (the previous page is the back side of the cover)

SPRAY

© 2001 Wolfgang Theiss

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: January 2016 in (whereever you are located)

Publisher

...enter name...

Managing Editor

...enter name...

Technical Editors

...enter name...

...enter name...

Cover Designer

...enter name...

Team Coordinator

...enter name...

Production

...enter name...

Special thanks to:

All the people who contributed to this document, to mum and dad and grandpa, to my sisters and brothers and mothers in law, to our secretary Kathrin, to the graphic artist who created this great product logo on the cover page (sorry, don't remember your name at the moment but you did a great work), to the pizza service down the street (your daily Capricciosas saved our lives), to the copy shop where this document will be duplicated, and and and...

Last not least, we want to thank EC Software who wrote this great help tool called HELP & MANUAL which printed this document.

Table of Contents

Foreword	7
Part I Introduction	8
1 About SPRAY.....	8
2 Overview.....	9
Why using spectral ray tracing?	9
How to find how it works	11
The new user-interface of version 2.4	11
SPRAY algorithm	18
Principle	18
Simulation logic.....	19
General properties of SPRAY objects.....	21
Part II Optical constants	22
1 Optical constants.....	22
Part III Scatterers	23
1 Overview	23
2 General scatterers.....	24
3 Mie scatterers.....	25
4 Extended Mie scatterers.....	27
5 Fluorescent scatterers.....	32
6 Fluorescent Mie scatterers.....	36
7 Composite scatterers.....	36
8 The RT file format.....	38
9 The View_RT utility.....	39
Part IV Interfaces	43
1 Overview	43
2 Pre-defined interfaces.....	43
3 Ideal diffusor.....	44
4 Specular and diffuse reflection.....	46
5 Layer stacks.....	47
Part V Geometric objects	56
1 Overview	56
2 Light sources.....	58
Overview	58
Point light source	58
Rectangular light source	60
Circular light source	62

Volume light source	64
Complex light source	65
3 Detectors.....	67
Surface detectors	67
Rectangular detector.....	67
Screen	71
Arrays	76
Linear array	76
Spherical detector arrays.....	79
Volume detectors	85
Grave	85
Cemetery	85
Absorbing material.....	85
4 Interface objects.....	85
Overview	85
Rectangular interface	86
Triangle	88
Circle	89
Sphere	90
Sphere segment	90
Cylinder (closed)	92
Cylinder (open)	94
Cone	94
Rectangular box	96
Prism	97
ATR crystal	99
Ellipsoid segment	101
Paraboloid segment	103
Circular aperture	106
Converging lens	106
Diverging lens	108
User-defined surface: Rectangular basis	111
User-defined surface: Circular basis	113
Periodic surface texture	115
Complex objects	122
Complex objects: Introduction.....	122
Complex objects: Subobject types	125
Complex objects: Creating input data.....	127
Importing objects from CAD programs.....	129
5 Special objects.....	133
Overview	133
Polarizer	133
Part VI Cameras	134
1 Overview.....	134
2 Rendered view.....	134
Part VII Simulation options	140
1 Spectral range and angle resolution.....	140
2 How many rays do you need?.....	141
3 Start options.....	142

Part VIII Distributed computing	143
1 Overview.....	143
2 Master PC.....	144
3 Client PCs: The tool NIGHTSHIFT.....	145
4 Strategy for distributed computing.....	147
5 OLE automation demo.....	148
Part IX OLE automation	150
1 Overview.....	150
2 Handling the OLE server.....	150
3 Object parameters.....	151
4 Simulation parameters.....	154
5 Retrieving results.....	155
6 Video generation.....	155
Video generation	155
Demo video	159
Part X Automated parameter fitting	160
1 Introduction.....	160
2 Step-by-step example.....	160
SPRAY model	160
Starting configuration	163
Preparing the parameter fit	164
Running the fit	169
Part XI References	172
1 References.....	172
Index	173

Foreword

This is just another title page
placed between table of contents
and topics

1 Introduction

1.1 About SPRAY



Spectral Ray Tracing

© W.Theiss

W. Theiss – Hard- and Software
Dr.-Bernhard-Klein-Str. 110, D-52078 Aachen, Germany
Phone: + 49 241 5661390 Fax: + 49 241 9529100
e-mail: theiss@mtheiss.com web: www.mtheiss.com

August 2008

This text was written using the program Help&Manual (from EC Software). With this software we produce the printed manual as well as the online help, PDF files and HTML code for internet documents - with exactly the same text input! This is a very productive feature and makes the development of the documentation quite easy. However, for this reason the printed manual sometimes contains some 'strange' text fragments which seem to have no relation to the rest of the text. These might be hypertext jumps in the online help system which - of course - loose their function in the printed version of the manual.

Start here!

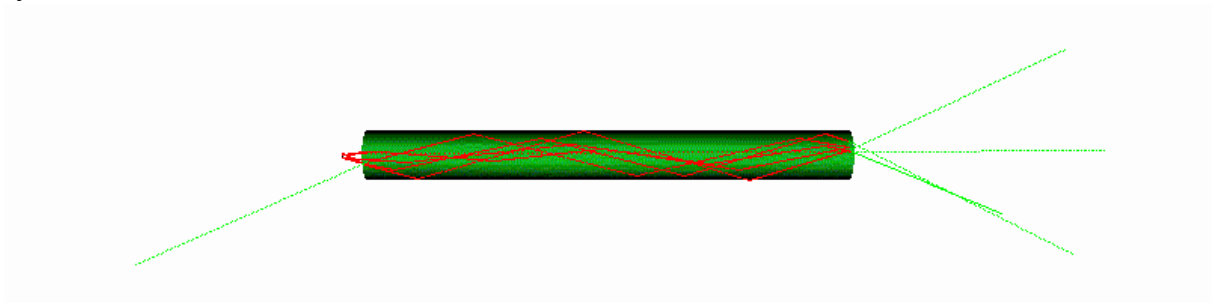
1.2 Overview

1.2.1 Why using spectral ray tracing?

A complete simulation of optical setups following the path of the radiation from the light source to the detector(s) can be useful in several situations, in particular in the design phase of an arrangement or to investigate an unexpected behaviour occurring in already existing components.

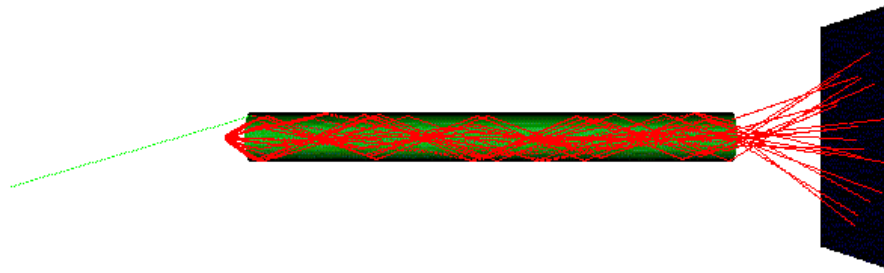
The very realistic simulations of optical spectra recorded in standard setups like external or internal reflectance or transmittance that can be done by the SCOUT software are based on some shortcuts which are good approximations in most cases. SCOUT works with a well defined, perfectly parallel incoming beam of radiation and assumes that all the radiation being reflected or transmitted reaches the detector, or - at least - that the normalization procedure in the experiment efficiently takes into account all loss effects.

Nevertheless, sometimes it is not possible to take this simple road and one has to deal with effects which cannot be taken into account straightforwardly. Here is a simple example. A light source on the left side illuminates the side of a solid cylinder with known refractive index. The radiation is emitted into a cone with a given opening angle. How much radiation comes out at the right side of the cylinder?

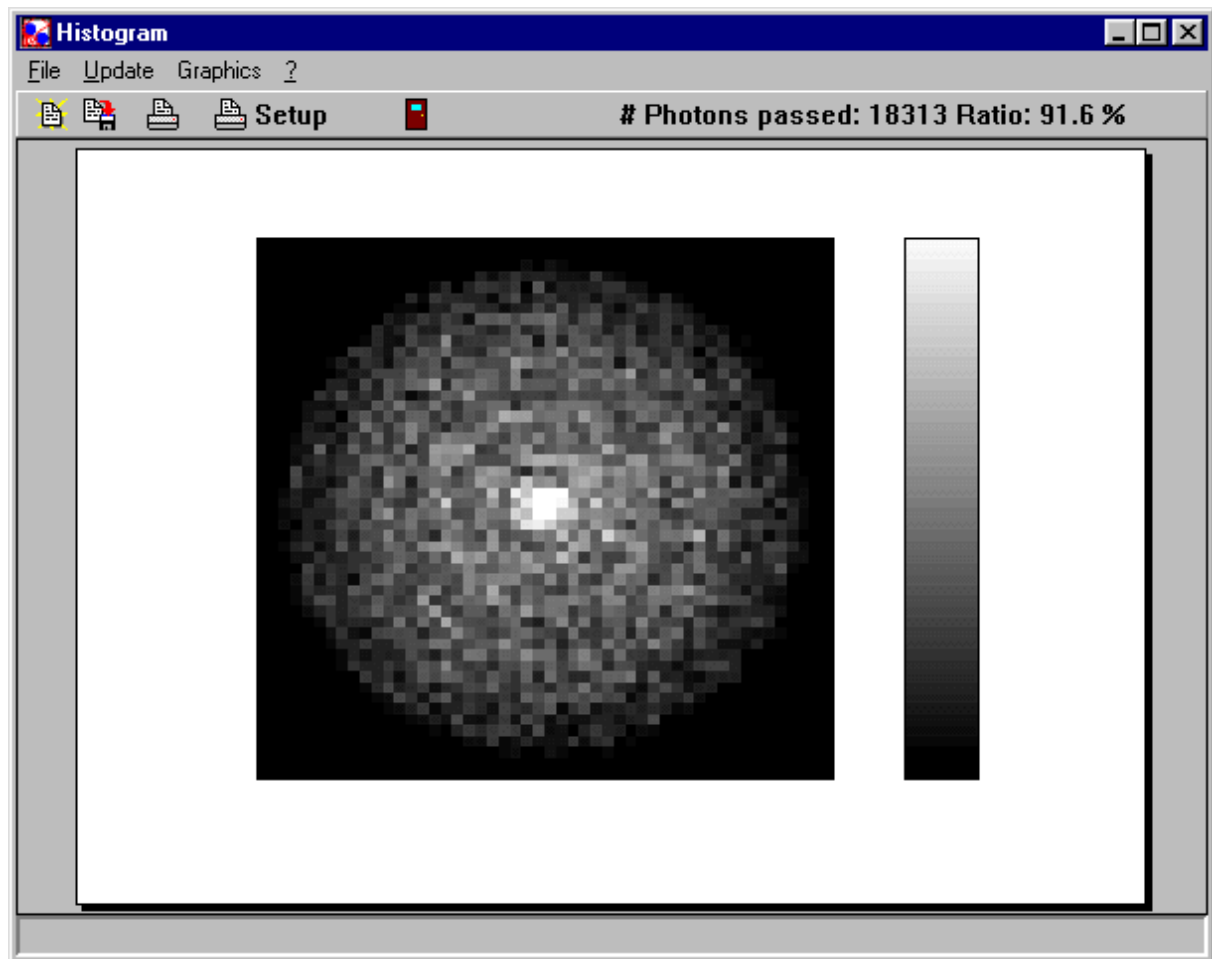


Well, it's not so hard. First compute the reflectance and transmittance of the vacuum-cylinder interface for various angles of incidence and polarizations. Then try to find out which angles of incidence occur with which weight and do a reasonable averaging. Then you know how much radiation enters the cylinder. Inside the cylinder total reflection occurs and no radiation is lost - unless some fraction of the light is incident at angles above the critical angle. How much is that? Well, the surviving radiation hits the right end of the cylinder and escapes. Unless it is reflected back to the left. Write down the transmitted amount and follow the reflected one which might be reflected at the left side again and come back ...

If you agree that this back and forth of radiation (that gets more and more complicated the longer you think about it) should be followed by a computer you are ready for SPRAY. You tell the program what objects do you have, how they are arranged and what you want to know, and then SPRAY sends for you photon after photon into the setup and records what happens to them. In our example you could place a screen to the right of the cylinder,



send a few thousand rays and see after a few seconds the distribution of radiation:



The throughput of the given setup is 91.6 %.

You can do this kind of analysis with frequency dependent optical constants of all involved materials which allows realistic computations of optical setups. This application example is discussed in detail in the introductory tutorial (see separate documentation). Some typical SPRAY applications are discussed in a separate documentation as well.

The next section gives a SPRAY overview.

1.2.2 How to find how it works

If you never worked with SPRAY before you should follow the first *SPRAY tutorial* (separate documentation) step by step. You will be guided through the most important sections of the program - as short as possible.

You should also inspect the discussion of *SPRAY examples* (separate documentation) in order to get an impression of what you can do with the program.

If you are an experienced SPRAY user, please read the section "What's new in version 2.4" about major SPRAY changes in 2008.

More detailed information is given in the following sections:

- Description of the [SPRAY algorithm](#)
- In most cases you will need optical constants for the materials used in your scenery
- SPRAY contains a very powerful technique to include light scattering media
- Materials and light scattering media are separated by interfaces
- Your scenery is composed of geometric objects which are usually covered by interfaces
- Objects called Cameras can be used to visualize your SPRAY scenery
- Having defined and positioned all required objects you can start the ray-tracing simulation
- Starting with SPRAY 2.0 you can work on a SPRAY simulation with the computational power of several PCs. See the section on Distributed computing for further information.
- If you have to perform many simulations you should automate your work with OLE automation.
- In order to optimize an optical system you can automatically adjust parameters of SPRAY objects. Be prepared, however, that the optimization will take a long time.

General information:

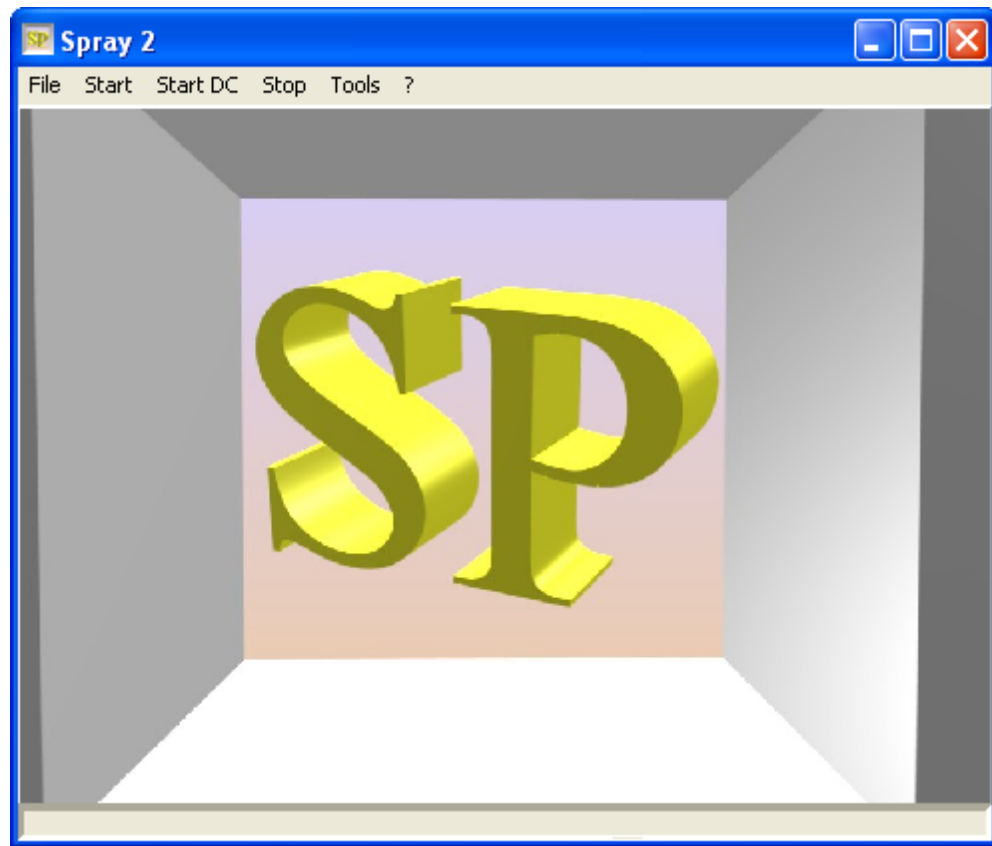
- SPRAY uses many **lists** - you can get information about working with list in the SCOUT technical manual. Your SPRAY package very likely contains a printed version of this document.
- Objects defining optical constants as well as detectors display spectra in 2D or 3D graphs. See the separate documentation 'A **graphics course**' to learn how to handle these.

1.2.3 The new user-interface of version 2.4

In August 2008 SPRAY has been upgraded to version 2.4. This section gives a short summary of the most important program changes.

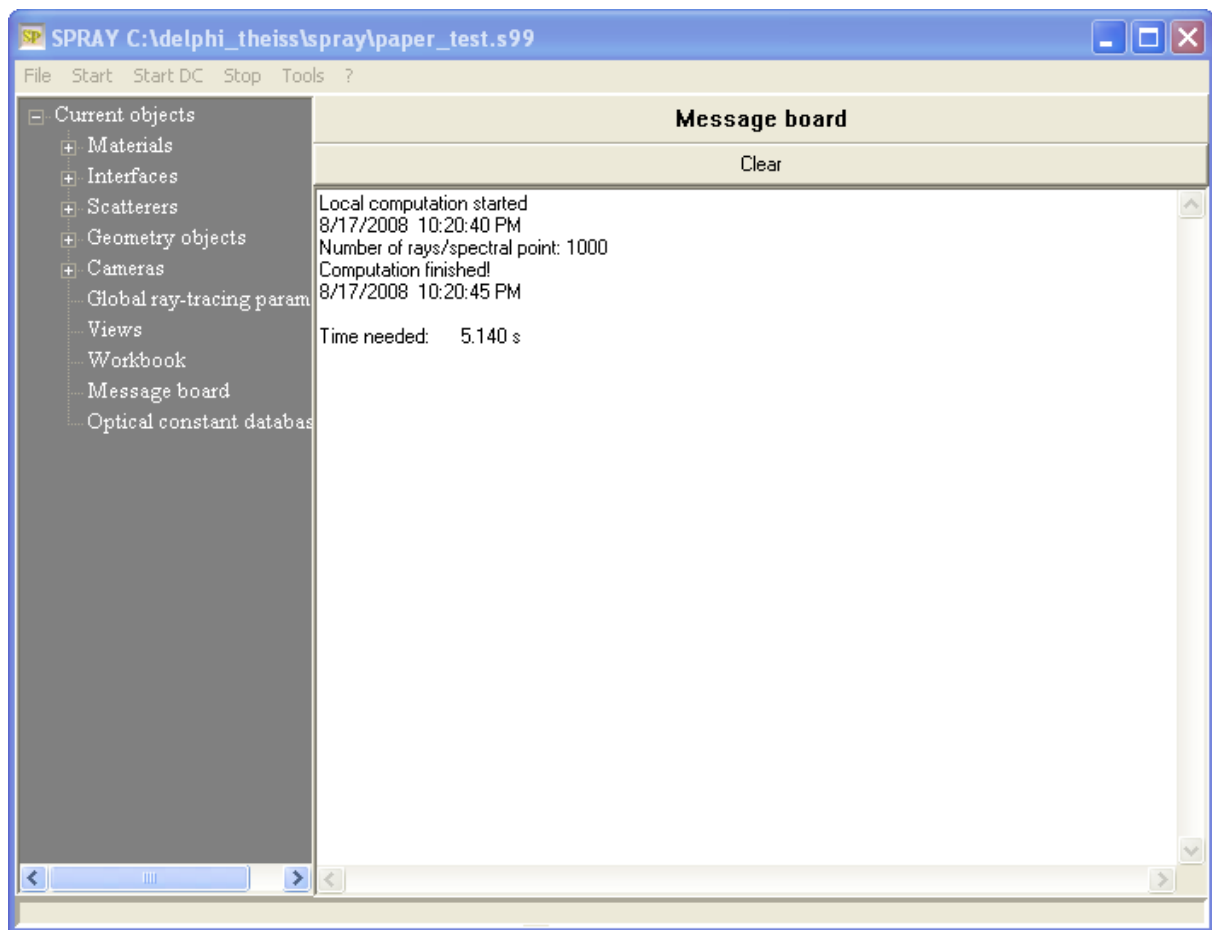
Main window

In order to indicate that SPRAY 2.4 significantly differs from previous versions, the main window looks different now:



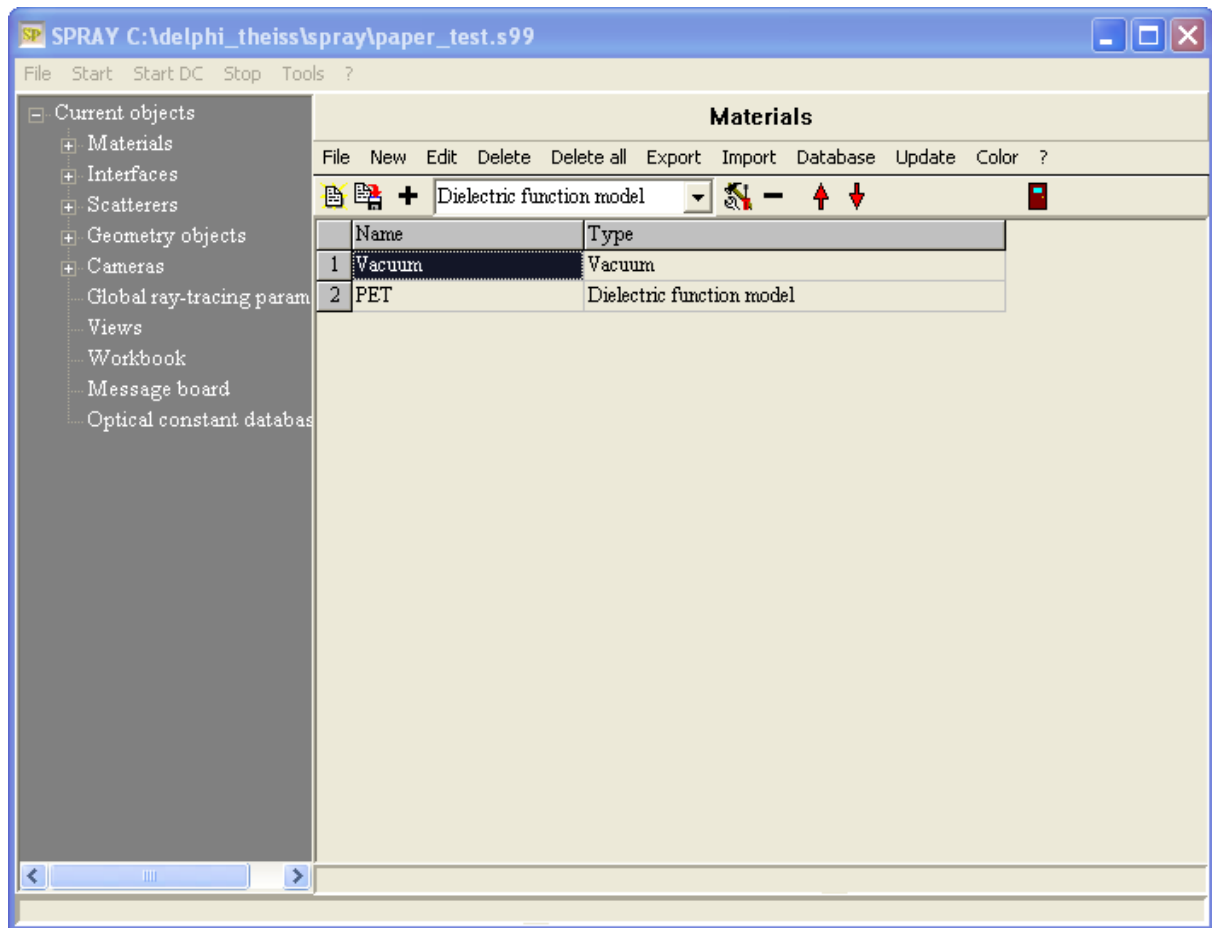
Like our SCOUT thin film analysis software, you can work in SPRAY on two levels: The main window starts in the so-called 'main view level' (as shown above). Unlike in SCOUT, the main view level in SPRAY is still under construction, and you cannot use it at the moment. In the future you will be able to define your own views on SPRAY objects and create flexible user interfaces. What is ready is the so-called treeview level which you enter by pressing F7. With F7 you toggle between the treeview and the main view level.

In the treeview level, you see a treeview of the current SPRAY objects on the left. The branches of the object tree are the various lists that build up the SPRAY configuration:

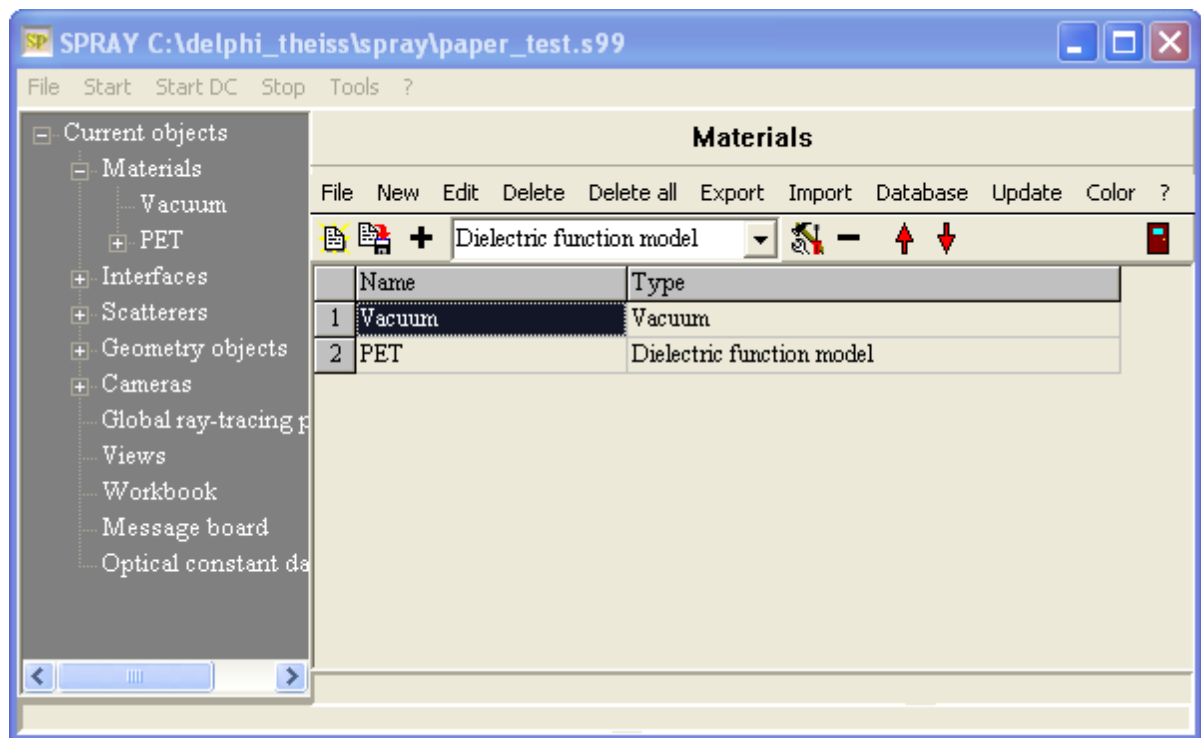


With a **right mouse click** you can open an object in the treeview - it will show its content in the right half of the main window. The example above shows the situation after a right-click on 'Message board' (The message board has replaced the 'Information panel' that was shown in the main window of previous SPRAY versions.).

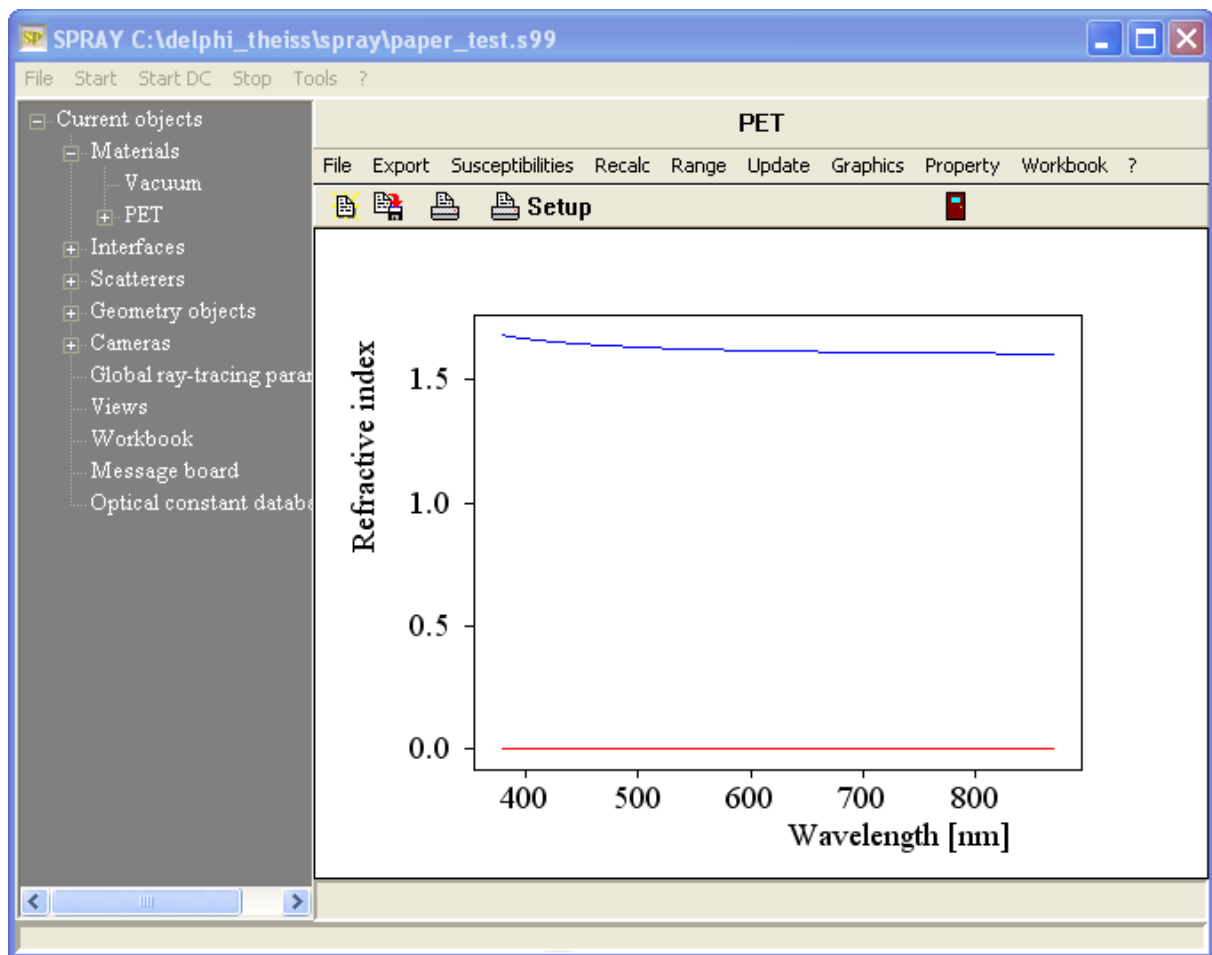
A right click on Materials, for example, opens the list of materials that you may recognize from older versions of SPRAY:



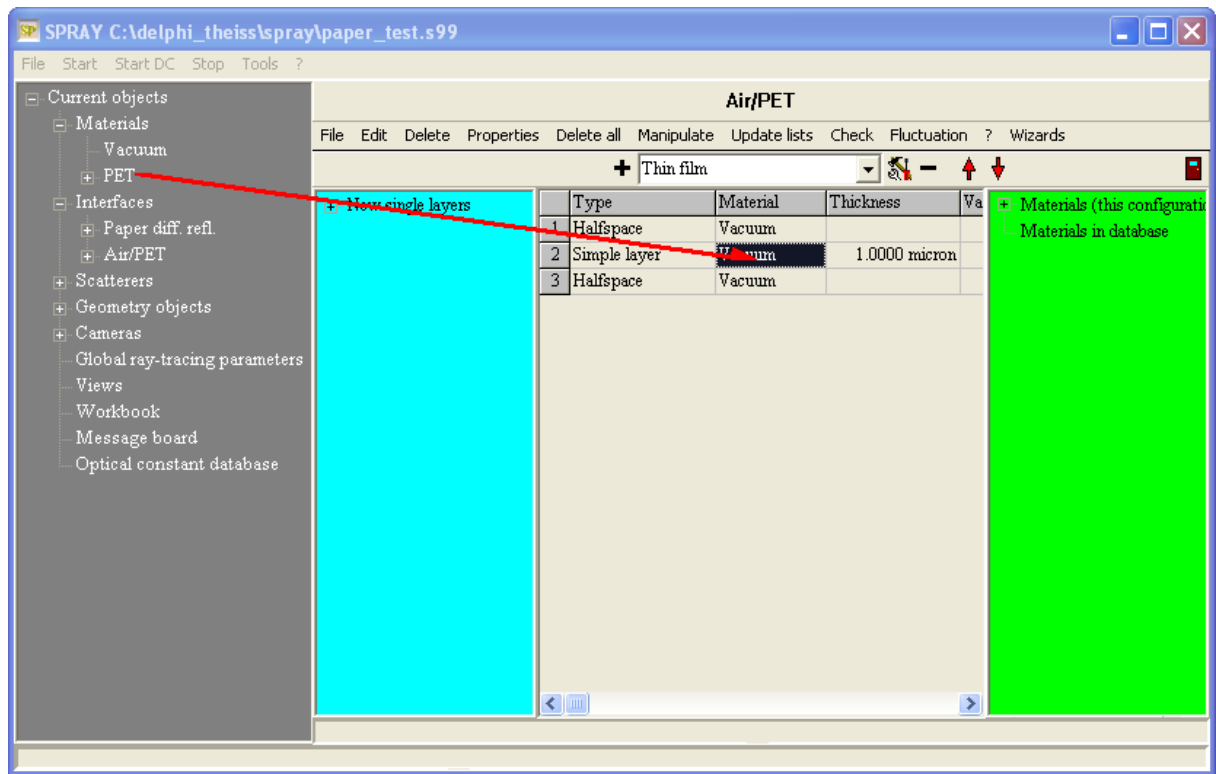
Most list objects appear in the treeview as well if you open the treeview branch by a left mouse click on the + to the left of a tree branch. If you expand the tree branch **Materials**, for example, you will get the following situation:



A right-click on the material PET in the treeview opens this object:

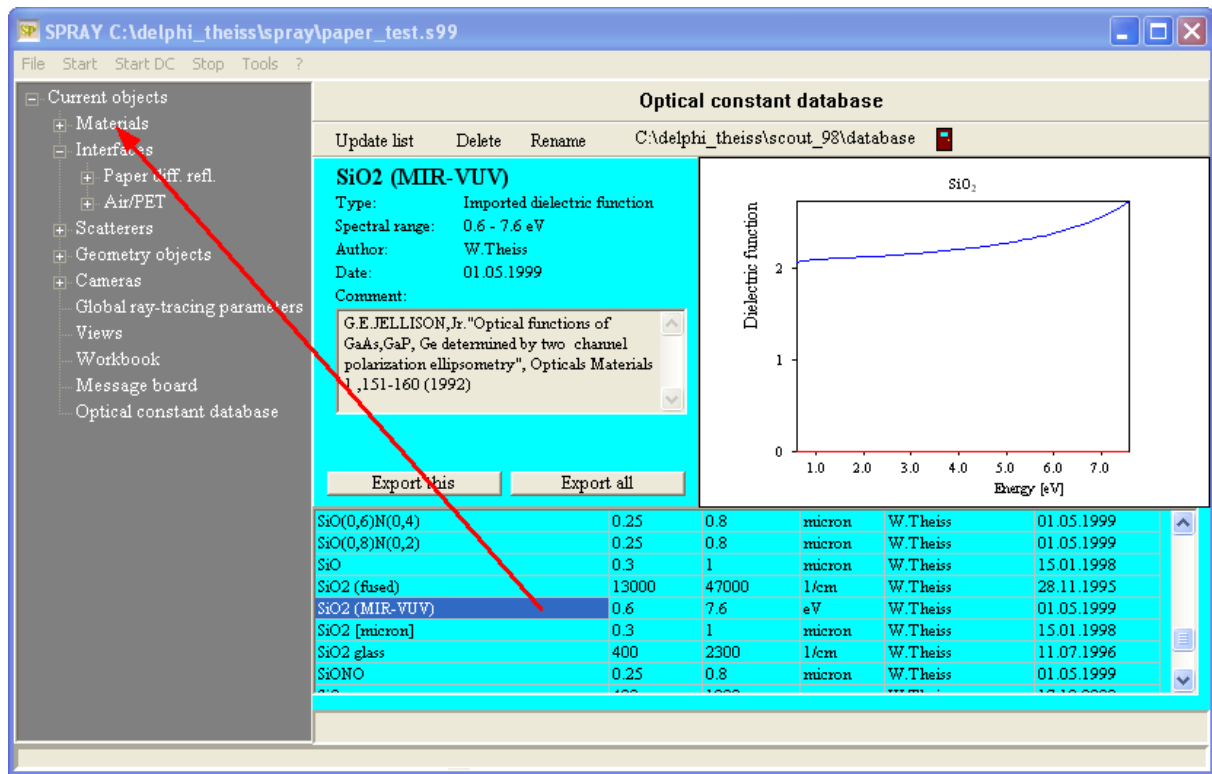


Drag&Drop operations are done in the following way: Open the source object of the drag operation in the treeview and then open the target object for the drop operation with a right-mouse click in the treeview. Then drag from the source object in the treeview to the destination in the right half of the window. Here is an example: In order to assign a material to a layer in a layer stack, expand the treeview branch **Materials** and open the layer stack object as follows:



The red arrow shows the direction of the drag&drop movement.

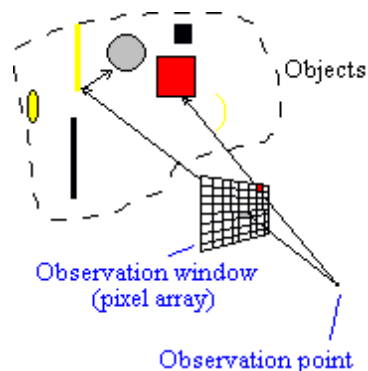
There is only one drag&drop exception: If you want to move a material object from the **Database of optical constants** you have to drag the material from the database grid on the right to the **Materials** branch in the treeview on the left. Here is an example:



1.2.4 SPRAY algorithm

1.2.4.1 Principle

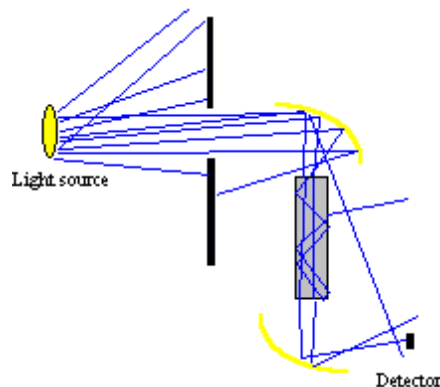
Widely used ray tracing algorithms for a more or less realistic simulation of pictures ('virtual reality') start with rays from an observation point, send them through the pixels of a pixel array and trace back which object of the collection of things in the defined scenery is hit first (see the figure below).



The color and brightness of the pixel is set according to the color and brightness of the hit point which - in turn - depends on its illumination from the surrounding light sources and the light reflections due to neighbored objects. In case of polished surfaces one has to split up the ray into a reflected and a transmitted contribution and follow up their paths through the scenery - typically not more than

five propagation iterations are taken into account. Usually quite crude assumptions on the specular or diffuse reflection properties of the surfaces are made - the quality of the result depends of course on the quality of the component's optical properties that are input to the software.

The 'virtual reality ray tracing method' discussed above is used in SPRAY only for taking pictures of the setup (called 'rendered views'). For a quantitative simulation of optical devices a different approach is more appropriate which is sketched below. A direct simulation of what happens in reality is done: The light source emits light (rays) in various directions and the objects in the scenery absorb or re-direct rays until finally the rays are absorbed or vanish to 'infinity' (if no object is hit any more).



Working with rays is of course only an approximation: Actually light sources emit quite complex radiation fields and generally one must take into account phase relations between partial waves (as is apparent in optical interferometry or diffraction effects). The ray-tracing approach neglects phases of electric and magnetic fields and describes light propagation in terms of 'intensity needles' with well defined direction and negligible dimensions perpendicular to the propagation direction.

Using light rays the quite complex 'real life' (inter)actions can be separated into actions of distinct objects characterized by certain geometric properties and abilities to process incoming rays. This is a situation tailored for object oriented programming (OOP) which has been used to realize the SPRAY program. OOP allows the definition of objects with certain properties and possible actions which directly correspond to real objects: The SPRAY actors are light sources, mirrors, detectors and so on which are controlled and coordinated by an object manager sending and receiving messages.

1.2.4.2 Simulation logic

The basic algorithm is based on the following scheme which is given here almost like it is defined in the program's source code:

Repeat

Manager selects frequency. SPRAY starts with the high energy end of the selected spectral range.

Manager tells all objects the current frequency.

Objects do frequency dependent initialization if necessary.

Repeat

Manager tells light source to create a ray.

Light source creates ray with an initial position and direction depending on the type of the light source. The initial polarization is random. The ray travels in a certain material assigned to the light source. The material has a refractive index, an absorption coefficient and may contain small scattering and absorbing particles (which are simply called scatterers in SPRAY) in a user-defined concentration.

Repeat

Manager asks all objects whether they get hit by the present ray.

All objects determine if ray hits them and return position of hit.

Manager selects closest hit point. If two objects report the same distance (within some tolerance) the one with higher priority wins.

Manager determines whether an absorption, scattering or fluorescence process occurs on the ray's path from its present position to the closest hit point (if the ray 'moves' in a material which may absorb, scatter or absorb and re-emit light)

If scattering occurs the ray is started at the position of the scattering event with a new direction which is determined by the scattering characteristics.

If fluorescence occurs the new frequency (lower than the current frequency) is computed according to the current fluorescence properties. The starting point of the new ray is added to a waiting queue of rays to be processed when the lower frequency is going to be processed.

If no absorption, scattering or fluorescence occurred:

Manager asks object with closest hit what happens with ray.

Object absorbs ray or changes its direction and/or possibly changes the material in which the ray moves (in case of an 'interface' object).

until ray is absorbed or no hit with one of the objects occurs.

until a user-defined number of rays has been processed

Repeat

Manager gets a ray from the 'fluorescence waiting queue' and starts it at its location in a random direction and with random polarization

Repeat

Manager asks all objects whether they get hit by the present ray.

All objects determine if ray hits them and return position of hit.

Manager selects closest hit point. If two objects report the same distance (within some tolerance) the one with higher priority wins.

Manager determines whether an absorption, scattering or fluorescence process occurs on the ray's path from its present position to the closest hit point (if the ray 'moves' in a material which may absorb, scatter or absorb and re-emit light)

If scattering occurs the ray is started at the position of the scattering event with a new direction which is determined by the scattering characteristics.

If fluorescence occurs the new frequency (lower than the current frequency) is computed according to the current fluorescence properties. The starting point of the new ray is added to a waiting queue of rays to be processed when the lower frequency is going to be processed.

If no absorption, scattering or fluorescence occurred:

Manager asks object with closest hit what happens with ray.

Object absorbes ray or changes its direction and/or possibly changes the material in which the ray moves (in case of an 'interface' object).

until ray is absorbed or no hit with one of the objects occurs.

until all rays in the waiting queue (generated by fluorescence events) are finished.

Manager tells all objects that the current frequency simulation is done.

Objects do clean-up if necessary.

until all frequency positions have been processed.

1.2.4.3 General properties of SPRAY objects

All objects (existing ones and those of the future) must be able to react to a number of manager commands and to perform certain calculations and user interactions. Here is an overview to give you an idea about SPRAY objects' structure and abilities.

Interfacing with the user and SPRAY:

- set geometrical data in a user dialog

- determine frequency dependent properties in a user dialog

- objects write their data to SPRAY binary files

- objects read their data from SPRAY binary files

Frequency scan:

- objects adjust their behaviour when a new frequency is selected

- objects eventually update their internal data when the simulation for a frequency is done

Ray-tracing:

- objects calculate possible hit points for arbitrary rays (if there are more than one hit points the

 - object selects the first one, i.e. the one closest to the origin of the ray)

- objects decide what happens to a ray after a hit

Whenever you would like to have a type of object in SPRAY that doesn't exist presently you can suggest to implement it - it might be useful for other users, too. The fastest implementation is achieved when you can even supply a suitable way to compute the hit point for arbitrary rays - this is usually the most difficult part, especially for objects with complicated shapes.

The optical properties of all materials that are used in SPRAY are defined using the same objects as in the SCOUT software package. As in SCOUT, you can connect to a database of optical constants and layer stacks and use its items.

Since SPRAY does a fully three-dimensional ray-tracing you as a user must specify all the geometric data of all the objects which can be a tedious job. All lengths are specified in 'cm' - there is no option to change this basic unit up to now.

2 Optical constants

2.1 Optical constants

The quality of optical simulations may critically depend on the choice of the underlying optical constants. SPRAY contains very powerful dielectric constant models as well as a large database of literature data and pre-defined models.

The definition of optical constants (or dielectric functions) in SPRAY is exactly the same as in our spectrum simulation program SCOUT. Please refer to the SCOUT technical manual (online version: www.mtheiss.com/docs/scout2/index.html) for details. A printed version of the SCOUT manual is included in your SPRAY documentation.

Optical constants enter SPRAY simulations in the following ways:

- Each light source is embedded in a certain material. Its optical constants are assigned to the light source object by drag&drop from the list of materials. The ray continues to move in that medium unless it crosses an interface which defines a change of the optical constants.
- Interfaces may consist of layer stacks. An optical constant object from the list of materials must be assigned to each layer of a layer stack. The stack of layers is embedded between a top and a bottom 'halfspace'. If the materials assigned to top and bottom halfspace are different and a ray crosses the interface, it continues its motion in the new material.

3 Scatterers

3.1 Overview

Explicit, individual scattering objects

Scattering of light can be simulated by curved geometric objects. You could add a glass sphere to your scenery, and rays hitting the sphere will change their direction according to the laws of reflection and refraction. Inside the sphere absorption may occur. If you want to introduce many explicit scatterers into your SPRAY model, please consider using a complex object.

Continuous scattering media

If you want to treat systems with many scattering objects like dust clouds or paints, and if these objects are small compared to the other geometric objects of the model, you can probably work with a continuum of scatterers. If SPRAY light rays move through such a continuum there is a certain chance for scattering and absorption per distance, depending on the shape, the optical constants and the density of the scatterers. If a scattering event occurs, the ray will take a new direction the probability of which is also a function of the particle's shape and optical constants.

The scattering and absorbing medium is characterized by the following quantities:

$$S = \frac{\text{Scattering probability}}{\text{distance}} \quad [S] = \frac{1}{\text{cm}}$$

$$K = \frac{\text{Absorption probability}}{\text{distance}} \quad [K] = \frac{1}{\text{cm}}$$

$$W(\Theta) = \text{probability for scattering angle } \Theta \quad [W(\Theta)] = \frac{1}{\text{cm}}$$

$$S = \iint W(\Theta) \sin \Theta d\Theta d\Phi$$

The scattering angle Θ is the angle between the direction of the incoming ray and that of the scattered ray. Up to now SPRAY assumes cylindrical symmetry of the scattered intensity around the initial direction - hence there is no Φ -dependence.

The following types of scatterers are implemented at present:

- General scatterers
- Mie scatterers
- Extended Mie scatterers
- Fluorescent scatterers
- Fluorescent Mie scatterers
- Composite scatterers

In the main window of SPRAY there is a button labeled **Scatterers** which gives access to the **list of scatterers**. This list manages the various scatterers of a SPRAY model.

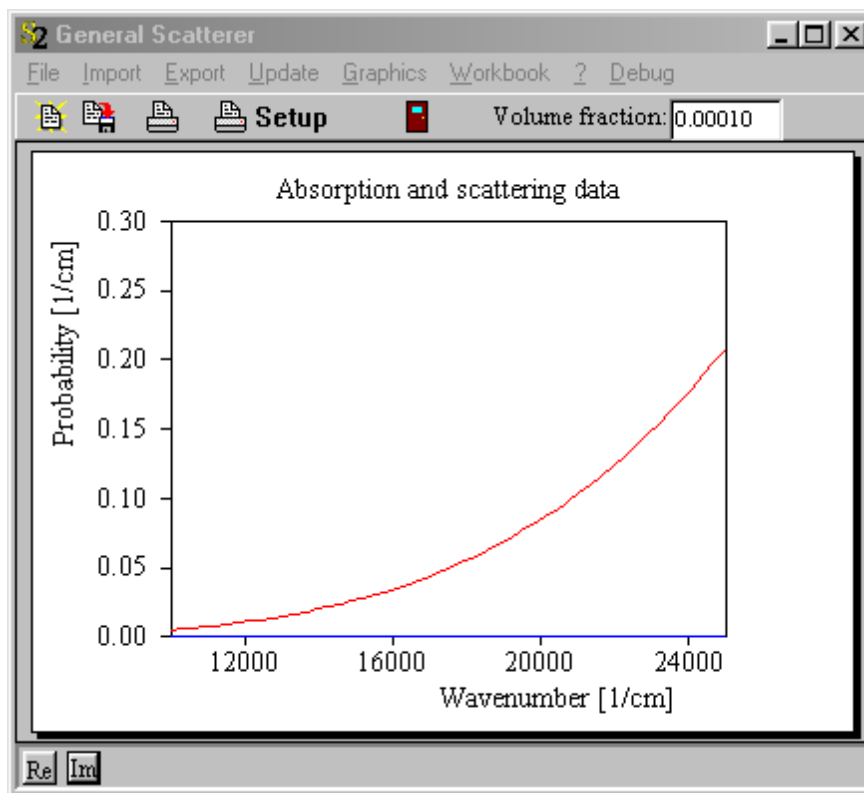
Please consult the documentation on **SPRAY examples** for some applications of scatterers.

3.2 General scatterers

This type of scatterers simply imports the required spectral data to compute K , S and $W(\Theta)$ from external data files. The external files must be created by other programs.

For practical reasons, the external data must be provided in a form that the volume fraction can be changed afterwards. In fact, the volume fraction of the scatterers is the only parameter of general scatterers.

General scatterer windows look like this:



Scattering data are imported using the **Import** menu command. The following file formats for data input are supported:

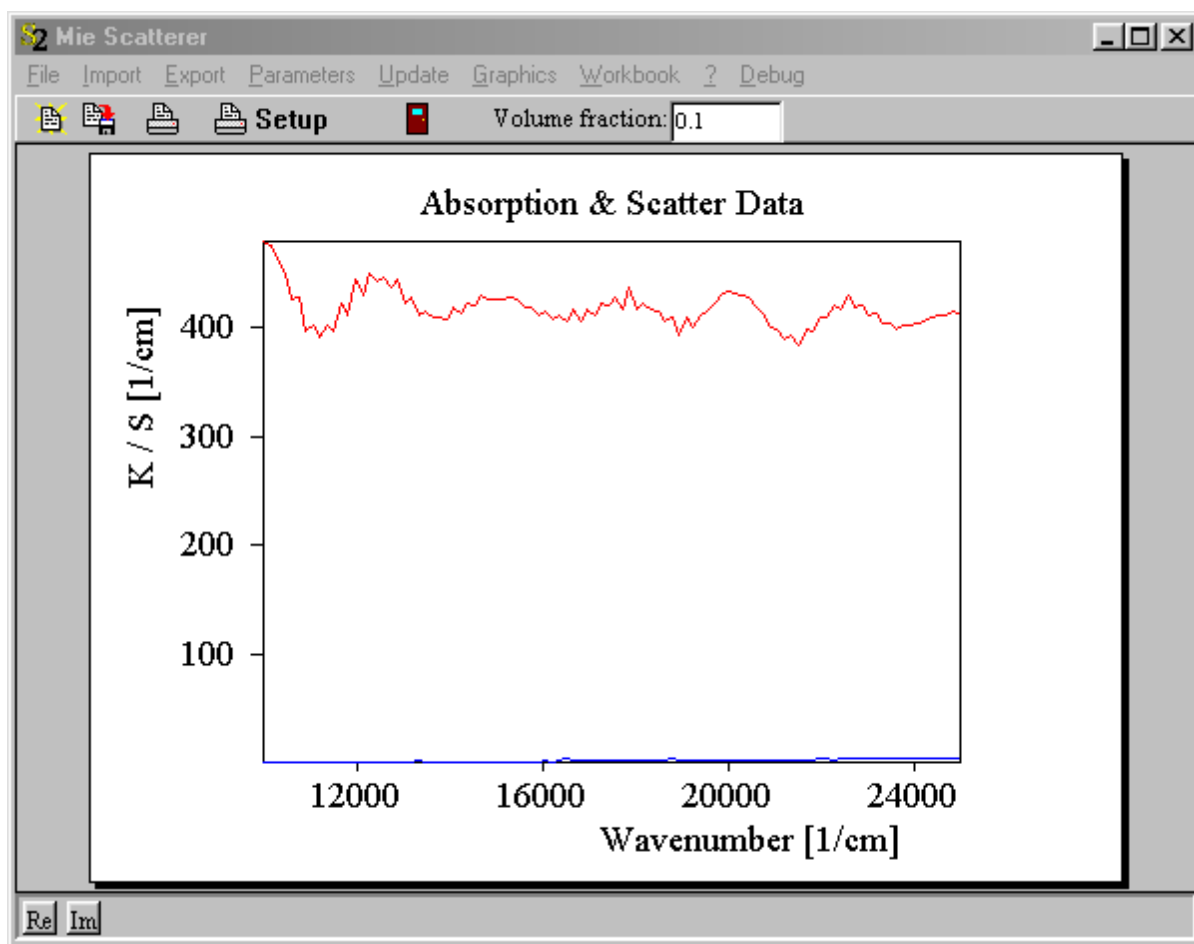
- RT file format (RT = Radiation Transfer), described below

After setting the **volume fraction** in the corresponding input box you have to use the **Update** menu command to compute K , S and $W(\Theta)$. K and S are displayed in the window (blue and red curve). At present, the angle dependence cannot be viewed in general scatterer objects but has to be

inspected with the external program View_rt delivered with SPRAY.

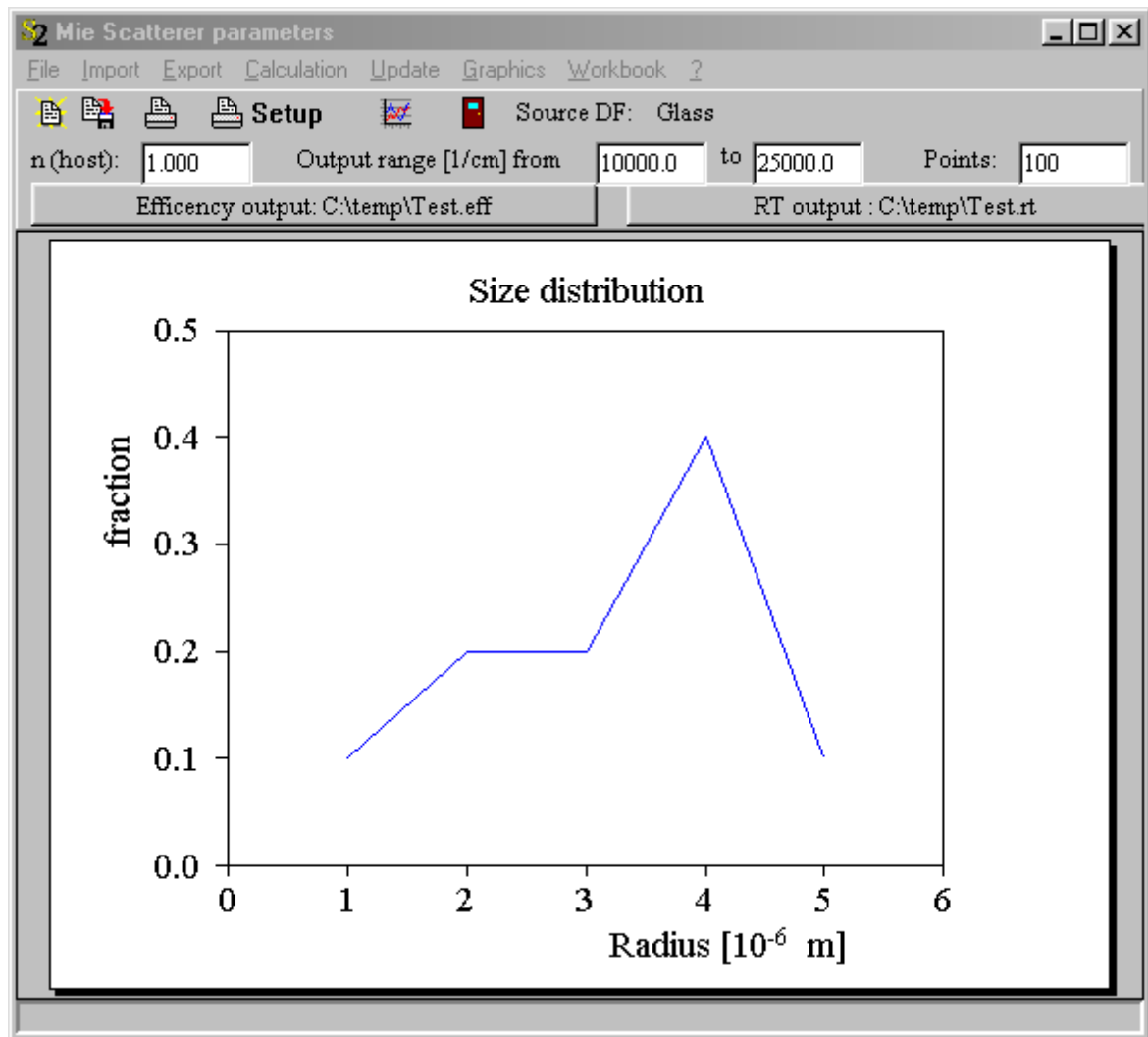
3.3 Mie scatterers

Objects of this type compute the average absorption and scattering characteristics of collections of spheres. The main window is this:



Like with general scatterers you can import rt-files and set the volume fraction of the scattering particles. In addition, you can compute your own absorption and scattering data using an external Mie program (the quite old-fashioned DOS program RTMIE, delivered with SPRAY) which will do the computational work.

In a subwindow of Mie scatterer objects which you open by the **Parameter** command you can set the required parameters for the Mie computation:



You have to specify the following items:

- The optical constants of the sphere material: Open the list of dielectric functions and drag an entry to the text field right to the label 'Source DF:'. Make sure that the name of the right dielectric function is displayed (in the example above: Glass).
- The refractive index of the host material surrounding the spheres: Enter the value of the refractive index in the field next to the label 'n(host)'. The refractive index must be constant and real. If you use the generated scattering data later on in a SPRAY simulation you must use the same surrounding refractive index as used here in the Mie computation.
- The wavenumber range for the output data: For historical reasons the spectral unit to be used in Mie computations must be wavenumbers. You have to specify the minimum and maximum value and the number of data points.
- The files for the computed efficiency data (see below) and the RT output data (that are later used in SPRAY).
- The radius distribution of the spheres: The Mie computations are performed for a distribution of sphere sizes which is displayed in the graph of the window. The radius distribution shown above

has the following meaning: If you take a collection of 1000 spheres you will find 100 particles with radius 1 micron, 200 with radii 2 and 3 microns, 400 with radius 4 microns and 100 with radius 5 microns. You can use the **Import** command to import the radius distribution from a file.

Alternatively you can import a distribution from the workbook using the **Workbook** commands. Note that the sphere radius has be entered in 'm'.

In order to perform Mie computations with the RTMIE program **the program file rtmie.exe must be present in the directory c:/mie**. This is an old FORTRAN program running in a DOS box. **It cannot handle long filenames and filenames or directory names with blanks correctly.** Hence you should be careful with the choice of filenames in your Mie work.

Activation of the **Calculation** command starts the following actions:

- SPRAY will create files that contain the dielectric function and the radius distribution of the spheres.
- The filenames and the other parameters are written to a small textfile called rtmie.ctl which is stored in the fixed directory c:/mie.
- RTMIE is started. It will run in a DOS box, load the input data, show the progress of the work and finally create the required output data.

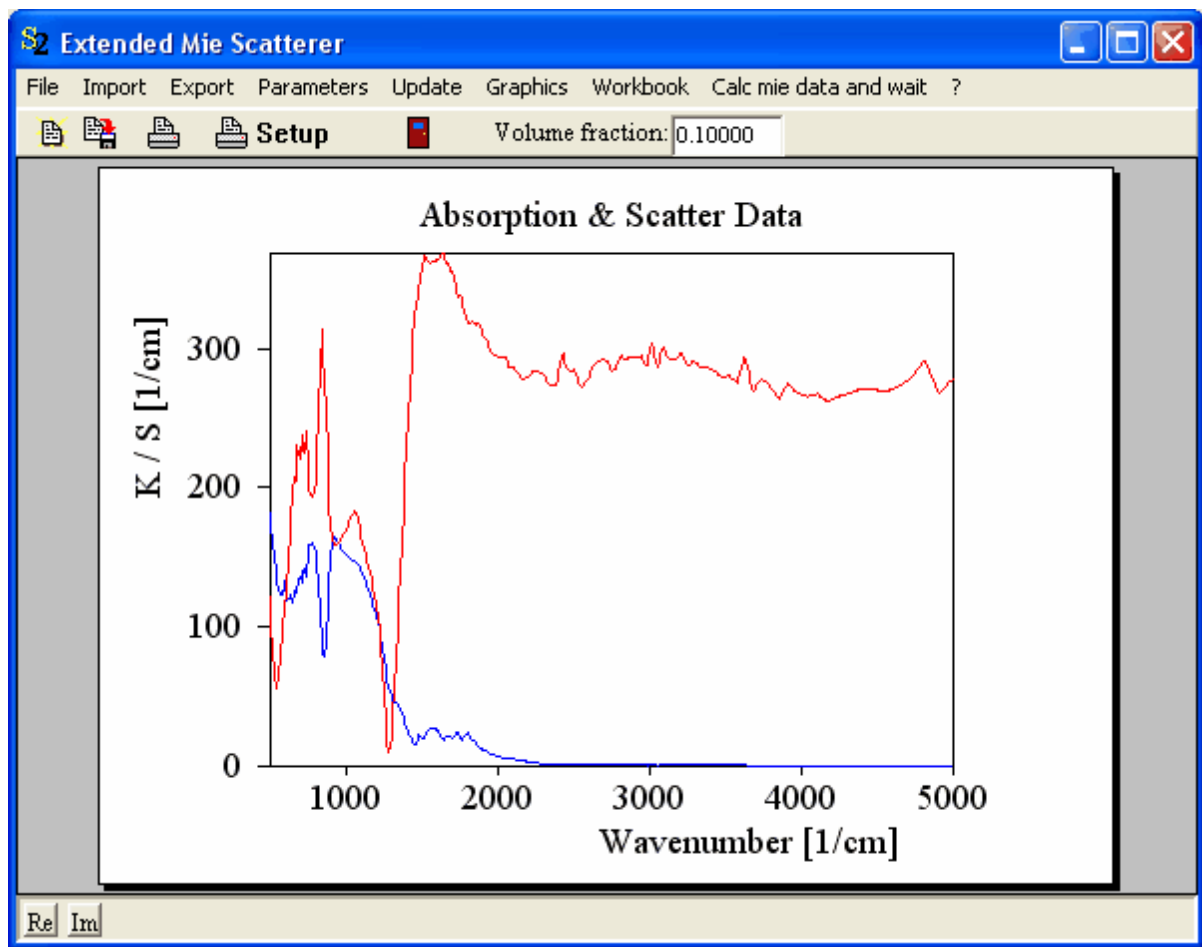
When the RTMIE program quits you can start to use the computed RT data for your SPRAY computations.

3.4 Extended Mie scatterers

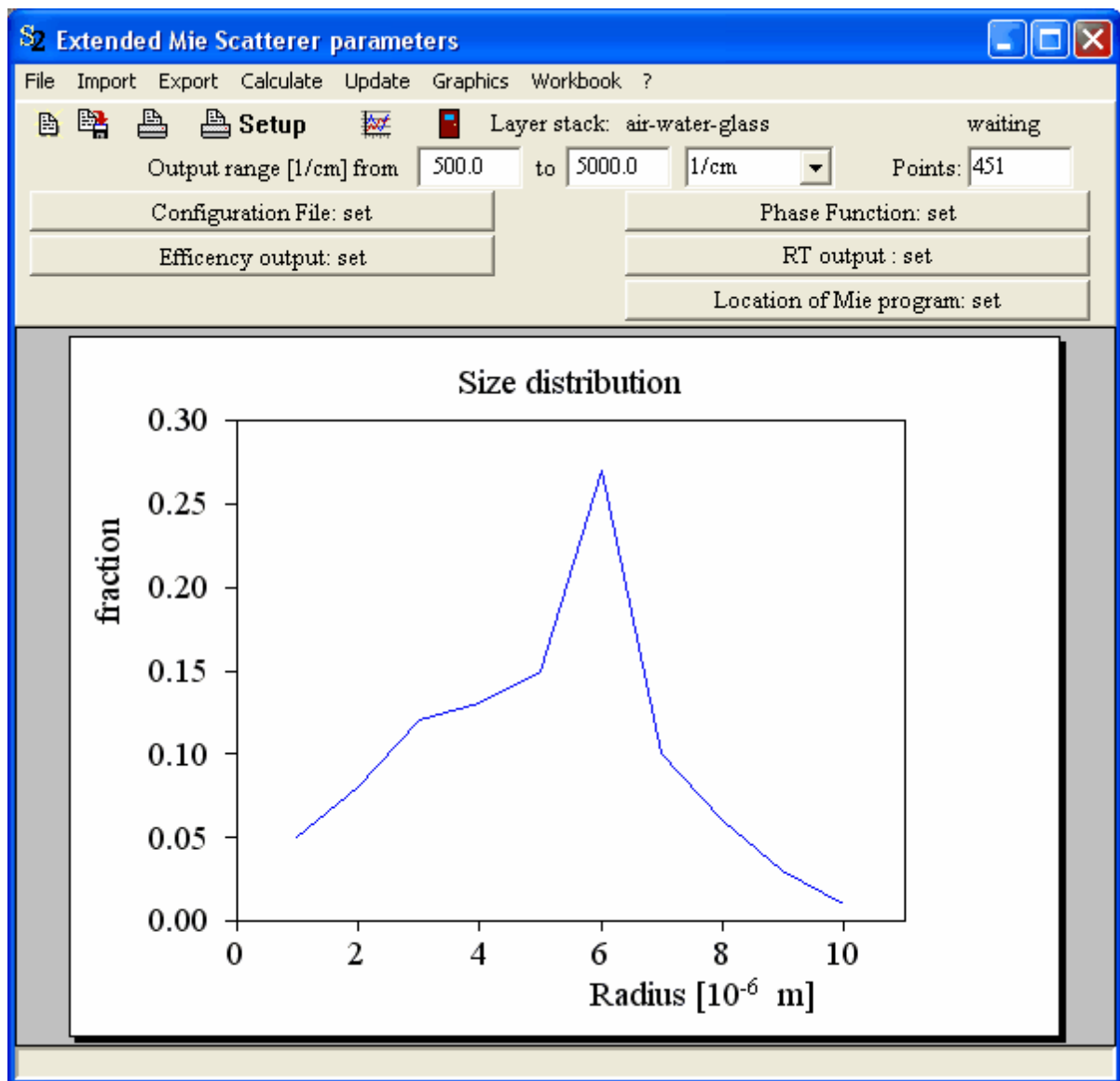
This program feature does not work in all SPRAY versions. The external program layeredpheres.exe must have been included in your SPRAY purchase.

Objects of this type are very similar to Mie scatterers. You can compute the scattering and absorption characteristics of multiply layered spheres and use these data in your SPRAY multiple scattering calculations.

The main window looks like this:



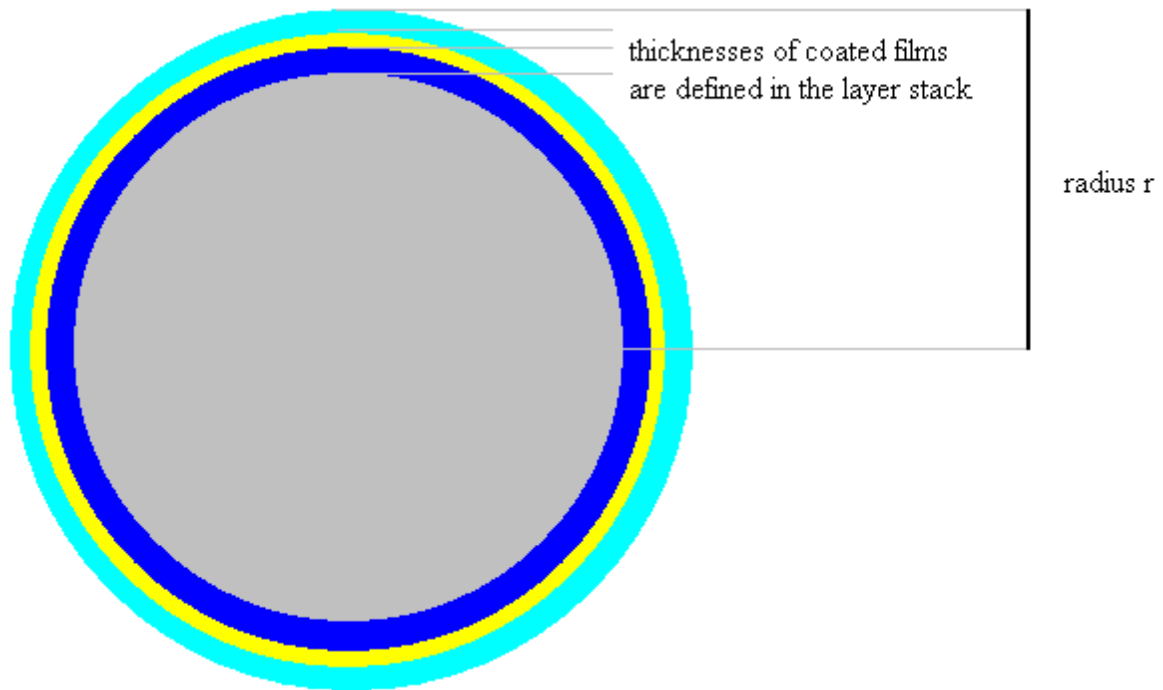
In blue you see the absorption and in red the scattering coefficient. As indicated, you can modify the **volume fraction** of this type of scatterers in this window. All settings for the Mie computation for coated spheres are done in a separate window that you can open by the **Parameters** command:



In this window you have to specify the coating, the sphere size distribution and the spectral range for the Mie computation. In addition, some files must be specified for the transfer of information to and from the external Mie program that performs the numerical work.

Radius distribution

The graph shows the radius distribution of the coated spheres. Radius means in this case the total radius including the homogeneous core and the total thickness of all layers that are coated on the sphere:



The radius distribution can be imported from data files (use the **Import** command) or from the workbook. You must specify two columns of data the first of which holds the total radius of the sphere, the second one the probability for this radius to occur in the distribution. Here is a workbook example:

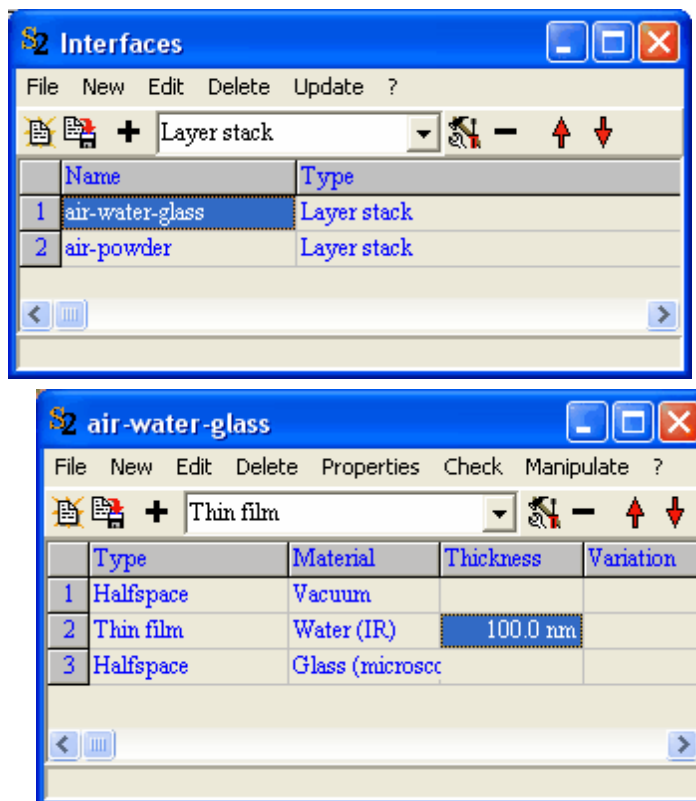
Workbook				
File Copy Paste Clear Graph Autosize cells				
	A	B	C	
1	1E-006	0.05		
2	2E-006	0.08		
3	3E-006	0.12		
4	4E-006	0.13		
5	5E-006	0.15		
6	6E-006	0.27		
7	7E-006	0.1		
8	8E-006	0.06		
9	9E-006	0.03		
10	1E-005	0.01		
11				
12				

To import these data, place the workbook cursor in the cell A1 and then use the command **Workbook|Import xy**.

The radius distribution should be normalized. A probability of 0.05 for the radius 1 micron means that in a collection of 1000 spheres there are 50 with radius 1 micron.

Coating

The coating of the spheres is defined in the list of interfaces as a layer stack. Here a 100 nm thick water film on glass spheres is defined:



The top halfspace fills the volume outside the sphere, the bottom halfspace defines the material of the sphere core. In between an arbitrary number of thin films can be used.

Once the layer stack is defined, you can drag it from the list of interfaces to the 'Mie definition window' and drop it onto the label called 'Layer stack:' in the speed button panel. Verify that the right interface has arrived.

Spectral range

In the section labeled as **Output range** you can specify the wavenumber minimum and maximum, and the number of data points. At present wavenumbers only are allowed here.

Transfer files

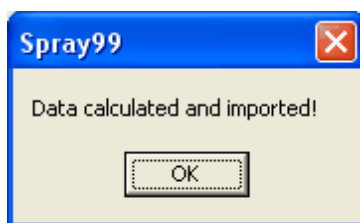
Finally some filenames for the data transfer to and from the Mie program must be entered. Please click on the buttons **Configuration file**, **Efficiency output**, **Phase functions**, and **RT output** to set these filenames. If you like you can use the same filename in each case. SPRAY will add different file extensions automatically.

Once you have to tell SPRAY where the Mie program called 'LayeredSphere.exe' is located in your system. Usually you will find the program in the directory where the SPRAY program has been installed.

When the Mie computation is started (see below), the configuration file will be passed to the Mie program in order to inform it about the requested computations. The obtained scattering and absorption efficiencies, the phase functions and the scattering and absorption characteristics ready for SPRAY in form of an RT-file are stored in the corresponding files. Especially the files containing RT data and phase functions can be quite large. It might be a good idea to check from time to time if you can delete some of these files.

Starting the Mie computation

Having done all settings you can close the Mie parameter window and go back to the previous one which will receive and display the results. In order to start the computation activate the command **Calc Mie data and wait**. The Mie program (which is a console program running in a DOS box which may flash up for a while) is started several times (once for each size class of the radius distribution). When the numerical data are ready you are informed by a short dialog:



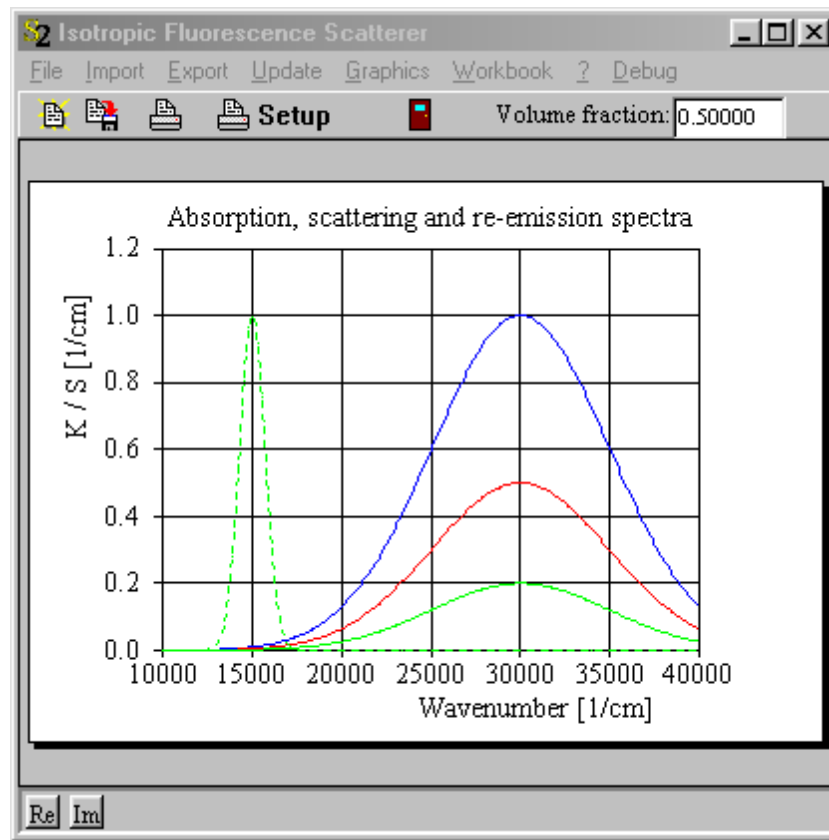
You can inspect the spectral and angular distribution of the scattered light using the View_RT utility which is described below.

The use of this kind of object for the computation of diffuse reflectance spectra is explained in SPRAY tutorial 1, example 2.

3.5 Fluorescent scatterers

Fluorescent scatterers do not only absorb or scatter radiation, but also re-emit radiation at a frequency different from the frequency of the absorbed radiation. In the present implementation the energy of the emitted rays must be lower than the one of the absorbed light. This should be a good approximation in most cases. The energy shift between absorbed and emitted radiation requires some refinement of the SPRAY algorithm to compute spectra which is shortly described below.

The properties of fluorescent scatterers are defined in the following window:



The graph shows four spectra:

- The blue curve gives the probability / distance (in cm) for absorption without further re-emission.
- The red spectrum is the probability / distance for isotropic scattering without energy shift (sometimes called 'Elastic scattering').
- The solid green line shows the probability / distance for absorption and subsequent re-emission at lower energy (fluorescence).
- The dashed green line gives the spectral composition of the re-emitted low frequency radiation.

At present, all four probability curves are imported from text files (extension *.ifd) which have the following structure:

```
10000 0.0002849304889 0.0001424652444 5.698609778E-005 1.388794386E-011
10100 0.0003091027739 0.000154551387 6.182055478E-005 3.737571328E-011
10200 0.0003351888926 0.0001675944463 6.703777852E-005 9.859505576E-011
10300 0.0003633281703 0.0001816640852 7.266563406E-005 2.54938188E-010
10400 0.0003936690407 0.0001968345203 7.873380813E-005 6.461431773E-010
10500 0.0004263695576 0.0002131847788 8.527391152E-005 1.605228055E-009
10600 0.0004615979303 0.0002307989652 9.231958606E-005 3.908938434E-009
10700 0.0004995330806 0.0002497665403 9.990661611E-005 9.330287575E-009
10800 0.0005403652241 0.000270182612 0.0001080730448 2.182957795E-008
10900 0.0005842964753 0.0002921482376 0.0001168592951 5.006218021E-008
11000 0.0006315414766 0.0003157707383 0.0001263082953 1.125351747E-007
11100 0.0006823280528 0.0003411640264 0.0001364656106 2.479596018E-007
```

.

.

The first column is the spectral position. It must be specified using wavenumbers (inverse wavelength, the unit is 1/cm). The second column is the probability / distance (in cm) for absorption without further re-emission (the blue curve above), followed by the probability for isotropic scattering without energy change (red curve) and the absorption probability with energy shift (green solid line). Finally, the last column gives the frequency distribution of the re-emitted fluorescent radiation (which is isotropic concerning the direction distribution).

The input data for fluorescent scatterer objects can conveniently be created using the **Data Factory** utility (which is delivered with SPRAY) and the built-in workbook. Once you have computed and copied all required data columns into the right sequence you can export the workbook data using the text file format 'Tabbed text'.

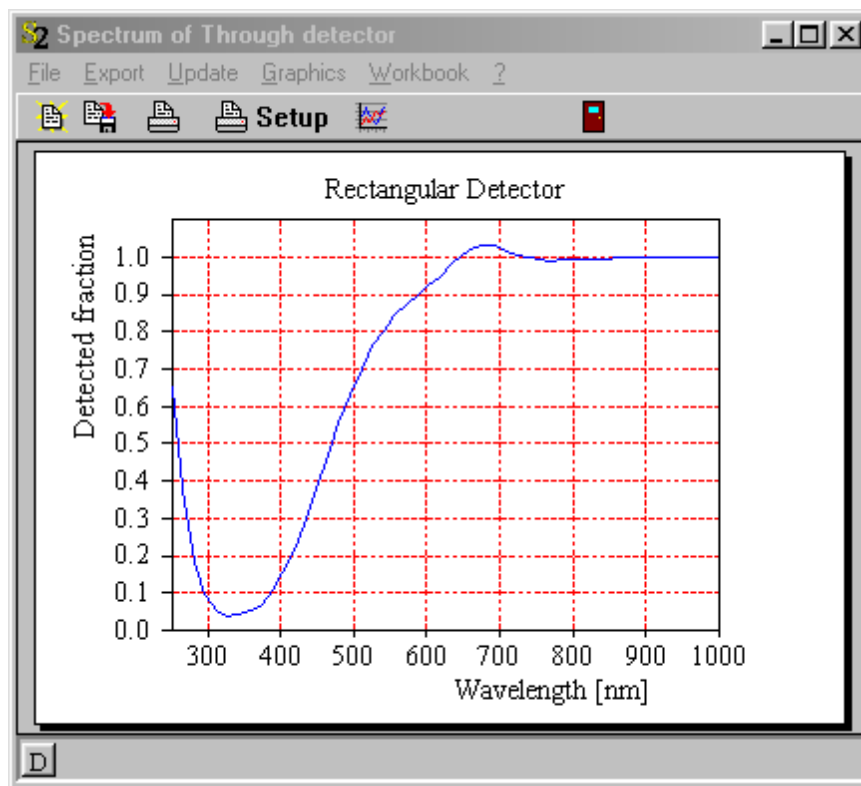
The SPRAY algorithm to do ray-tracing simulations has been extended in order to include the energy shifts caused by fluorescent scatterers. SPRAY always does the loop through all spectral points starting at the highest energy. If you use eV, 1/cm or THz, it will start with the highest value. If wavelengths (nm or microns) are selected, SPRAY will start with the lowest value.

During the simulation, SPRAY collects all rays which are absorbed by a fluorescent scatterer and for which re-emission occurs. The rays are added to a queue at the energy selected for re-emission. After all rays emitted by the light source are processed for a spectral point, SPRAY checks if there are 'fluorescent' rays in the queue to be re-started at this frequency. Of course, the rays are started where there were absorbed previously. This way the algorithm includes multiple absorption-emission cycles. However, only energy shifts towards lower energies are possible at the moment.

A simple example demonstrates the use of fluorescent scatterers. A circular light source illuminates a cylinder filled with the scatterers. The probabilities shown in the graph above are used.

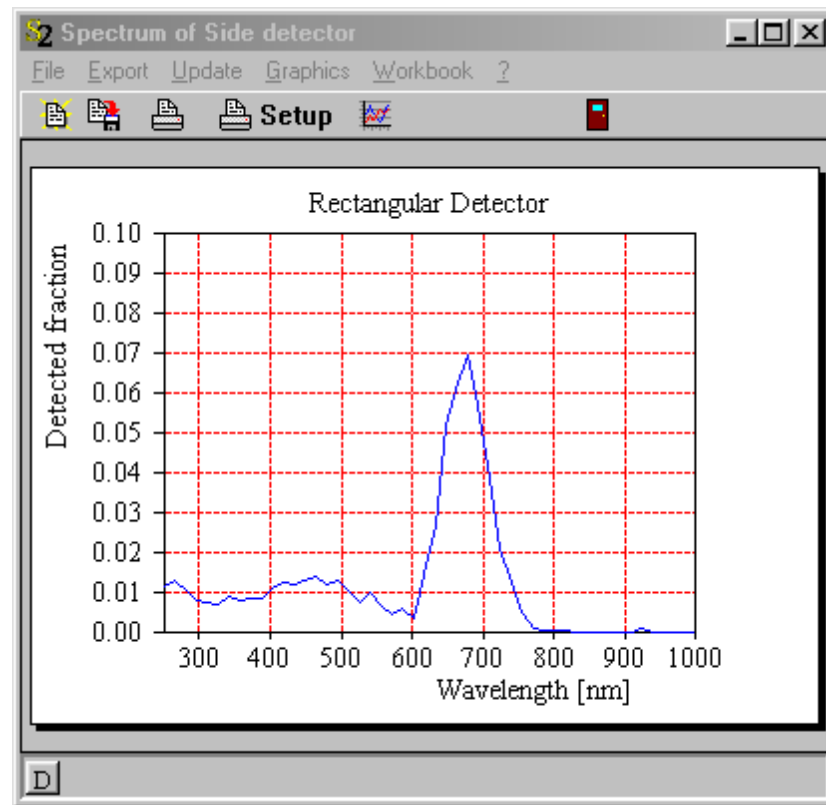
Two detectors register radiation: One collects rays moving in the direction of the radiation emitted by light source (mainly un-absorbed and un-scattered rays are found here), one 'sees' rays scattered to the side (elastically scattered radiation as well as fluorescence light).

The 'forward detector' spectrum is this:



Below 600 nm the spectrum is dominated by the large absorption/scattering losses whereas in the infrared region almost all rays move through the cylinder without modification. Around 680 nm the sum of re-emitted rays from higher energies and unscattered and elastically scattered rays is larger than the number of rays emitted by the light source in this spectral range. Hence the detector signal is larger than 1 which may occur in the case of fluorescence.

The detector recording the radiation scattered sideways has the following spectrum:



Here we see the superposition of elastically scattered light (the broad distribution below 800 nm) and the sharper peak of fluorescence light around 680 nm.

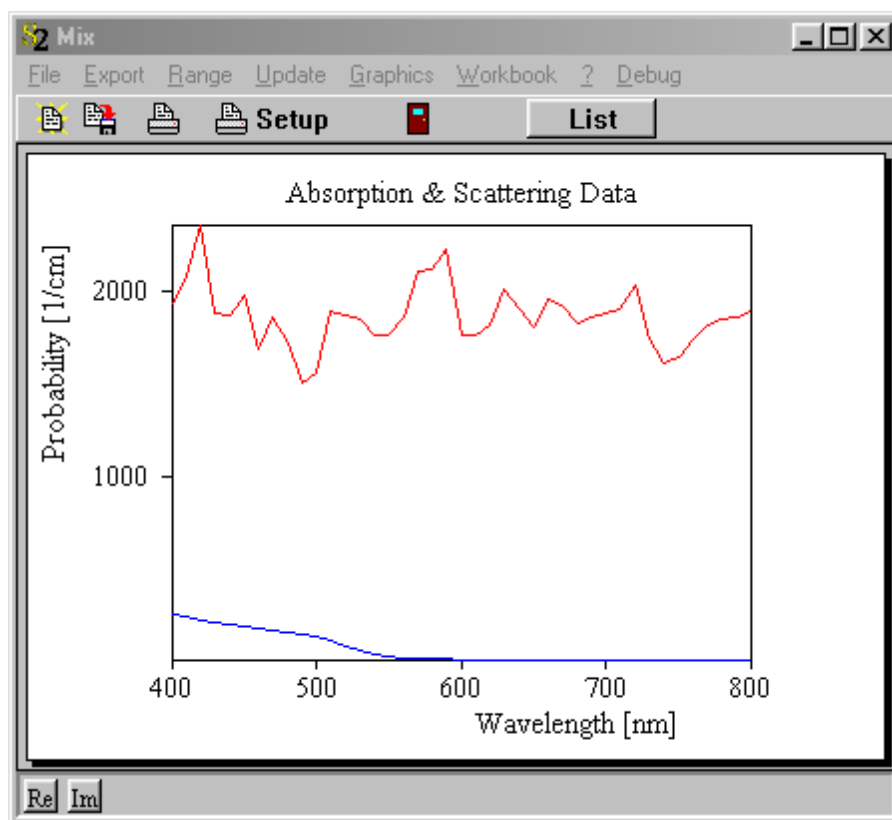
3.6 Fluorescent Mie scatterers

This object type combines extended Mie scatterers with fluorescence. An extended Mie scatterer subobject is used to compute the absorption and light scattering properties of the scatterers. The absorbed radiation is re-emitted with a probability given by the quantum efficiency subobject and a spectral distribution defined by the subobject 'Spectral distribution of emitted radiation'.

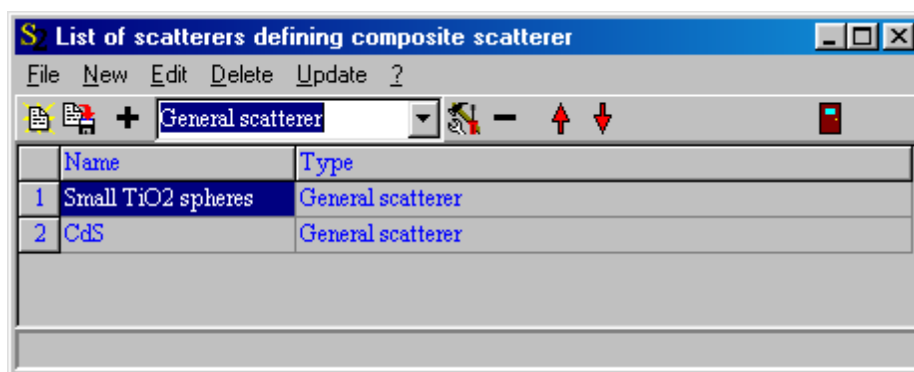
3.7 Composite scatterers

Composite scatterers are mixtures of several general scatterers. If a paint, for example, is composed of several types of inclusions in a host, you can compute the absorption and scattering behaviour of each type individually and then import these data in general scattering objects. Then you create an object of type Composite scatterer which simply adds up the individual absorption and scattering probabilities.

The main window of composite scatterers is this:



It shows the scattering and absorption probabilities K and S of the mixture. The spectral range can be set by the **Range** command. It may be different from the ranges of the individual scatterers. Composite scatterers use a list to manage the general scatterers that make up the composition. The list is opened pressing the **List** button:



First, with the button labeled '+' you create as many scattering types as you want to mix. Then you fill each entry with the wanted general scatterer by a drag&drop operation from the list of scatterers. This list is viewed by pressing the **Scatterer** button in the main window of SPRAY.

3.8 The RT file format

The RT file format used to store average single scattering characteristics is a text file format. Basically it contains one long column of data structured in a way described below. Some details of the file format may appear a little strange - well, it is a quite old format and its history will not be discussed here.

An example of an RT file is shown first:

```
0.0
180.0
  9150
    .94395262E-22    .10000000E+05    .25000000E+05    50    .10000000E+01
    .382960E-19
    .488527E-18
    .597476E-19
    .597382E-19
    .597103E-19
    .596638E-19
    .595988E-19
    .595154E-19
    .594137E-19
    .592939E-19
    .591560E-19
    .590003E-19
    .588269E-19
    .586360E-19
    .584280E-19
    .582030E-19
    .579613E-19
    .577033E-19
.
.
.
```

The individual columns have the following meaning:

0.0

Line 1: Minimum angle (should be 0 in all cases)

180.0

Line 2: Maximum angle (should be 180 in all cases)

9150

Line 3: Total number of data points: This is the product of the number of spectral points and angular points

.94395262E-22 .10000000E+05 .25000000E+05 50 .10000000E+01

Line 4: This line holds 5 numbers. The first one is the parameter t which is the average volume of the scattering particles. The unit to be used here is m^3 . The next three numbers define the spectral range (which must in wavenumbers!): Wavenumber minimum, wavenumber maximum and number of data points. The last number is the refractive index of the host material surrounding the scattering particle. Dividing the total number of data points (9150) by the number of data points of the spectral range (50) one gets 183. This means that there are 183 numbers for each wavenumber in the file: The cross section for absorption (in m^2), the cross section for scattering (in m^2 as well) and 181 data points covering the

angle range from 0 to 180 degrees in 1 degree steps.

Starting with the minimum wavenumber, these 183 data points will now follow. Then the 183 numbers for the next wavenumber point are stored in the file, and so on until all wavenumber points are processed.

.382960E-19

Line 5: Absorption cross section for 10000 1/cm

.488527E-18

Line 6: Scattering cross section for 10000 1/cm

.597476E-19

Line 7: Scattering probability W(0°)

.597382E-19

Line 8: Scattering probability W(1)

.597103E-19

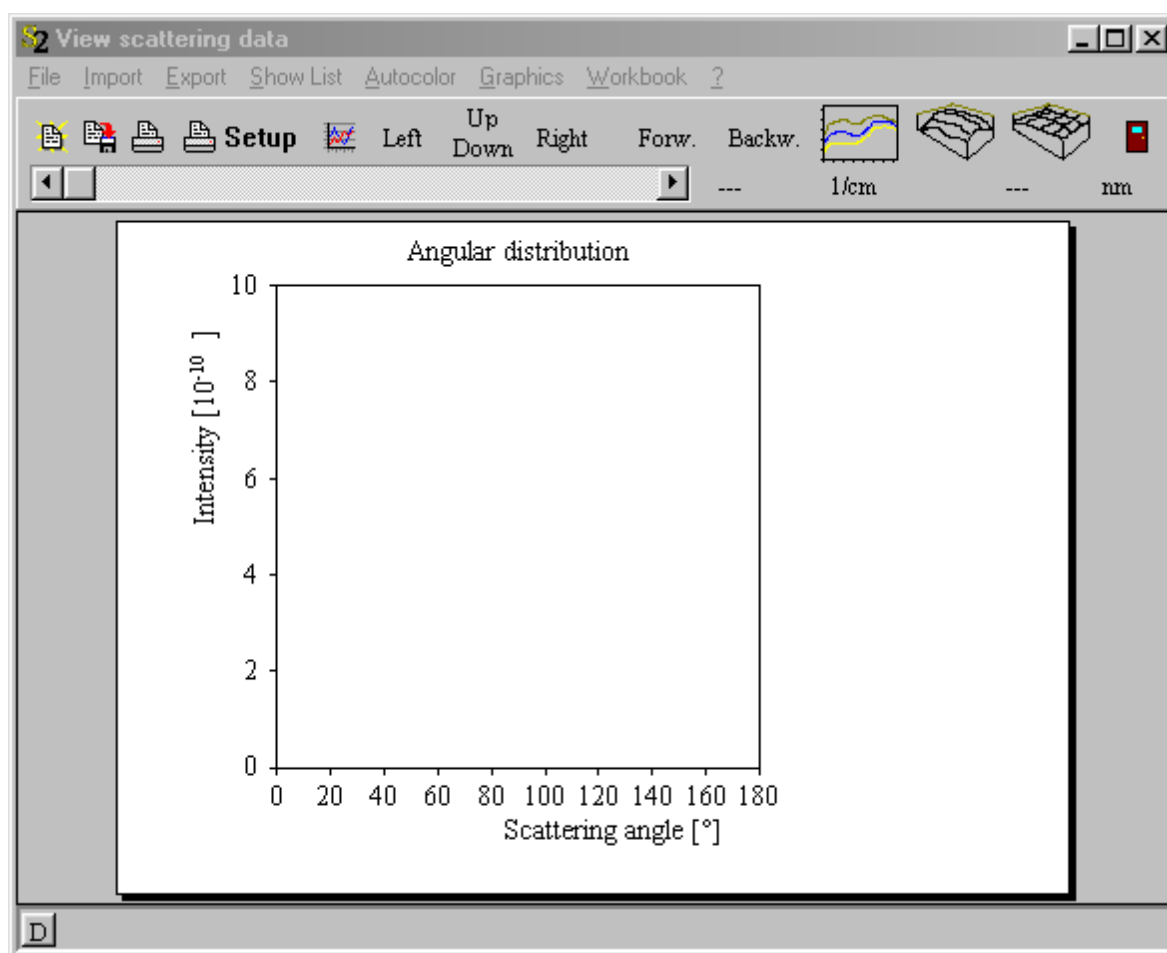
Line 9: Scattering probability W(2)

.
.
.

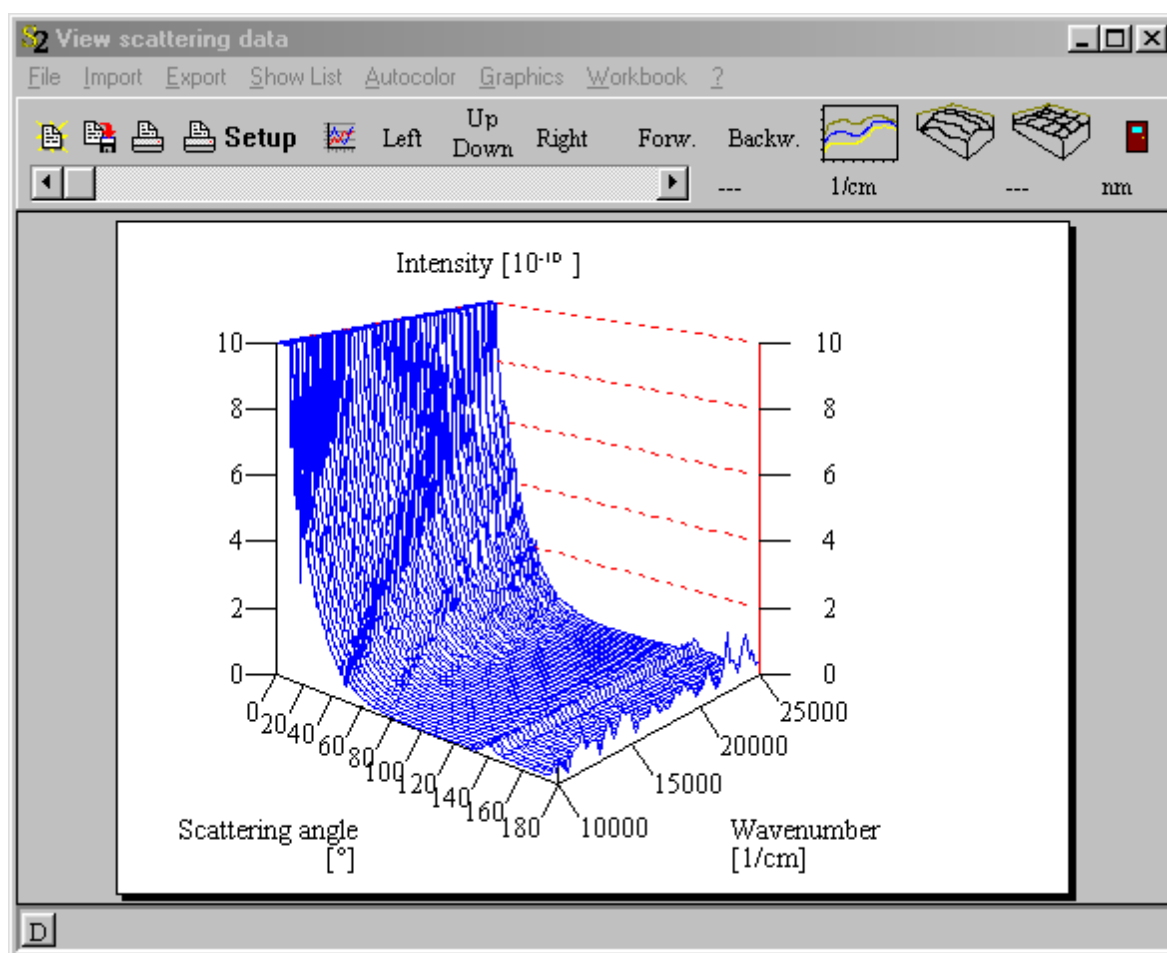
You can inspect the data stored in RT files using the View_RT utility program that is delivered with SPRAY.

3.9 The View_RT utility

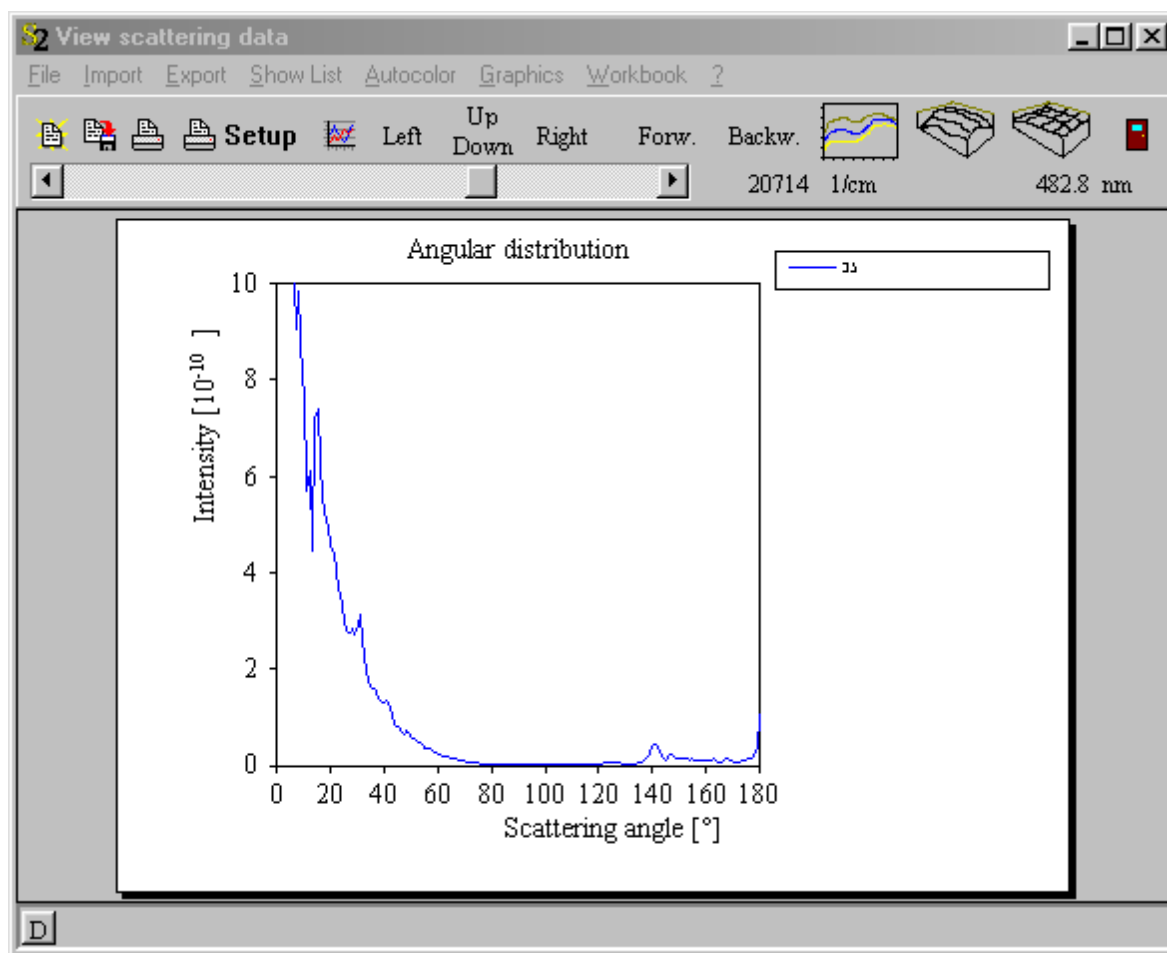
The angular and spectral dependence of the scattering probabilities can be visualized using the View_RT program which is a small utility delivered with SPRAY:



Use the **Import** command and select the RT data that you want to inspect. Here is an example:



The View_RT program is very much like the Collect program. You can generate 3D and 2D views. In 2D mode you can use the slider to move through the wavenumber dependence of the scattering data. Here is a screenshot:



The numbers right to the wavenumber slider tell you where you are on the wavenumber and nanometer scale.

4 Interfaces

4.1 Overview

Most of SPRAY's geometrical objects are so-called 'Interface objects' which means that their surface may be covered with a user-defined interface. The interface determines what happens to rays that hit the surface. You can select from some simple pre-defined interfaces and from (in general more complicated, but also more useful) user-defined interfaces.

These interfaces are implemented at present:

- Perfect absorber (pre-defined)
- Perfect mirror (pre-defined)
- Ideal diffusor (user-defined)
- Arbitrary layer stack (user-defined)

4.2 Pre-defined interfaces

The following interfaces are pre-defined and work without any further parameter input:

Perfect absorber:

Any ray that hits a surface covered with a perfect absorber is absorbed, independent of angle of incidence, polarization or frequency.

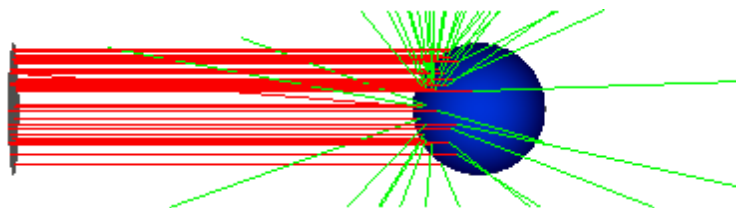
The following example shows a sphere coated with a perfect absorber. It is illuminated by a parallel beam:



Perfect mirror:

Any ray hitting a surface covered with a perfect mirror is reflected with a probability of 1, independent of angle of incidence, polarization or frequency.

A sphere coated with a perfect mirror behaves like this:



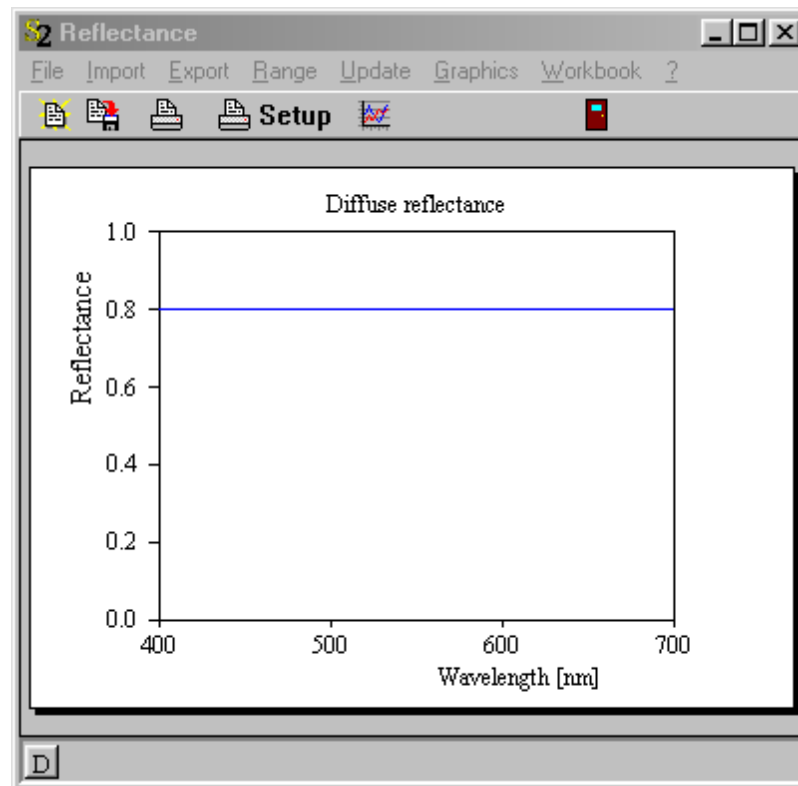
4.3 Ideal diffusor

Interfaces of type 'Diffuse' are used to describe diffusely reflecting and transmitting surfaces in a simple, phenomenological way. Diffuse reflection of light is caused by inhomogeneities like edges, scratches or small particles. In many cases the multiple scattering of light by the inhomogeneities leads to a so-called Lambertian scattering characteristics: The intensity of the radiation has a cosine-like distribution with the scattering angle (where 0 deg is the surface normal).

If the light diffusion takes place on a scale much smaller than your geometric objects you can skip a SPRAY simulation of the microscopic details and use as a shortcut interfaces of type 'Diffuse': These produce Lambertian characteristics with user-defined wavelength-dependent reflectance and transmittance coefficients. The user-dialog looks like this:



The buttons labeled 'Reflectance' and 'Transmittance' open subwindows which are used to define the spectral dependence of the reflectance (or transmittance) coefficient:

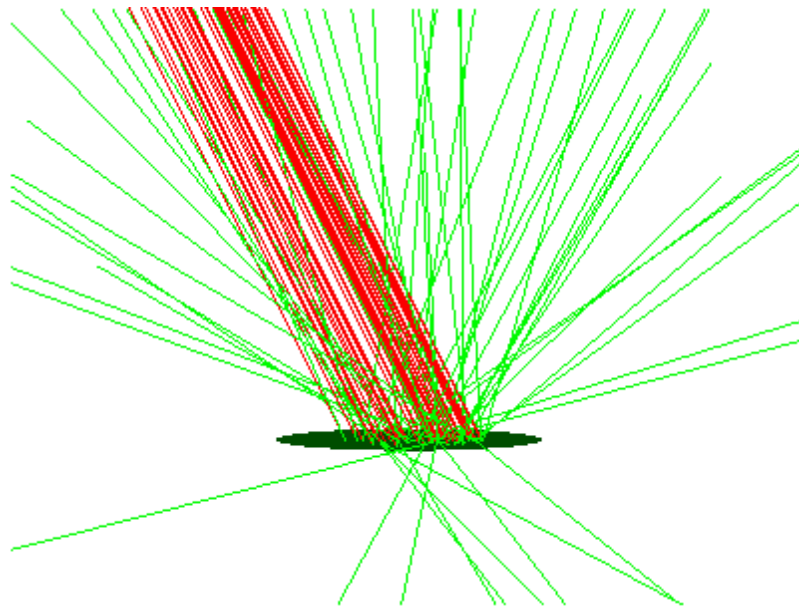


You can generate the reflectance data in various ways:

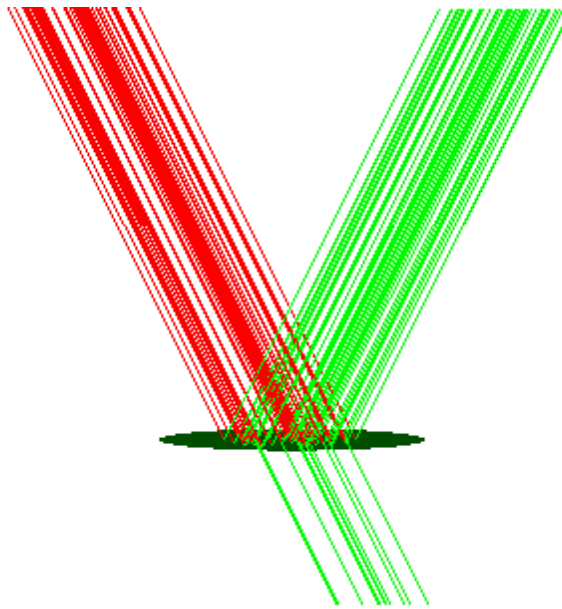
- Create the diffuse reflectance spectrum by the *Data Factory* (a utility that comes with SPRAY) and drag them into the reflectance window (see the SCOUT technical manual for Drag&Drop operations)
- Create data with Excel or similar spreadsheet programs and import them into SPRAY via the workbook (see the SCOUT technical manual for a description of workbook features and operation).
- Type the data directly into the workbook and import them from there.
- Digitize literature data with our *Digit* program (included in your SPRAY package) and drag them into SPRAY.

Of course, the transmittance data are created in the same way as the reflectance data. Make sure that the sum of the reflectance and transmittance coefficients does not exceed 1. If the sum is smaller than 1 part of the light is absorbed at the interface.

For a constant reflectance of 0.8 and a transmittance of 0.2 the situation looks like this (illumination with a parallel beam from the upper left corner):



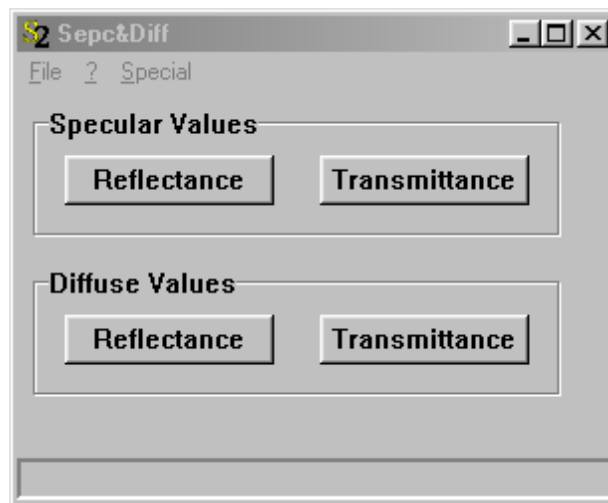
Using the **Properties** command you can switch from diffuse to specular reflectance and transmittance. In this case the radiation is reflected according to the law of reflection of perfectly smooth interfaces:



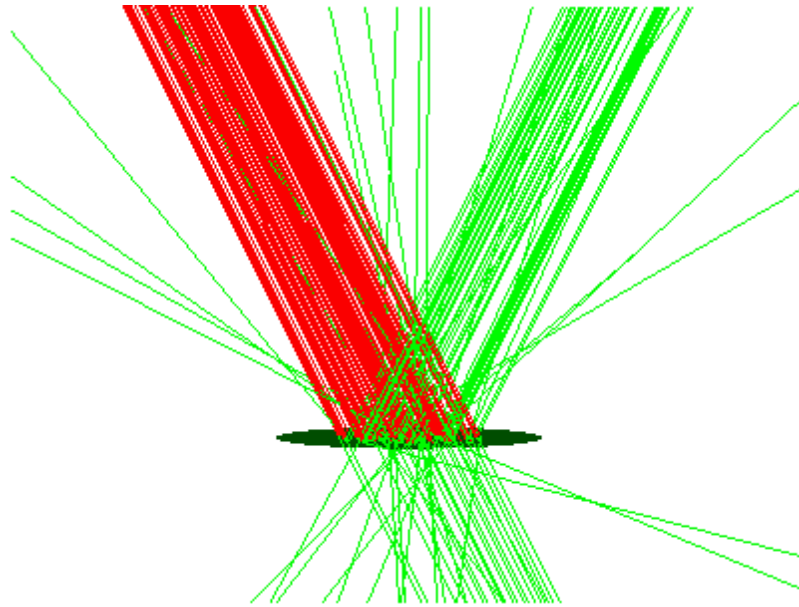
In the **Properties** submenu you can also set the scatterers present on either side of the interface (see the explanation above).

4.4 Specular and diffuse reflection

Interfaces of type 'Specular and diffuse' are used to combine specular and diffuse (Lambertian) reflectance (and transmittance). The dialog has four buttons to open the corresponding windows:

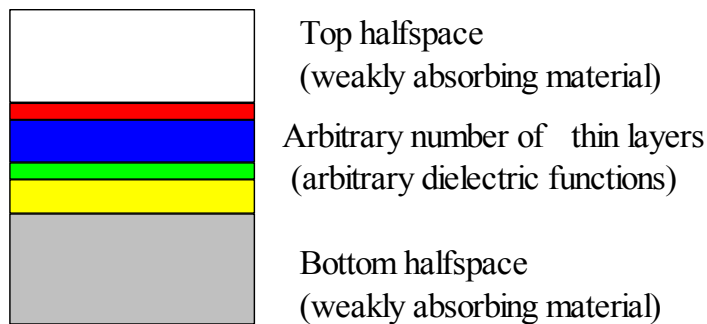


For a specular reflectance of 0.3, specular transmittance of 0.2, diffuse reflectance of 0.2 and diffuse transmittance of 0.2 (i.e. the probability for absorption at the interface is 0.1) the interface behaves like this:

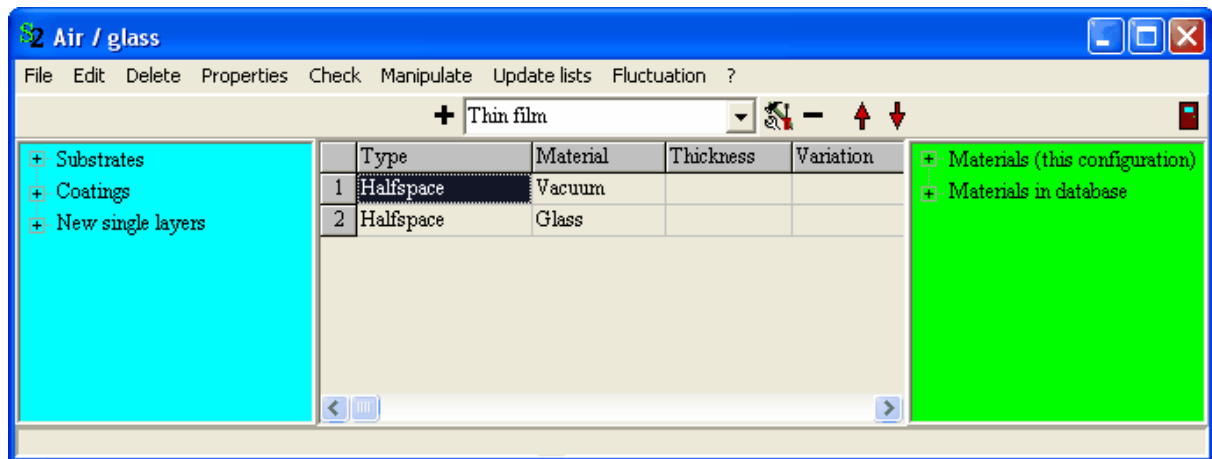


4.5 Layer stacks

The surface of interface objects can be covered with arbitrary layer stacks. Each layer is 'filled' with optical constants taken from the list of dielectric functions. The top halfspace and the bottom halfspace should have optical constants with small absorption coefficients:



The window which lets you define the layer stack and some other properties (scatterers, surface tilt angle distribution) looks like this:



The definition of layer stacks in SPRAY is exactly the same as in SCOUT. Please consult the SCOUT technical manual for details.

How it works

If a ray hits an interface with a layer stack SPRAY must decide if the ray gets reflected, transmitted or absorbed. This is done the following way:

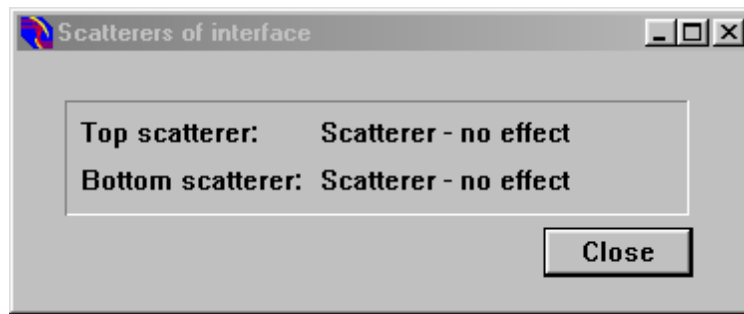
Before the ray-tracing for a new spectral point is started the angle dependence of the reflectance and transmittance of the layer stack is computed for both s- and p-polarization. The angle resolution for these calculations is set in the Parameters section.

During the ray-tracing procedure, if a ray hits the surface of an object that is covered with the layer stack, the projection of the ray's polarization onto the direction of s-polarization and p-polarization is determined. With probabilities proportional to these projections the polarization is changed to either pure s- or p-polarization. For the present angle of incidence and polarization, the reflectance and transmittance coefficient is looked up in the previously computed tables. Linear interpolation is used between angles for which the data have been calculated. Based on the values for reflectance and transmittance, it is decided if the ray is reflected, transmitted or absorbed. In the case of reflection the new direction is computed according to the law of reflection. The new direction of a transmitted ray is set applying the law of refraction, taking into account the real part of the index of refraction of top and bottom halfspace.

Layer stacks are considered to be infinitely thin with respect to the geometrical dimensions of the objects in the ray-tracing scenery. To be consistent with this assumption, the total thickness of the layer stacks used in a SPRAY simulation must be small enough.

Scatterer assignment

All user-defined interfaces can be used as separation between continuum scatterers. Usually the **Properties** submenu contains a command called **Scatterer assignment** which opens the following dialog:



From the list of scatterers you can drag the appropriate scatterers to the top and the bottom halfspace, respectively. The two halfspaces are separated by the interface, of course. The default selection for the scatterers is 'No effect' which means there are no scatterers on both sides of the interface.

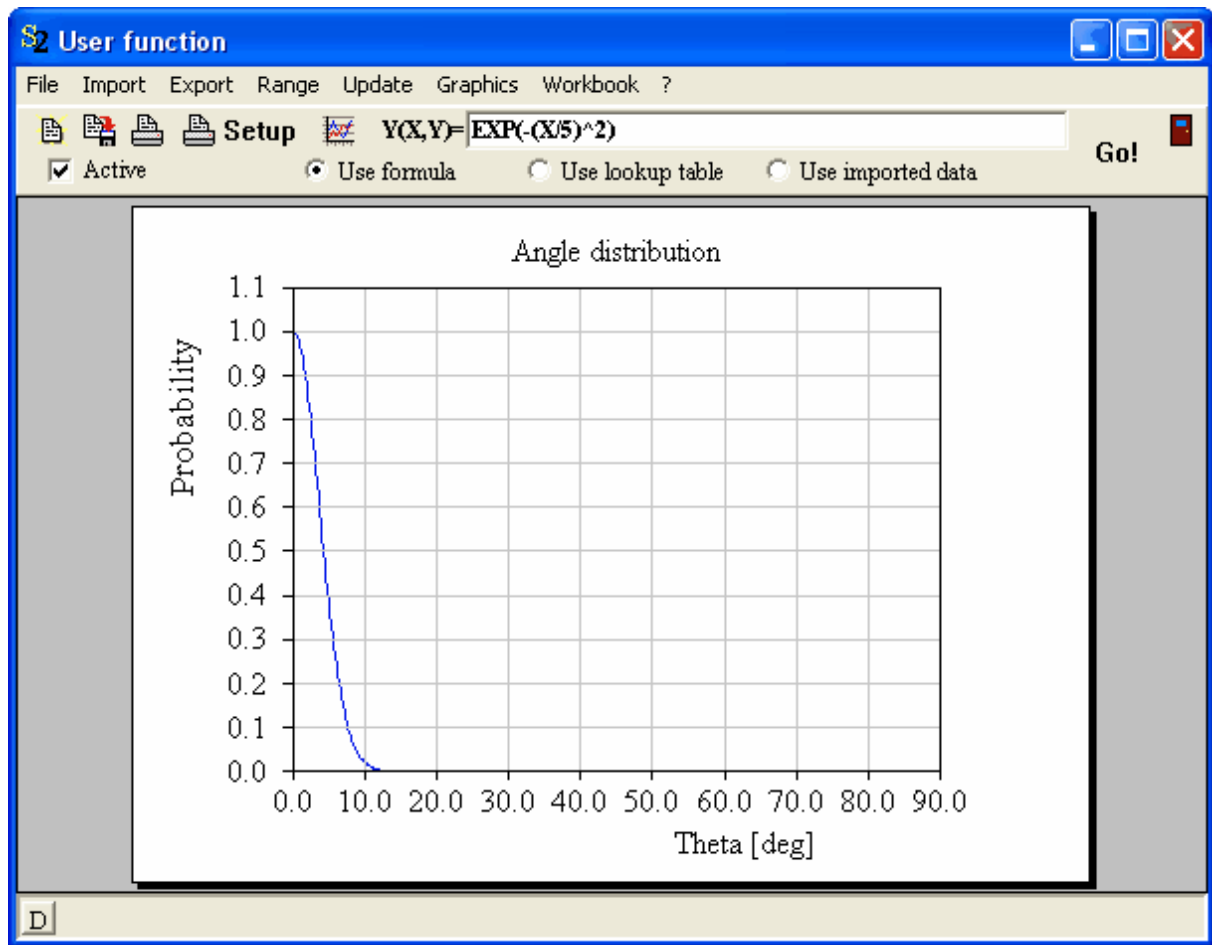
The picture below shows an example: The upper interface (viewed from the side) switches from a non-scattering medium (top) to a light scattering continuum (bottom):



Introducing surface roughness: Tilt of surface normal

You can introduce surface roughness to the SPRAY model in a simple, statistical way. Instead of explicitly defining a certain surface shape (which is possible with user-defined surface shape objects) you can tilt the surface normal according to a statistical distribution whenever a ray hits the surface. The point where the hit occurs is still determined by the geometric object but the surface normal tilt angle is taken statistically.

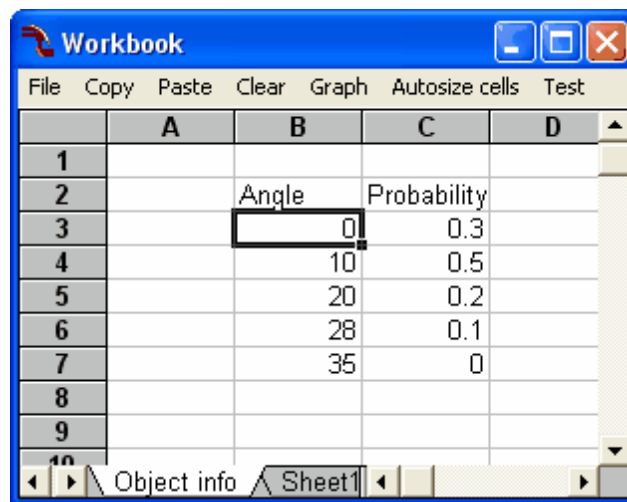
Using the menu command **Properties|Distribution of surface normal tilt angles** you can open a subwindow which is used to define the tilt angle distribution:



Please note that this window blocks SPRAY until it is closed again: You cannot access other parts of the program as long as this window is open.

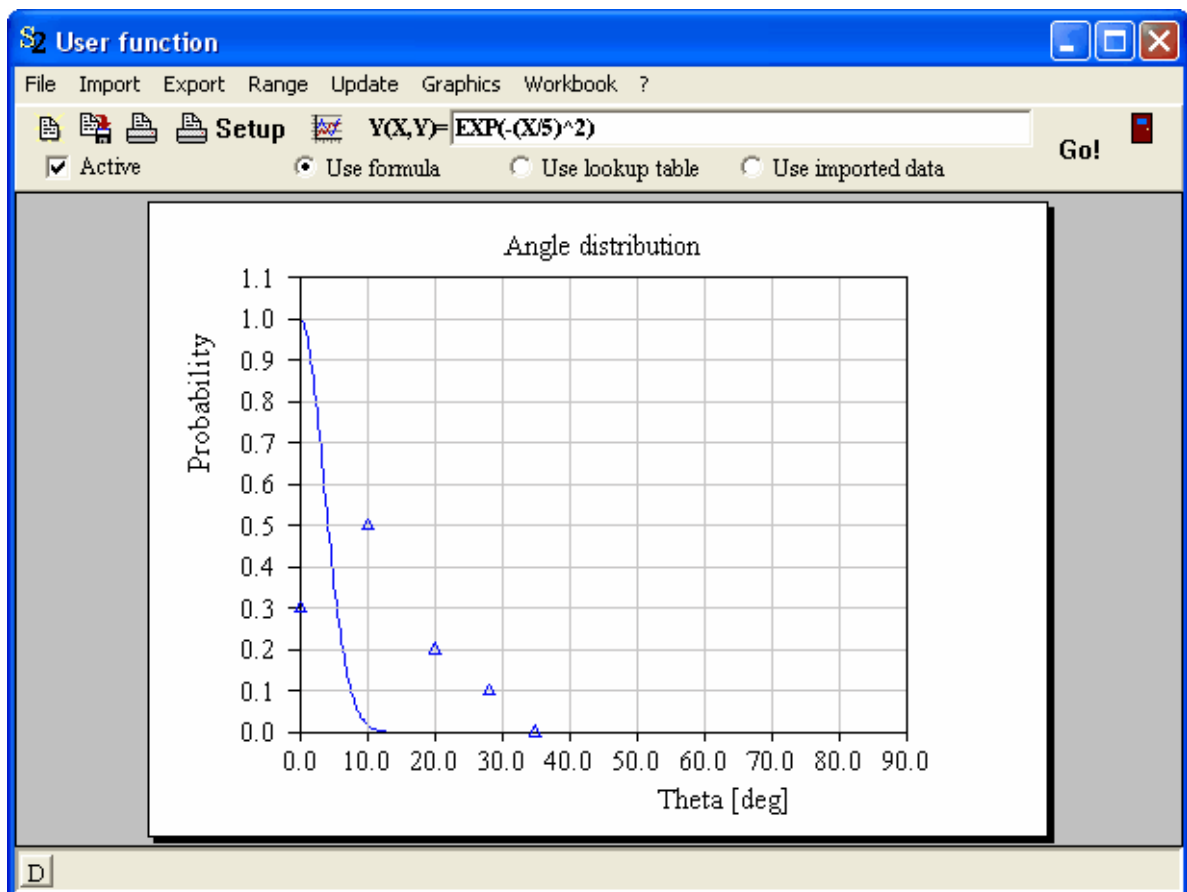
The tilt angle distribution is used only if the checkbox 'Active' is checked. It is ignored otherwise. Depending on the settings of the checkboxes 'Use formula', 'Use lookup table' and 'Use imported data' the angle distribution is computed from a

- user-defined formula: Type in a formula in the edit box to the right of 'Y(X,Y) = ' in the top of the window. In the formula, use the term 'X' to refer to the angle. Click on the menu command **Range** in order to set the angle range where the formula is going to be evaluated. This can be 0 ... 90 degree or a subrange. Use a large number of points in the range dialog, e.g. 500 or 1000, in order to display the user-defined curve. Finally press **Go** or **Update** in order to compute the distribution.
- a lookup table: The distribution is computed using a lookup table. The lookup table is imported from the workbook using the command **Workbook|Import lookup table from data columns** or **Workbook|Import lookup table from data rows**. Here is an example of a workbook page with a lookup table:



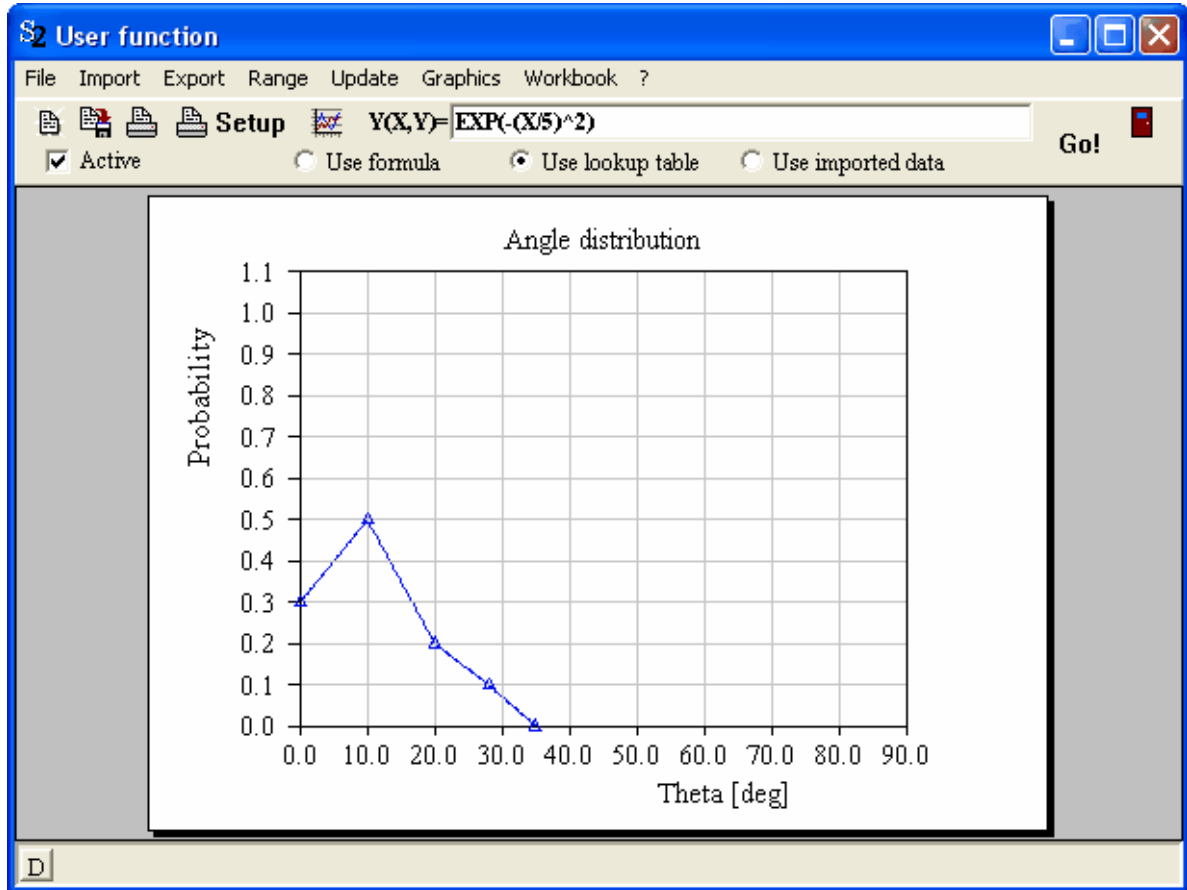
	A	B	C	D
1				
2		Angle	Probability	
3		0	0.3	
4		10	0.5	
5		20	0.2	
6		28	0.1	
7		35	0	
8				
9				
10				

Place the cursor into cell B3 and use the command **Workbook|Import lookup table from data columns** in order to read the table values. Afterwards the points of the lookup table are displayed in the graph:



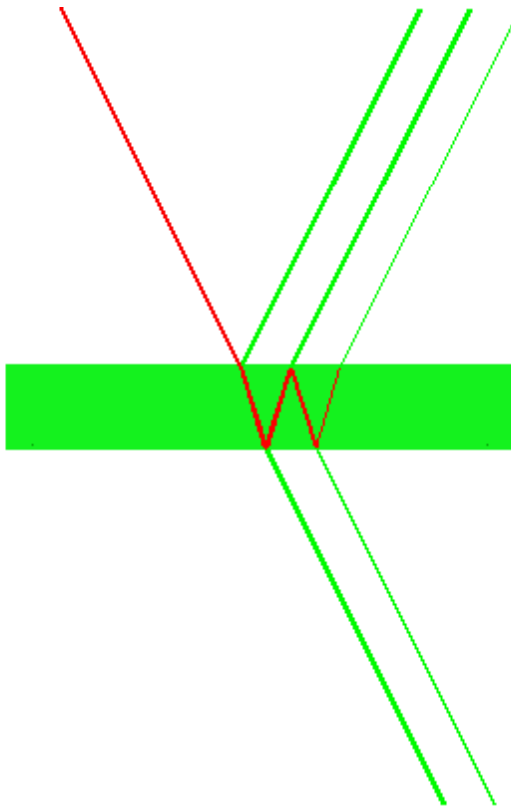
Check the option **Use lookup table** and press **Go**. Now the distribution is computed using linear interpolation in between the points of the lookup table and constant extrapolation outside the range

of the lookup table:

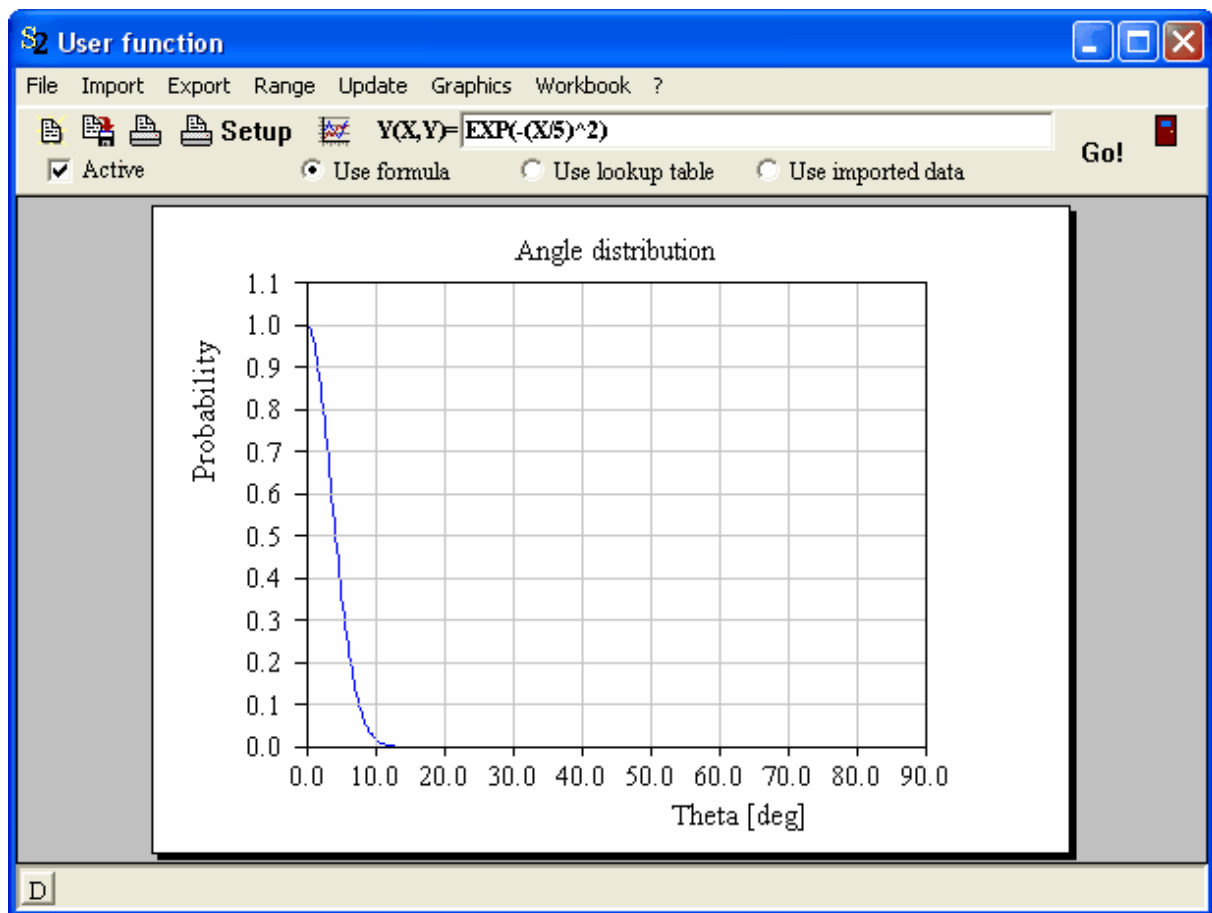


- a set of imported data points: Use the **Import** menu command to import the complete angle distribution from a data file. Alternatively, you can import the data using the command **Workbook|Import xy** (this will read data columns from the workbook, starting at the cursor position). Please read the section 'Technical notes|Import formats' in the SCOUT technical manual for more information on file import data formats.

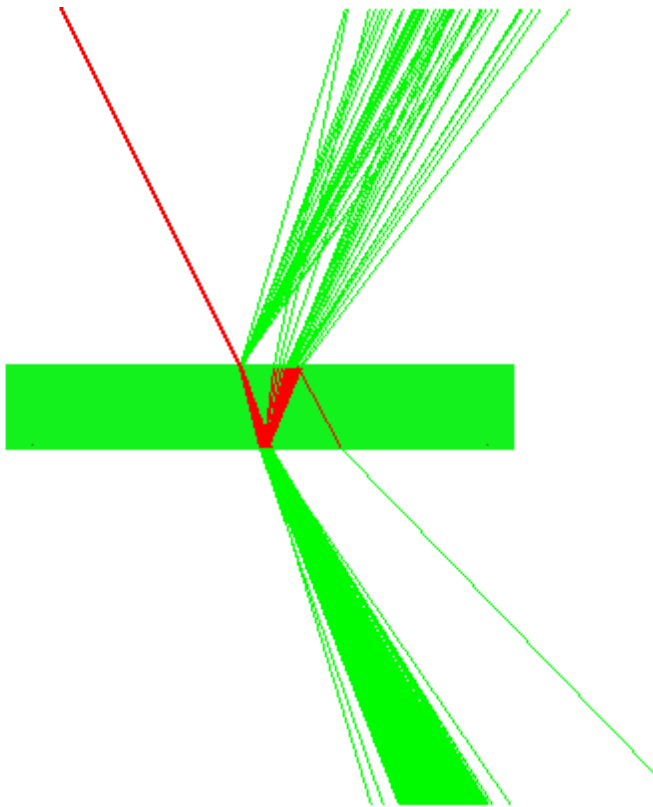
Here is a simple example demonstrating the introduction of surface roughness using this mechanism of surface normal tilting: A light source illuminates a glass plate from the upper left. Without surface roughness the ray-tracing looks like this:



Using the formula $\exp(-(x/5)^2)$ the following distribution is created:



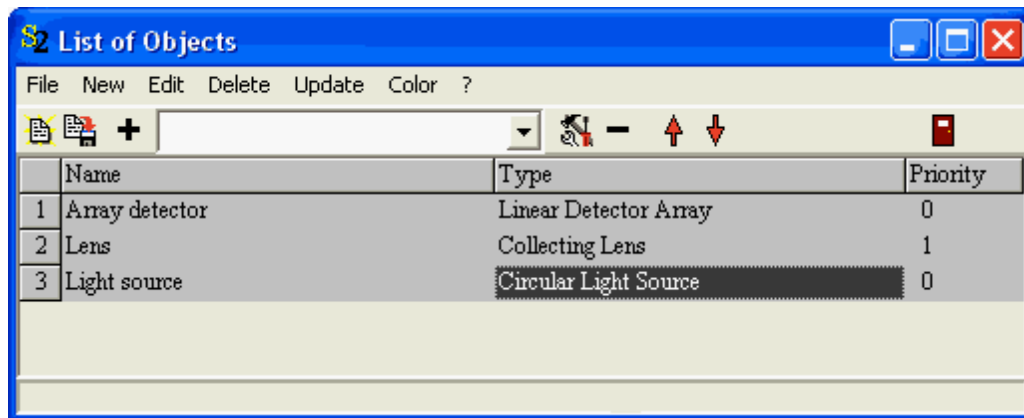
Activating the surface tilt by checking the **Active** checkbox the ray-tracing changes to this:



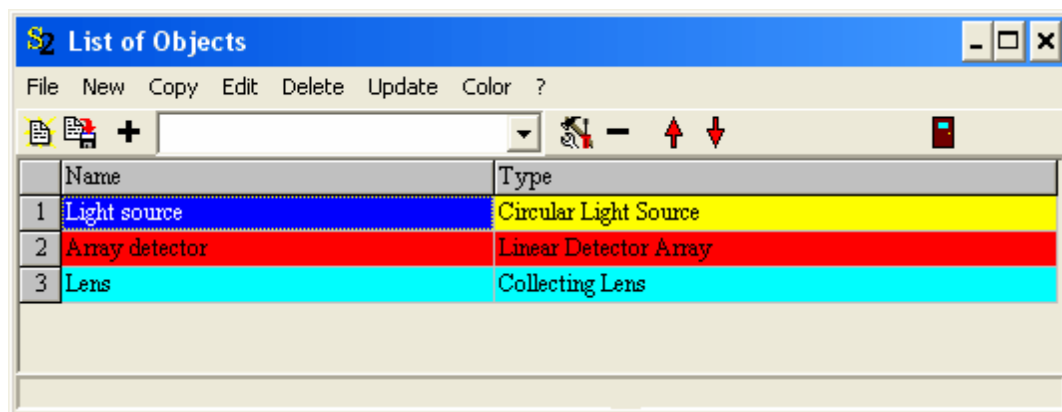
5 Geometric objects

5.1 Overview

The geometric objects of SPRAY are managed by the 'List of objects'. The list is opened with the button 'Objects' in the main window. Here is an example:

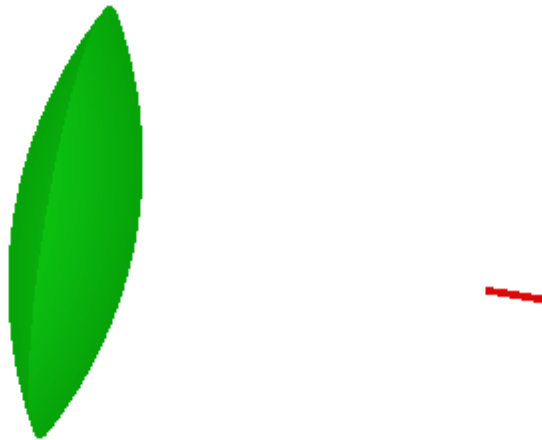


In the list, you can overwrite the name of the objects and set their priority. If, during the ray-tracing, two objects report the same distance to the starting point of the current ray, the one with the highest priority will get the order to handle this ray (see the description of the SPRAY ray-tracing algorithm). Like in almost all lists, a double click on a row lets you select a color for the corresponding object:



Assigning colors to objects can make long lists easier readable. In addition, you can use the assigned colors of the objects for camera views. The global SPRAY option 'Use object colors for camera views' (Use the command **File|Options|Use object colors for camera views** in order to check or uncheck this option) is used to control the color assignment of objects in camera views: If the option is unchecked, default colors are used. Otherwise the user-defined color assignment is taken to display the objects in a view.

For the example configuration given above, a view with default colors looks like this:



Setting the option 'Use object colors for camera views' one gets



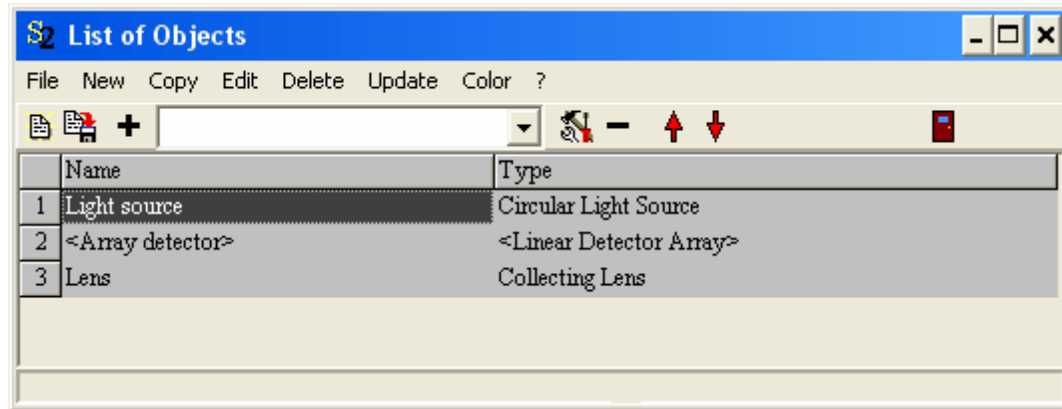
The available objects are classified as follows:

- Light sources (these objects emit rays)
- Detectors (these objects detect rays)
- Interface objects (these objects reflect, absorb or transmit rays. Depending on the assigned type of interface the reflection or transmission is specular or diffuse)
- Special objects (here objects are collected that do not belong to one of other the categories)

Activation / De-activation of objects

If you want to exclude an object from SPRAY simulations temporarily you can de-activate it by **clicking with the right mouse button** in the 2nd column of the object's row (the type column) **while the shift key is pressed** on the keyboard. To indicate the de-activation the object's name is

displayed in brackets like the 'Array detector' in the following example:



To turn an object back into the activated state right-click again with the shift key pressed.

Copy: Duplication of objects

Any object (except the light source) in the list of objects can be duplicated by the **Copy** command. Since a temporary file is used during the copy action, this command only works if you have the right to write to the local drive c: on your PC.

5.2 Light sources

5.2.1 Overview

Light sources emit rays. Some of the SPRAY light sources are transparent, some are absorbing. Every light source must be embedded in a material which is to be taken from the list of dielectric functions. If the light source is embedded in a light scattering material you have to set the corresponding scattering object.

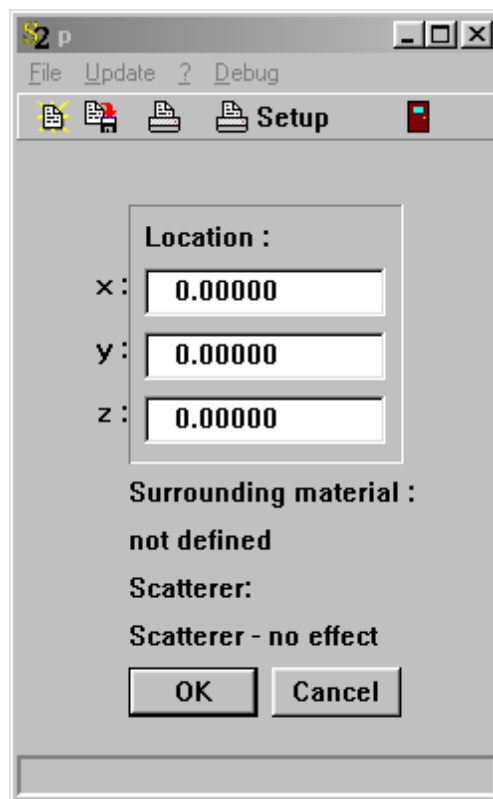
At present there must be **exactly one light source** in a SPRAY setup.

The following types of light sources are available:

- Point light source
- Rectangular light source
- Circular light source
- Volume light source

5.2.2 Point light source

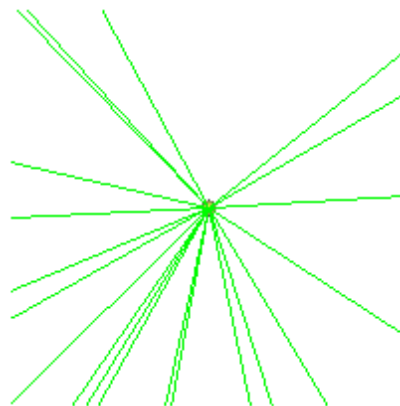
This is the simplest type of light sources emitting rays from a single point in space with random directions. The only geometric quantity to be specified is the position of the point light source. The dialog to do so is the following:



The rays emitted by the light source move in the 'surrounding material'. Open the list of dielectric functions and drag an entry from there over the text label 'not defined'. Drop it there and make sure that the label now displays the wanted material (vacuum in most cases).

If the light source is embedded in a scattering material, drop an item from the list of scatterers to the text label 'Scatterer - no effect'.

In rendered views point light sources appear as small spheres:



Access by OLE automation

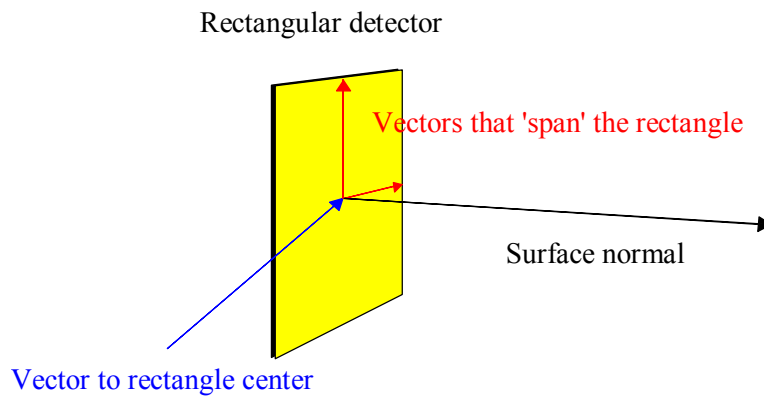
OLE automation controllers can modify a point light source named 'MyName' in the SPRAY object list by the following OLE commands:

object_parameter("MyName", "x"): read/write the x-coordinate of the position
 object_parameter("MyName", "y"): read/write the y-coordinate of the position
 object_parameter("MyName", "z"): read/write the z-coordinate of the position

5.2.3 Rectangular light source

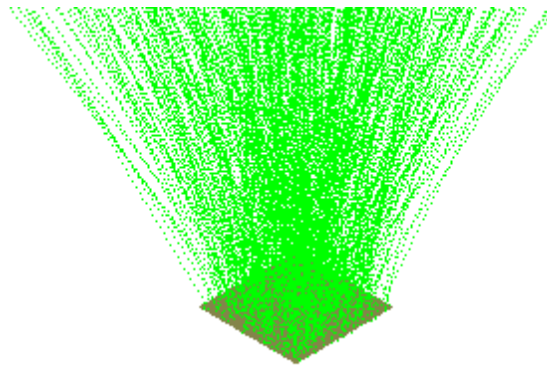
Rectangular light sources have a rectangular shape and emit rays on one side of the rectangle into a cone of directions. The cone angle can be specified by the user. To define the orientation of the rectangle and to set the cone angle the following dialog is used:

The vector called 'Location' defines the center of the rectangle (the blue arrow in the sketch below). The two other vectors should be perpendicular with respect to each other. They span the rectangle as indicated below by the two red vectors. The cross product of vector 1 and vector 2 gives the surface normal. The cone angle of the emitted radiation is measured from the surface normal. If the option 'Absorbing' is checked the rectangular light source absorbs any ray that hits the rectangle. Otherwise, the rectangle is transparent to radiation.



Emitted rays are started at a position chosen randomly on the rectangle. Their initial directions are selected randomly as well within the user-defined cone. The surrounding **Material** and **Scatterer** are set the same way as for point light sources.

The following graph shows the appearance of a rectangular light source in a rendered view, emitting some test rays:



Access by OLE automation

OLE automation controllers can modify a rectangular light source named 'MyName' in the SPRAY object list by the following OLE commands:

object_parameter("MyName", "x"): read/write the x-coordinate of the position
 object_parameter("MyName", "y"): read/write the y-coordinate of the position
 object_parameter("MyName", "z"): read/write the z-coordinate of the position

object_parameter("MyName", "x1"): read/write the x-coordinate of vector 1
 object_parameter("MyName", "y1"): read/write the y-coordinate of vector 1
 object_parameter("MyName", "z1"): read/write the z-coordinate of vector 1

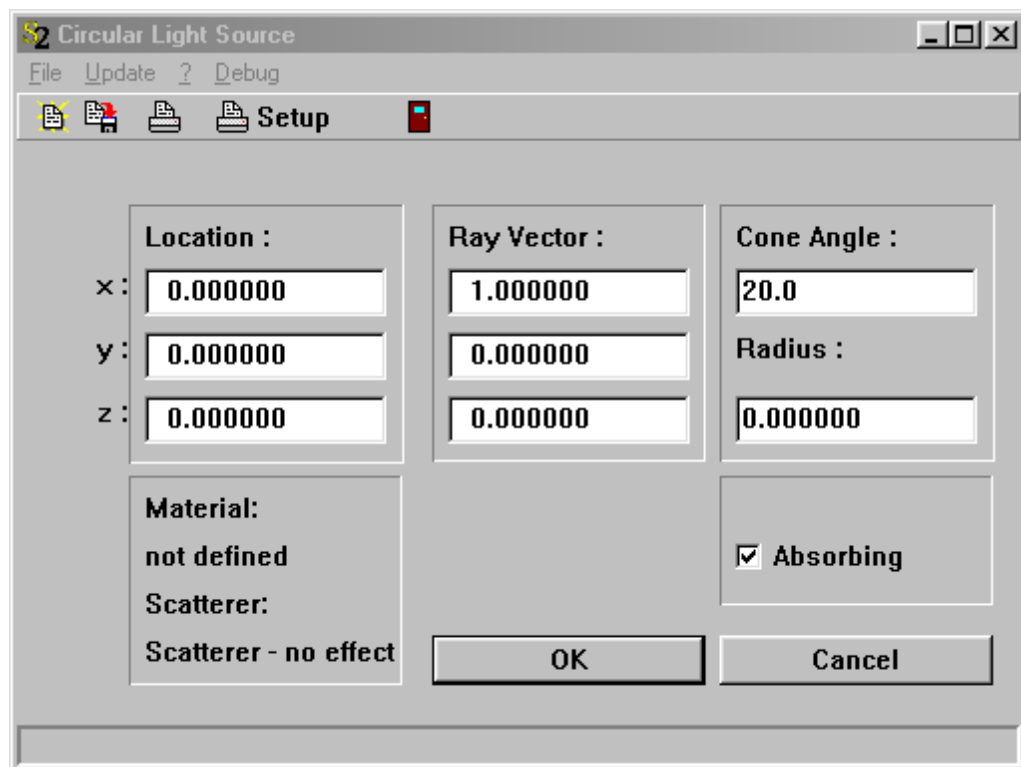
object_parameter("MyName", "x2"): read/write the x-coordinate of vector 2

object_parameter("MyName", "y2"): read/write the y-coordinate of vector 2

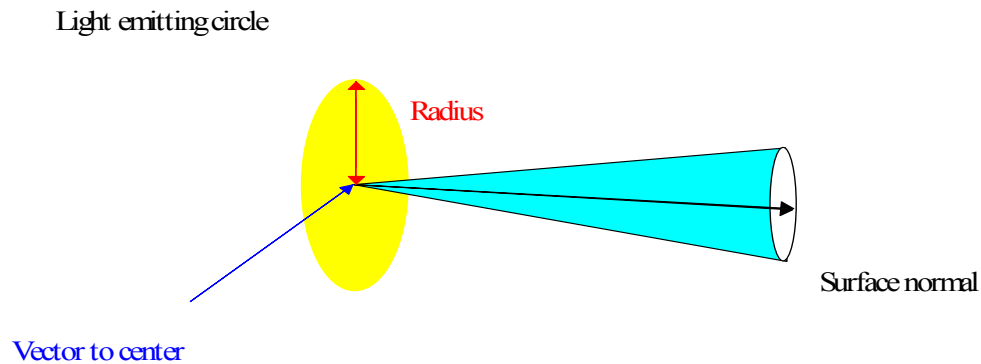
object_parameter("MyName", "z2"): read/write the z-coordinate of vector 2

5.2.4 Circular light source

Circular light sources have a circular shape and emit rays into a cone. The cone angle can be specified by the user. To define the orientation of the circle and to set the cone angle the following dialog is used:

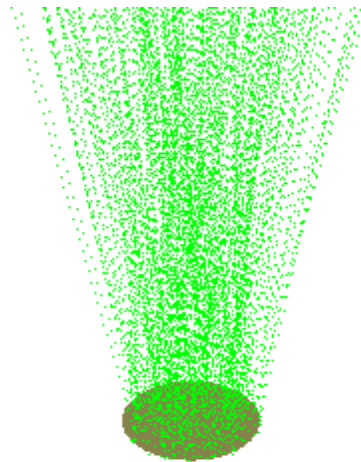


The center of the circle is defined by the 'Location' vector (blue in the sketch below). The surface normal is given by the vector called 'Ray vector' which is the black arrow. This is also the center of the emission cone. The 'Radius' parameter is the radius of the circle.



If the option 'Absorbing' is checked the circular light source absorbs any ray that hits the circle. Otherwise, the rectangle is transparent to radiation.

Appearance of a circular light source in rendered views:



Access by OLE automation

OLE automation controllers can modify a circular light source named 'MyName' in the SPRAY object list by the following OLE commands:

`object_parameter("MyName", "x")`: read/write the x-coordinate of the position

`object_parameter("MyName", "y")`: read/write the y-coordinate of the position

`object_parameter("MyName", "z")`: read/write the z-coordinate of the position

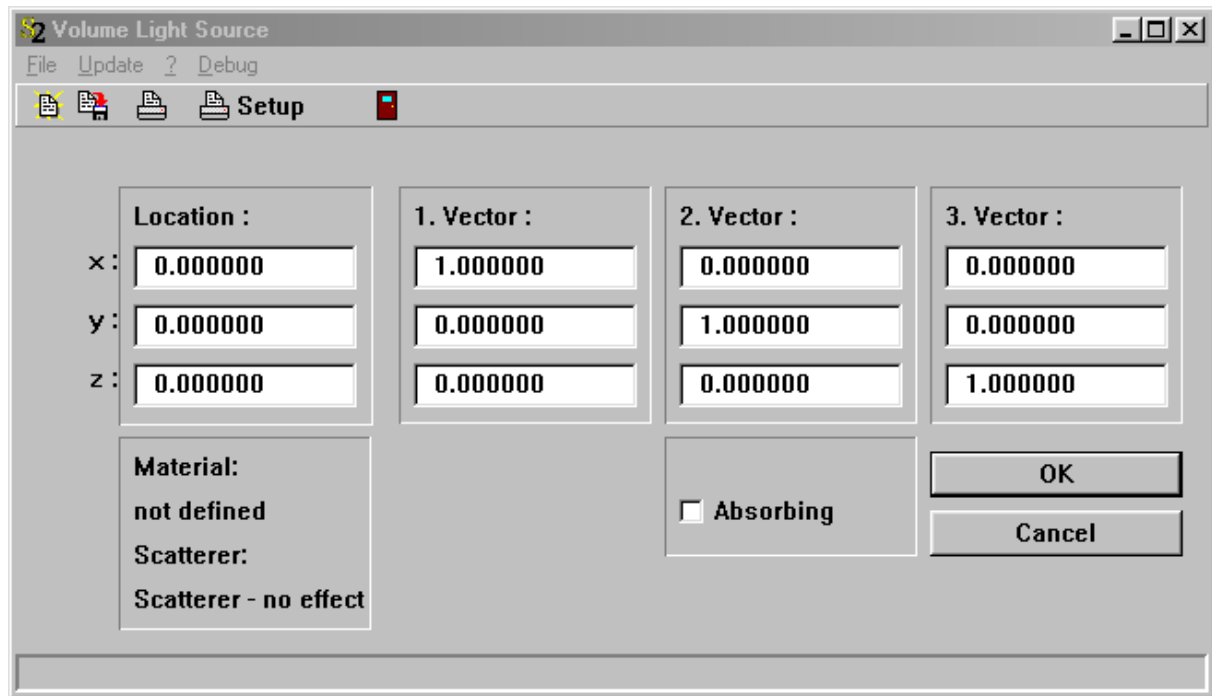
`object_parameter("MyName", "x1")`: read/write the x-coordinate of surface normal

`object_parameter("MyName", "y1")`: read/write the y-coordinate of surface normal

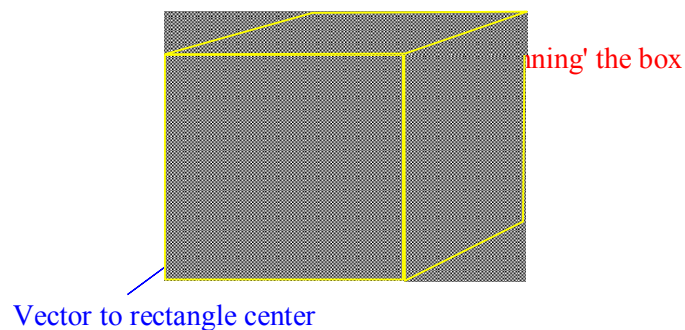
`object_parameter("MyName", "z1")`: read/write the z-coordinate of surface normal

5.2.5 Volume light source

Volume light sources are light emitting boxes. The orientation and size of the box and the other required parameters are specified in the following dialog:



The 'Location' vector gives the box center, whereas the other three vectors point from the center of the box to the centers of three rectangles that define the boundary of the box:



Access by OLE automation

OLE automation controllers can modify a volume light source named 'MyName' in the SPRAY object list by the following OLE commands:

```
object_parameter("MyName", "x"): read/write the x-coordinate of the position
object_parameter("MyName", "y"): read/write the y-coordinate of the position
object_parameter("MyName", "z"): read/write the z-coordinate of the position
```

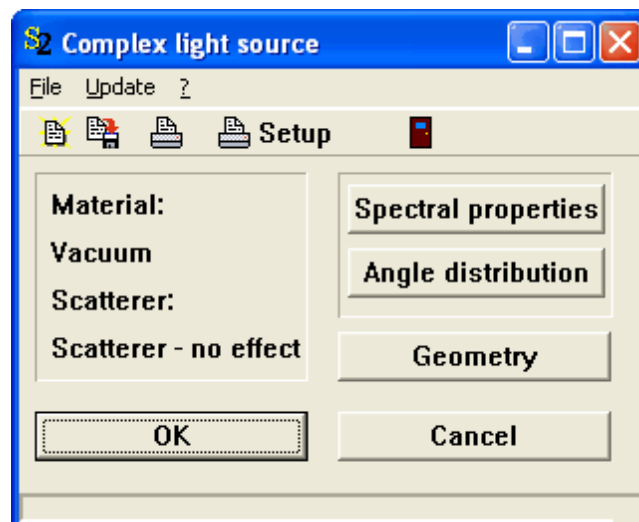

object_parameter("MyName", "x1"): read/write the x-coordinate of vector 1
object_parameter("MyName", "y1"): read/write the y-coordinate of vector 1
object_parameter("MyName", "z1"): read/write the z-coordinate of vector 1

object_parameter("MyName", "x2"): read/write the x-coordinate of vector 2
object_parameter("MyName", "y2"): read/write the y-coordinate of vector 2
object_parameter("MyName", "z2"): read/write the z-coordinate of vector 2

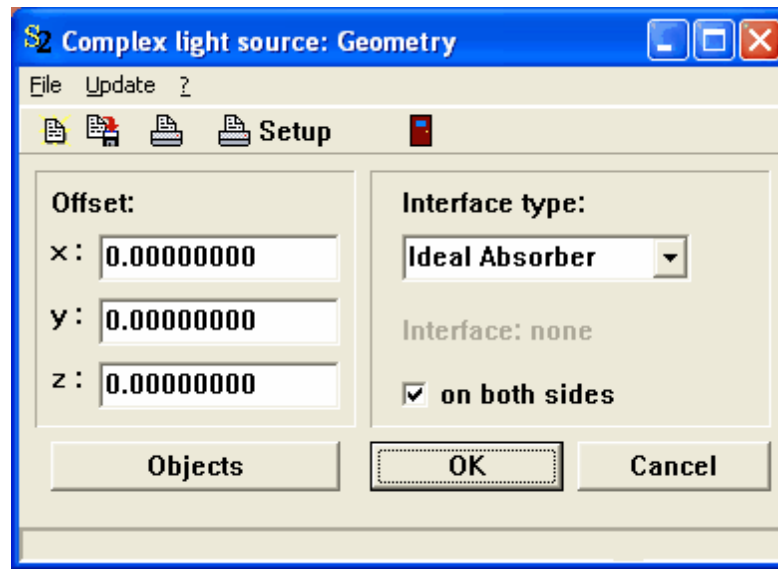
object_parameter("MyName", "x3"): read/write the x-coordinate of vector 3
object_parameter("MyName", "y3"): read/write the y-coordinate of vector 3
object_parameter("MyName", "z3"): read/write the z-coordinate of vector 3

5.2.6 Complex light source

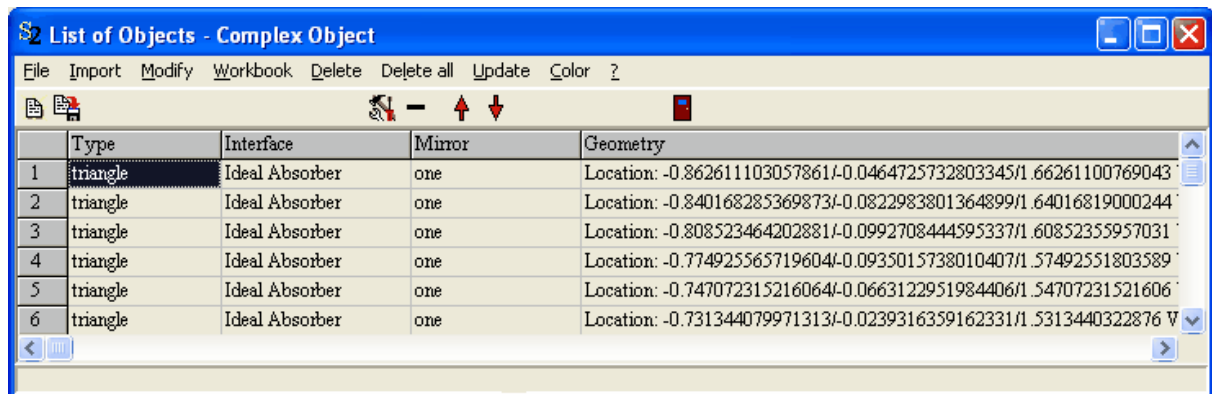
Complex light sources consist of several emitting surfaces. They are generated like geometric objects of type 'complex object' (see below). All subsurfaces must be embedded in the same material but are completely free to be positioned and oriented in space. The amount of rays started at the individual surfaces is computed according to the relative surface area of each particular surface with respect to the total area of the light source.



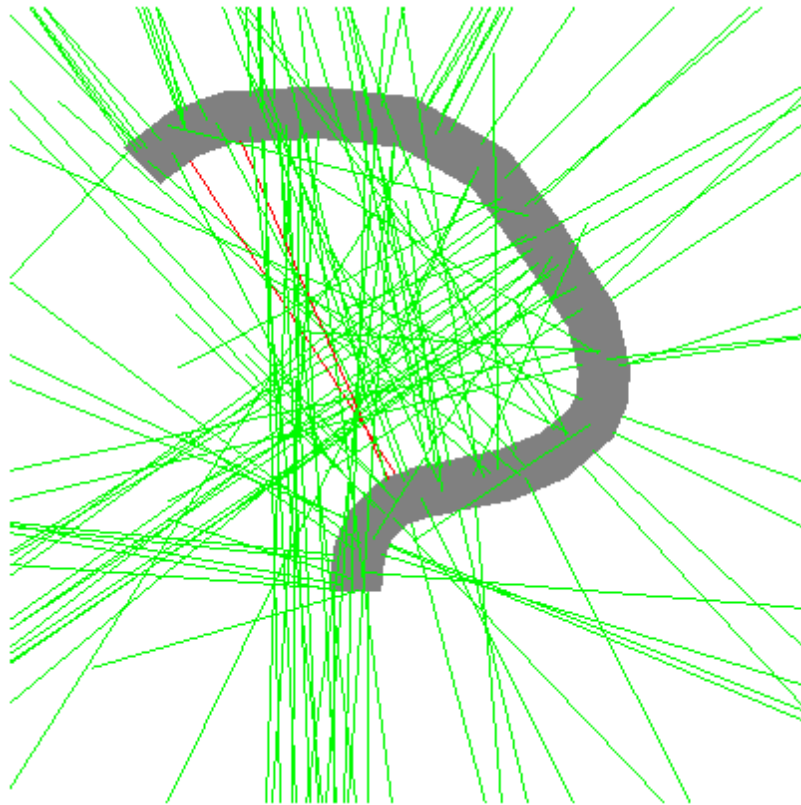
In the object's dialog, use the **Geometry** button to open a subwindow which allows the definition of the subobjects exactly the same way as described for complex objects:



The **Objects** button opens the list of the individual surfaces - here its a large collection of triangles obtained from a CAD program:



Here is the example of an emitting bended fiber:



5.3 Detectors

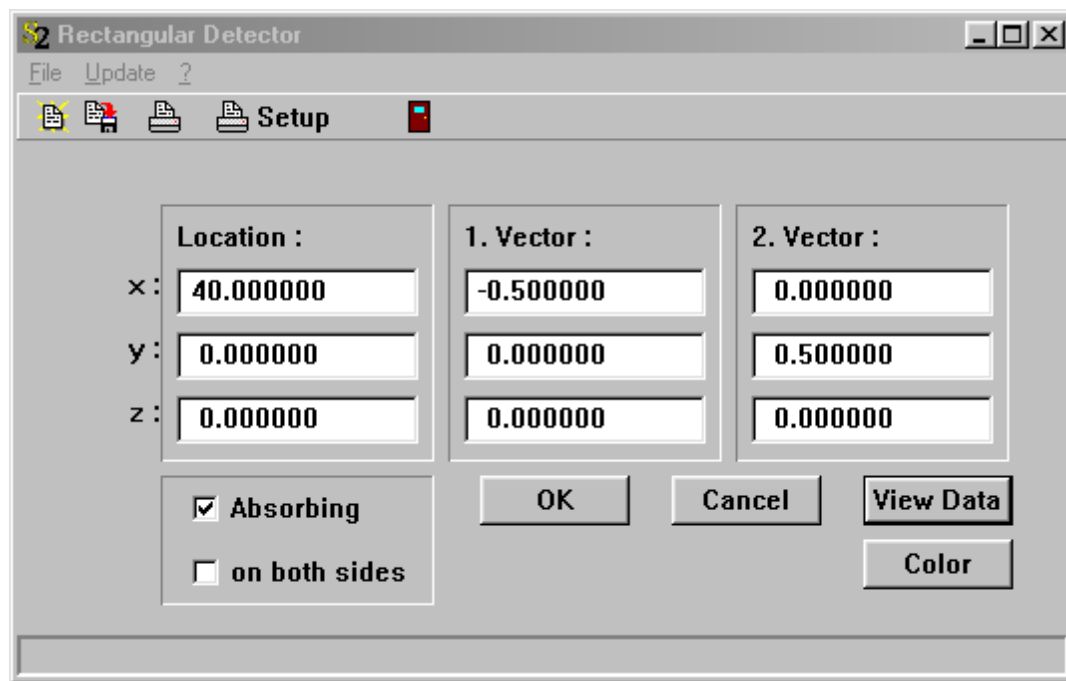
In most cases, detectors are the objects that produce the results that you are interested in. They tell you how much rays arrive at certain positions in space or how many rays are absorbed in a certain volume.

5.3.1 Surface detectors

Detectors of this kind count rays which are crossing a surface.

5.3.1.1 Rectangular detector

This type of detector is based on a rectangle that counts arriving rays. The dialog to set the object's geometry is this:

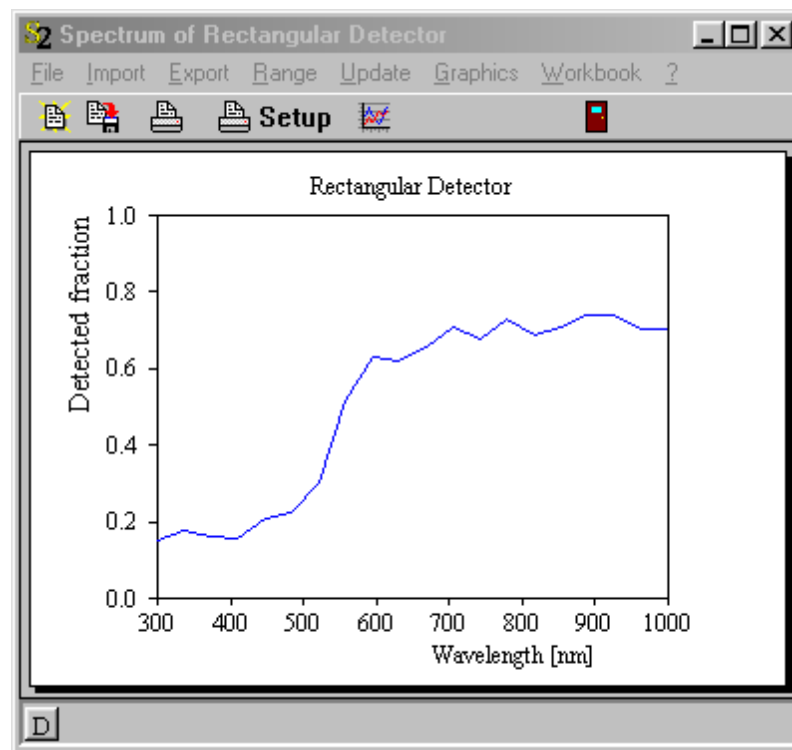


The meaning of the vector coordinates is exactly the same as for the rectangular light source discussed previously.

The option '**Absorbing**' determines how incoming rays are treated: If it's checked then arriving rays are counted and absorbed, otherwise counted and transmitted without any change of direction or polarization.

If you check the option 'on both sides' the detector will count any rays that hit the rectangle. If not, only those rays arriving from the side into which the surface normal points are recorded.

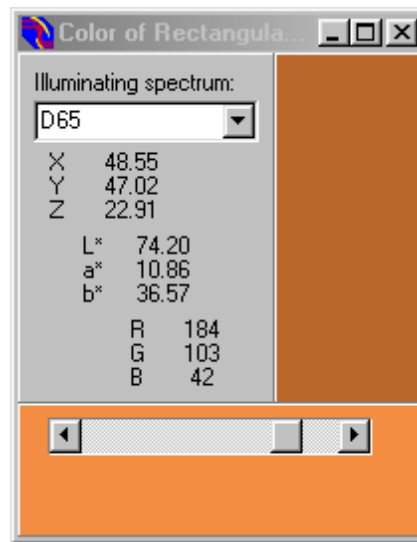
The button **View Data** opens a window that shows the results after a SPRAY simulation:



After the simulation all the detectors' spectral ranges are the same, namely the one specified for the SPRAY simulation. For each spectral point, the fraction of rays counted by the detector is computed, i.e. the ratio of the detected rays and the rays emitted by the light source. Please see the section 'How many rays do you need' for a short remark about the noise level of SPRAY spectra. The display of the data (please consult the 'Graphics course' if you don't know how to set and modify the graph) can be printed or copied to the clipboard (as Windows metafile, a vector graphics format).

You can export the spectrum to a data file using the **Export** command. The SCOUT technical manual contains a description of the available file formats. The same document also shows how to write the spectrum to the **Workbook** which is a good way to store and compare results of various SPRAY simulations. Alternatively, you can **drag&drop** the spectrum to the *Collect* and *Data Factory* utilities.

If the spectral range of the SPRAY simulation covers the visible spectral range you might be interested in the color corresponding to the recorded spectrum. Pressing the **Color** button (see the dialog of the geometrical parameters above) opens a small window showing color information:



This window is exactly the same as the one used to display color information in our Coating Designer (CODE) thin film software. Please consult the CODE manual for details. The detected spectrum is treated like a reflectance or transmittance spectrum of a layer stack, and the color coordinates are computed using the selected illuminating spectrum (Here: D65). SPRAY computes and displays X,Y,Z, L*, a* and b*. The color coordinates are converted to RGB values in order to paint a rectangle (upper right section of the window). This gives only a rough, qualitative impression of the color since the conversion is done in a simple, not very accurate way. The lower section shows a rectangle with the same a* and b* values, but a user-defined L* value which you can vary by moving the slider. If the original L* value is too small the color impression on the screen may appear to be almost black, and you can use the slider to get a brighter version of the color.

Of course, you can use as many detectors as you like in a SPRAY scenery.

Access by OLE automation

OLE automation controllers can modify a rectangular detector named 'MyName' in the SPRAY object list by the following OLE commands:

object_parameter("MyName", "x"): read/write the x-coordinate of the position
 object_parameter("MyName", "y"): read/write the y-coordinate of the position
 object_parameter("MyName", "z"): read/write the z-coordinate of the position

object_parameter("MyName", "x1"): read/write the x-coordinate of vector 1
 object_parameter("MyName", "y1"): read/write the y-coordinate of vector 1
 object_parameter("MyName", "z1"): read/write the z-coordinate of vector 1

object_parameter("MyName", "x2"): read/write the x-coordinate of vector 2
 object_parameter("MyName", "y2"): read/write the y-coordinate of vector 2
 object_parameter("MyName", "z2"): read/write the z-coordinate of vector 2

object_parameter("MyName", "color_x"): read the color coordinate x

object_parameter("MyName", "color_y"): read the color coordinate y
 object_parameter("MyName", "color_z"): read the color coordinate z
 object_parameter("MyName", "l_star"): read the color coordinate l_star
 object_parameter("MyName", "a_star"): read the color coordinate a_star
 object_parameter("MyName", "b_star"): read the color coordinate b_star
 object_parameter("MyName", "color_r"): read the color coordinate R
 object_parameter("MyName", "color_g"): read the color coordinate G
 object_parameter("MyName", "color_b"): read the color coordinate B

simple_detector_value("MyName", 2200): read the detected fraction at 2200 1/cm

5.3.1.2 Screen

Screens measure and display the intensity distribution of radiation like CCD cameras. A rectangle that can be positioned anywhere in the scenery is filled with a two-dimensional array of small rectangular pixels. Each pixel counts incoming rays, independent of frequency. After a SPRAY simulation the pixel signals are displayed using gray levels to indicate the measured number of rays - this way intuitive pictures of the intensity distribution are obtained.

The geometrical settings are done in the following dialog:

S2 Rectangular Screen

File Update ?

Setup

	Location :	1. Vector :	2. Vector :
x :	0.100000	0.000000	0.000000
y :	0.000000	-4.000000	0.000000
z :	0.000000	0.000000	4.000000

Resolution of Data:

x :	50
y :	50
Color :	20

☐ Absorbing

☐ on both sides

View Data

OK Cancel

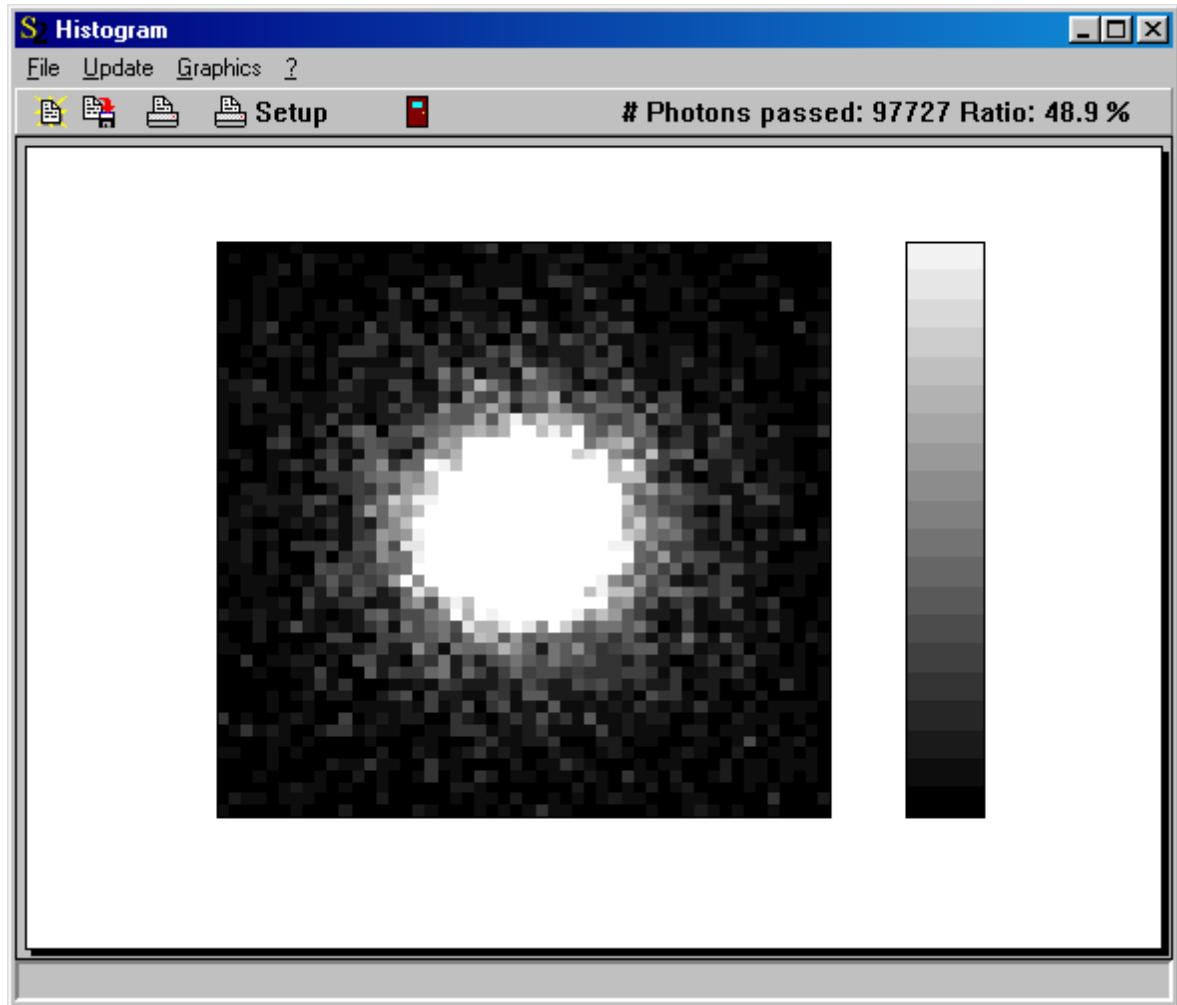
The vectors '**Location**', '**Vector 1**' and '**Vector 2**' define the rectangle as explained for the rectangular light source. In the section '**Resolution**' you can set the number of pixels to be used in the two directions: The pixels in the direction of '**Vector 1**' will be drawn horizontally, those in the '**Vector 2**' direction vertically. The '**Gray levels**' parameter determines how many different gray

levels are used for the drawing of each pixel.

If you check the '**Absorbing**' item, the rays hitting the screen will be absorbed. If not, they are counted and transmitted without any modification.

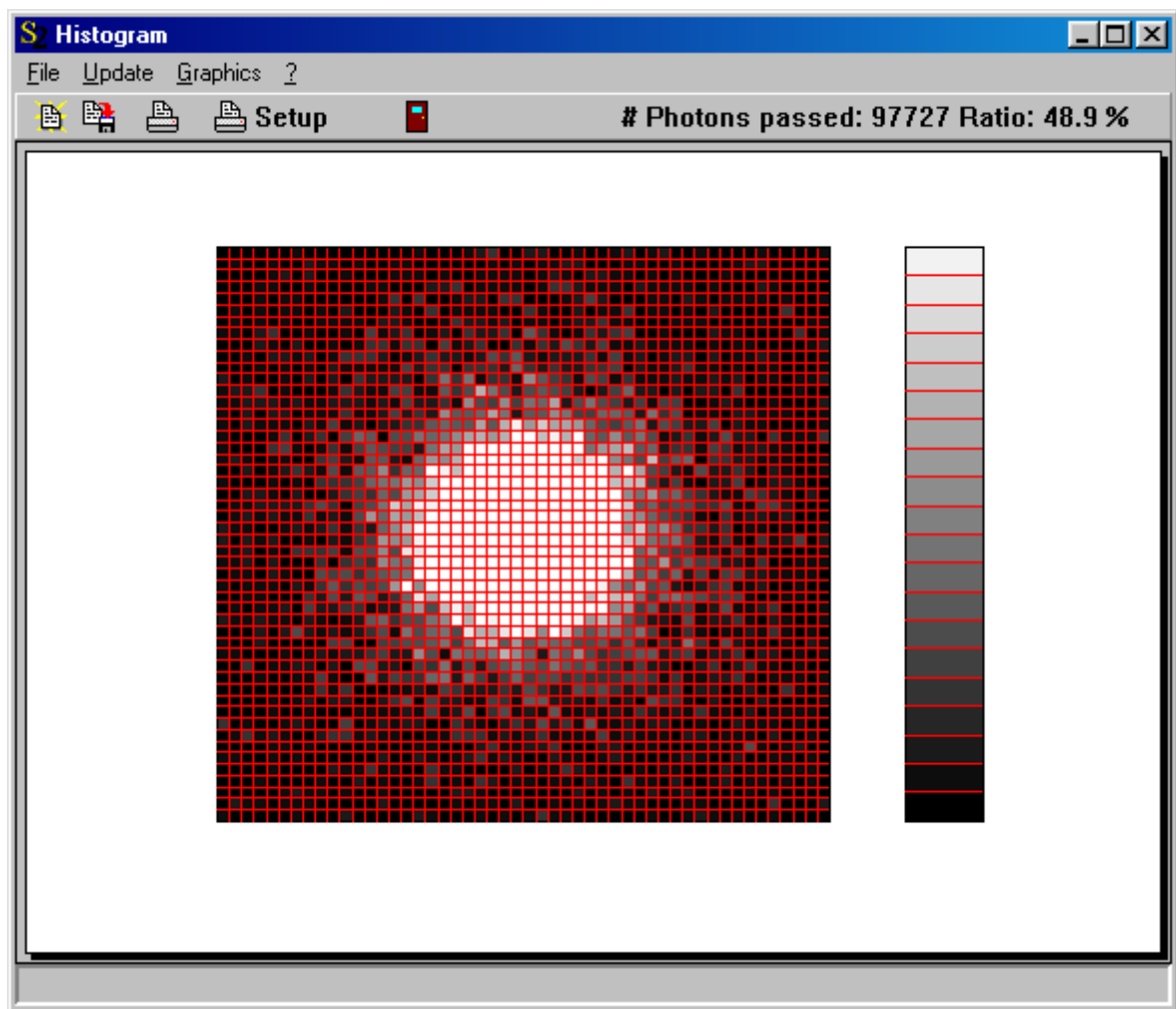
If '**on both sides**' is activated the rays are counted independent of their direction. If this property is turned off, only those rays are counted that approach from the side into which the surface normal points.

The '**View Data**' button opens the graphics window displaying the intensity distribution:

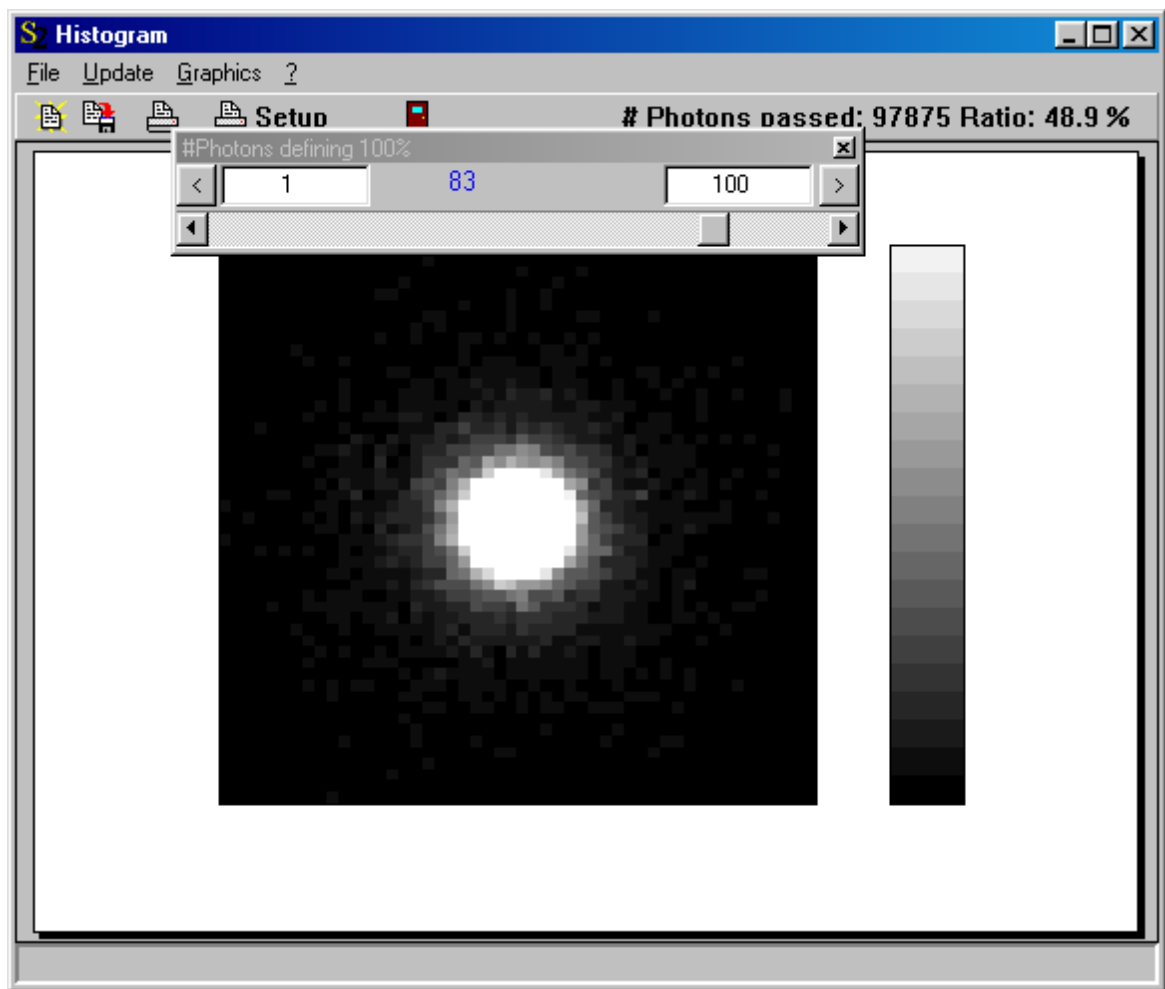


The window informs you about the number of rays that passed the screen (and the ratio of the total number of rays counted by the screen to the number of rays sent by the light source).

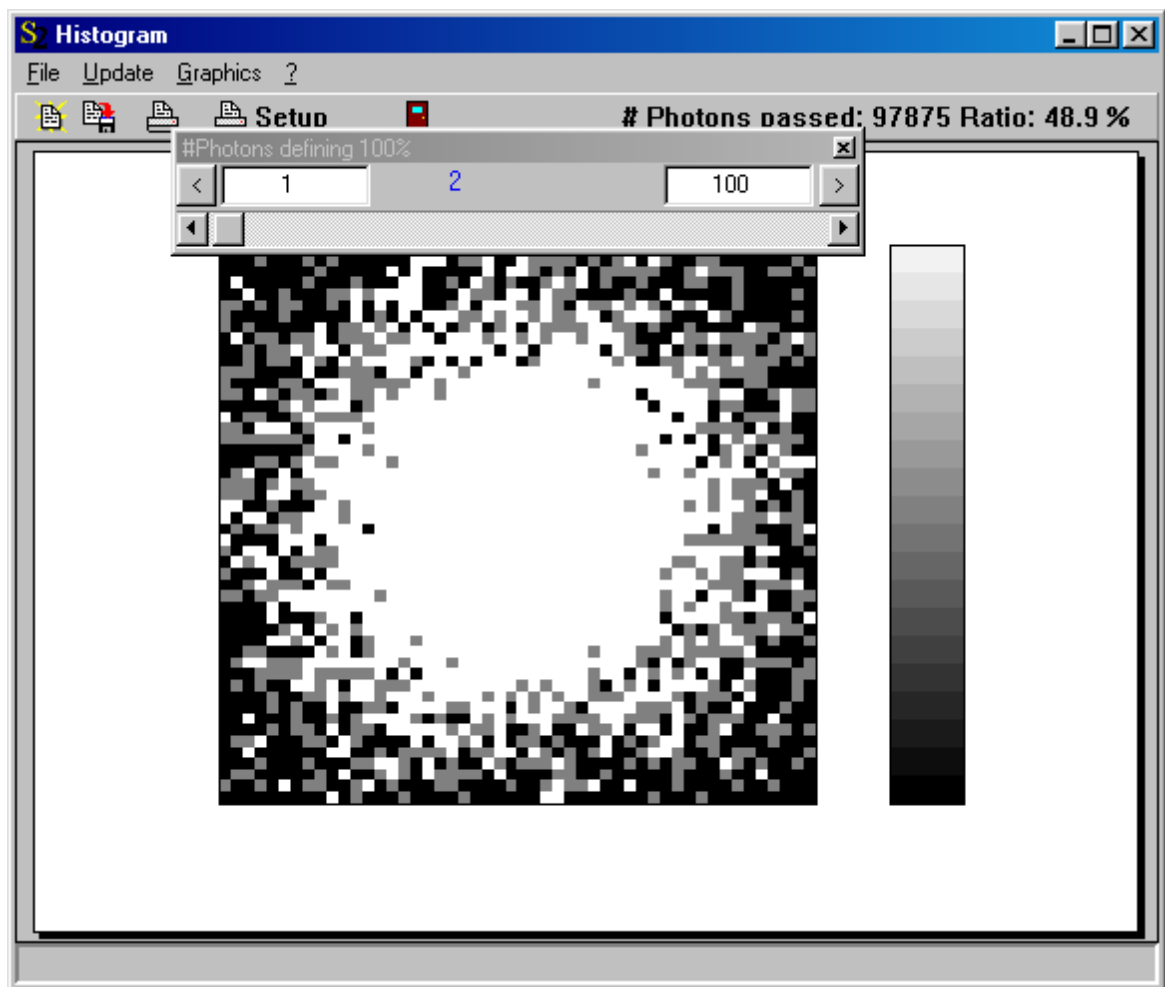
With the 'red lines' option in the Graphics menu you can turn on/off red lines separating the individual pixels:



Clicking the **right mouse button** somewhere in the graph opens a slider to modify the gray levels:



Moving the slider you can set the number of rays/pixel that is used as 100% (white color) for the picture. This way you can generate pictures with different 'brightness' to view the data at various intensity levels:



Access by OLE automation

OLE automation controllers can modify a screen named 'MyName' in the SPRAY object list by the following OLE commands:

`object_parameter("MyName", "x")`: read/write the x-coordinate of the position

`object_parameter("MyName", "y")`: read/write the y-coordinate of the position

`object_parameter("MyName", "z")`: read/write the z-coordinate of the position

`object_parameter("MyName", "x1")`: read/write the x-coordinate of vector 1

`object_parameter("MyName", "y1")`: read/write the y-coordinate of vector 1

`object_parameter("MyName", "z1")`: read/write the z-coordinate of vector 1

`object_parameter("MyName", "x2")`: read/write the x-coordinate of vector 2

`object_parameter("MyName", "y2")`: read/write the y-coordinate of vector 2

`object_parameter("MyName", "z2")`: read/write the z-coordinate of vector 2

5.3.1.3 Arrays

5.3.1.3.1 Linear array

Linear arrays are collections of rectangular detectors placed one after the other in a row. You can create a linear detector array selecting the new object type '**Array detector I**' in the list of objects and then pressing the '+' button.

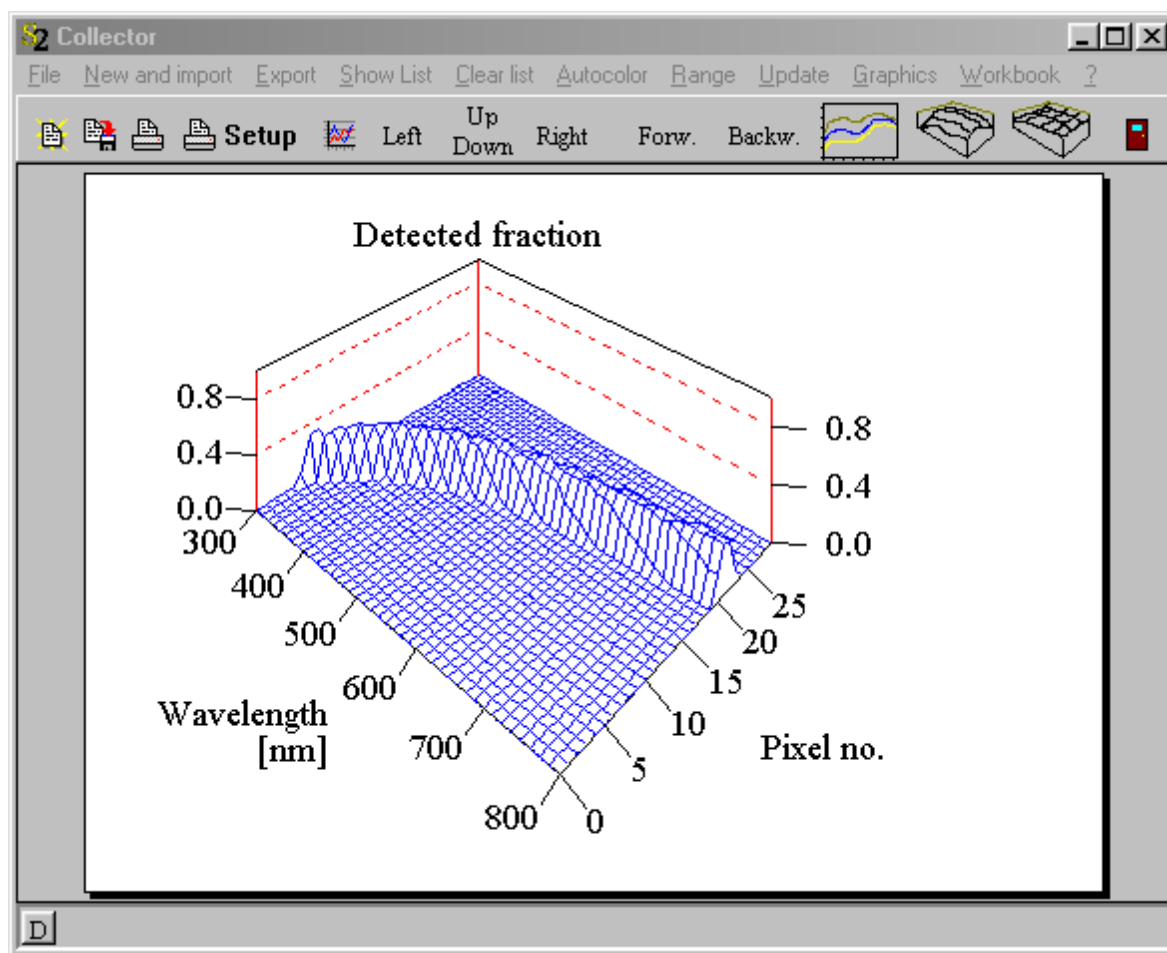
The geometric properties of linear arrays are defined in the following dialog:

The vectors **Location**, **Vector 1** and **Vector 2** define the total rectangle covered by the array, as described previously for rectangular light sources. The large rectangle is divided into individual pixels along the direction of vector 1. The number of pixels is set by the parameter '**# Pixels**'. The example given in the dialog above means the following: The pixel array has total dimensions of 0.3 cm along the x-axis and 0.04 cm along the y-axis. There are 30 pixels each of which has a length of 0.01 cm in x-direction and a width of 0.04 cm in the y-direction.

If **Absorbing** is checked incoming rays are counted by the corresponding pixel and absorbed. Otherwise rays are counted but transmitted without any modification.

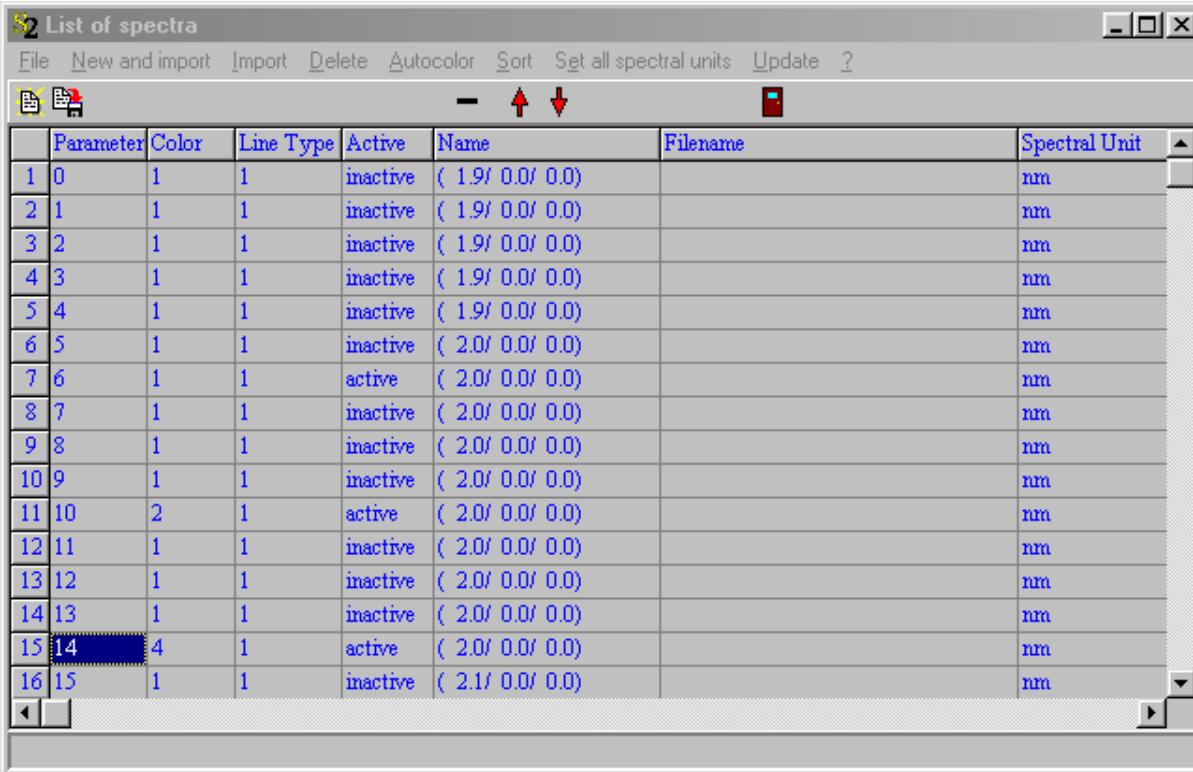
If **on both sides** is checked all rays are counted hitting the array both from the frontside and the backside. If not, only those rays arriving from the side of the surface normal are registered.

After a SPRAY simulation you can inspect the detector signals for all pixels pressing the **View data** button. A window identical to the main window of our *Collect* utility program opens:



With *Collect* you can represent data in 3D and 2D graphs (see the section about the use of the workbook in the SCOUT technical manual for more details about *Collect* graphs).

In 2D graphs you can compare spectra of selected pixels as shown below if you de-select spectra in the list of spectra (that opens by the **Show list** command):

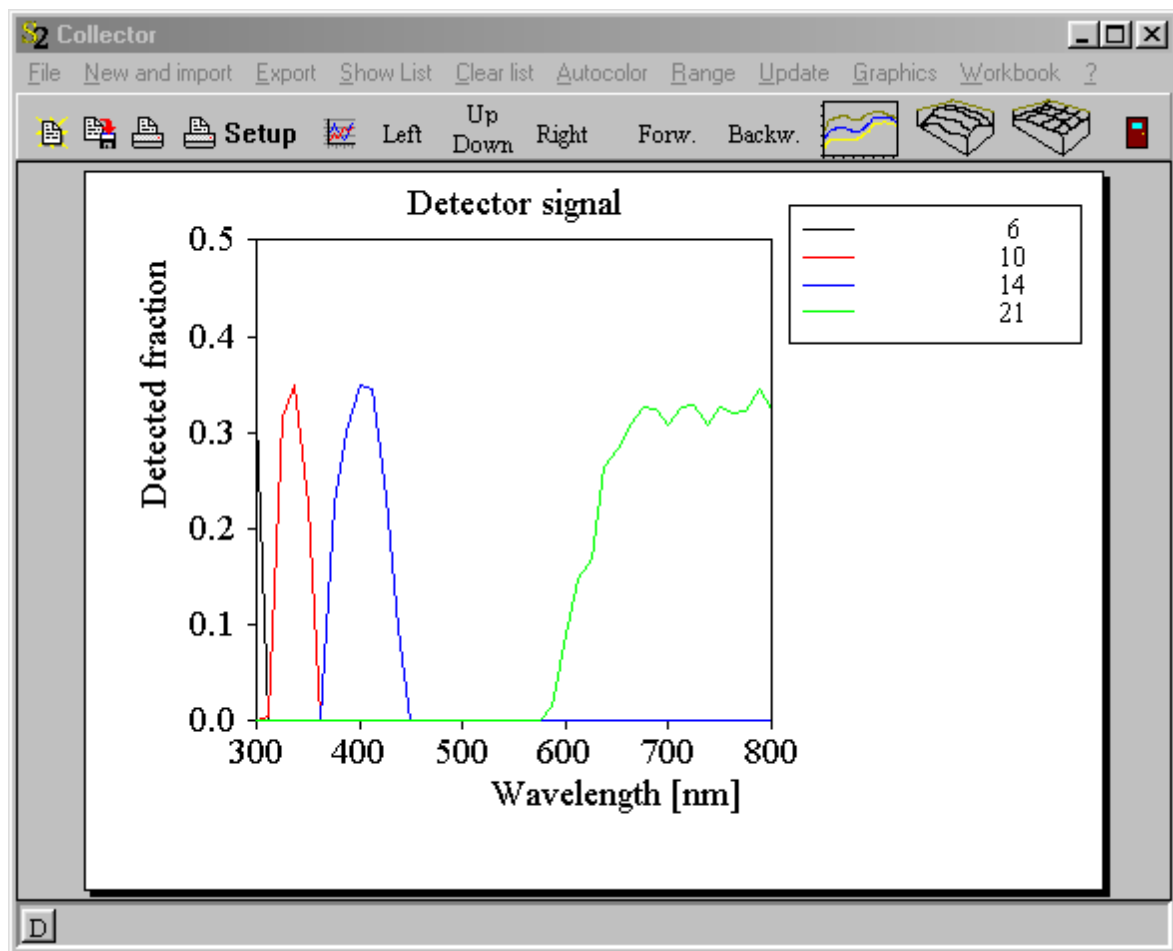


List of spectra

File New and import Import Delete Autocolor Sort Set all spectral units Update ?

	Parameter	Color	Line Type	Active	Name	Filename	Spectral Unit
1	0	1	1	inactive	(1.9/ 0.0/ 0.0)		nm
2	1	1	1	inactive	(1.9/ 0.0/ 0.0)		nm
3	2	1	1	inactive	(1.9/ 0.0/ 0.0)		nm
4	3	1	1	inactive	(1.9/ 0.0/ 0.0)		nm
5	4	1	1	inactive	(1.9/ 0.0/ 0.0)		nm
6	5	1	1	inactive	(2.0/ 0.0/ 0.0)		nm
7	6	1	1	active	(2.0/ 0.0/ 0.0)		nm
8	7	1	1	inactive	(2.0/ 0.0/ 0.0)		nm
9	8	1	1	inactive	(2.0/ 0.0/ 0.0)		nm
10	9	1	1	inactive	(2.0/ 0.0/ 0.0)		nm
11	10	2	1	active	(2.0/ 0.0/ 0.0)		nm
12	11	1	1	inactive	(2.0/ 0.0/ 0.0)		nm
13	12	1	1	inactive	(2.0/ 0.0/ 0.0)		nm
14	13	1	1	inactive	(2.0/ 0.0/ 0.0)		nm
15	14	4	1	active	(2.0/ 0.0/ 0.0)		nm
16	15	1	1	inactive	(2.1/ 0.0/ 0.0)		nm

The corresponding 2D graph is the following:



Access by OLE automation

OLE automation controllers can modify a linear array detector named 'MyName' in the SPRAY object list by the following OLE commands:

`object_parameter("MyName", "x")`: read/write the x-coordinate of the position

`object_parameter("MyName", "y")`: read/write the y-coordinate of the position

`object_parameter("MyName", "z")`: read/write the z-coordinate of the position

`object_parameter("MyName", "x1")`: read/write the x-coordinate of vector 1

`object_parameter("MyName", "y1")`: read/write the y-coordinate of vector 1

`object_parameter("MyName", "z1")`: read/write the z-coordinate of vector 1

`object_parameter("MyName", "x2")`: read/write the x-coordinate of vector 2

`object_parameter("MyName", "y2")`: read/write the y-coordinate of vector 2

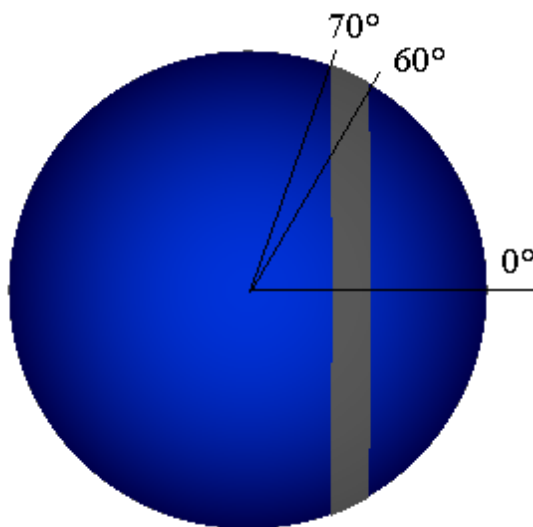
`object_parameter("MyName", "z2")`: read/write the z-coordinate of vector 2

5.3.1.3.2 Spherical detector arrays

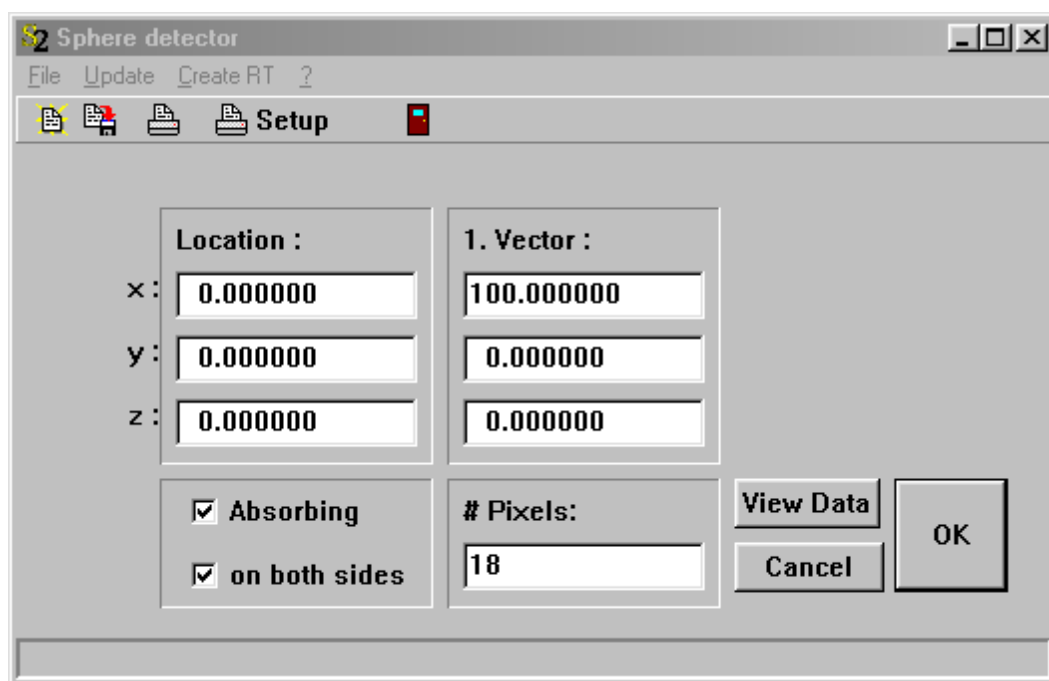
So-called 'Spherical detector arrays' are collections of detectors used to record the angular dependence of radiation. You can create a spherical detector array selecting the new object type

'Array detector III' in the list of objects and then pressing the '+' button.

The individual pixels of spherical detector arrays are sphere segments that divide the sphere according to the following scheme:



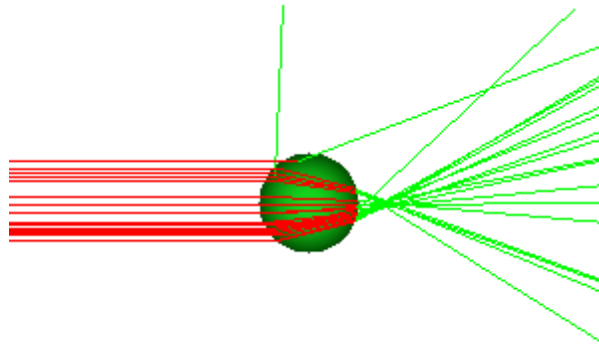
The geometric parameters are set in the following dialog:



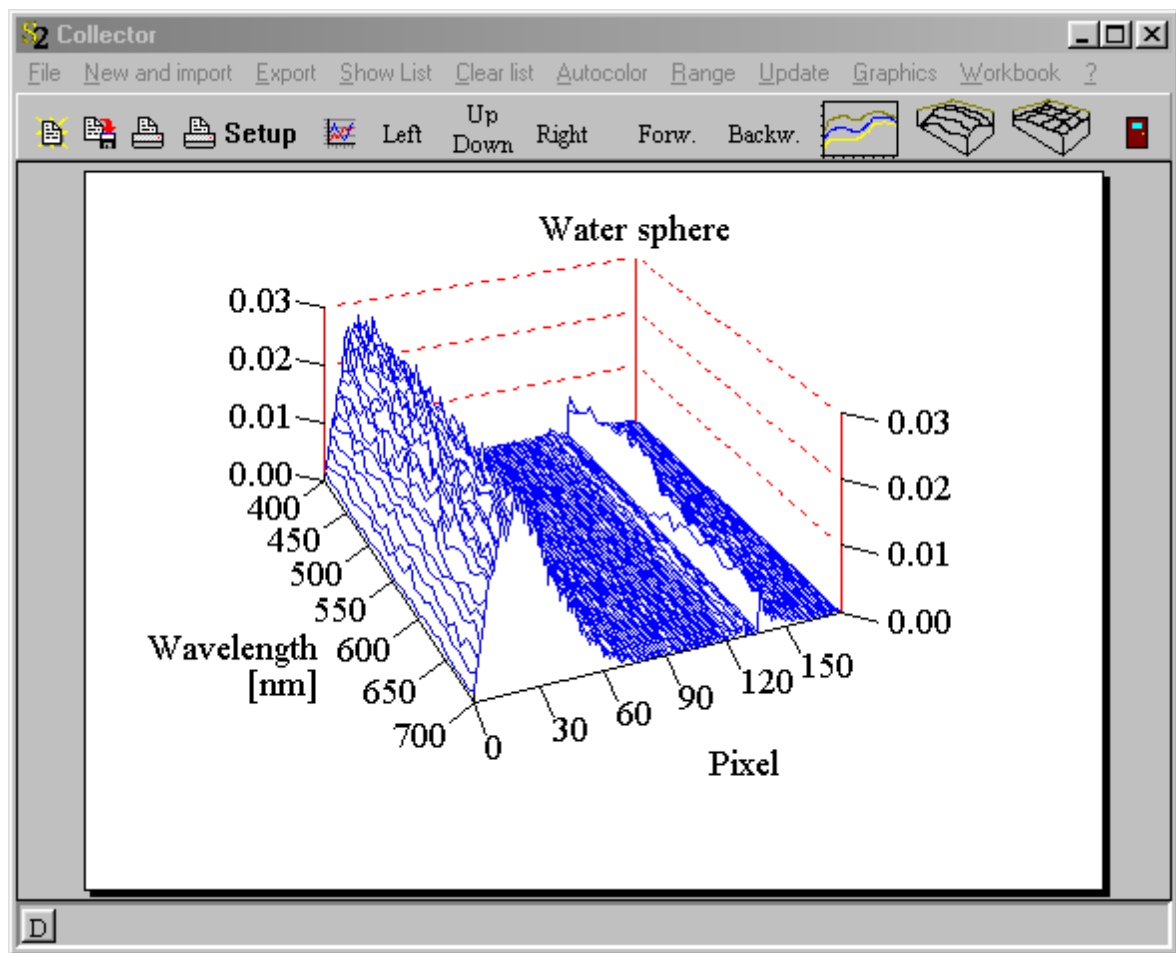
Location denotes the center of the sphere, **Vector 1** the direction for which the angle is 0°. At the same time, the length of vector 1 is the sphere radius.

The parameter **# Pixels** sets the number of sphere segments to be used. If - as shown above - you use 18 pixels the angle resolution is 10°.

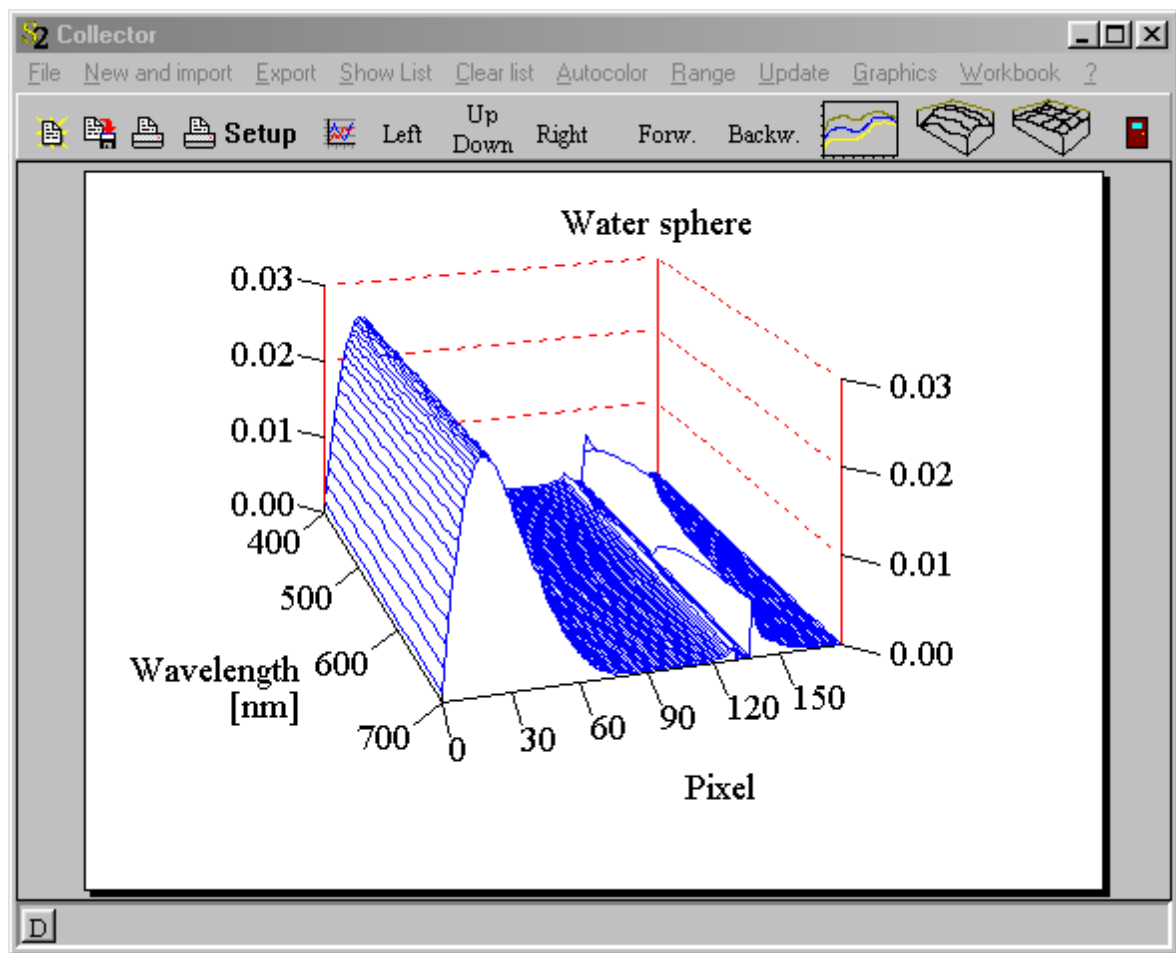
The following example shows how spherical detector arrays can be used to record the angular dependence of scattered radiation. Consider the simple example of a water sphere illuminated by sunlight from the left:



To 'measure' the angle dependence of the radiation you can place a huge spherical detector array (with a radius 100 times larger) around the water sphere. Using 180 pixels you will get an angular resolution of 1° . However, since some of the sphere segments (those close to 0° and 180°) will have a small area, you will need a lot of rays in order to get a significantly low noise level. Doing a SPRAY simulation with 10000 rays/spectral point in the range 400 to 700 nm with 31 spectral points (i.e. 10 nm wavelength resolution) the following angle distribution is obtained for a water sphere of 10 micron radius:

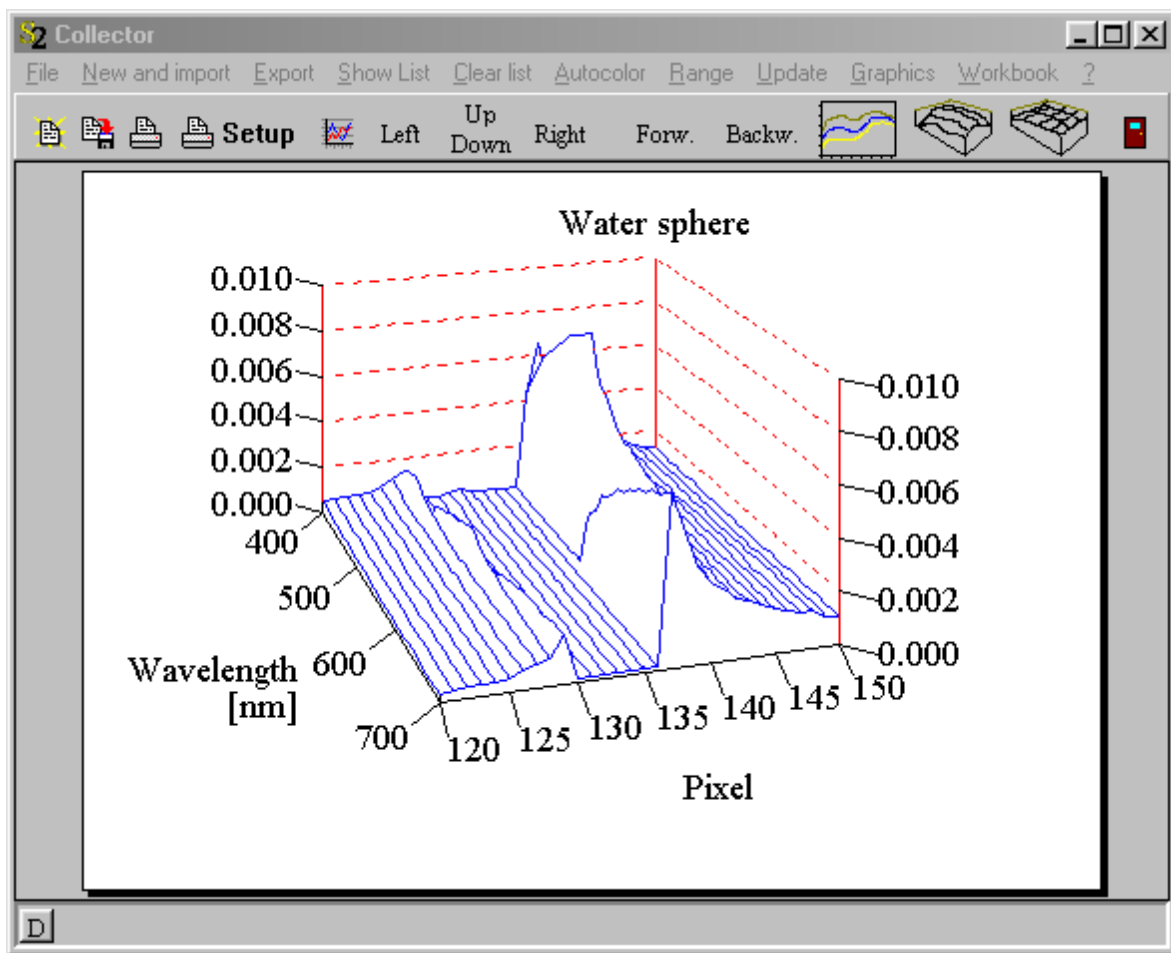


Increasing the number of rays to 1000000 /spectral points a much smoother distribution is obtained:



There is large forward scattering contribution and a sharp peak of radiation around 140° . The latter is the origin of the rainbow, of course. As you can see the peak for red light (700 nm) occurs at a slightly lower angle as for blue light (400 nm). Watching water spheres in the sky with the sun behind you a strong contribution of red light comes from a different direction than the blue maximum, with all other colors in between.

You can also reproduce the second rainbow (scattering angles around 130°) which shows the reverse sequence of colors. In between the two rainbows there is a dark zone with almost no radiation:



Generating RT files for spheres with spherical detector arrays

You can use spherical detector arrays to generate single scattering data for spheres that you can later use for multiple scattering investigations in scatterers. You must be aware, however, that you are using ray-tracing to do so which may lead to wrong results in some cases. For example, you will miss phase effects (interference of partial waves) in the angle distribution. Also, you will miss the refraction of radiation 'around' the sphere which leads for large spheres to a very intense, but narrow forward scattering peak.

However, there are also a large number of cases where the ray-tracing approach is very useful to determine single scattering characteristics. Here are some rules for RT file generation with SPRAY:

- Use a circular light source which creates a parallel light bundle with exactly the same radius as the sphere (i.e. the light source radius equals the sphere radius and the cone angle of the light source is 0°)
- The sphere may be covered with an arbitrary multilayer stack as long as the total thickness of the coating is much smaller as the sphere radius
- The direction of 0° of the spherical detector array must be in the direction of the light source emission
- The light source should be transparent

- Do the simulation using a spectral range defined in wavenumbers
- Use 180 pixels

After the SPRAY simulation finishes you have to open the dialog of the sphere detector (see above) and select the menu item called '**Create RT**'. You will be asked for the sphere radius and the refractive index of the surrounding medium. Finally you are asked for the filename of the RT file to be saved. The RT file can then be re-imported by General scatterer objects.

5.3.2 Volume detectors

These detectors count rays which are absorbed in a volume.

5.3.2.1 Grave

This is a rectangular box object which counts all rays which end inside by absorption. You may call this kind of object "voxel".

Normalizing to the number of emitted rays for each spectral position an absorption spectrum is computed after the ray-tracing.

5.3.2.2 Cemetery

A cemetery is an array of grave objects. After the ray-tracing you will have an array of absorption spectra.

5.3.2.3 Absorbing material

Although managed by the list of geometry objects, objects of this type are not really geometric objects - they do not have geometric dimensions. Instead they count how many rays are absorbed by one of the materials. The material is assigned by drag&drop from the list of materials.

After the ray-tracing the absorption spectrum is computed by normalizing the absorbed rays to the number of emitted rays for each spectral position.

Please note that 'Absorbing material' objects count only those absorption events which occur inside a macroscopic volume. Absorption inside thin films of interfaces is not taken into account.

Objects of this type have been introduced to SPRAY in order to be able to 'record' absorption spectra of silicon wafers with textured surface.

5.4 Interface objects

5.4.1 Overview

The following geometrical objects in SPRAY may be covered by simple pre-defined interfaces or user-defined interfaces (details of interfaces have been given above). The objects are divided into two classes: Some (like spheres or solid cylinders) completely enclose a certain volume with their surface. These are called **Closed volumes**. Others (like rectangles or circles) are referred to as 'Open structures'. These objects cannot enclose a volume alone.

'Open' structures:

- Rectangle
- Triangle
- Circle
- Sphere segment
- Cylinder (open)
- Cone
- Ellipsoid segment
- Paraboloid
- Circular aperture
- User-defined surface: Rectangular basis
- User-defined surface: Circular basis

Closed volumes

- Sphere
- Cylinder (closed)
- Rectangular box
- Prism
- ATR crystal
- Converging lens
- Diverging lens

5.4.2 Rectangular interface

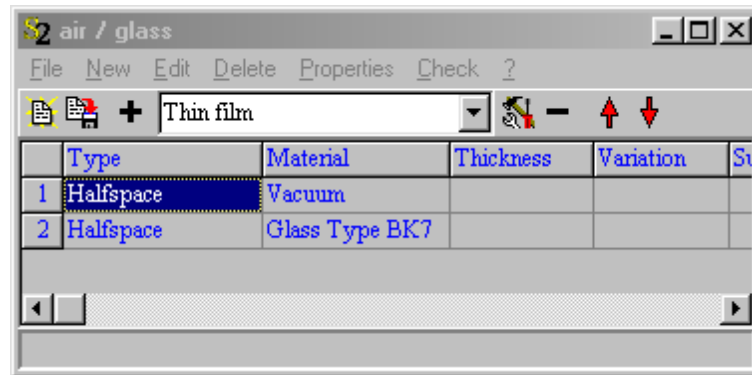
Objects of this type are rectangles the geometry of which is defined in the following dialog:

The dialog box is titled "Rectangle". It contains the following fields and controls:

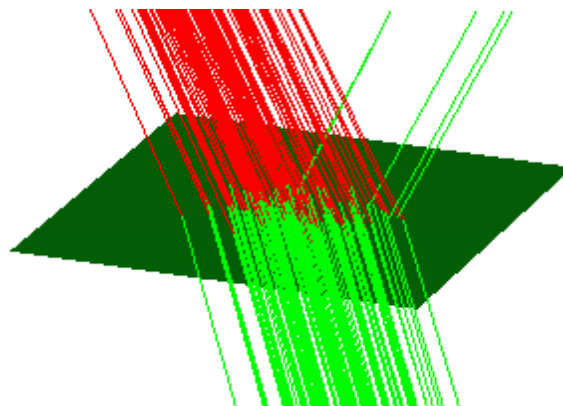
- Location :**
 - x : 0.000000
 - y : 0.000000
 - z : 0.000000
- 1. Vector :**
 - 1.000000
 - 0.000000
 - 0.000000
- 2. Vector :**
 - 0.000000
 - 1.000000
 - 0.000000
- Interface:** air / glass
- User-defined:** Dielectric Interface (dropdown menu)
- ☐ on both sides
- OK
- Cancel

The meaning of the vectors is exactly the same as explained for rectangular light sources.
The dropdown-box below 'User-defined' let's you select between a perfectly absorbing, perfectly

reflecting or a user-defined interface. If you choose 'Dielectric interface' you have to drag&drop an entry of the list of interfaces to the text label to the right of 'Interface:'. In the example shown above the interface named 'air / glass' is selected to cover the rectangle. The interface is simply the boundary between two halfspaces of air and glass, respectively:



In a rendered view this rectangle looks like this (illumination from the air side, refraction into the glass side):



If an interface is a boundary between two different materials (like in the example shown above) you as SPRAY user are responsible for consistency: Under no circumstance should a ray return from the glass side back into air without passing another interface. This is physically impossible, of course, and your model should be setup in a way that unphysical situations are avoided.

The safest and best way to do realistic simulations is to completely enclose a material with interfaces, e.g. to build a closed volume with several rectangular interfaces (see also the pre-defined 'volume objects' below). However, you can shortcut and work with 'open structures' like the interface in the picture above if you are sure that definitely no ray will return once it travels on the glass side.

Access by OLE automation

OLE automation controllers can modify a rectangular interface named 'MyName' in the SPRAY object list by the following OLE commands:

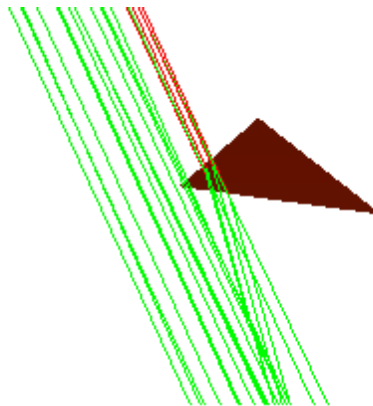
```
object_parameter("MyName", "x"): read/write the x-coordinate of Location
```

object_parameter("MyName", "y"): read/write the y-coordinate of Location
 object_parameter("MyName", "z"): read/write the z-coordinate of Location

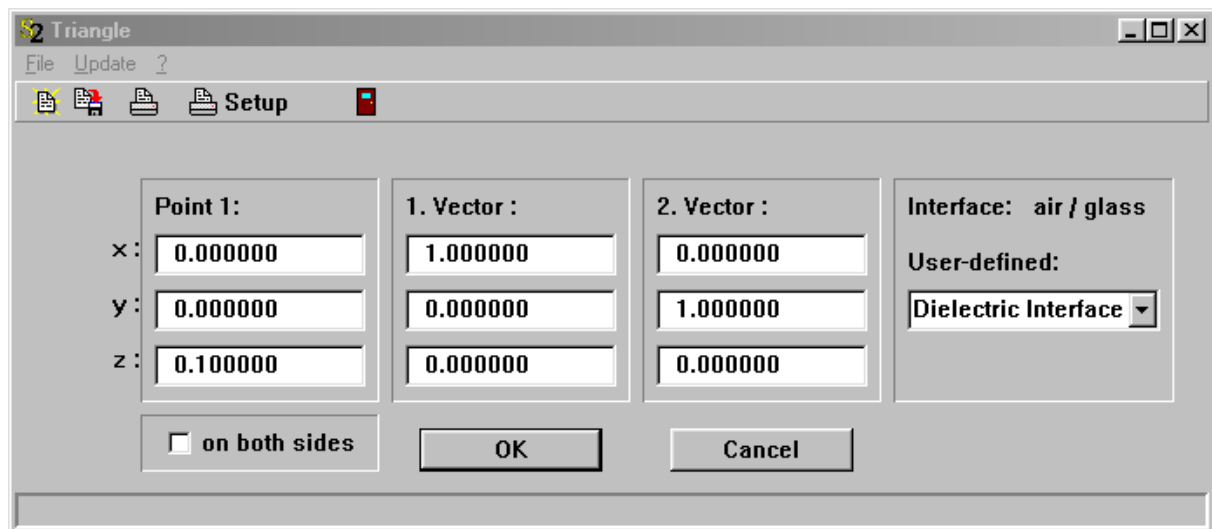
object_parameter("MyName", "x1"): read/write the x-coordinate of vector 1
 object_parameter("MyName", "y1"): read/write the y-coordinate of vector 1
 object_parameter("MyName", "z1"): read/write the z-coordinate of vector 1

object_parameter("MyName", "x2"): read/write the x-coordinate of vector 2
 object_parameter("MyName", "y2"): read/write the y-coordinate of vector 2
 object_parameter("MyName", "z2"): read/write the z-coordinate of vector 2

5.4.3 Triangle



Triangles are defined specifying one corner (called '**Point 1**') and two vectors from that corner to the other corners (called '**Vector 1**' and '**Vector 2**'). The cross product of vector 1 and vector 2 is the surface normal of the triangle. The settings are done in the following dialog:



The definition of the interface covering the triangle is the same as explained for rectangles.

Access by OLE automation

OLE automation controllers can modify a triangular interface named 'MyName' in the SPRAY object list by the following OLE commands:

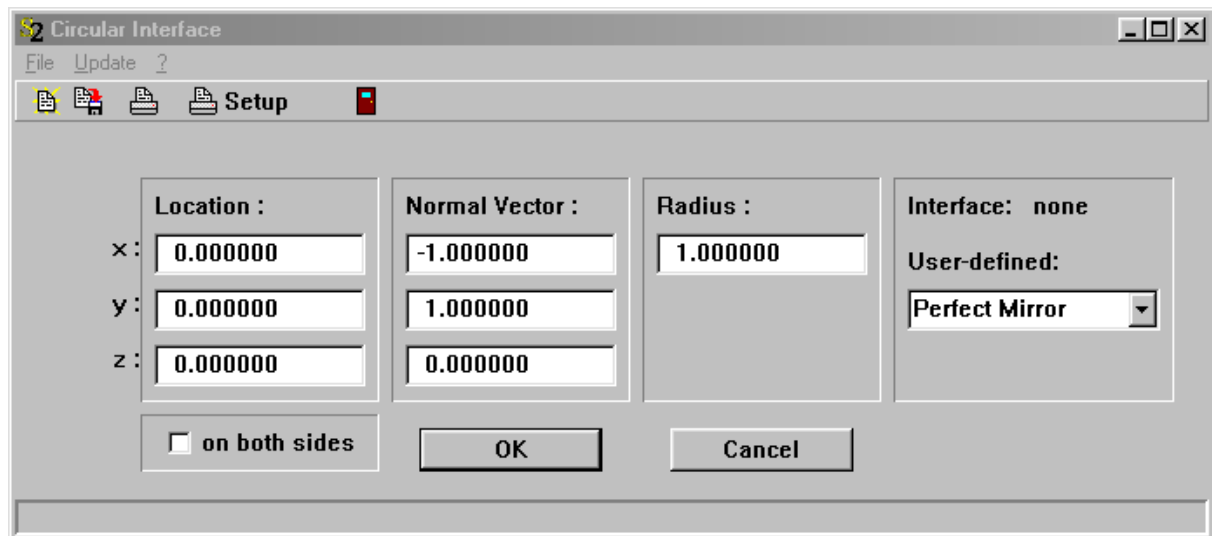
object_parameter("MyName", "x"): read/write the x-coordinate of Point 1.
object_parameter("MyName", "y"): read/write the y-coordinate of Point 1.
object_parameter("MyName", "z"): read/write the z-coordinate of Point 1.

object_parameter("MyName", "x1"): read/write the x-coordinate of vector 1
object_parameter("MyName", "y1"): read/write the y-coordinate of vector 1
object_parameter("MyName", "z1"): read/write the z-coordinate of vector 1

object_parameter("MyName", "x2"): read/write the x-coordinate of vector 2
object_parameter("MyName", "y2"): read/write the y-coordinate of vector 2
object_parameter("MyName", "z2"): read/write the z-coordinate of vector 2

5.4.4 Circle

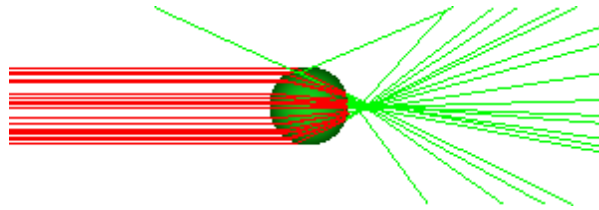
Circular interfaces (create an object of type 'Interface circular stop') are defined in the following dialog:



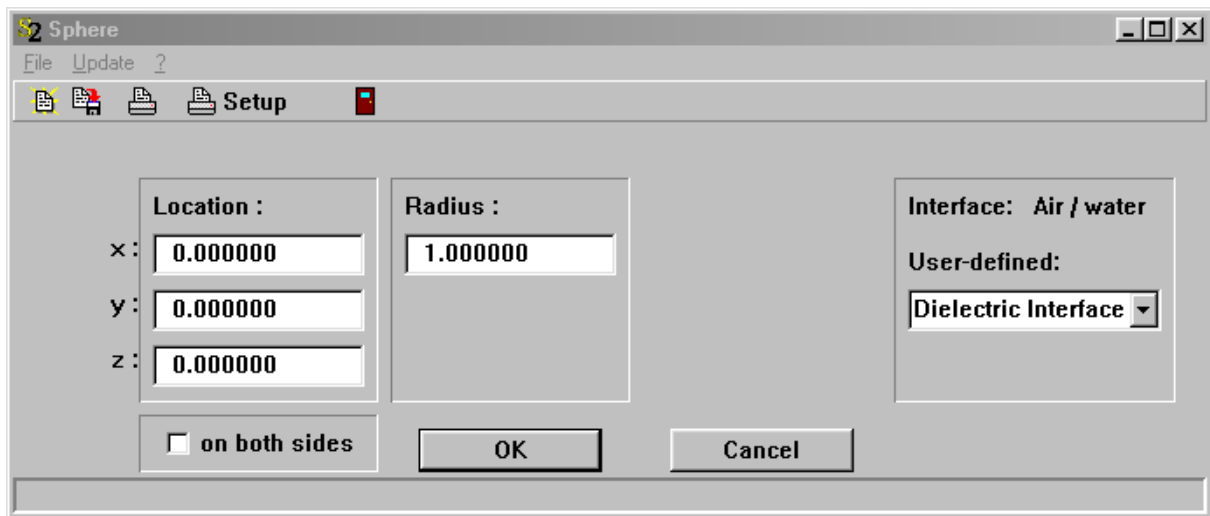
Location points to the center of the circle, **Normal Vector** is the surface normal, and **Radius** the radius of the circle.

The definition of the interface covering a circle is the same as explained for rectangles.

5.4.5 Sphere



These objects are spheres the parameters of which are set in this dialog:



Location is the center of the sphere, **Radius** the radius (of course).

The definition of the interface covering the sphere is the same as explained for rectangles.

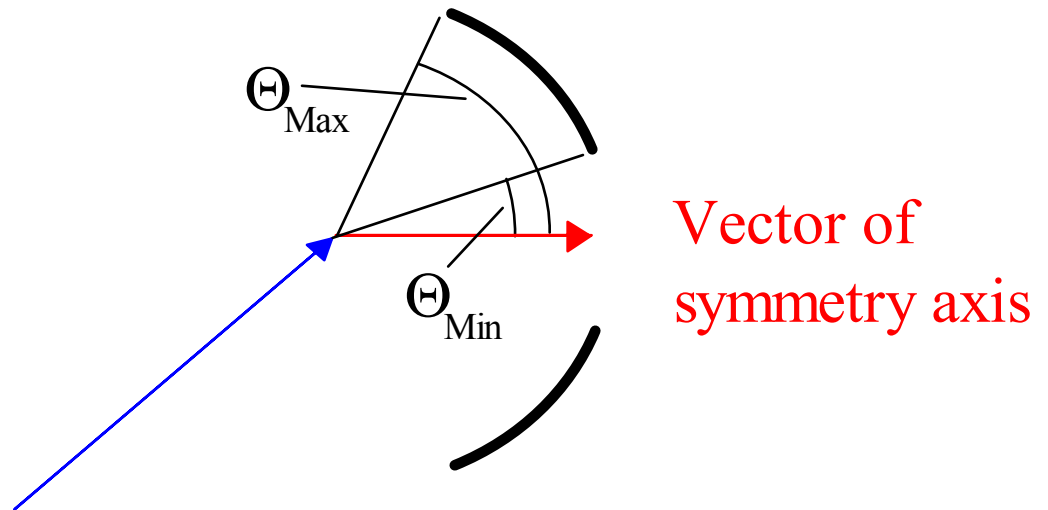
5.4.6 Sphere segment



Objects of this type are sphere segments, defined by the center of the sphere, the symmetry axis and

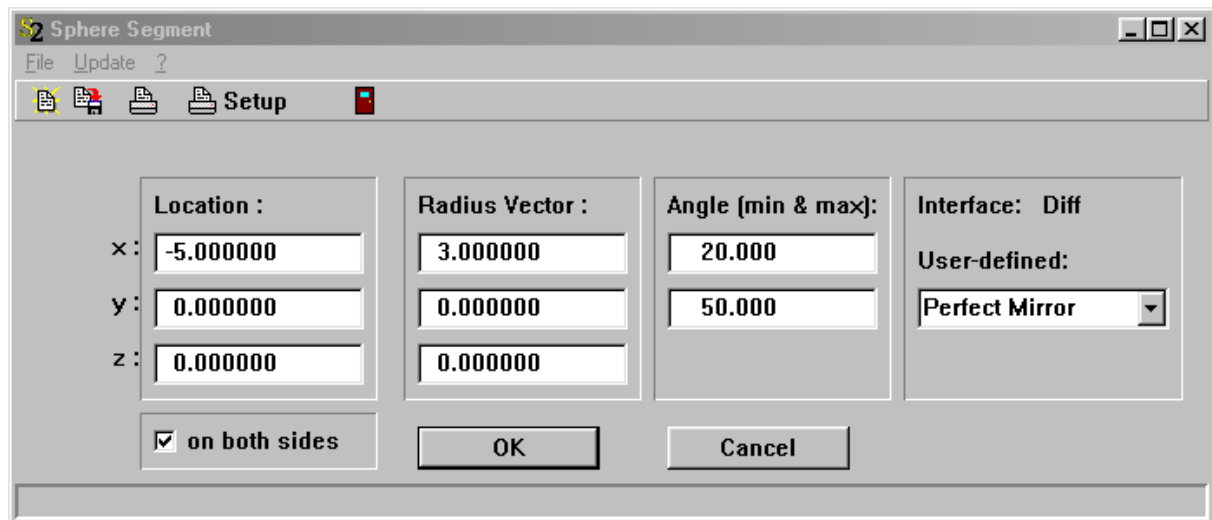
two angles Θ_{Min} and Θ_{Max} according to the following scheme:

Sphere segments



Vector to sphere center

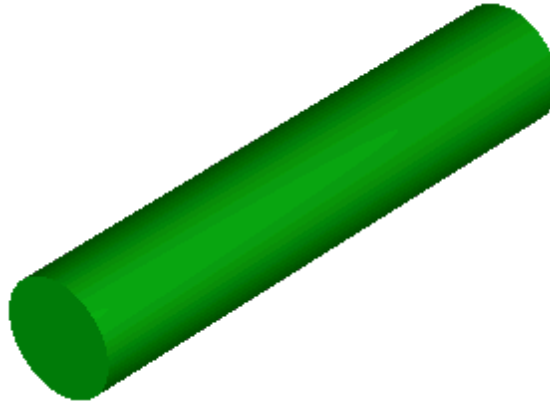
The dialog to set the object's parameters is this:



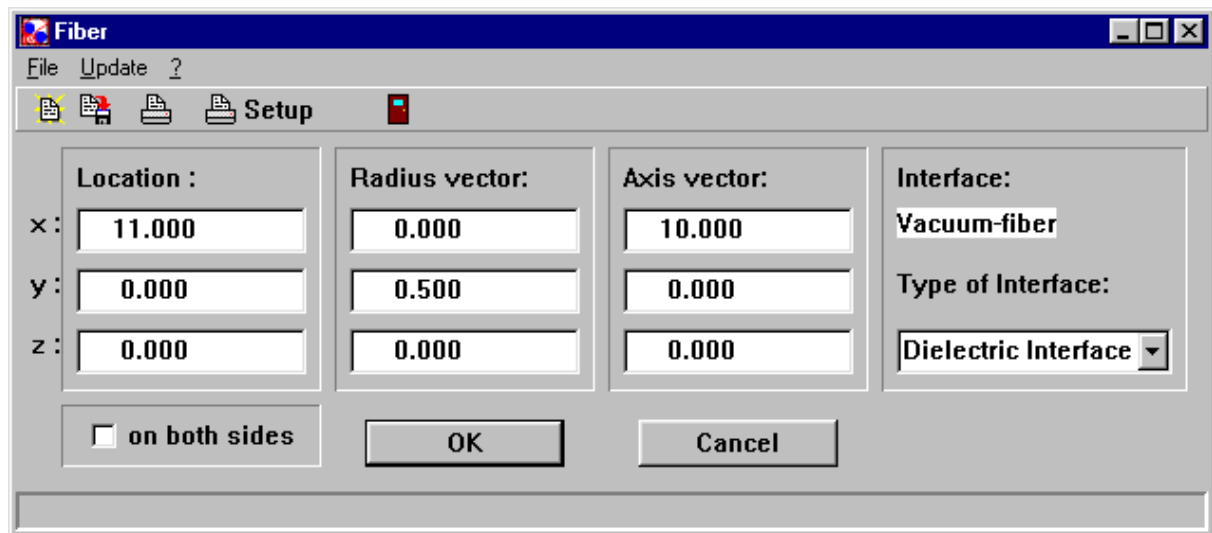
Here **Location** is the vector to the center of the sphere. The **Radius vector** points along the symmetry axis. Its length equals the radius of the sphere. The two angles are Θ_{min} and Θ_{max} (top and bottom, respectively), given in degrees. The angle range is $0^\circ \dots 180^\circ$.

The definition of the interface covering sphere segments is the same as explained for rectangles.

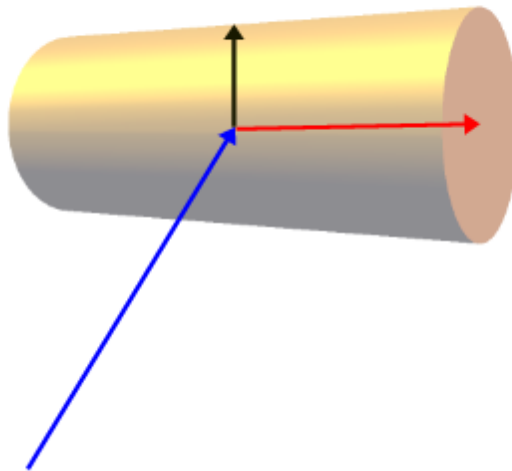
5.4.7 Cylinder (closed)



Closed cylinders are defined in dialogs like this:



Location is the vector to the center of the cylinder. The **Axis vector** points along the main symmetry axis (the red arrow in the sketch below) whereas the **Radius vector** points from the center to the cylindrical surface (black arrow). Axis vector and radius vector must be perpendicular to each other.



The definition of the interface covering cylinders is the same as explained for rectangles. The cylindrical surface as well as the two circular areas have the same interface. If you need different interfaces, you have to use a combination of an open cylinder and two circular interfaces.

Access by OLE automation

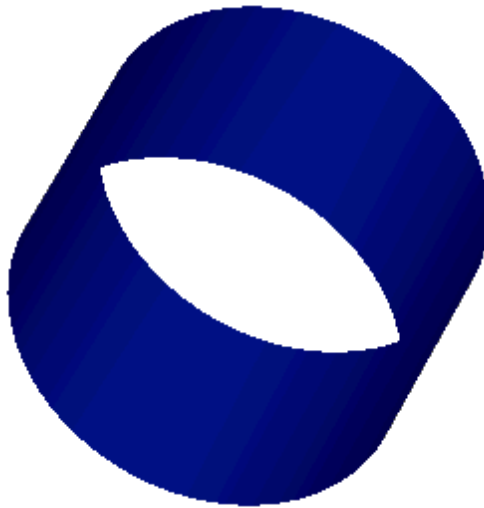
OLE automation controllers can modify a closed cylinder named 'MyName' in the SPRAY object list by the following OLE commands:

object_parameter("MyName", "x"): read/write the x-coordinate of Location.
object_parameter("MyName", "y"): read/write the y-coordinate of Location.
object_parameter("MyName", "z"): read/write the z-coordinate of Location.

object_parameter("MyName", "x1"): read/write the x-coordinate of Radius vector.
object_parameter("MyName", "y1"): read/write the y-coordinate of Radius vector.
object_parameter("MyName", "z1"): read/write the z-coordinate of Radius vector.

object_parameter("MyName", "x2"): read/write the x-coordinate of Axis vector.
object_parameter("MyName", "y2"): read/write the y-coordinate of Axis vector.
object_parameter("MyName", "z2"): read/write the z-coordinate of Axis vector.

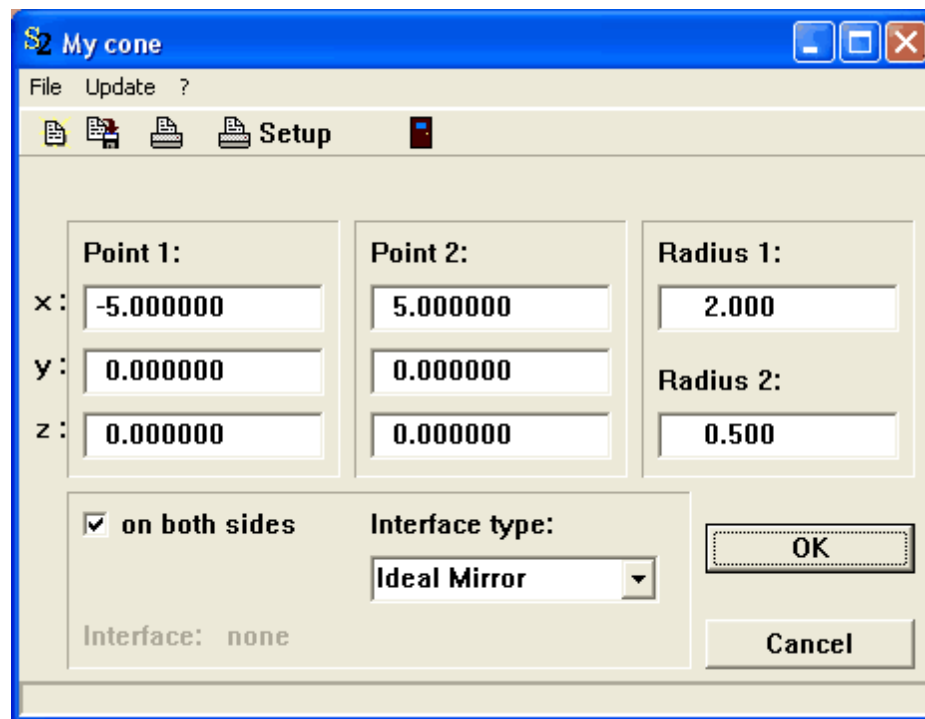
5.4.8 Cylinder (open)



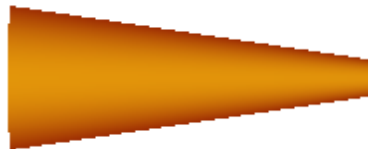
Open cylinders are - of course - very similar to closed cylinders discussed above. Only the two circular areas closing the cylinder at its top and bottom side are missing.

5.4.9 Cone

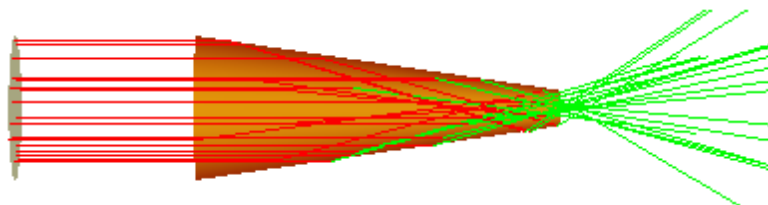
Cone objects are similar to open cylinders, with the difference that the radius of the cylinder changes linearly from one end to the other. Cones are defined by setting the two end points and the two radii of the circles at each end. Here is an example:



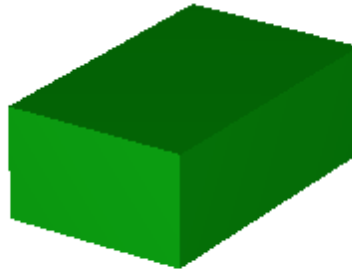
The cone defined by these settings looks from side like this:



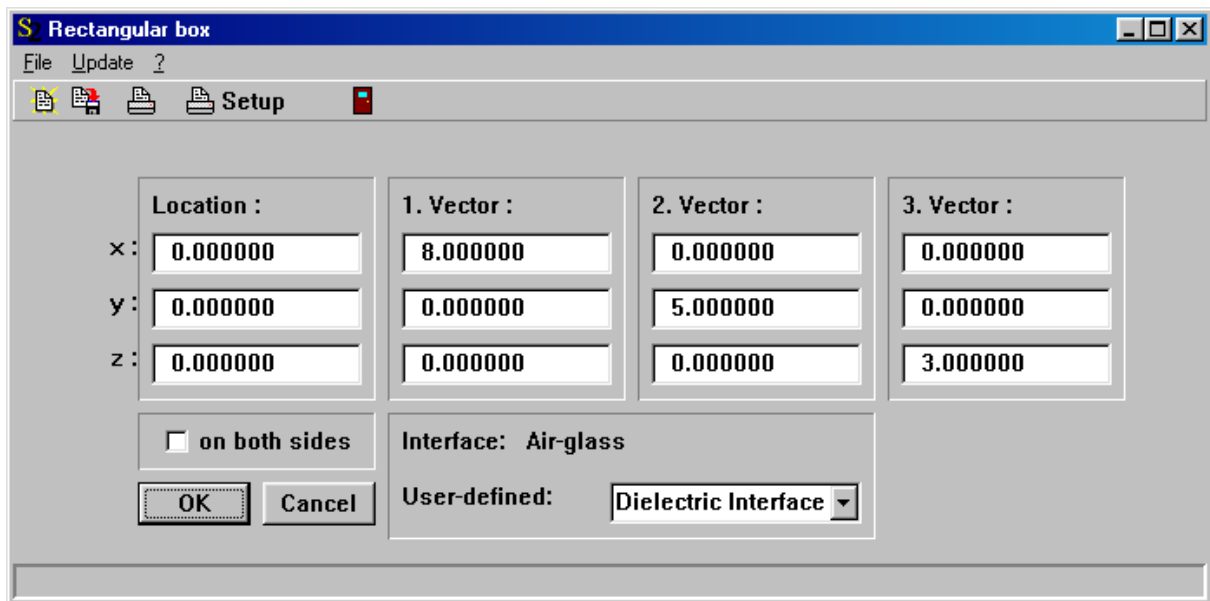
Illuminated by a parallel beam from the left shows the concentrating effect of cones:



5.4.10 Rectangular box

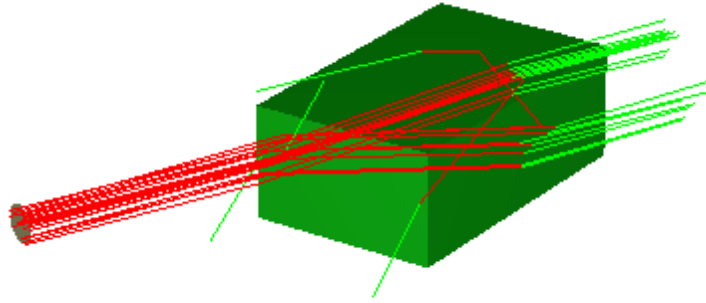


This type of object (select object type **Quader**) is a rectangular box the geometry of which is defined as follows:



The parameters have the same meaning as those explained for volume light sources.

The definition of the interface covering the box is the same as explained for rectangles. All 6 rectangles have the same interface, with the surface normal pointing outward. The picture below shows some rays travelling through a glass brick:



5.4.11 Prism



Prism objects are based on a triangle. The volume of the prism is created by moving the base triangle along a vector called height vector. In most cases the height vector will be perpendicular to the triangle but it need not be. The parameters of prisms are set in dialogs like the following:

Location :		1. Vector :		2. Vector :		Height Vector :	
x :	0.000000		0.500000		-0.500000		0.000000
y :	1.000000		-2.000000		-2.000000		0.000000
z :	0.000000		0.000000		0.000000		1.000000

☐ on both sides

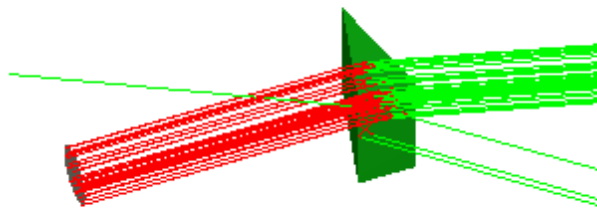
OK Cancel

Interface: Air / glass

User-defined: Dielectric Interface

Location, **Vector 1** and **Vector 2** define the base triangle as describe previously for flat triangles. **Height vector** gives the direction and size of the shift of the triangle. Note that the shift is done in both directions, upwards and downwards. In the example given above the total height of the prism in z-direction is 2 cm.

The definition of the interface covering the prism is the same as explained for rectangles. All 5 prism 'subsurfaces' have the same interface, with the surface normal pointing outward. The picture below shows some rays travelling through a glass prism:



5.4.12 ATR crystal



Recording ATR (Attenuated Total Reflection) spectra is a standard technique in infrared spectroscopy. In order to simulate ATR experiments (which regularly raise questions to be answered by spectroscopic ray-tracing) a very special volume object called 'ATR crystal' has been implemented in SPRAY. The dialog to set the object's parameters is this:

Location :		1. Vector :		2. Vector :		3. Vector :	
x :	0.000000		0.500000		0.000000		0.000000
y :	0.000000		0.000000		0.500000		0.000000
z :	0.330000		0.000000		0.000000		0.200000

Top Surface:	Vacuum / silicon	Angle: 20.0	OK
Bottom Surface:	Vac / absorber / silicon		
Other Surfaces:	Vacuum / silicon		

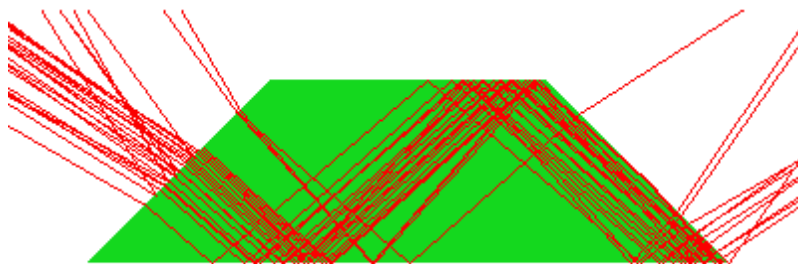
Cancel

The parameter have the following meaning. **Location**, **Vector 1** and **Vector 2** define a base rectangle as described before. Along **Vector 3** (which should be perpendicular to Vector 1 and Vector 2) the rectangle is moved a distance given by the length of Vector 3. During this 'motion' the length of the rectangle (size in direction of Vector 1) is decreased (see below). Finally, the base rectangle is moved in the negative Vector 3 direction increasing its size. This way the shape of the ATR crystal is obtained:

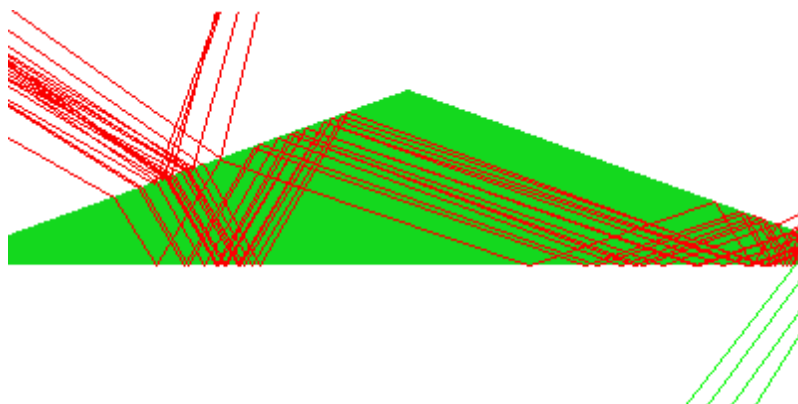
The following sequence of side views (plane of Vector 1 and Vector 3) shows how the decrease/increase of the rectangle is controlled by the parameter **Angle** which sets the angle of the lower left

corner of the ATR crystal. The test rays are send to a silicon prism (refractive index 3.4):

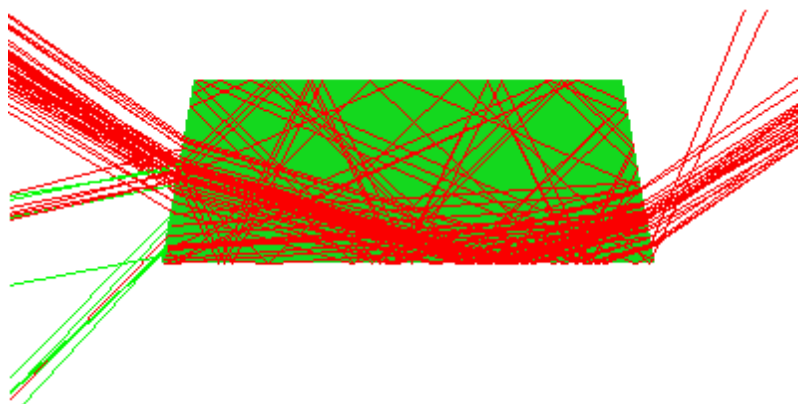
45° :



20°

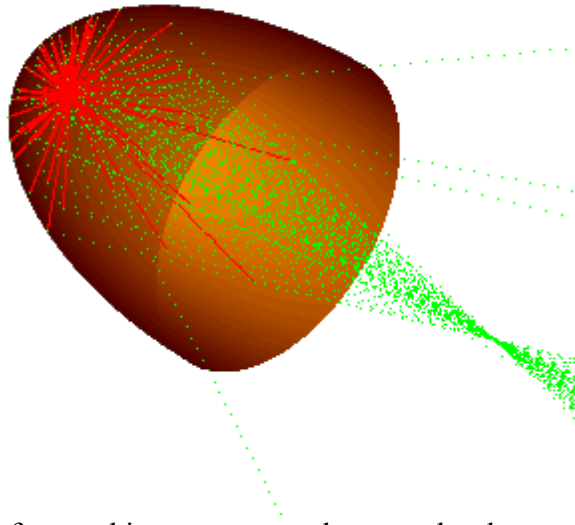


80°



You can assign different interfaces to the top, bottom and side surfaces of the ATR crystal. Only user-defined interfaces (dragged&dropped from the list of interfaces) are allowed.

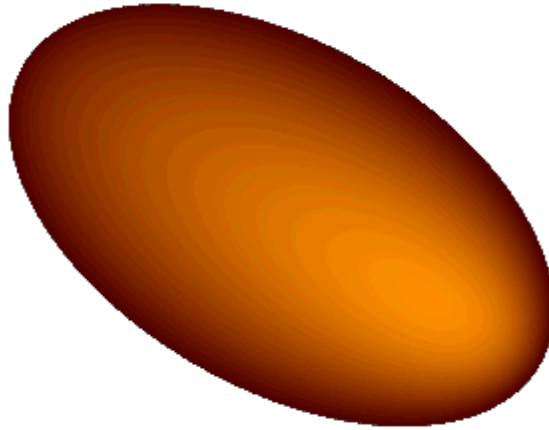
5.4.13 Ellipsoid segment



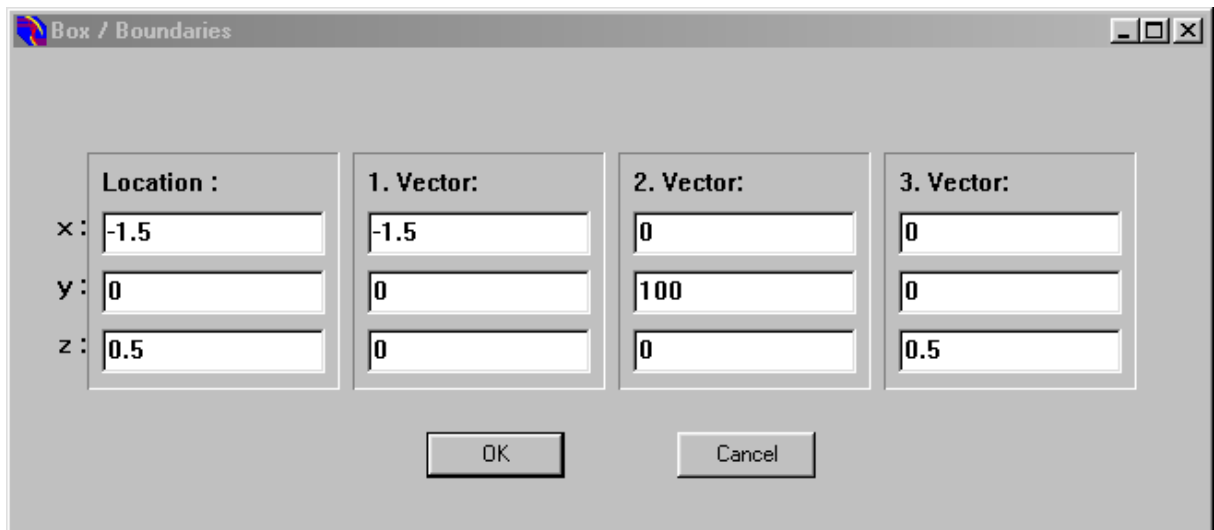
Segments of ellipsoids are often used in spectroscopy because they have two focal points. They are ideal elements for the transportation of radiation from one point in space (first focal point) to another (second focal point).

Ellipsoids in SPRAY are defined in two steps: First the full ellipsoid is defined specifying the center and three vectors (perpendicular to each other) that point from the center into the three principal directions:

This way you describe the full ellipsoid:



With the **Show box data** command you open a second dialog which defines a rectangular box:

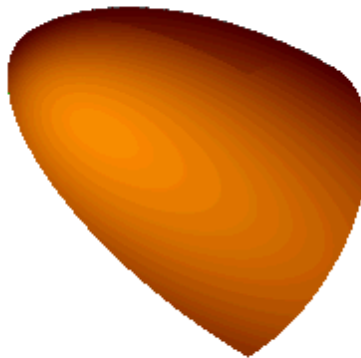


	Location :	1. Vector:	2. Vector:	3. Vector:
x :	-1.5	-1.5	0	0
y :	0	0	100	0
z :	0.5	0	0	0.5

OK Cancel

Location is the center of the box, **Vector 1, 2 and 3** start at the center and point to the centers of three adjacent rectangles, as already shown for volume light sources.

The ellipsoid segment is now defined by those parts of the ellipsoidal surface which are **inside the rectangular box**:



The definition of the interface covering ellipsoids is the same as explained for rectangles.

5.4.14 Paraboloid segment

Parabolic objects are called 'Paraboloid segments' in SPRAY. They are defined in two steps. First a paraboloid surface is defined using four vectors, then a rectangular box is defined with four vectors as well. Only those parts of the paraboloid surface are taken that are inside the box.

The dialog to define the paraboloid is this:

Location :		1. Vector:	2. Vector:	3. Vector:
x :	0.000000	3	0.000000	0.000000
y :	0.000000	0.000000	3.000000	0.000000
z :	0.000000	0.000000	0.000000	1.000000

Interface: none

User-defined:

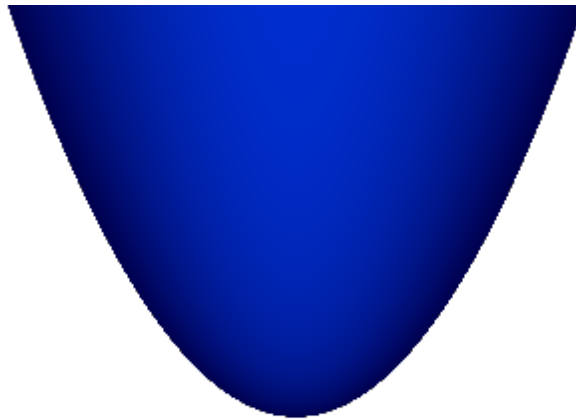
Perfect Mirror

☒ on both sides

OK Cancel Show Box Data

The center of the paraboloid is given by the vector called '**Location**', the **symmetry axis** is defined by **vector 3**. **Vectors 1** and **2** must be perpendicular to vector 3. The length of vector 1 determines the distance you have to go in the direction of vector 1 (starting at the center) in order to reach a height of 1 (measured in the direction of vector 3, with respect to the plane spanned by vector 1 and vector 2). The meaning of vector 2 is the same (only in the direction of vector 2).

The above settings lead to the following paraboloid:



Setting the box parameters one can now cut out the wanted piece of the paraboloid. The dialog is opened by the **Show box data** command:

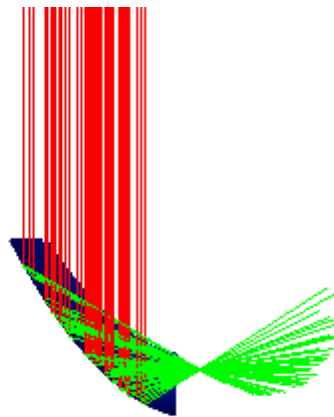
	Location :	1. Vector:	2. Vector:	3. Vector:
x :	-5.000	4.000	0.000	0.000
y :	0.000	0.000	5.000	0.000
z :	4.000	0.000	0.000	4.000

OK Cancel

This cuts out an off-axis paraboloid segment. The new piece is this:

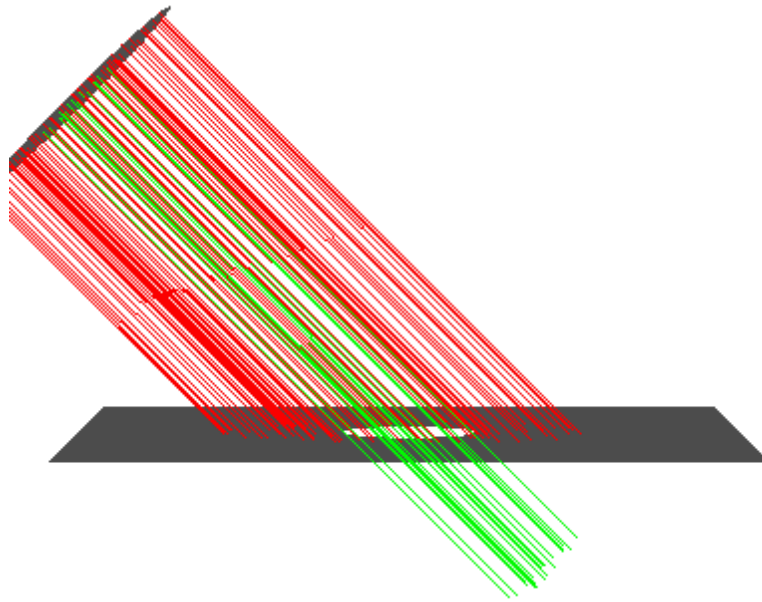


Sending a parallel beam downwards from above the mirror one gets a nice focus (which is the task of parabolic mirrors):



The definition of the interface covering paraboloids is the same as explained for rectangles.

5.4.15 Circular aperture



Objects of this type (select object type **Interface circular aperture**) are rectangles with a circular aperture in the center. The objects' parameters are set in the following dialog:

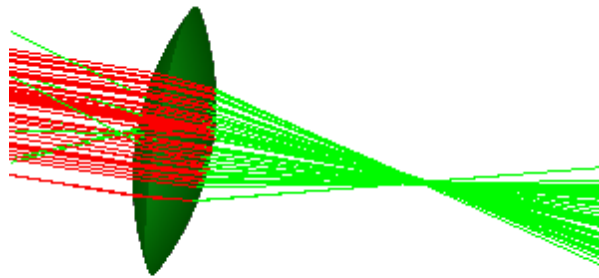
Interface (Rectangular with circular aperture)			
File Update ?			
[Icons] Setup [Color Icon]			
Location :	1. Vector :	2. Vector :	Interface: none
x : 0.000000	5.000000	0.000000	User-defined:
y : 0.000000	0.000000	5.000000	Perfect Absorber
z : 0.000000	0.000000	0.000000	
<input type="checkbox"/> on both sides	Radius of Aperture:	1.000000	OK Cancel

In addition to the parameters of 'normal' rectangles (see above) there is the **Radius of Aperture**, which is the radius of the circular aperture, of course.

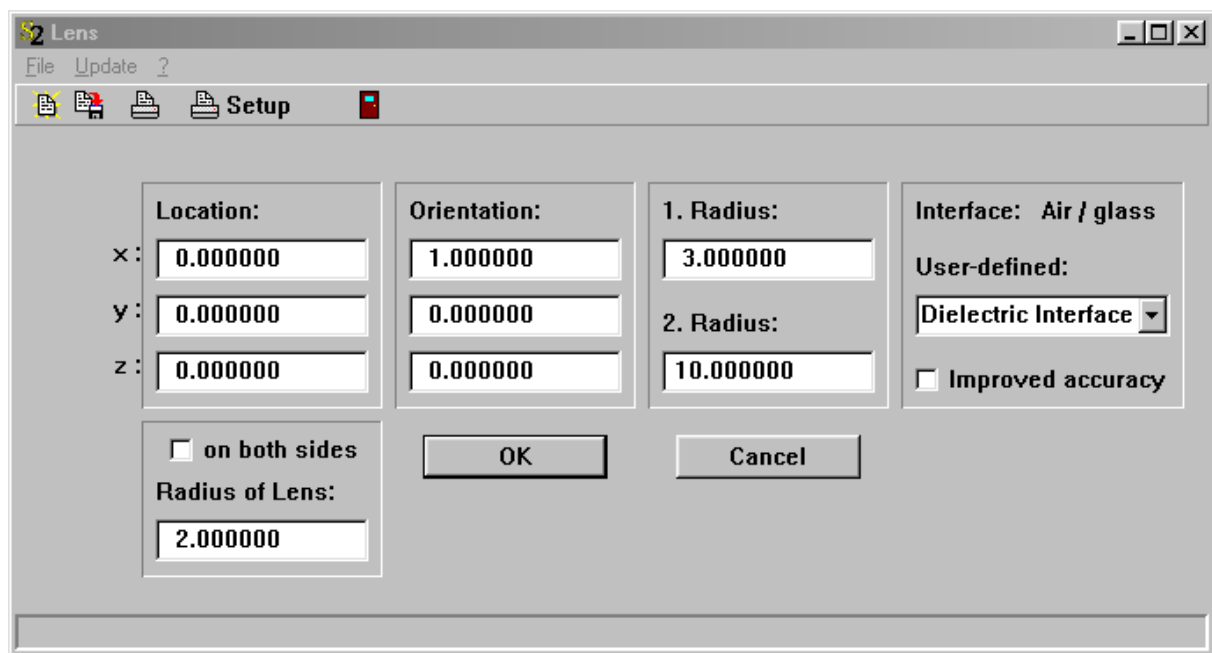
The definition of the interface covering the surface is the same as explained for rectangles.

5.4.16 Converging lens

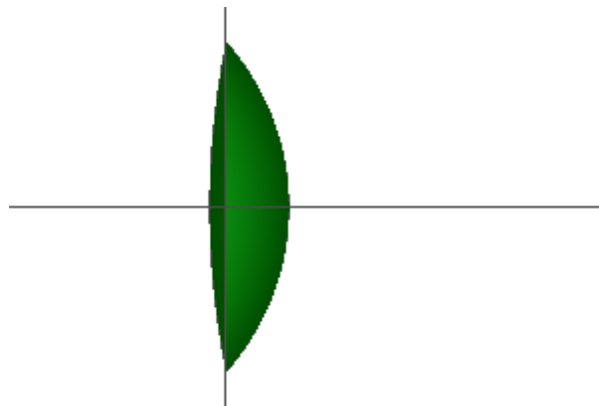
These objects are composed of two sphere segments. Assigning an appropriate layer stack to the interface (e.g. an air-glass transition) you can define a lens:



The dialog to set the object's parameters is this:



The parameters of the dialog are explained using the following side view of a converging lens:



The horizontal line is the symmetry axis of the lens. The centers of both sphere segments are on this line. You define this direction by the **Orientation** vector.

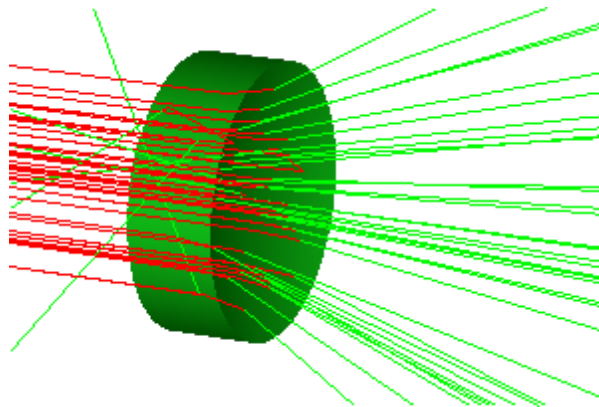
The vertical line represents the plane where the two sphere segments are 'glued' together. **Radius 1** gives the radius of the sphere segment shown on the right side. The center of this sphere is on the left. **Radius 2** is the radius of the sphere segment on the left. The parameter called **Radius of the lens** is the radius of the circle common to both sphere segments: This number determines the size of the lens.

Location gives the intersection of the horizontal and vertical line, i.e. it lies on the symmetry axis and in the plane where the two sphere segments are put together.

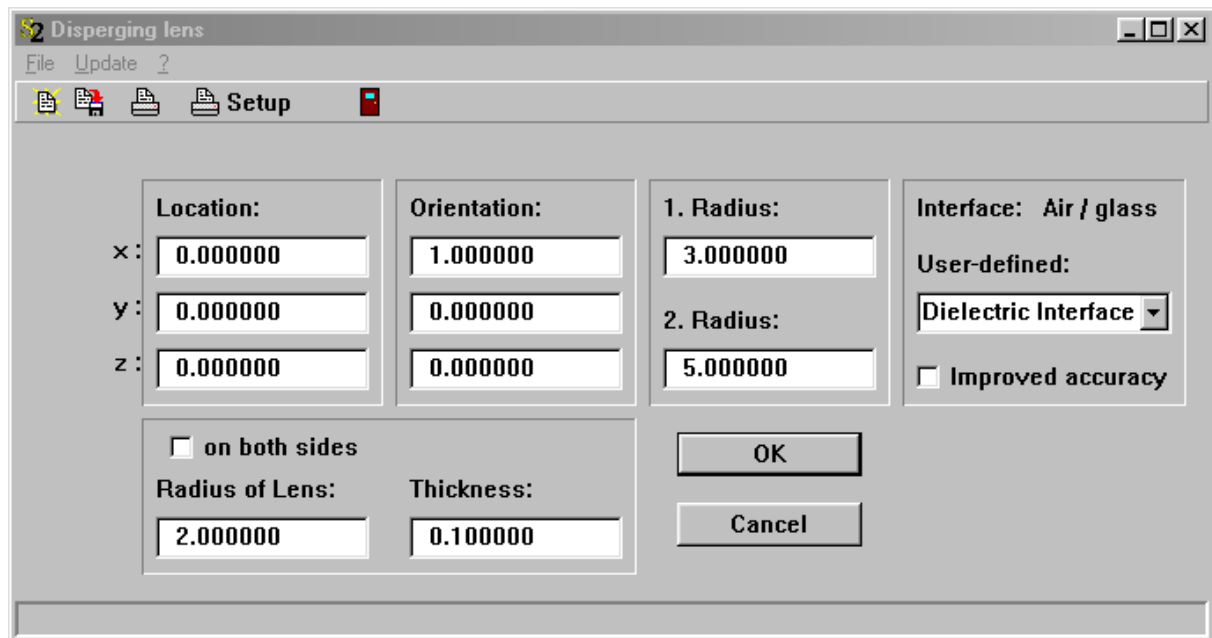
Note that **Radius 1** and **Radius 2** must always be larger than **Radius of the lens**.

The definition of the interface covering lenses is the same as explained for rectangles.

5.4.17 Diverging lens

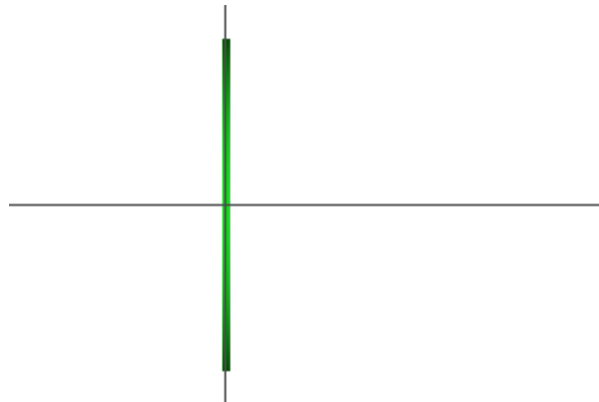


Diverging lenses are - similar to converging lenses - composed of two sphere segments. In addition, there is a cylindrical 'envelope'. The parameters are set in the following dialog:

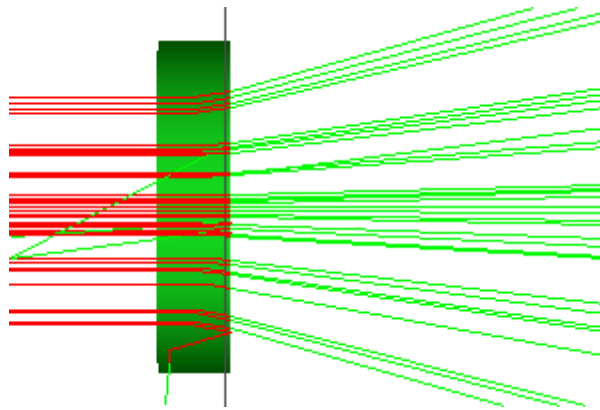


The meaning of the parameters is explained using the following construction of the final lens:

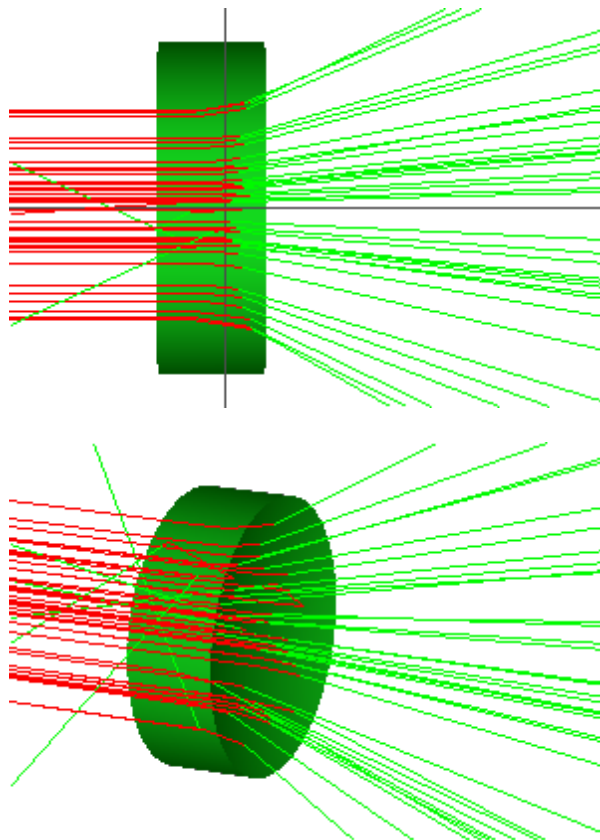
1.
Start with a cylinder. It has a radius given by **Radius of lens** and a thickness given by **Thickness**. The cylinder is centered around the vector **Location**. Its symmetry axis is given by the direction **Orientation**:



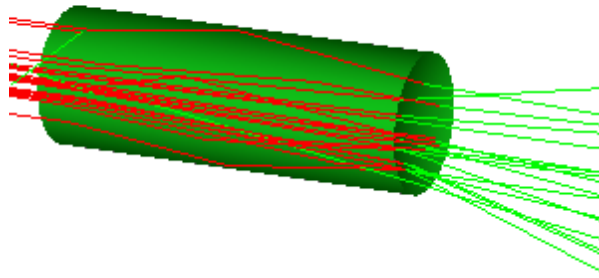
2.
Now add on the left side a sphere segment. The center of the sphere is on the symmetry axis of the cylinder, the radius is given by **Radius 1**. The distance of the sphere center to the left side of the cylinder is given by **Radius 1** as well - the sphere segment just touches the cylinder:



3.
Finally add the sphere segment on the right:



Note that **Radius 1** and **Radius 2** must always be larger than **Radius of lens**.
If you choose a large value of thickness the lens becomes an optical fiber:



All three surfaces (the two sphere segments as well as the cylinder) are covered with the same interface. The definition of that interface is the same as explained for rectangles.

5.4.18 User-defined surface: Rectangular basis

Objects of this type (select object type **Interface curved I**) implement almost arbitrary surfaces - created by a two-dimensional function defined over a rectangular basis. The dialog is this:

Curved Interface (rectangular)

File Update ?

Icons: [File] [Print] [Setup] [Red Flag]

Location :	1. Vector :	2. Vector :	Interface: Air / glass
x : 0.000000	1.000000	0.000000	User-defined: Dielectric Interface ▼
y : 0.000000	0.000000	1.000000	
z : 0.000000	0.000000	0.000000	

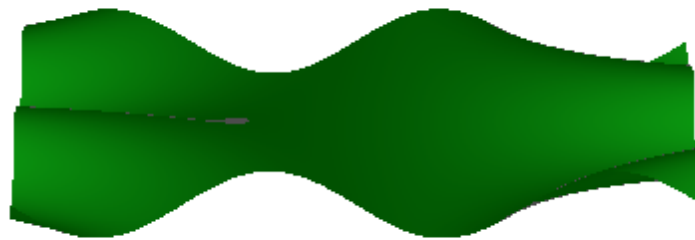
☐ on both sides

Function: 0.1*SIN[X*2*PI*Y]

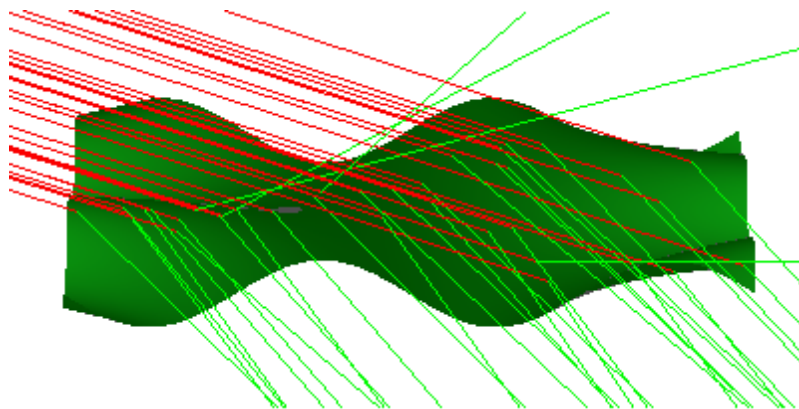
OK Cancel

The parameters are the same as already explained for the rectangle. In addition, there is a user-defined **Function** that creates the surface shape. As variables you can use X and Y: X represents the distance from **Location** (the center of the rectangle) in direction of **Vector 1**, Y the distance in direction of **Vector 2**. As all geometrical distances, X and Y are measured in cm. The terms you can use in a formula are described in the *Data Factory* manual which is distributed with SPRAY.

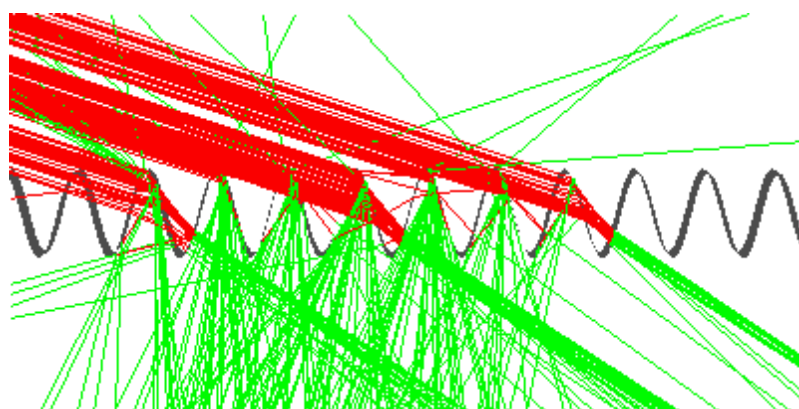
The parameters shown above lead to the following surface shape:



Sending some test rays (air - glass interface):



Here is another example, showing a side view of the sinusoidal surface profile defined by $0.3 * \sin(X * 4 * \pi)$. Again, the interface is between air and glass, and the illumination is from the upper left:



The definition of that interface covering the user-defined surface is the same as explained for rectangles.

Access by OLE automation

OLE automation controllers can modify an object of type 'Interface Curved I' named 'MyName' in

the SPRAY object list by the following OLE commands:

```
object_parameter("MyName", "x"): read/write the x-coordinate of Location
object_parameter("MyName", "y"): read/write the y-coordinate of Location
object_parameter("MyName", "z"): read/write the z-coordinate of Location
```

```
object_parameter("MyName", "x1"): read/write the x-coordinate of vector 1
object_parameter("MyName", "y1"): read/write the y-coordinate of vector 1
object_parameter("MyName", "z1"): read/write the z-coordinate of vector 1
```

```
object_parameter("MyName", "x2"): read/write the x-coordinate of vector 2
object_parameter("MyName", "y2"): read/write the y-coordinate of vector 2
object_parameter("MyName", "z2"): read/write the z-coordinate of vector 2
```

```
object_parameter_string("MyName", "surface_formula"): read/write the function definition as string
```

5.4.19 User-defined surface: Circular basis

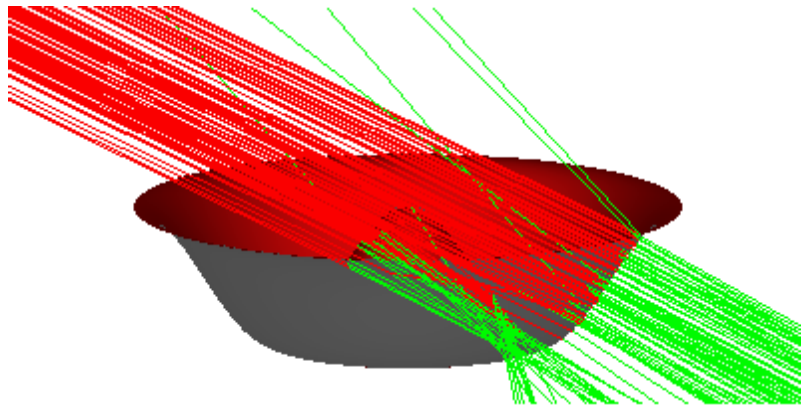
Objects of this type (select object type **Interface curved II**) are very similar to user-defined interfaces on a rectangular basis that were discussed before. The difference is the shape of the basis which is a circle instead of a rectangle. The dialog to set the objects' parameters is this:

Location defines the center of the circle over which the surface profile is computed. **Normal Vector** is the surface normal, and **Radius** the radius of the circle.

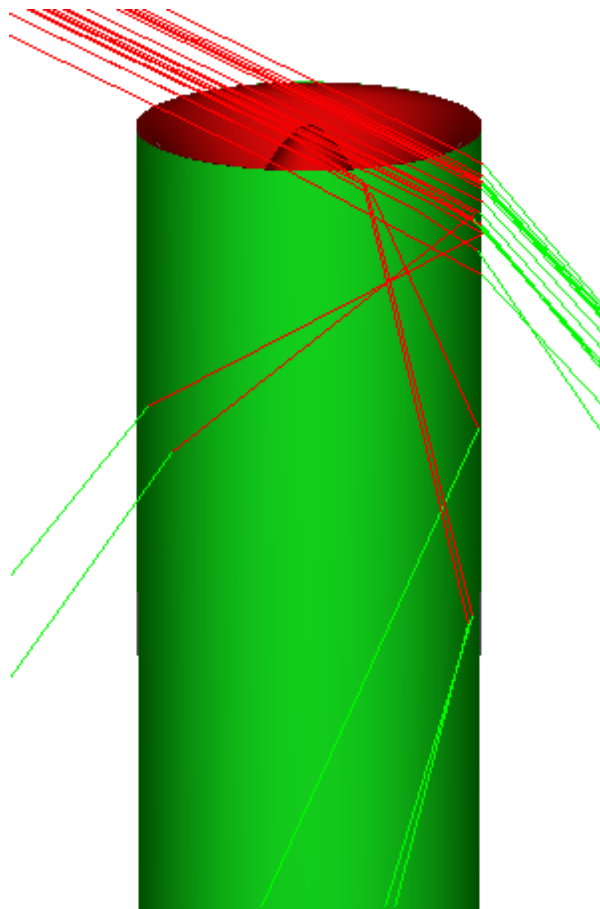
The **Function** describes how the surface profile, i.e. the height of the curve, depends on the distance from the center. This distance must be called 'R' in the formula. The example leads to the following surface:



Some test rays (air-glass interface) show the consequences:



As the following example shows, user-defined interfaces with circular basis can be combined with open cylinders to study various shapes of fiber heads:



The definition of that interface covering the user-defined surface is the same as explained for rectangles.

5.4.20 Periodic surface texture

This object type allows easy modeling of surfaces with periodic macroscopic texture. Typically it is used for glass or plastic surfaces.

The basic shape is a rectangle which is divided into many small rectangles. Each of the small rectangles has the same surface topology. You can select from several pre-defined textures. The object dialog looks like this:

Location: x : 0.00000000 y : 0.00000000 z : 0.00000000			Vector 1: 7.80000019 0.00000000 0.00000000			Vector 2: 0.00000000 7.80000019 0.00000000			Cancel OK
--	--	--	--	--	--	--	--	--	------------------

☐ on both sides
 Interface type:
 User-defined

Interface: Air-glass

Texture pattern: Inverted pyramid

Periodicity x-direction: 0.100000 **Periodicity y-direction:** 0.100000

Texture pattern amplitude: 0.05

—
 —
 —

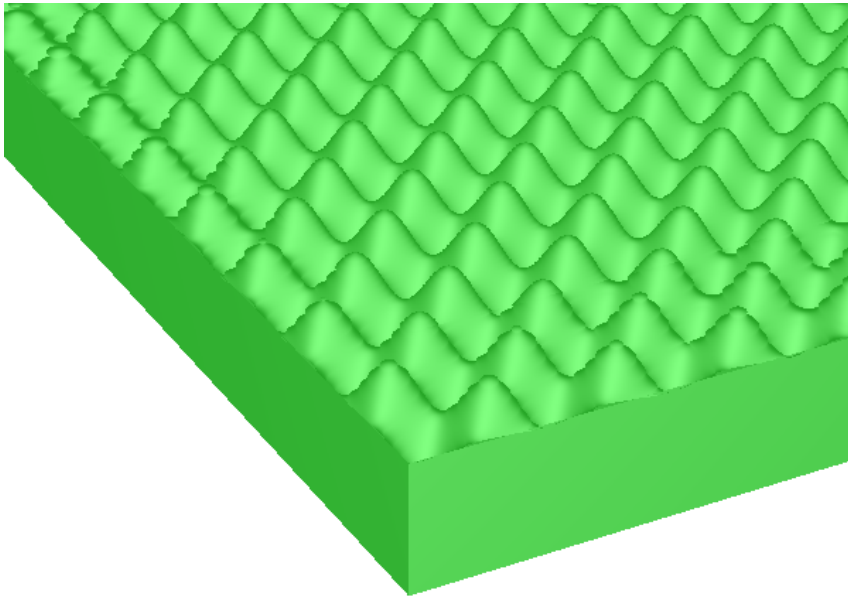
The top editors are used to define the location and size of the base rectangle. The definition of the interface is the same as for all other interface objects.

The second half of the dialog allows the user to define the type of texture in the dropdown box as well as the texture period in x- and y-direction. Both quantities are entered in cm.

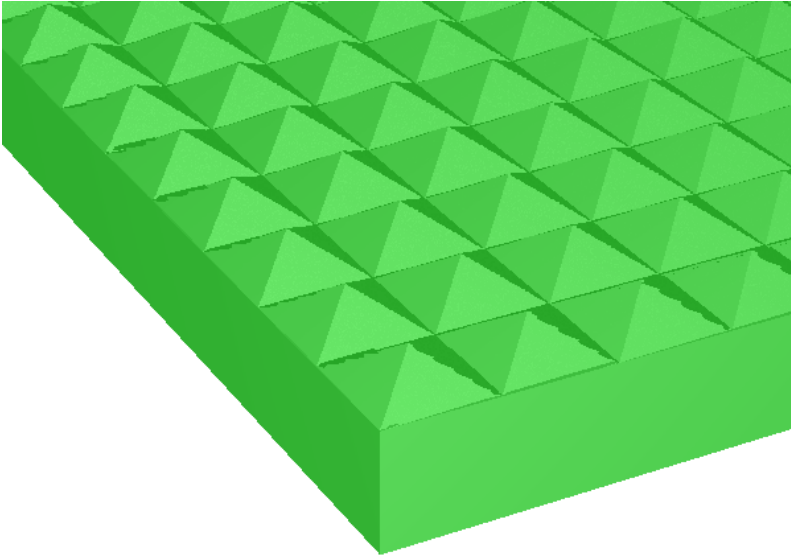
The amplitude of the texture (also defined in cm) determines the difference between the highest and the lowest point of the surface shape. Depending on the selection of the texture type there are more parameters shown at the bottom of the dialog.

The following texture types are available:

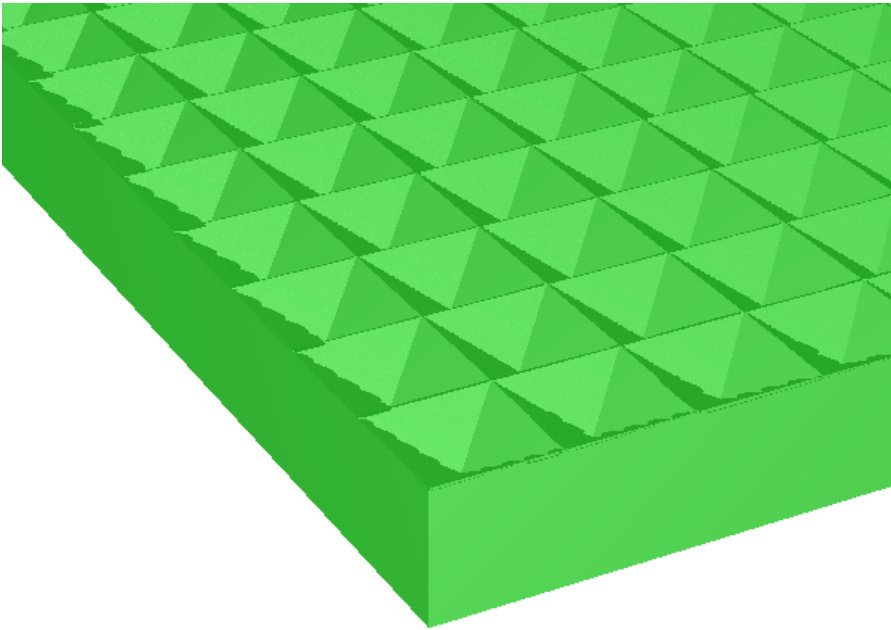
Product of sine profiles in x- and y-direction:



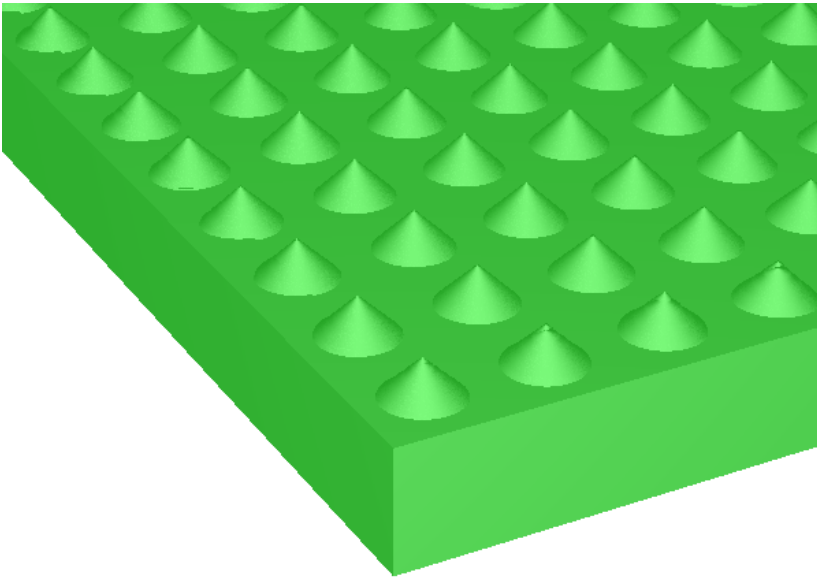
Sine profiles in x-direction (or in y-direction)
Pyramids:



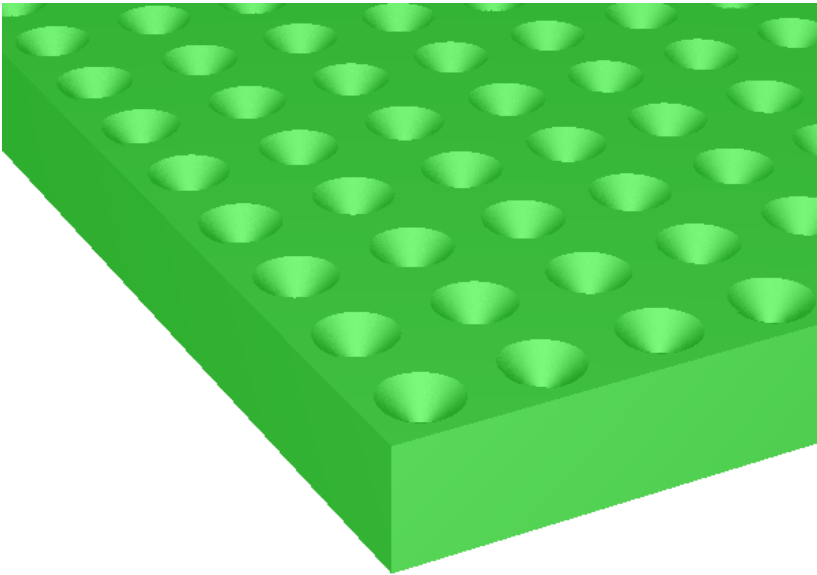
Inverted pyramids:

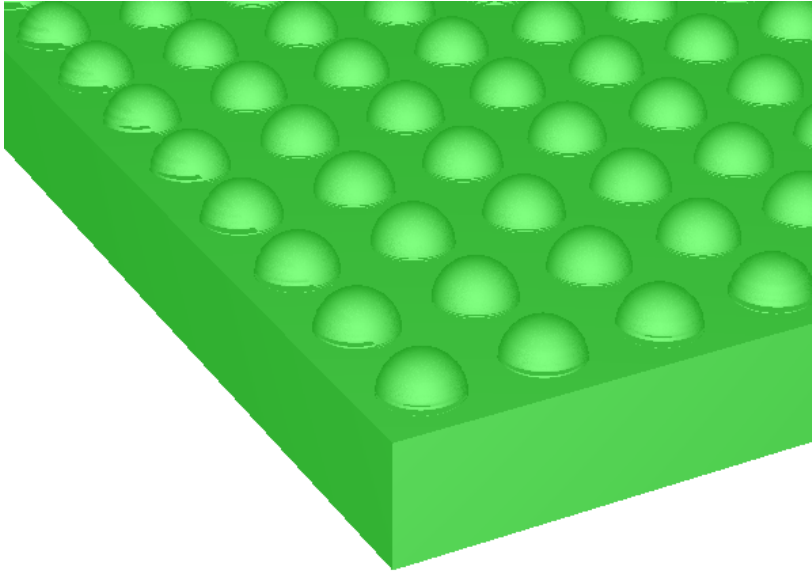
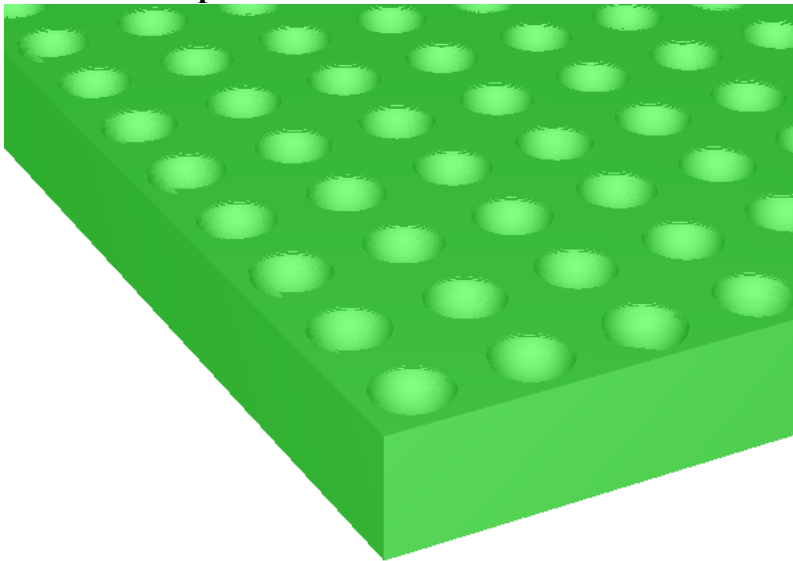


Cones:



Inverted cones:



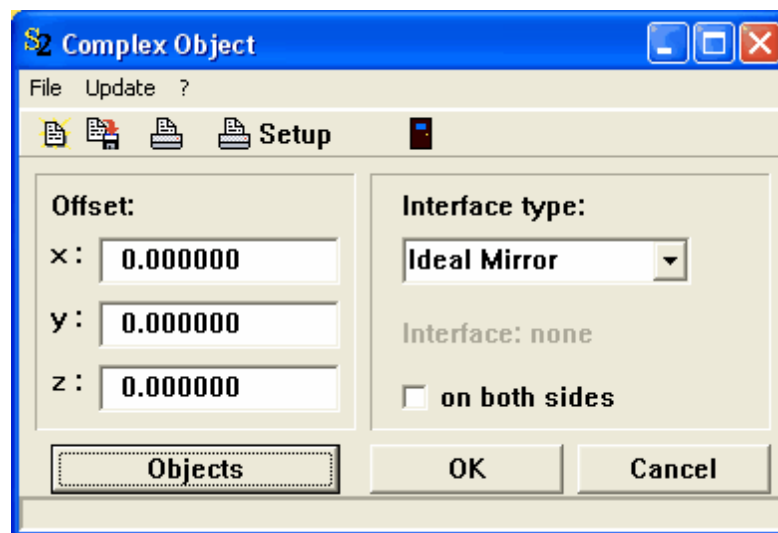
Half spheres:**Inverted half spheres:**

5.4.21 Complex objects

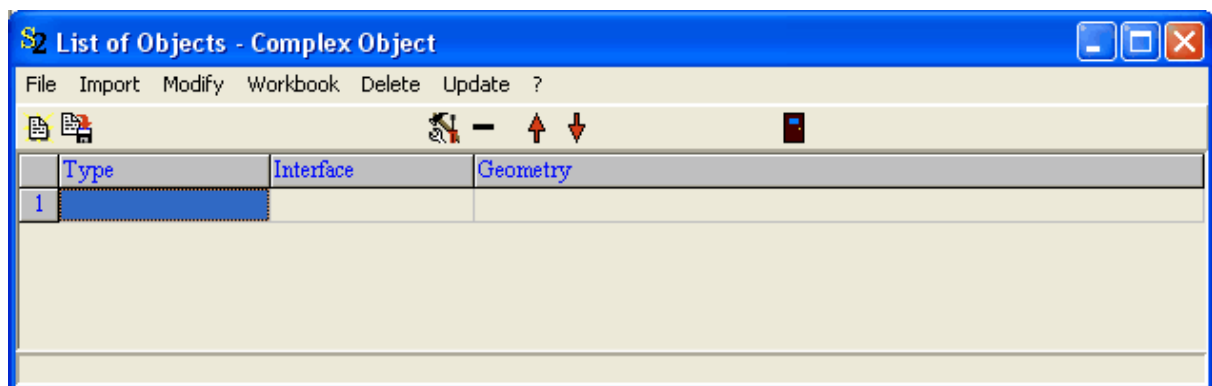
5.4.21.1 Complex objects: Introduction

Objects of this type are composed of several subobjects of simple shapes (rectangles, triangles, circles, ...). It is recommended to use this object type if you have to work with many interface objects of the same shape or if you want to group objects for easier manipulation.

If you edit complex objects the following window opens:



When subobjects are created (see below) they are shifted from their original position by the vector specified here as '**Offset**'. All newly created objects are automatically covered with the interface defined in this dialog. You can change the interface assignment later, if you like (see below). The subobjects are handled by a list which is opened clicking the **Objects** button:



Subobject creation

Subobjects are created by importing geometry information from text files of workbook sections. The structure of the text files and corresponding workbook sections is identical. It is a simple sequence of lines where each line holds the information of one object to be created. If text files are used the individual items in a line must be separated by tab stops. In workbook sections the individual items are stored in corresponding cells, of course. Every line has to start with the specification of the object type that is to be created. Then the required coordinates follow in a sequence which is described below. The recommended extension for the text files is *.ol (for object list). Here is an example of a text file defining a sphere and a rectangle:

```
sphere 5      0      0      3
rectangle vectors      -2      0      0      1      0      0      0      0      1
```

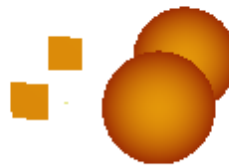
Use the **Import** command to load object information from text files. You are then asked to select a file. Make sure you specify the file type '**Object list**' which is the first import filter. After the file is processed the object list looks like this:



In a view, these objects look like this:



If you set the y-coordinate of the **Offset** vector (see above) to 10 cm and import the object information from the text file again, you obtain a shifted copy of the objects:



Besides reading subobject information from text files you can read from the workbook. Open the workbook with the **Workbook|Show** command and select the row from which you want to start to import subobjects. In this case, it's cell A1:

	A	B	C	D	E	F	G	H	I	J	K
1	sphere	5	0	-5	3						
2	rectangle vectors	-2	0	-5	1	0	0	0	0	1	
3	sphere	5	10	-5	3						
4	rectangle vectors	-2	10	-5	1	0	0	0	0	1	
5											
6											
7											

Now apply **Workbook|Import**. The four new objects are added to the subobject list.

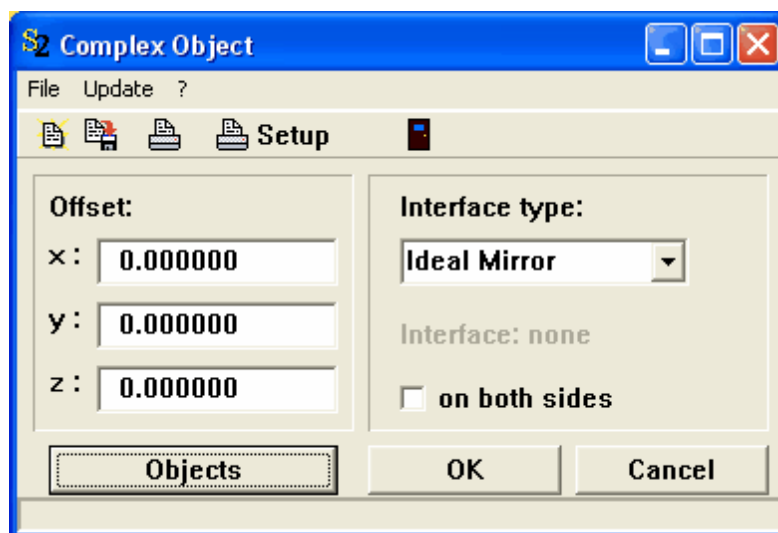
Remove subobjects

Like in any other SPRAY list, you can delete subobjects by selecting their rows and clicking the '-' button. Alternatively you can execute the menu command **Delete|Delete selected**. The command **Delete|Delete all** obviously deletes all objects.

Modify interface assignment

Once you have created a list of subobjects you can change the interface assignment in one step for all subobjects, or individually.

A global change is done this way: Open the complex object's window



and select the wanted interface. Then go to the subobject list and execute the command **Modify|Assign interface to all objects**. This will just do what it says.

To do individual assignments, first select the subobject that you want to assign a different interface to. Pressing the function key **F6** (or **F7**) you can toggle between 'Ideal mirror', 'Ideal absorber' or a user-defined interface. If a user-defined interface is selected, you can iterate through all user-defined interfaces by the **F4** and **F5** functions keys (forward or backward, respectively).

The following section gives details about the implemented types of subobjects that can be used in complex objects.

Warning: Using complex objects you can easily add many objects to the SPRAY configuration. This can dramatically increase the computational time for SPRAY spectra.

5.4.21.2 Complex objects: Subobject types

These subobject types are available:

Rectangle

These objects can be defined in two ways:

1.

Define the center and the two perpendicular vectors that span the rectangle (as described in the section about rectangular interfaces). In this case the definition line has to start with the keyword 'Rectangle vectors'. Then the location and the two directional vectors follow, as in the following example:

```
rectangle vectors    -2    0    -5    1    0    0    0    0    1
```

Note that the coordinates must be separated by tab stops if they are stored in text files.

2.

Sometimes it is more appropriate to specify the four corners of a rectangle. This can be done using the keyword 'Rectangle points' which must be followed by the coordinates of the four corners. In order to get the correct surface normal you can work with the following trick:

- Imagine you sit in the center of the rectangle
- Move up a little along the wanted surface normal
- Turn around and look back on the rectangle below you
- Start with one of the corners as first point and then add the others counterclockwise

An example of this data format is this:

```
rectangle points    0    0    0    2    0    0    2    5    0    0
                   5    0
```

Triangle

Here you have two possibilities as well, using spanning vectors or end points:

1.

As described in the section about triangular interfaces, the triangle is defined with a vector to one of the corners as a reference point and the two vectors from that point to the remaining corners. The surface normal is given by the cross product of the two vectors. Here is an example:

```
triangle vectors    0    0   -5    1    0    0    0    0    1
```

2.

You can also write down the three corners, using the same orientation rule as explained for rectangles (see above). An example is given here:

```
triangle points    0    0    0    2    0    0    2    5    0
```

Circle

Circles are defined by their center, their surface normal and the radius:

```
circle -3    0    0    0    0    1    0.34
```

Sphere

Spheres are simple defined by the center coordinates followed by the radius:

```
sphere -3    0    0    0.5
```

Cylinder

This object type is a closed cylinder, defined the usual way by specifying the center, the radius vector and the axis vector:

```
cylinder    20    0    0    0    0    1    4    0    0
```

Ellipsoid

This subobject type defines full ellipsoids, specified by the center and the three principal directions:

```
ellipsoid      -10   0   0   0   0   2   0   2   0   0   0
                3
```

Cone

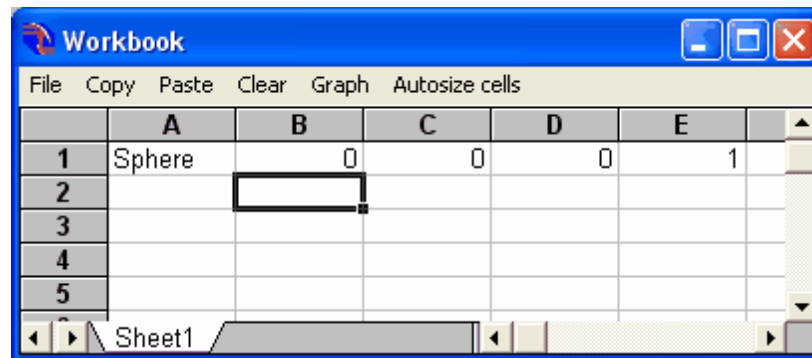
This subobject type defines a cone, specified by the two end points and the corresponding radii of the circles at each end:

```
cone          -5   0   0   5   0   0   2   0.5
```

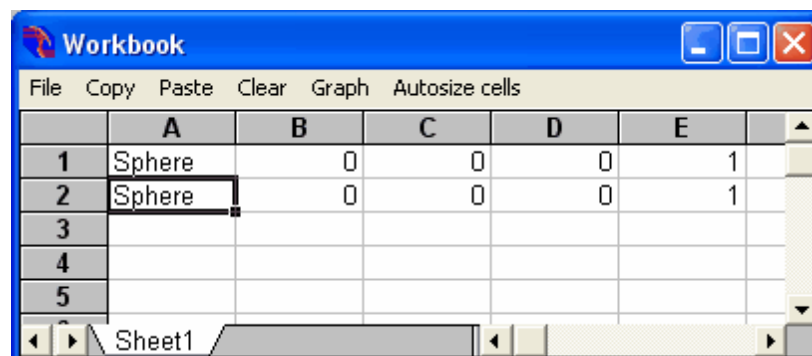
5.4.21.3 Complex objects: Creating input data

Workbook

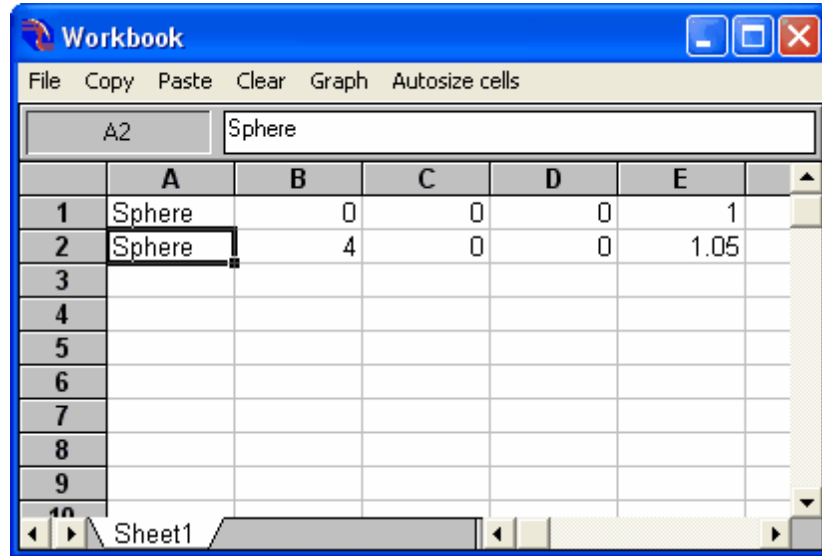
The easiest way to create input data is to use the workbook and its built-in functions. If you, for example, want to create a sequence of spheres with increasing radius, start with the first sphere in the first row by typing in its coordinates explicitly:



Now select the cells that define the first sphere and use the **Copy** command to copy them to the clipboard. Move the cursor into cell A2 and execute **Paste**. Now you have a copy of the first sphere:

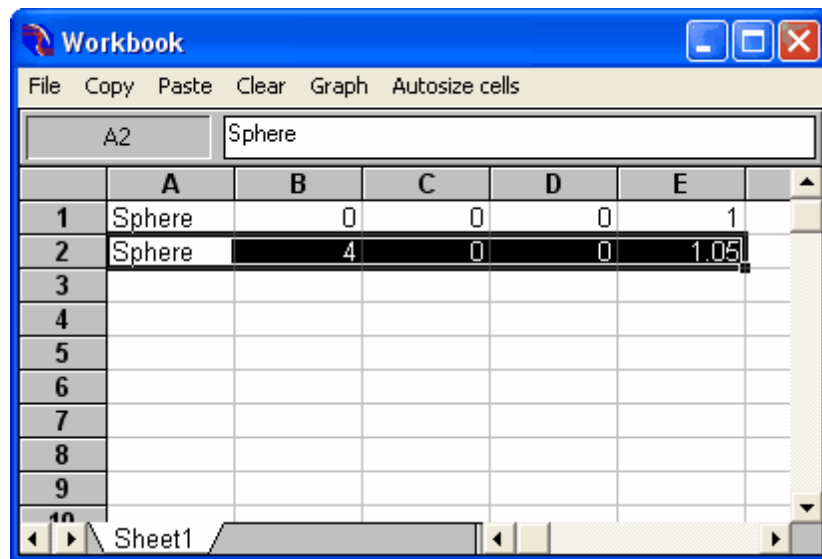


Replace the content of cell B2 with the formula `'=B1+4'` and that of cell E2 with `'=E1*1.05'`. Now the workbook should look like this:



	A	B	C	D	E
1	Sphere	0	0	0	1
2	Sphere	4	0	0	1.05
3					
4					
5					
6					
7					
8					
9					
10					

The second sphere is shifted along the x-axis by 4 cm and its size is increased by 5%.
Now select the workbook range A2:E2:



	A	B	C	D	E
1	Sphere	0	0	0	1
2	Sphere	4	0	0	1.05
3					
4					
5					
6					
7					
8					
9					
10					

You can now create many copies of the selected cells by dragging down the small black square at the bottom right corner of cell E2. The data and formulas are copied downwards until you release the mouse button. The workbook now could look like this:

	A	B	C	D	E
1	Sphere	0	0	0	1
2	Sphere	4	0	0	1.05
3	Sphere	8	0	0	1.1025
4	Sphere	12	0	0	1.157625
5	Sphere	16	0	0	1.2155063
6	Sphere	20	0	0	1.2762816
7	Sphere	24	0	0	1.3400956
8	Sphere	28	0	0	1.4071004
9					

Now you can import the new objects into the complex objects the way described above. Here is the result:



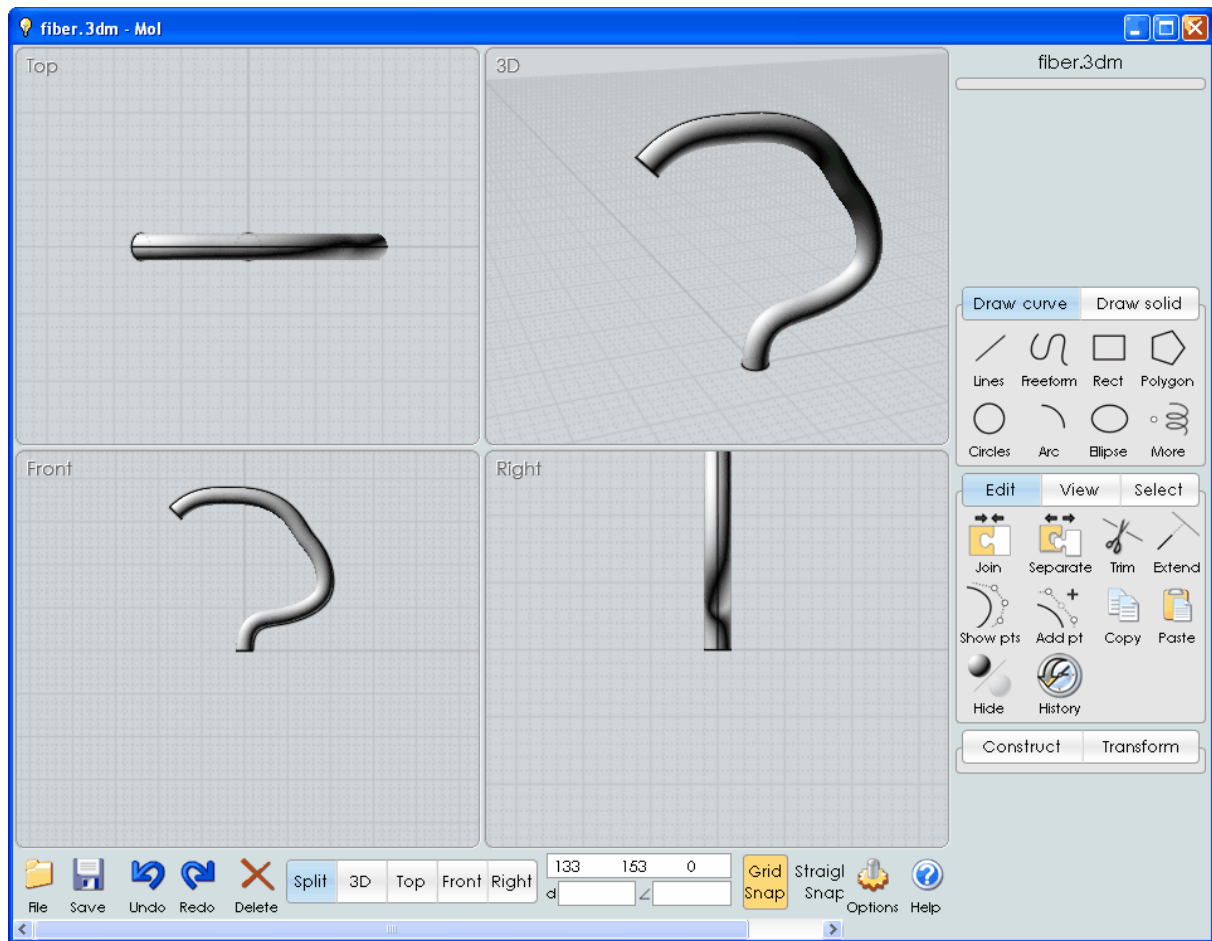
External programs

In many cases it is advantageous to use a programming language to compute the geometric data for complex objects. You could use, for example, Excel's VisualBasic to compute the required input tables for SPRAY. If you have to do this only a few times, you can copy the tables manually to the clipboard and paste them into the SPRAY workbook.

If you have to copy the computed data many times into SPRAY you should create text files with the input data and import them into the complex object of SPRAY by OLE automation. The necessary OLE automation commands will be implemented soon.

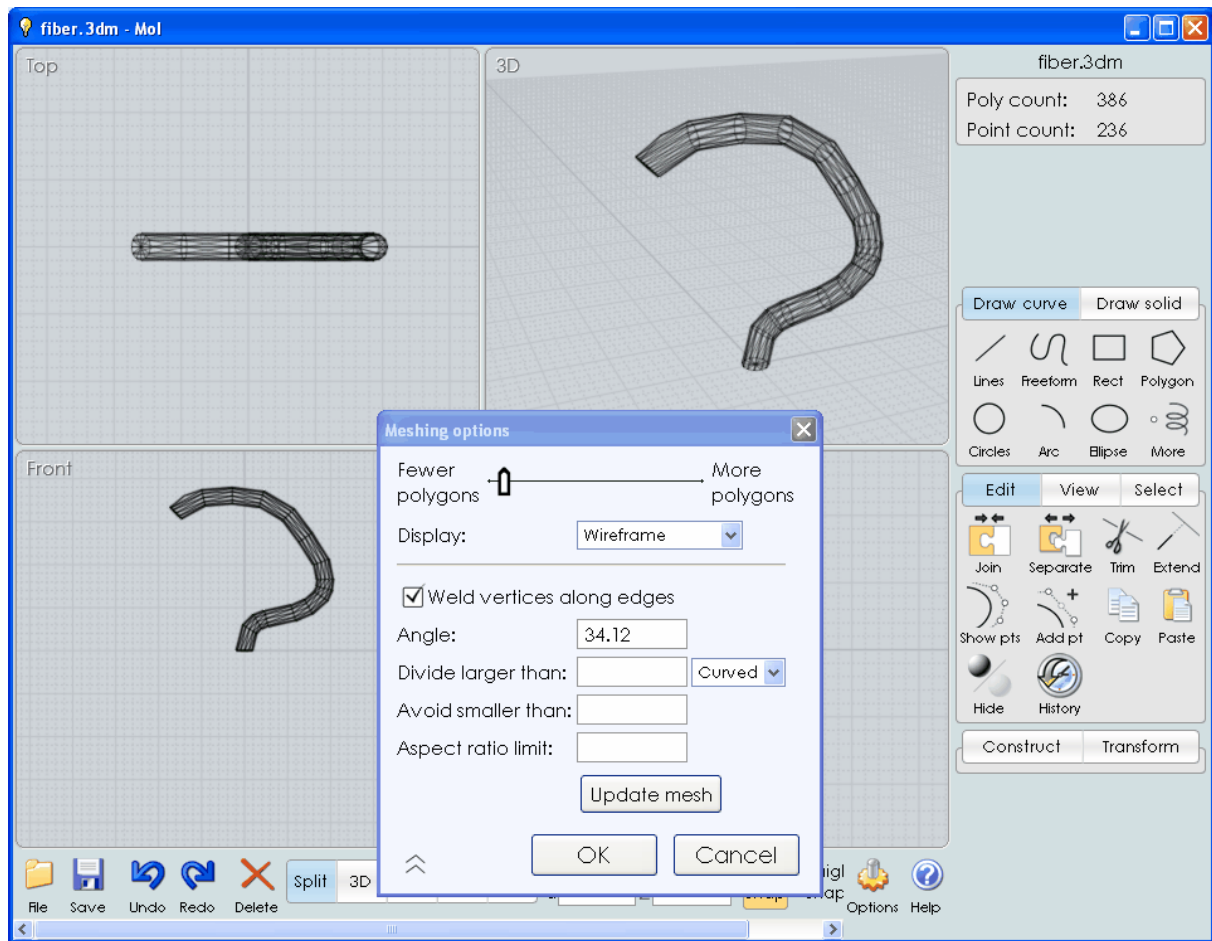
5.4.21.4 Importing objects from CAD programs

You can import objects from CAD systems into SPRAY if your construction software can generate a mesh of triangles and export the coordinates using the STL format. Here is a screenshot of the Mol software used to generate a bended fiber:



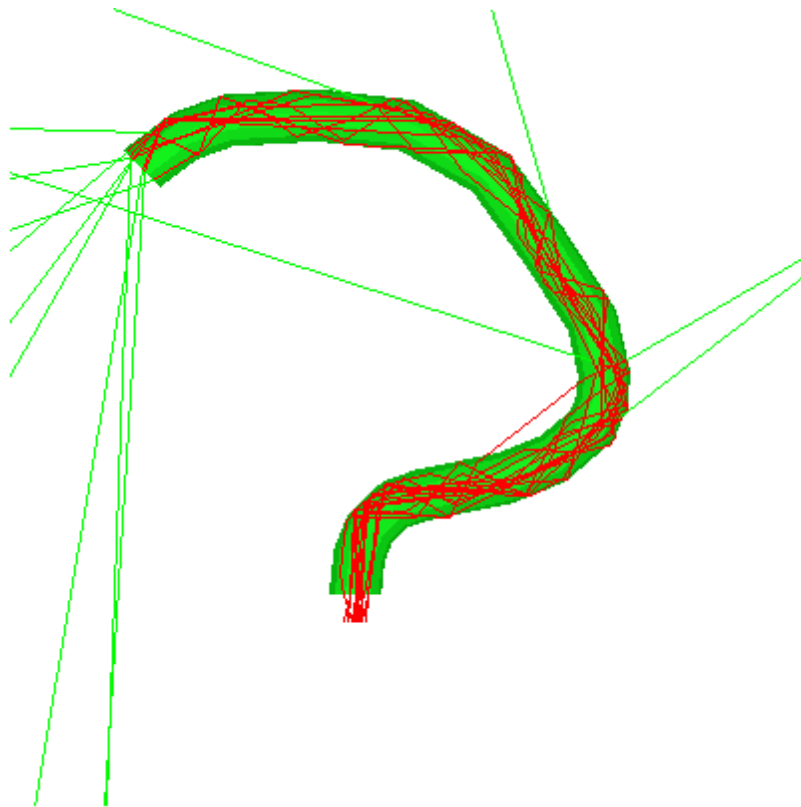
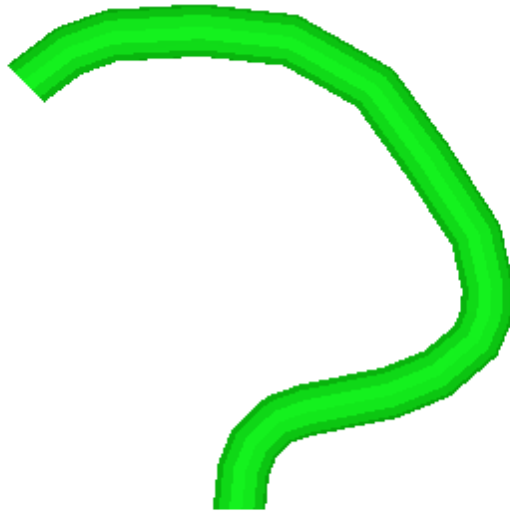
When saving the fiber as a mesh of triangles be sure that the number of triangles is not too high - otherwise SPRAY computations will take a very long time, and camera views will be extremely slow!

So please try to find a reasonable compromise of quality and time efficiency.



Once you have generated a STL file, you can import the data into SPRAY objects of type 'Complex object' using the Import command in its object list and selecting one of the STL formats (binary or Ascii). The triangle data are expected to give dimensions in mm. SPRAY takes the three nodes of each triangle in the sequence given by the STL file, and computes the surface normal according to the right hand rule - it ignores the surface normal specified in the STL data.

The following graphs show the behaviour of a glass fiber import from the MoI software illuminated from the bottom:



5.5 Special objects

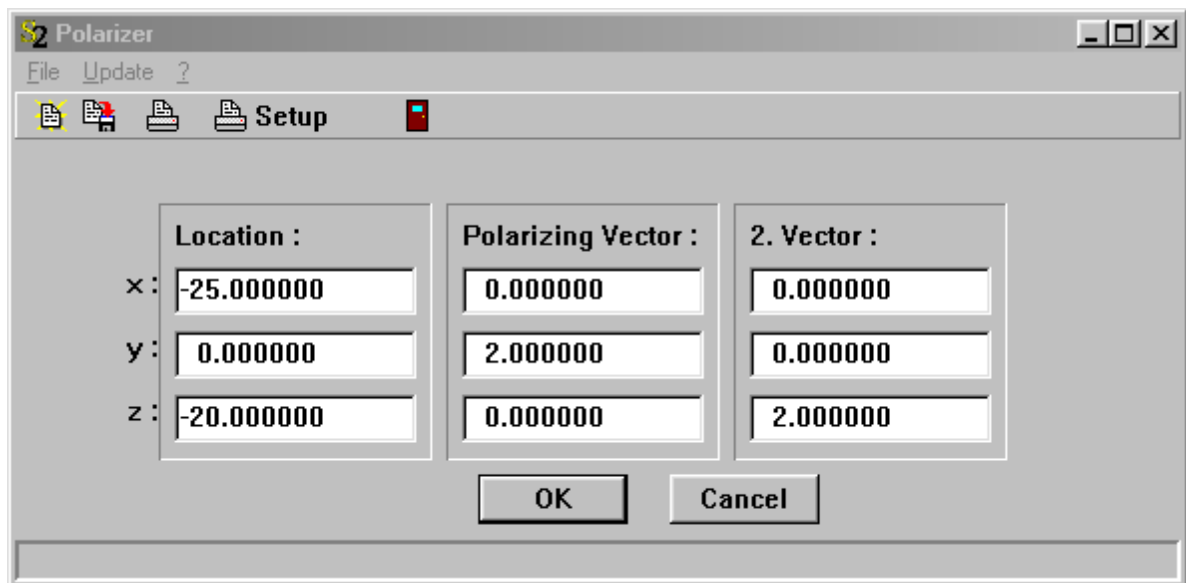
5.5.1 Overview

The following special objects are implemented in SPRAY at present:

- Polarizer

5.5.2 Polarizer

Polarizers are rectangles which are defined this way:



The meaning of the parameters is the same as explained previously for rectangular light sources (where the term Vector 1 is used instead of Polarizing vector).

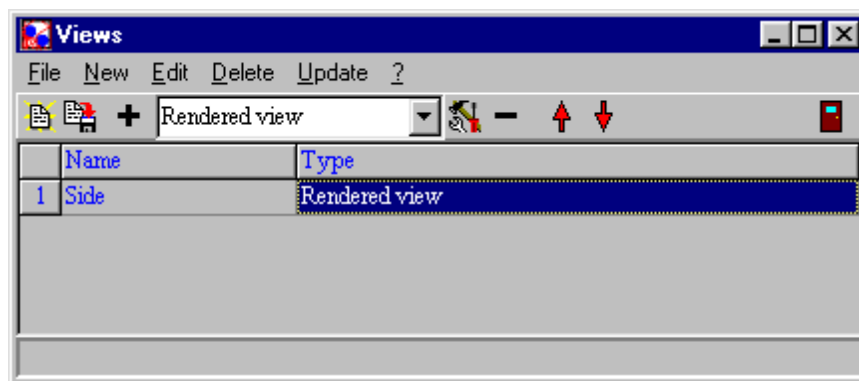
In ray-tracing simulations polarizers act as ideal polarizing rectangles: If a ray hits the rectangle the projection of its polarization vector into the direction of the **Polarizing vector** is computed. The absolute value of the projection is a number between 0 and 1. Then a random number between 0 and 1 is taken and compared to the projection. If the random number is larger the ray is absorbed. Otherwise the ray is transmitted with a polarization vector in the direction of the Polarizing vector.

6 Cameras

6.1 Overview

Cameras take pictures of the present geometric scenery in SPRAY. They are used to check the setup and to visualize its performance watching the path of some test rays. Of course, cameras are also valuable for the documentation of SPRAY results.

Cameras are managed by a list which you open clicking the **Cameras** button in the main window of SPRAY:

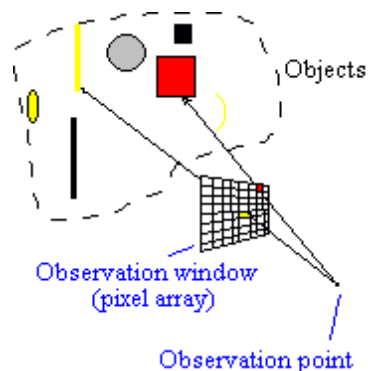


You can create as many cameras as you like. At present, the following types of cameras are implemented:

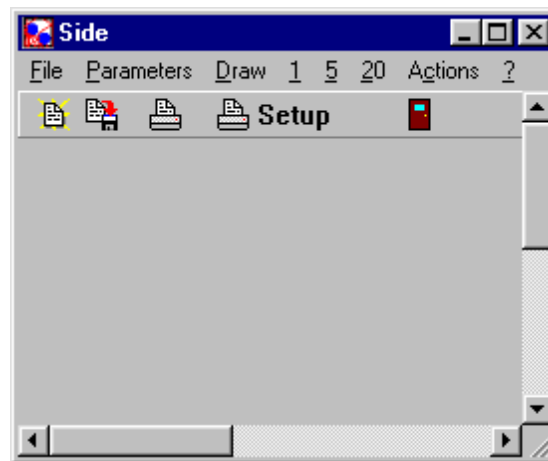
- Rendered views

6.2 Rendered view

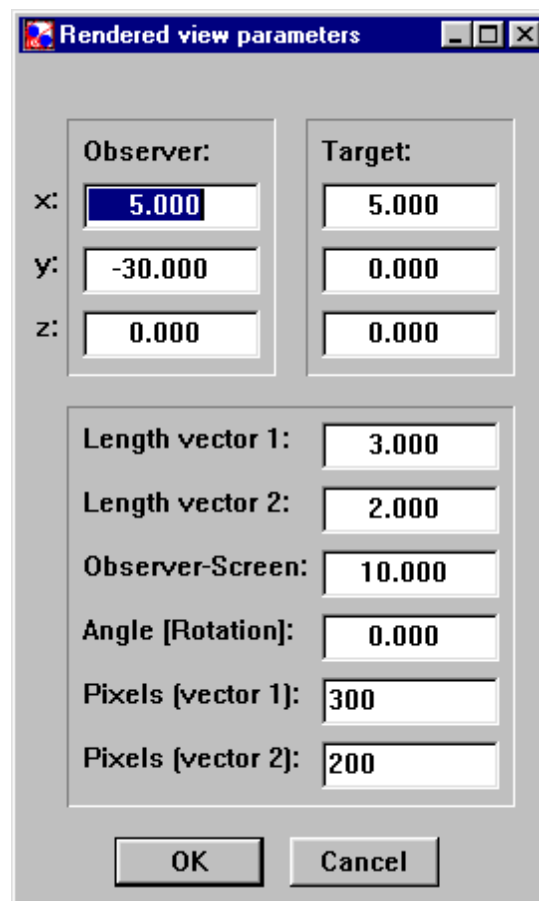
Rendered views are 'photographs' of the SPRAY setup taken the following way. Rays from the observation point are drawn through each pixel of the observation window. The closest hitpoint with an object (if there is any) is taken to determine the color of the pixel:



The pixel array is drawn as bitmap in the rendered view window:



The **Parameters** menu command opens the following dialog which gives access to the relevant parameters:

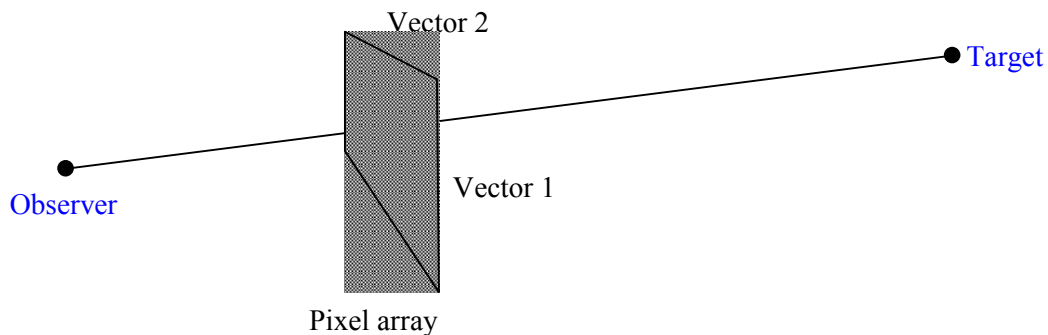


The parameters of rendered views are these:

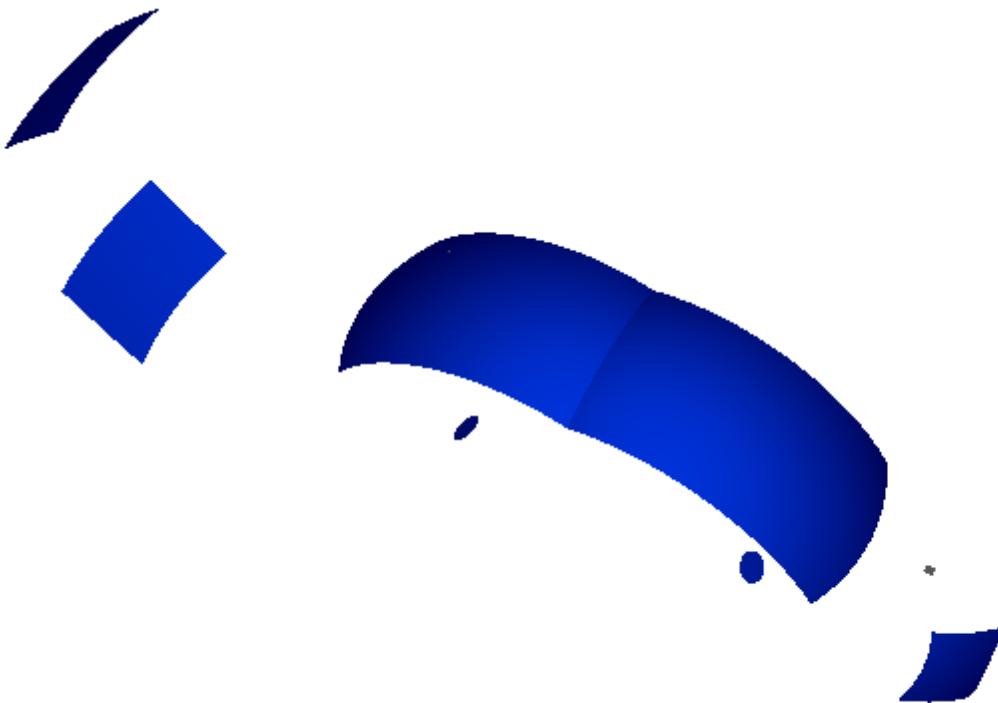
- Observer (x,y,z): Coordinates of the observer position
- Target (x,y,z): The observer looks in direction of the target, i.e. a line drawn from the observation point through the center of the pixel array will arrive at the target position.

- Length vector 1: This is the length of vector 1 (see sketch below). Vector 1 starts in the center of the pixel array, is perpendicular to the line observer-target and lies in the x-y-plane (i.e. its z-component is zero), at least if the rotation angle is zero (see below).
- Length vector 2: Length of vector 2 which starts at the center of the pixel array and is perpendicular both to the line observer-target and vector 1.
- Observer-Screen: Distance between the observer and the center of the pixel array. Changing this distance you can easily change the viewing field.
- Angle (Rotation): The angle between vector 1 and the x-y-plane. This can be used to rotate the 'camera'. Vector 2 will follow automatically.
- Pixels (vector 1): Number of pixels in the direction of vector 1.
- Pixels (vector 2): Number of pixels in the direction of vector 2.

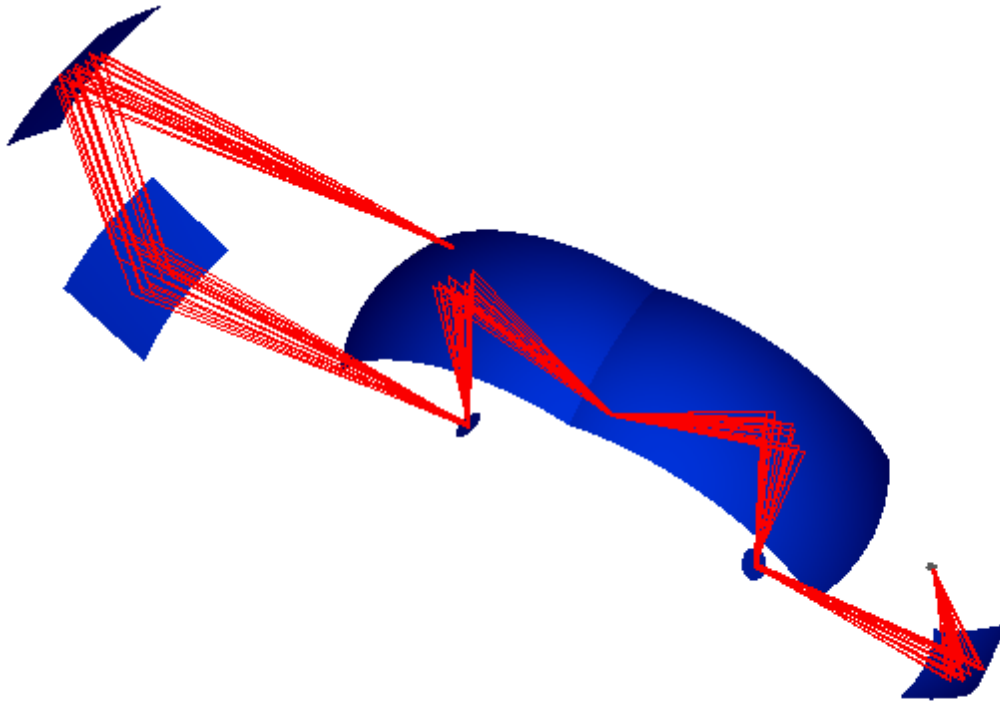
To avoid picture distortions the ratio of the lengths of vectors 1 and 2 should be the same as the ratio of the corresponding number of pixels.



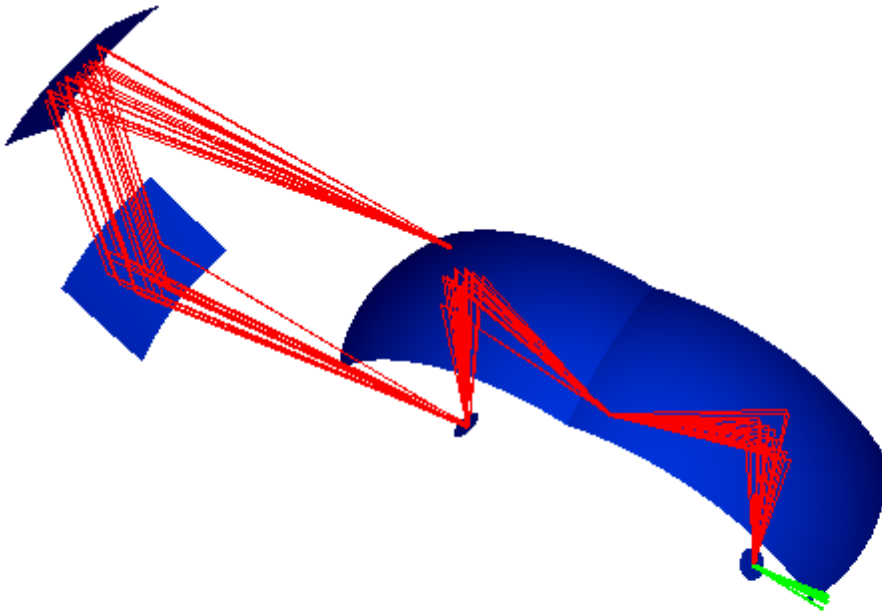
The **Draw** command starts the computation of the picture. Note that the required computational time can be significant, in particular if you work with a high number of pixels. For 600 by 600 pixels SPRAY needs to determine 360000 closest hit points.



With the commands **1**, **5** and **20** you can send 1, 5 or 20 test rays through the scenery. **SPRAY** performs the ray-tracing and the rendered view draws the path of the rays into the bitmap. The minimum of the selected spectral range (see below) is taken for the test rays. You can repeatedly use the test ray commands to create more and more rays in the picture:



It is not checked if parts of the rays are hidden by some objects - SPRAY simply draws the complete paths. Rays that do not hit any object of the scenery and escape to infinity are drawn as short lines in a different color:



If you have opened several rendered views (with different observation points and directions, for example) the test rays are drawn in each view.

7 Simulation options

7.1 Spectral range and angle resolution

The button labeled **Parameters** in the main window of SPRAY opens the following dialog which gives access to important simulation parameters:

Spectral Range & Angle Resolution

Spectral range :

Minimum Unit

Maximum

Number of points

Angle resolution :

Number of points

Number of photons per spectral point :

Number of points

Max. number of interactions:

Number per photon

OK Cancel

In the '**Spectral range**' section you can set the spectral range for the ray tracing simulation. Specify the minimum, the maximum, the number of spectral points and the unit. For the latter you can choose between 1/cm (wavenumbers), nm (wavelength), eV (energy), micron (wavelength) and THz (frequency).

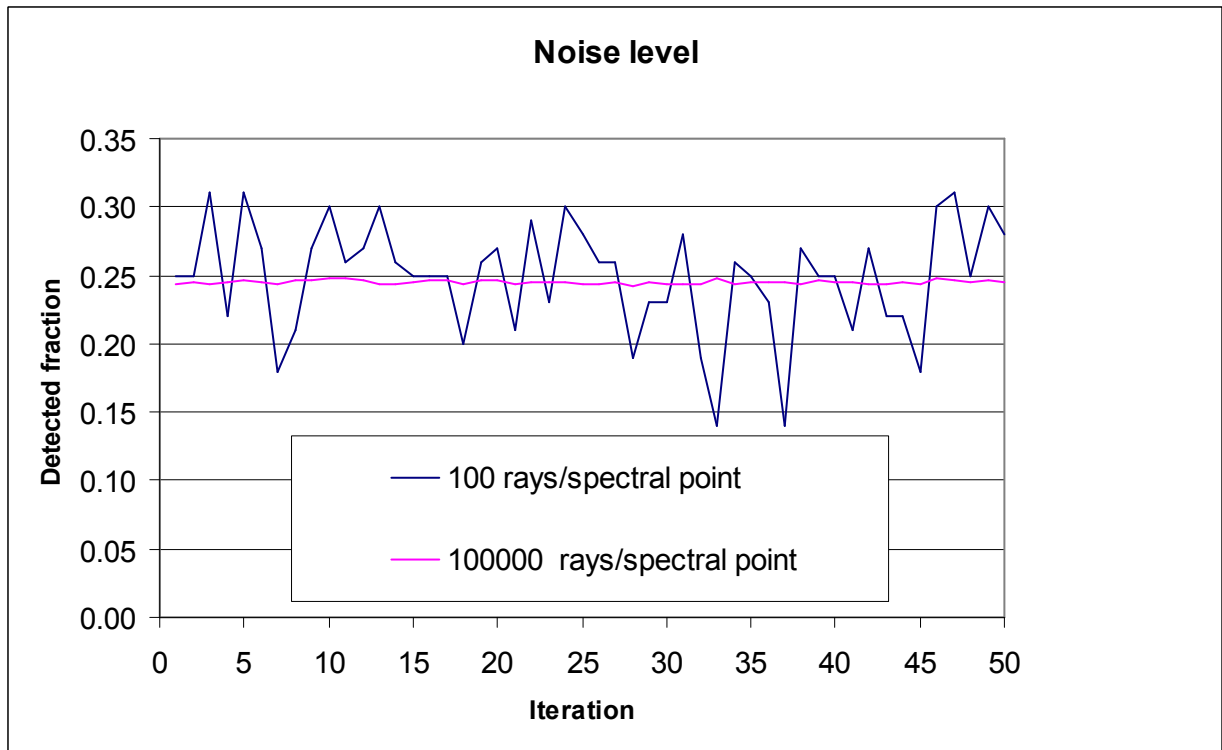
The '**Angle resolution**' is important for interface objects. These compute (before the simulation is started) the angular dependence of the reflectance and transmittance using the specified number of points for the range 0 ... 180 degrees. Usually this does not take too long compared to the subsequent ray-tracing with possibly many thousands of rays. Hence there is no reason to go to a low angle resolution unless you have a large number of interfaces in your model.

The '**Number of photons per spectral point**' determines how many rays are processed for each spectral point. How many rays you need depends very strongly on the questions that you have to answer.

Finally you can set the '**Max. number of interactions**' for each ray. After a ray has been emitted by the light source SPRAY counts how many interactions this ray has with the objects of the setup. If the specified maximum value of interactions is reached before the ray reaches infinity or is absorbed the tracing of this ray is stopped. This is to avoid situations where a ray is reflected back and forth forever between two ideal mirrors, for example. Usually you do not have to change this value unless you study very special setups.

7.2 How many rays do you need?

Solving problems by optical ray-tracing involves some randomness - consequently there is noise in the results. The following graph shows an example: The same SPRAY simulation has been repeated 50 times, and the chart shows the detected fraction obtained with 100 and 100000 rays:



Obviously, working with more rays reduces the noise and increases the quality of the result. In most cases you can work with the following simple relations to get an idea of the noise in your SPRAY results.

Suppose you work with N rays/spectral point, and on the average a fraction f of the emitted rays reach the detector. Then the amplitude of the noise is about

$$\Delta f = f \frac{\sqrt{f N}}{f N} = \sqrt{\frac{f}{N}}$$

In the example shown above f is about 0.24 and the expected noise amplitudes are $\Delta f = 0.049$ for $N = 100$ and 0.00155 for $N = 100000$. For comparison, the standard deviation of the 50 experiments

in the case of 100 rays is 0.04122, for 100000 a value of 0.00126 was found. These numbers are in rough agreement with the noise amplitudes given by the simple formula above, and also with the 'experimental' results.

7.3 Start options

There are two buttons to finally start the SPRAY ray-tracing simulation:

- **Simulation:** This starts the simulation and blocks all other windows of SPRAY until the computation has finished. You will see a progress bar indicating the percentage of the work that has been done already. This is the fastest and safest way to do simulations.
- **Simulation (thread):** The simulation is started in a separate thread. This way of doing the simulation is slower compared to the first one. During the computation, SPRAY will react to mouse clicks and other user actions. You can close or move windows and inspect object data. However, you should not change any parameters that may have influence on the simulation.

8 Distributed computing

8.1 Overview

SPRAY simulations may take quite a long time. You can increase the computational speed by adding the power of several PCs that work together on a SPRAY calculation.

What you need for SPRAY distributed computing is

- several PCs connected by a network
- a common network folder (all PCs must be able to write and read from this folder)
- SPRAY installations on all PCs (the minimal installation without database, tools and help files is sufficient)
- the *NIGHTSHIFT* tool running on every PC (this program is installed with SPRAY)
- one 'master' PC which runs SPRAY as distributed computing server.

How it works

On the master PC you load the SPRAY configuration that you want to process. You can start the distributed computing run either by hand or by OLE automation. SPRAY on the master PC divides the computational work into several tasks which are stored in a subfolder named 'untouched' of the common network folder. Thereafter it waits for incoming results in the subfolder 'results' which are added up to the total solution. If the results for all tasks have been received the simulation is ready. The *NIGHTSHIFT* tools on the client PCs constantly scan the network folders for SPRAY tasks. If a task is detected, a local SPRAY program (started by OLE automation) does the computation. This activity is indicated by moving the task from the subfolder 'untouched' to the subfolder 'started'. When the computational work of the task is finished and the task is still present in the subfolder 'started', the results are stored in the subfolder 'results' and the task is deleted in the folder 'started'. Having processed a task, *NIGHTSHIFT* looks for another task in the subfolder 'untouched'. If there is none, it looks for tasks in the subfolder 'started'. Work on tasks in this folder are indicated by the remark 'Working on pending tasks'.

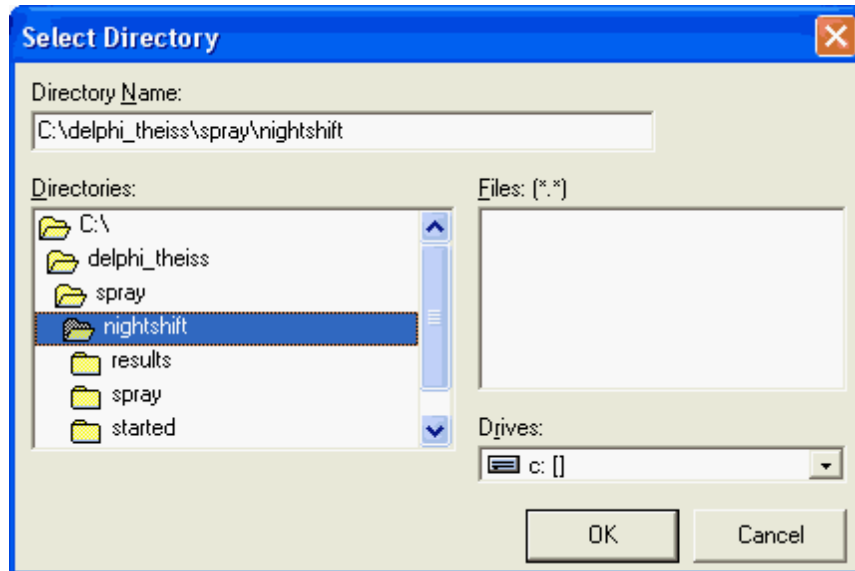
While SPRAY computations are running on a PC you cannot use the PC for other work because it is blocked almost completely by the heavy ray-tracing workload. Hence SPRAY simulations using office PCs should be done at night. With *NIGHTSHIFT* options you can split a day into daytime office hours (reserved for the owner of the desktop) and night time (available for SPRAY). At daytime the normal PC operation is not disturbed by *NIGHTSHIFT* which is just waiting. At night all the computational power is used for SPRAY.

The algorithm described above does not require that certain client PCs are available for tasks. Client PCs can join or leave the distributed computing at any time. If there is no client there is no progress, if there are many the progress is fast. Only the master PC must be running all the time.

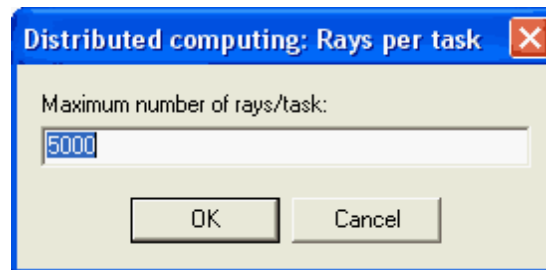
Before you try to work with distributed computing, please read the following sections about the master PC settings, the tool *NIGHTSHIFT* on the client PCs and strategy considerations for distributed computations. A VisualBasic demo shows how to control distributed computing from Excel.

8.2 Master PC

The configuration of the master PC is rather simple. Using the menu command **File|Options|Distributed computing: Network folder** you can set the common network folder in a dialog like the following:



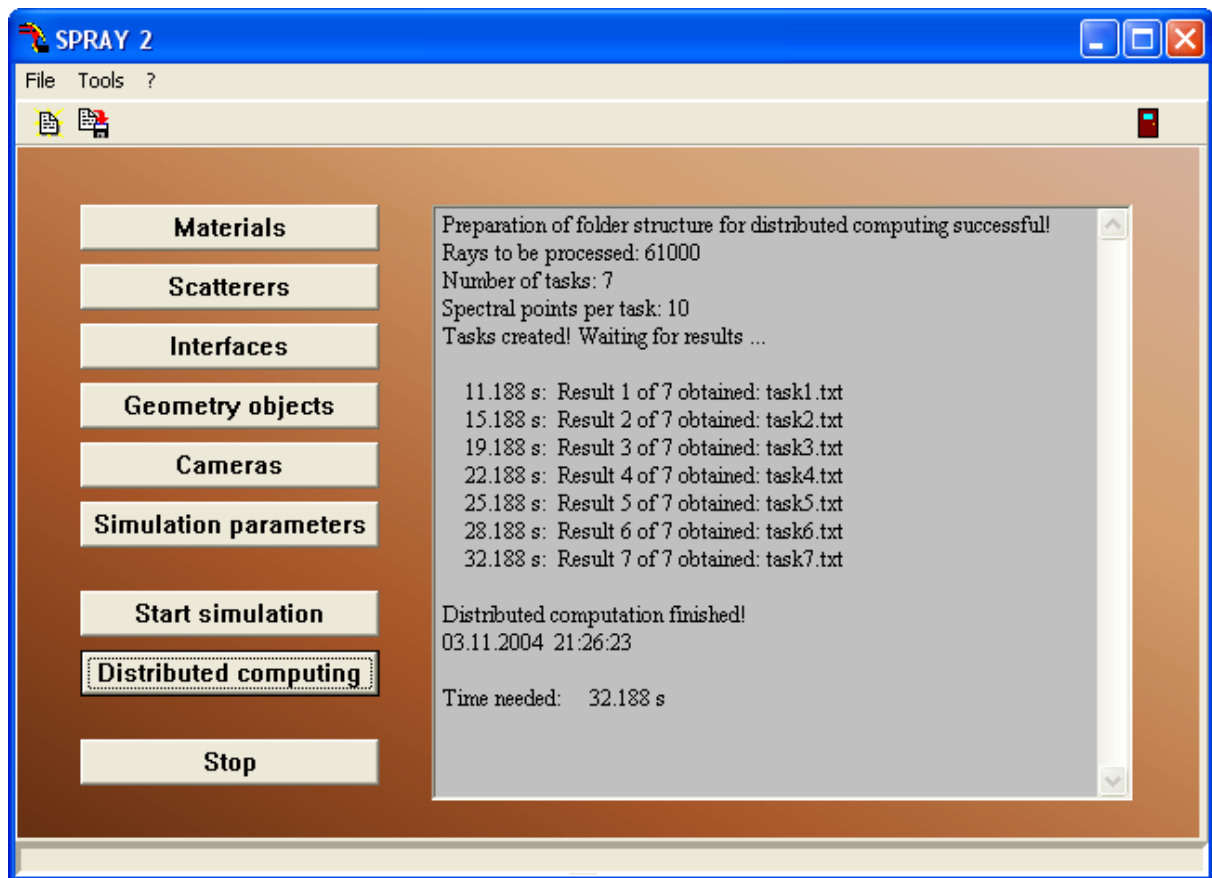
The size of the individual tasks is controlled by the parameter 'rays per task' which you can specify using the command **File|Options|Distributed computing: Rays per task**:



Please read the section 'Strategy for distributed computing' in order to make your choice of the number of rays per task.

This is all you have to configure on the master PC. The settings of the network folder and the number of rays per task are stored in the SPRAY configuration when you save it. You can now manually start the distributed computation pressing the button '**Distributed computing**' in the main window. If you control SPRAY on the master PC by OLE automation (for example by Excel's VisualBasic) the distributed computing is started using the command `spray.start_dc_computation` (here spray is the OLE automation object that has been assigned with `Set spray = CreateObject("Spray99.Spray_Remote")`). The numbers of rays per task can be set from VisualBasic as well, e.g. with the command `spray.dc_rays_per_task = 30000`. (See also the VisualBasic demo below)

On the master PC, a short distributed computing run looks like this:

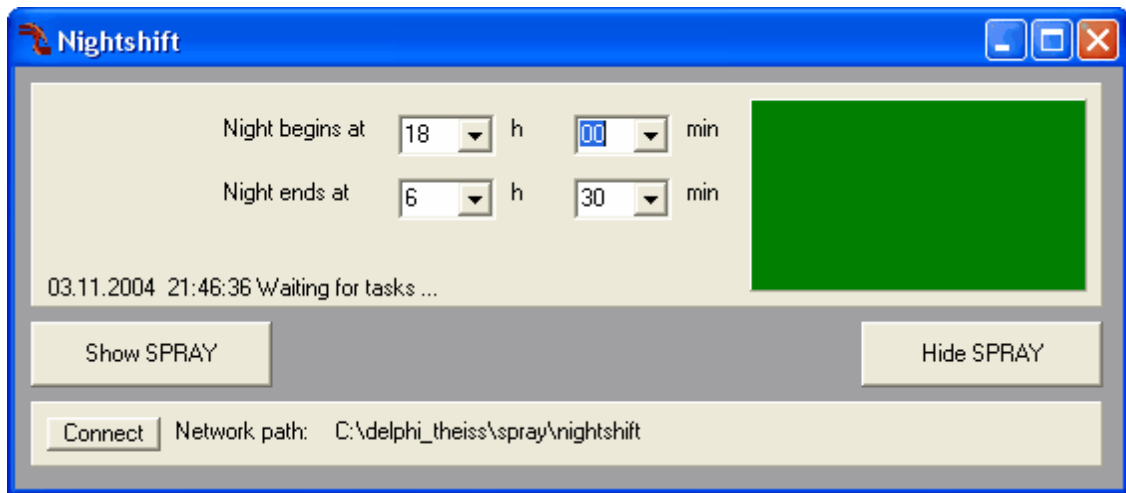


The task separation and the incoming results are logged. The time required for the complete simulation is given at the end. It is accessible from VisualBasic with the command `spray.seconds_last_run`.

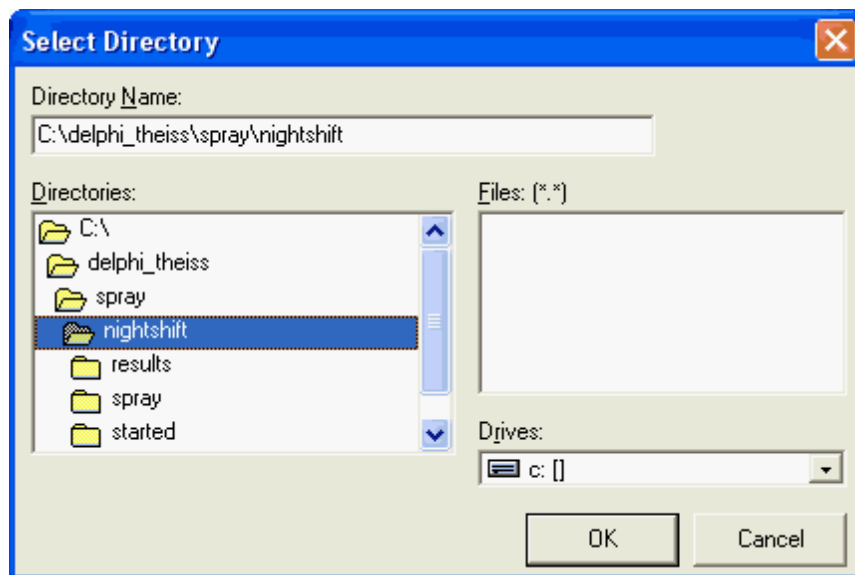
8.3 Client PCs: The tool NIGHTSHIFT

On every client PC you must install SPRAY. The minimal installation with the application files is sufficient. The installation routine copies a small program called nightshift.exe to the SPRAY directory.

After the installation, start and configure the small tool *NIGHTSHIFT*:

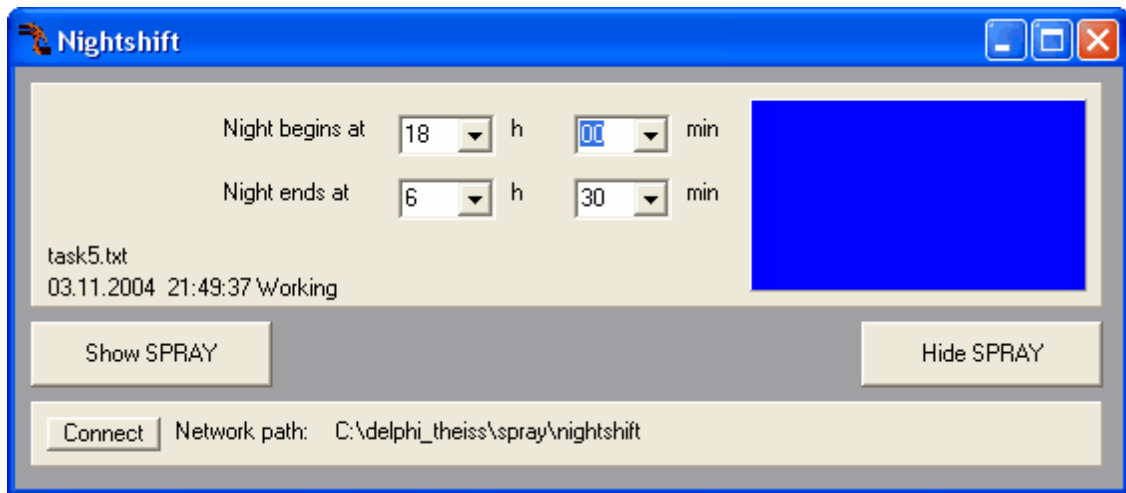


The controls in the upper half of the window are used to tell the program when it is night. At night, the program starts SPRAY as a hidden OLE automation server and passes tasks to it which are to be done. The tasks are looked for in the network folder which can be selected after pressing the **Connect** button:

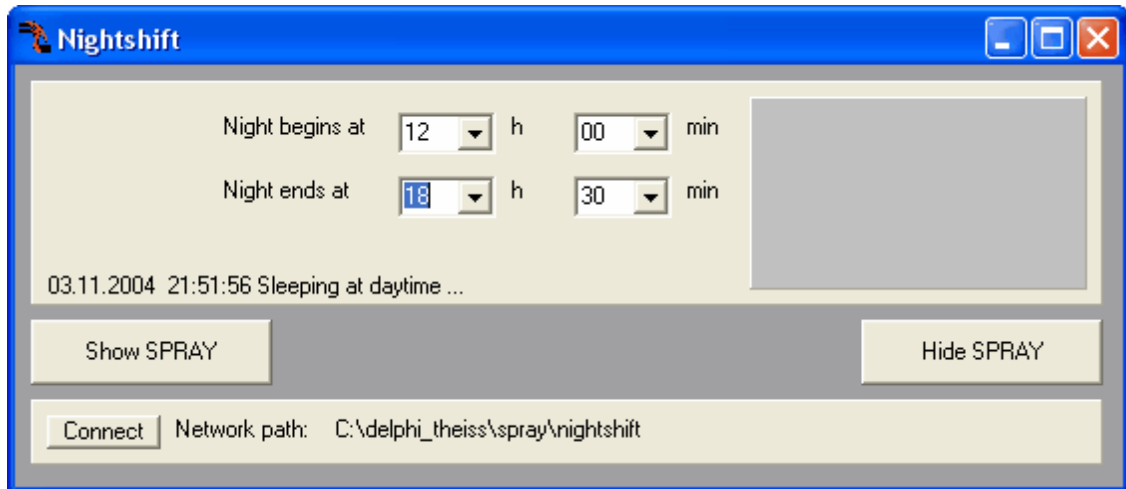


During the night, *NIGHTSHIFT* looks for tasks to be processed (this state is indicated by the green rectangle to the right).

While a task is processed the color changes to blue:



Working on pending tasks is indicated white. During the day *NIGHTSHIFT* looks like this:



If you like you can show the SPRAY server running in the background by pressing the **Show SPRAY** button. The server is hidden again if you press **Hide SPRAY**.

The settings of the *NIGHTSHIFT* program are stored in the file `nightshift.ini` located in the program directory (where the program file `nightshift.exe` has been installed to) when you close the program. The settings are read at program start.

8.4 Strategy for distributed computing

Once you have configured the common network path and SPRAY has been installed on all client PCs, everything is ready for distributed computing with SPRAY. The performance of the network depends on the speed of the individual computers and the size of the tasks. The following remarks can be used as a guideline for your own network experiments.

In addition to the numerical work to be performed for the ray-tracing the processing of the tasks

requires some extra time. The creation of the tasks on the master PC, the search for tasks on the client PCs and the exchange of configuration files and results cause a delay of a few seconds for every task. If a local computation on the master PC takes 10 seconds, and you split it up into 50 tasks to be processed in the network, it can easily take several minutes until you get the final result. Distributed computing is faster than local computing only if you have large simulation jobs and not too many tasks.

On the other hand, the number of tasks should not be too small. If you employ various computers which may differ in speed you should try to keep them all busy. If the individual tasks are too large the slow PCs will probably never give a useful contribution: While a slow PC is working on a too heavy computational load, the faster PCs may run out of work and start to work on the pending task of the slow PC as well, eventually overtaking it.

Current limitations in distributed computing:

- *NIGHTSHIFT* cannot stop its SPRAY OLE server properly when the master PC finishes the distributed computing. Therefore it may happen that a client PC still works on a task of a previous simulation while the next simulation has been started already. In this case, the results are not passed back to the master PC (it is checked if the results belong to the current simulation) but the time of the client PC needed to finish the 'old' task is lost.
- The master PC is not very busy in the case of distributed computing: It creates the tasks and then just adds up the results. Hence it is useful to start a *NIGHTSHIFT* program on the master PC as well in order to make use of its computational power. However, this does not work properly if you start the distributed computing manually by pressing the button 'Distributed computing'. We are working on a solution for this problem. If you control the master SPRAY by OLE automation (e.g. VisualBasic routines in Excel) everything works fine.

8.5 OLE automation demo

The following VisualBasic code shows the control of a simple distributed computing simulation with SPRAY. Please read the comments that explain what happens.

```
Sub dc_test()  
Dim nr As Long  
Dim seconds, signal As Single  
  
' Create the SPRAY Ole server  
Set spray = CreateObject("Spray99.Spray_Remote")  
  
' Load the configuration  
spray.load_configuration = "c:/temp/dc_test/dc_test.s99"  
' Show SPRAY
```

```
spray.Show
```

```
' Switch off Excel warnings
```

```
' this avoids the popup message that Excel waits for another application ...
```

```
DisplayAlerts = False
```

```
' Set the number of rays per spectral point
```

```
spray.photons = 10000
```

```
' Set the number of rays per task
```

```
spray.dc_rays_per_task = 20000
```

```
' Start the distributed computing simulation
```

```
spray.start_dc_computation
```

```
' Wait loop: spray.status is >=0 during the simulation and <0 when the job is finished
```

```
nr = spray.Status
```

```
While nr >= 0
```

```
    ' wait for 3 seconds
```

```
    Application.Wait Now + TimeSerial(0, 0, 3)
```

```
    ' ask SPRAY again
```

```
    nr = spray.Status
```

```
Wend
```

```
' get the time SPRAY needed for the last simulation
```

```
seconds = spray.seconds_last_run
```

```
' Now you should pick up the results ...
```

```
'  
'  
'
```

```
' Delete the SPRAY ole server
```

```
Set spray = Nothing
```

```
End Sub
```

9 OLE automation

9.1 Overview

SPRAY is an OLE automation server that exports some parameters and actions for external control by OLE automation clients. This offers very high flexibility to users who are able to develop programs or macros. In fact, only little programming knowledge is required to perform useful SPRAY computations.

The examples discussed in the following are coded in Excel's VisualBasic. The combination of a spreadsheet program with tables and charts and a macro language doing the SPRAY control operations is very useful for SPRAY work. However, if you do not want to purchase a spreadsheet program you can also use the Windows Scripting Host (WSH, a free Windows tool by Microsoft) to run VisualBasic scripts. In addition, OLE servers can be addressed from LabView or any modern programming language like C++ or Delphi.

The following sections describe the OLE automation features of SPRAY:

- Handling the OLE server
- Simulation parameters
- Object parameters
- Retrieving results
- Video generation

9.2 Handling the OLE server

Registration

Before you can use SPRAY as OLE server you have to start it once manually, i.e. run it using the corresponding Start menu command or executing the program `spray99.exe`. At program start, SPRAY will be registered as the OLE server 'Spray99.Spray_Remote' on your machine.

VisualBasic implementation

In the following the VisualBasic code is in **blue** whereas all comments are in black.

Declaration:

To use SPRAY in a VisualBasic macro you have to declare it as a public object:

```
Public Spray As Object
```

Create the OLE server:

Before you can call SPRAY commands and properties you have to create the OLE server with the `CreateObject` command:

```
Sub create()  
  Set Spray = CreateObject("Spray99.Spray_Remote")  
End Sub
```

Loading a configuration:

Once SPRAY is created you can load a SPRAY configuration by assigning a filename to the *load_configuration* property:

```
Sub load_file()  
    Spray.load_configuration = "C:\spray2\ole_test.s99"  
End Sub
```

Saving a configuration:

If the SPRAY settings have been modified by OLE automation you can save the configuration by assigning a filename to the *save_configuration* property:

```
Sub load_file()  
    Spray.save_configuration = "C:\spray2\ole_test2.s99"  
End Sub
```

Show SPRAY:

The *show* command shows SPRAY on the screen

```
Sub show_spray()  
    Spray.show  
End Sub
```

Hide SPRAY:

The *hide* command shows SPRAY on the screen

```
Sub hide_spray()  
    Spray.hide  
End Sub
```

Cleaning up:

To finish your work you should remove SPRAY from memory which is done in VisualBasic the following way:

```
Sub destroy()  
    Set Spray = Nothing  
End Sub
```

9.3 Object parameters

In order to optimize a setup or to investigate 'strange effects' it is quite helpful to vary a parameter (e.g. the position of a geometric element) and inspect the change of the detector signals. To do so, you have to go through the following sequence:

- Set the parameters of the model
- Perform the simulation and compute the detector spectra
- Read the required data and store them in appropriate tables
- Go back and do the next parameter modification

The parameters of SPRAY objects that can be controlled by OLE automation are described in this

section. Please tell us if you are missing some items here. Usually we can provide additional OLE features quite rapidly.

Scatterers:

scatterer_parameter(name:string, parameter:string): float

Using this property you can set or read parameters of scatterers. With *name* you have to specify the name of the scatterer, with *parameter* the property you want to pick. You can set, for example, the volume fraction by referring to the parameter 'volume_fraction'.

The following VisualBasic line sets the volume fraction of the scatterer named 'Water spheres' to 3%:

```
spray.scatterer_parameter("Water spheres","volume_fraction") = 0.03
```

The notation without underscore is also tolerated:

```
thevalue = spray.scatterer_parameter("Water spheres","volume fraction")
```

You can retrieve the absorption and scattering probabilities K and S the following way:

```
thevalue = scatterer_parameter("Water spheres","K at 1000 nm")
```

```
thevalue = scatterer_parameter("Water spheres","S at 4.53 eV")
```

Information about the size distribution

If the scatterer is of type 'Extended Mie scatterer' or 'Fluorescent Mie scatterer' you can get information about the current radius distribution and manipulate it. Get the number of radii using the command

```
number_of_points = scatterer_parameter("Water spheres","size classes")
```

The 'radius points' are counted 1,2,3, ...

Get the radius of point 7 using the command

```
radius = scatterer_parameter("Water spheres","radius size class 7")
```

Get the probability of point 7 using the command

```
prob = scatterer_parameter("Water spheres","probability size class 7")
```

Changing the size distribution

When you set the number of points of the radius distribution, the internal data array is initialized with zeroes:

```
scatterer_parameter("Water spheres","size classes") = 10
```

You should now use a loop and set the radius and the probability for all points:

```
scatterer_parameter("Water spheres","radius size class 1") = 20E-9
```

```
scatterer_parameter("Water spheres","probability size class 1") = 0.23
```

```
scatterer_parameter("Water spheres","radius size class 2") = 40E-9
```

```
scatterer_parameter("Water spheres","probability size class 2") = 0.134
```

scatterer_command(name:string, command : string, parameter: float)

Use this command to manipulate the size distribution. The following commands are defined up to now:

Normalize the distribution: `scatterer_command("Water spheres","Normalize",0)`

Shift the radius values of the distribution (by 20 nm, in this example): `scatterer_command("Water spheres","shift radius distribution",20e-9)`

Multiply the radius values of the distribution (by 1.1, in this example): `scatterer_command("Water`


```
spheres","multiply radius distribution",1.1)
```

scatterer_load_rt(name : string) : string

Passing a filename to this property causes the scatterer with the specified *name* to import RT data from the specified file.

Here is an example:

```
spray.scatterer_load_rt("Water spheres") = "C:\mie\water1.rt"
```

Geometric objects:

object_parameter(name:string, parameter:string): float

Using this general property you can set or read parameters of geometric objects. With *name* you have to specify the name of the object, *parameter* determines the property you want to set or read. Changing the position (x-coordinate) of a mirror named 'Mirror 2', for example, can be done like this:

```
spray.object_parameter("Mirror 2","x") = 14.3
```

Please consult the description of the individual geometric objects to see which parameters are available.

object_parameter_string(name:string, parameter:string): string

Using this property you can set or read string parameters of geometric objects. With *name* you have to specify the name of the object, *parameter* determines the property you want to set or read. Changing the formula of a user-defined surface shape named 'My shape', for example, can be done like this:

```
spray.object_parameter_string("My shape","surface_formula") = "0.3+0.3*(sin(0.2*x))"
```

Please consult the description of the individual geometric objects to see which parameters are available.

Views

In connection with automatic video generation or simply for taking some snapshots of the SPRAY scenery, rendered views can be modified by OLE automation as well. The command for doing so is this:

view_parameter(name:string, parameter:string): float

Name specifies the view you want to select, and *parameter* determines the property you want to read or set. Here are the parameters that you can access:

- 'observer_x' : x-coordinate of the observation point
- 'observer_y' : y-coordinate of the observation point
- 'observer_z' : z-coordinate of the observation point
- 'target_x' : x-coordinate of the target
- 'target_y' : y-coordinate of the target
- 'target_z' : z-coordinate of the target
- 'vector_1': Length of vector 1
- 'vector_2': Length of vector 2
- 'distance': Distance observation point - center pixel array
- 'angle': Rotation angle of the pixel array

9.4 Simulation parameters

The following parameters for the configuration of SPRAY simulations can be set by OLE automation controllers:

Number of rays/spectral point:

This parameter is set using the photons property (integer). The command
`spray.photons = 2000`
 sets the number of rays/spectral point to 2000. You can read the current value this way:
`a_variable = spray.photons`

Minimum of spectral range:

Change this parameter using the spectral_min property (float). The command
`spray.spectral_min = 435.3`
 sets the spectral minimum to 435.3 . You can read the current value this way:
`a_variable = spray.spectral_min`

Maximum of spectral range:

This parameter is set using the spectral_max property (float). The command
`spray.spectral_max = 1100`
 sets the spectral maximum to 1100. You can read the current value this way:
`a_variable = spray.spectral_max`

Number of spectral points:

The property (integer) 'spectral_points' has to be used when the number of spectral points for SPRAY computations is to be modified:

`spray.spectral_points = 120`
 The current value of this parameter is obtained this way:
`a_variable = spray.spectral_points`

Doing the simulation:

Ray-tracing simulations may take a long time. Some OLE automation clients like Excel do not wait long enough for lengthy computations to be finished, and raise a warning dialog which blocks the execution of the VisualBasic code. This behaviour completely destroys the automation: You have to click the OK button of the dialog every 10 minutes or so to continue in the VisualBasic code. In order to avoid this problem SPRAY creates a simulation thread which does the ray-tracing work in the background. In the foreground, SPRAY will continue to listen to OLE commands. Hence you can start the SPRAY simulation, and then let Excel execute a loop until SPRAY has finished its work. The SPRAY property *status* gives information about the number of processed rays. If it is negative, SPRAY has finished the simulation and Excel can close its waiting loop. The following few VisualBasic lines implement this strategy:

`dim rays as long ' variable holding the number of processed rays`

<code>Spray.start_simulation</code>	<code>'Start the SPRAY simulation</code>
<code>Do</code>	
<code> Application.Wait Now + TimeSerial(0, 0, 2)</code>	<code>'Excel waits for 2 seconds</code>

```
rays = Spray.Status                                'Excel reads the number of processed rays
Application.StatusBar = rays                        'Excel indicates the number of rays in the
status bar
Loop While (rays > -1)                             'Loop finished if SPRAY has done its work
```

You could now start routines that readout the results from detectors or screens.

9.5 Retrieving results

Having done SPRAY simulations controlled by OLE automation or manually; you can use the following commands to get access to detector signals.

Simple rectangular detectors

simple_detector_value(name : string, specpos : float) = float

Here you have to specify the name of the detector object and the spectral position. The latter must be given in wavenumbers (i.e. the inverse wavelength, measured in 1/cm). You will get back the detected fraction of the detector at this spectral point.

VisualBasic example:

```
result = spray.simple_detector_value("Backside detector",12500)
```

Array detectors

array_detector_value(name : string, pixel, specpos : float) = float

Parameters of this function are the detector name, the wanted pixel and the spectral position (in wavenumbers). The pixels are counted from 1 to the number of pixels of the array. The return value of the function is the detected fraction.

VisualBasic example:

```
result = spray.array_detector_value("My detector", 14,12500)
```

Screens

screen_value(name : string) = float

You specify the name of the screen and this function returns the fraction of rays that hit the screen. Screens add rays independent from spectral positions, i.e. there is no frequency information.

VisualBasic example:

```
result = spray.screen_value("Side")
```

9.6 Video generation

9.6.1 Video generation

With SPRAY, OLE automation and the third party shareware tool 'Platypus Animator' you can automatically generate videos like a flight around or through your scenery. Controlled by OLE automation, SPRAY computes the required pictures which are finally composed to a video sequence by the Platypus Animator program.

The following page shows a small and simple example. Note that the video download may consume quite some time ...

The following OLE commands are useful for picture and video generation:

Computation of view pictures

The command *start_grafx* performs the computation of all rendered view pictures. In VisulBasic, the line

`spray.start_grafx`
will do this.

Modifying pictures

Once the view pictures are generated you can modify them with the following commands.

The command *test_ray* (no parameters) will tell SPRAY to send one test ray (as if you had selected the **1** menu command in a rendered view). With a loop you can generate as many test rays as you like:

```
For i = 1 To 20
    spray.test_ray
Next i
```

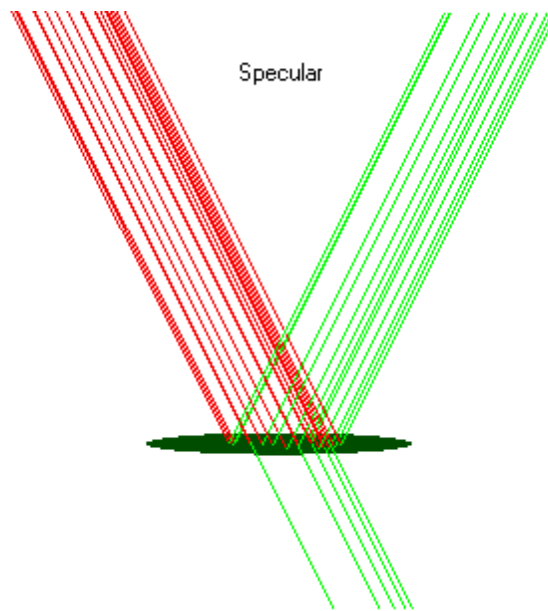
You can add some text using the property *text(name: string, the_text: string) = position (integer)*. Here is an example:

```
spray.Text("Side", "Specular") = 1
```

This command adds the text 'Specular' to the picture of the view named 'Side'. Assigning the value 1 (position) places the text in the upper center of the picture. The following values of position may be used:

- 1: Top, horizontally centered
- 2: Bottom, horizontally centered
- 3: Left, vertically centered
- 4: Right, vertically centered

Here is an example for position 1:



Saving pictures

The property *save_bitmap* of type string is used to save the computed view bitmaps. If you pass a filename to the property SPRAY saves the pictures of the views to the specified filename. The letters A, B, C, ... are added to the filename in order to separate between the individual views in the list of views.

The VisualBasic command

```
spray.save_bitmap = "c:\video\test\sideview"
```

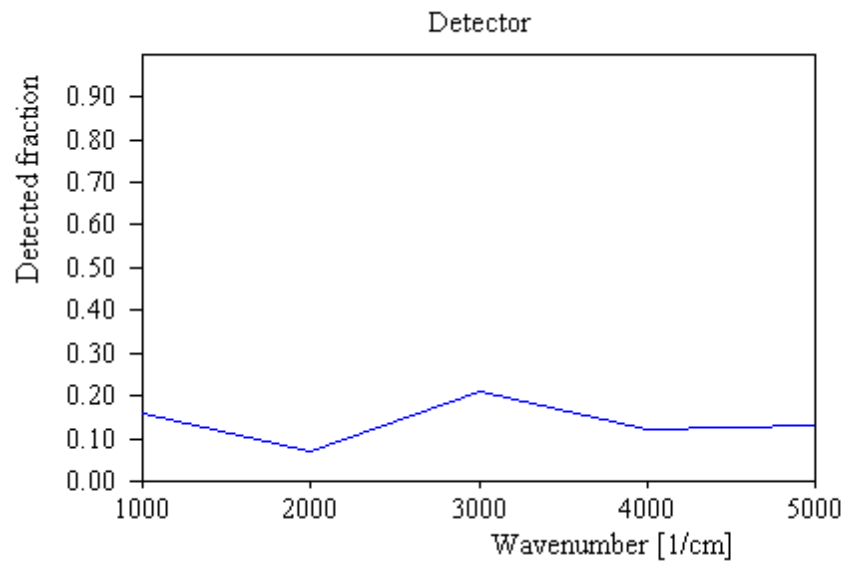
will create the files sideviewA.bmp, sideviewB.bmp, ... in the directory c:\video\test.

The property *save_detector_bitmap*(*x_pixels*, *y_pixels* : integer, *object_name* : string) = *filename* : string is used to generate a picture with a detector spectrum. The parameters *x_pixels* and *y_pixels* set the size of the bitmap, *object_name* specifies the name of the detector. The *filename* assigned to the property is used for saving the bitmap. The extension .bmp is automatically added to the filename.

A VisualBasic command like

```
spray.save_detector_bitmap(500, 300, "My detector") = "c:\temp\detector1"
```

creates a picture like the following:



The preparation of video sequences can be done best with the following commands:

save_bitmap_auto(filename : string) = frame : integer saves the bitmaps using a name which is a combination of the string you specify with the *filename* parameter and the number of the current picture in the sequence of the video, the *frame* parameter. Using this command in a loop (with increasing frame number) generates a sequence of pictures. The loop must count the frames starting at 0.

After the loop is finished you execute the command *write_ini_file(filename : string) = frame : integer* with exactly the same filename as you used in the *save_bitmap_auto* command. The parameter *frame* must be set to the last frame number in the loop. The *write_ini_file* command tells SPRAY to create a text files (one for each view in the list of views) which contains information for the *Platypus Animator* program. The name of the text files will be composed of the *filename* parameter, the letter 'A' for the first view, the letter 'B' for the second and so on, and the extension *.ini*.

Finally you have to pass the text files as parameter to the *Platypus Animator* program. It will create the video sequences and store it as *.avi* file.

Here is a complete example of a VisualBasic routine that creates a video sequence:

```
Public Sub rainbow()
Dim z As Single

For j = 0 To 50
' Light source is lifted up
z = j / 50
spray.object_parameter("Light source", "z") = z
' Compute picture
spray.start_grafx
' Send some test rays
For i = 1 To 100
    spray.test_ray
```

```
Next i
' Add a text (the current height of the light source)
spray.Text("Side", "Height: " + Format(z, "####0.0")) = 1
' save the bitmap
spray.save_bitmap_auto("c:\temp\rainbow") = j
Next j

' Write the ini-File for the avimaker program
spray.write_ini_file("c:\temp\rainbow") = 50

' call the avimaker program with ini-File as parameter
Call Shell("C:\video\avimaker\AviMaker.exe" + " " + "c:\temp\rainbowA.ini", 1)

End Sub
```

The result is shown on the next page.

9.6.2 Demo video

This short SPRAY video example shows some test rays travelling through a water sphere. A sharp light beam first hits the center of the sphere and is then moved up. The video shows rays being immediately reflected at the air/water surface, transmitted through the sphere, and leaving the sphere after one internal reflection. The latter are responsible for the appearance of the rainbow.

If you click on the rectangle below and you do not see the video, very likely your PC doesn't support this video format. Sorry!

10 Automated parameter fitting

10.1 Introduction

In order to optimize parameters of SPRAY objects we have integrated a fitting module into the software. Since the handling of parameter fits is not easy it should be applied by experienced SPRAY users only.

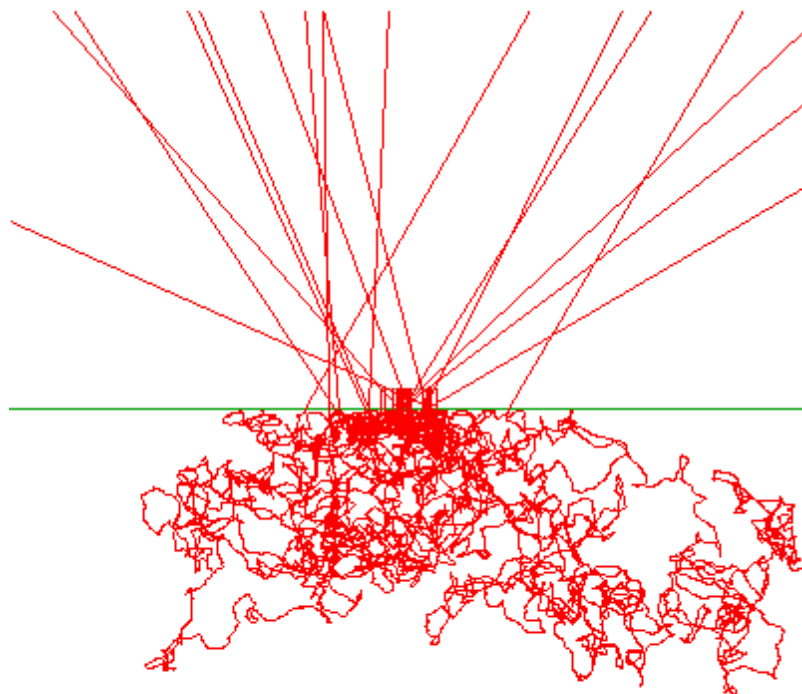
The following step-by-step example shows how the fit module is used to adjust optical constant parameters by a comparison of SPRAY simulations and diffuse reflectance measurements.

10.2 Step-by-step example

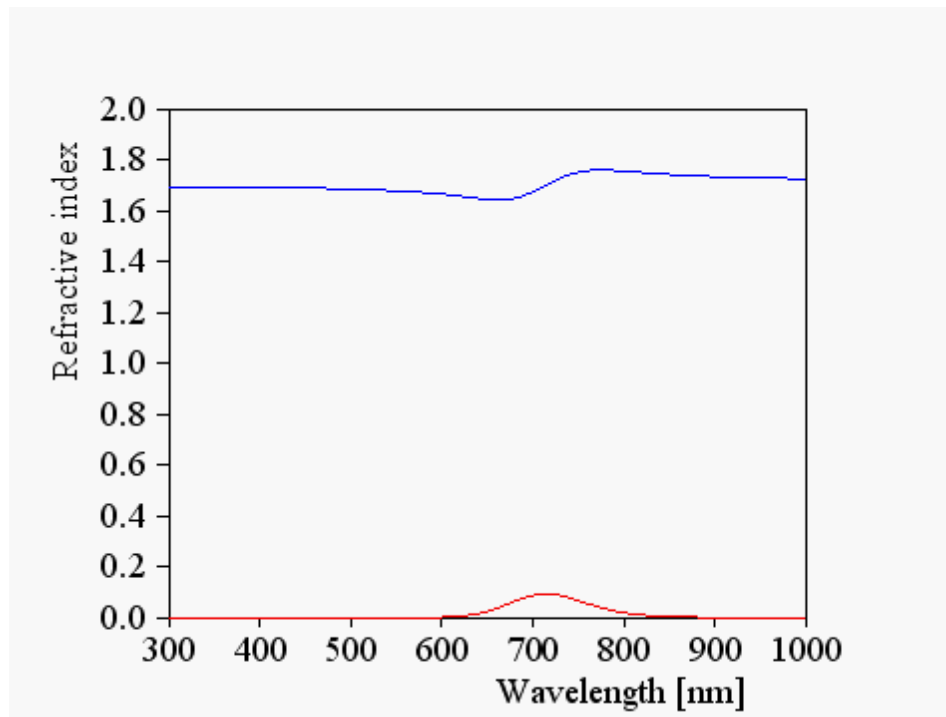
10.2.1 SPRAY model

The following simple SPRAY model is used to simulate a diffuse reflectance experiment: A circular (transparent) light source illuminates an interface from air to a solid material (binder). The binder contains spherical pigments with an absorption band in the visible spectral range. The binder itself absorbs in the UV. On top is a huge rectangular detector collecting all rays that escape into the upper halfspace.

Here is a side view of the system not showing the detector:



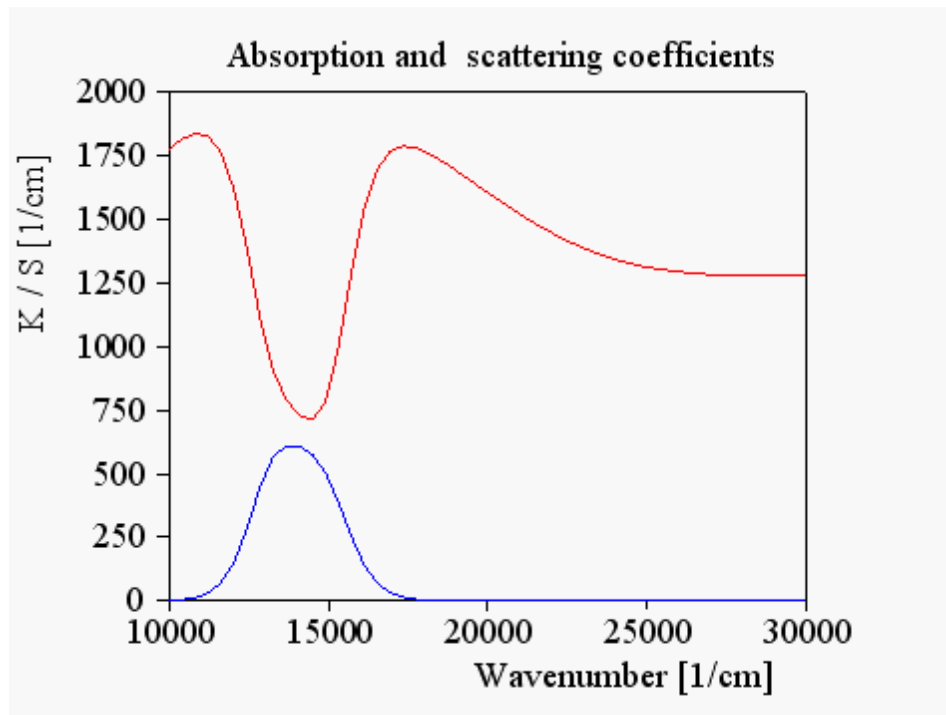
The optical constants of the pigments are the following:



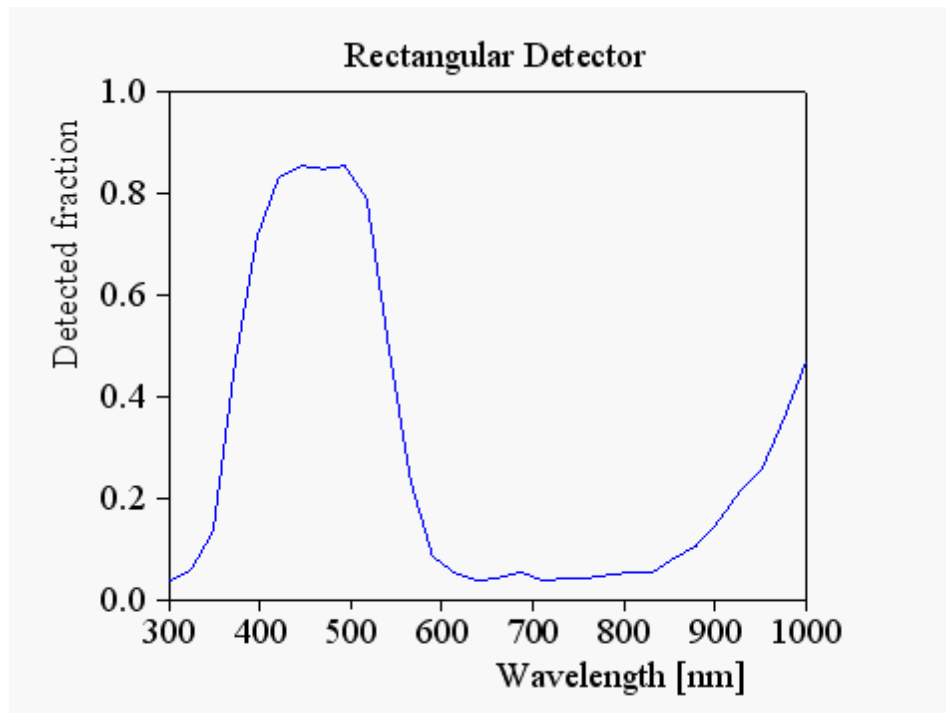
They are computed based on a constant and a Kim oscillator:

S2 Susceptibilities											
File New Edit Delete Update ?											
Dielectric background											
	Name	Type	Param.	Value	Param.	Value	Param.	Value	Param.	Value	Pa
1	Absorption band	Kim oscillator	Pos.	14000.0	Str.	3000.0	Damp.	2000.000	GL-switch	0.100	
2	n	Constant refrac		1.7000							

The scattering and absorption coefficients are computed using an extended Mie scatterer object with the following result:



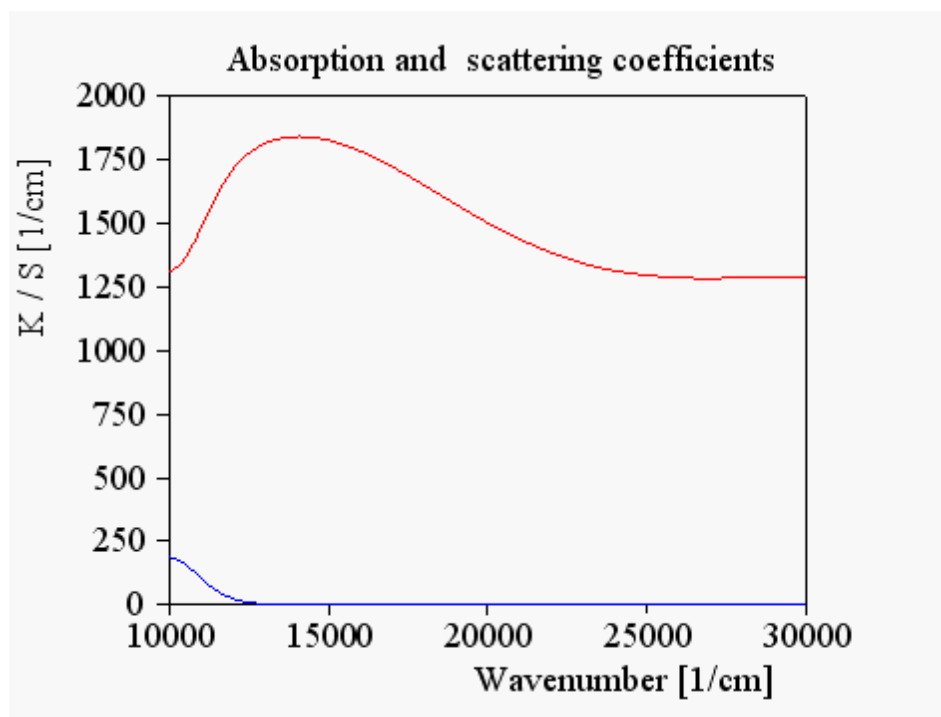
The corresponding diffuse reflectance spectrum has been computed with 30 spectral points in the range 300 ... 1000 nm sending 1000 rays at each spectral points:



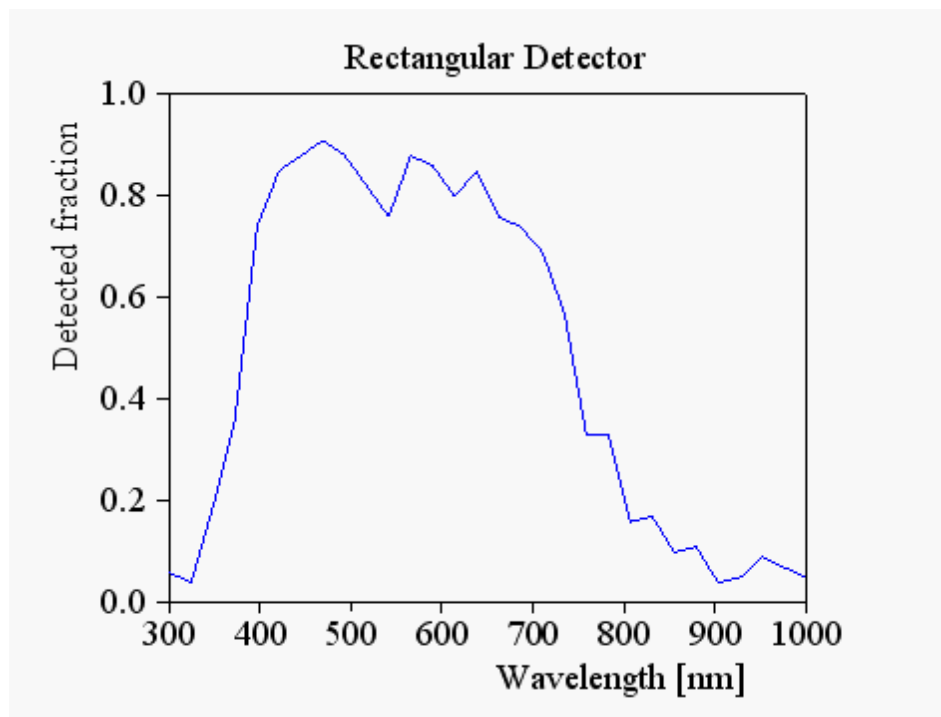
This spectrum is saved in the file `measured_spectrum.std` using the standard data format. It will serve as the 'measurement' in the following fit procedure.

10.2.2 Starting configuration

The SPRAY model that has been used to compute the 'measurement' is now modified a little: The resonance frequency of the Kim oscillator is changed from 14000 to 10000 1/cm, the oscillator strength from 3000 to 1000 1/cm. The absorption and scattering coefficients are now the following:



The diffuse reflectance spectrum is changed too, of course. The spectrum is now calculated with 100 rays per spectral point only in order to speed up the computations in the following parameter fit:



This configuration is saved in the file `optimize_start.s99` which is distributed with the help file.

10.2.3 Preparing the parameter fit

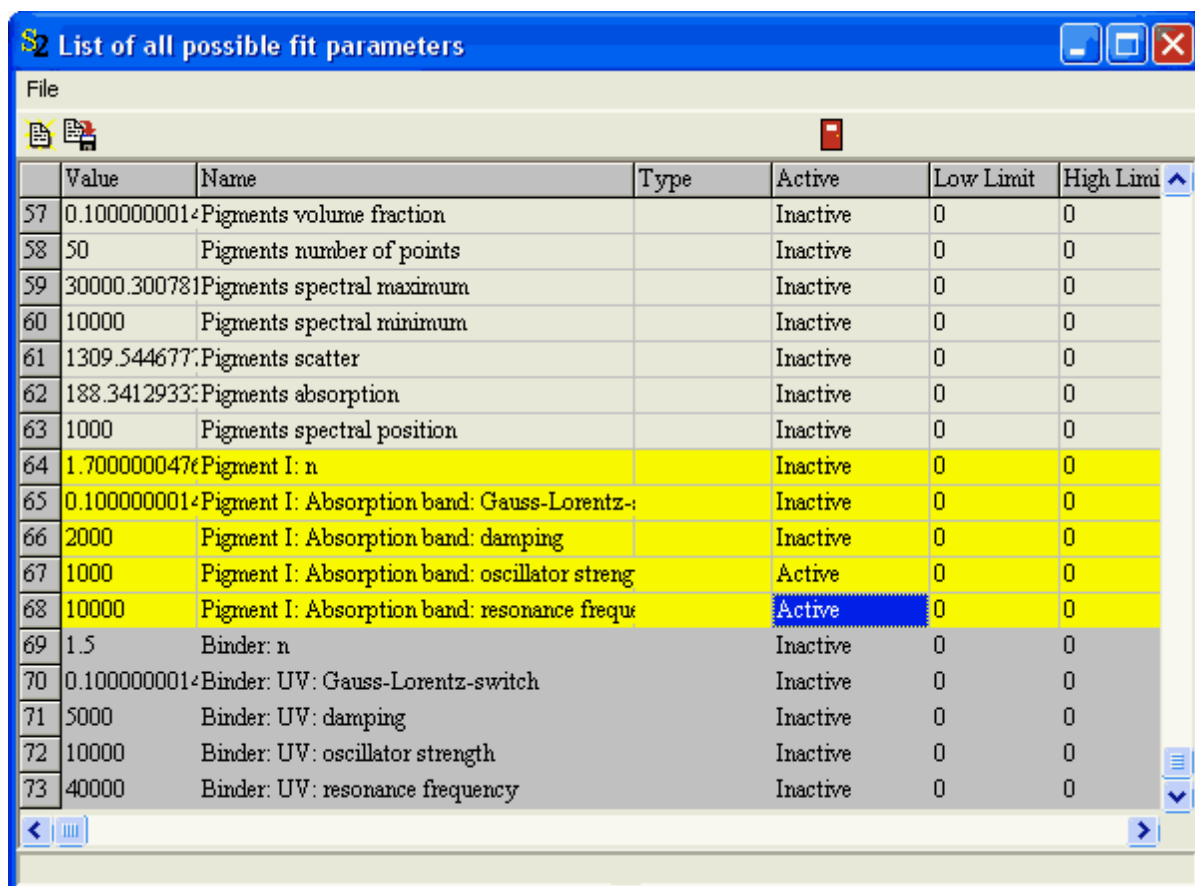
We can now prepare the parameter fit. Please follow exactly the next steps.

1. Open the fitting module with the menu command **Tools|Optimize**:



2. Push the button '**Select fit parameters**'. This will open a list which will show all possible fit parameters. Move down the list until the cursor reaches item 67. Move the cursor to the column 'Active' and change the value by pressing the **F4** key. Stop when the state of parameter 67 is 'Active'.

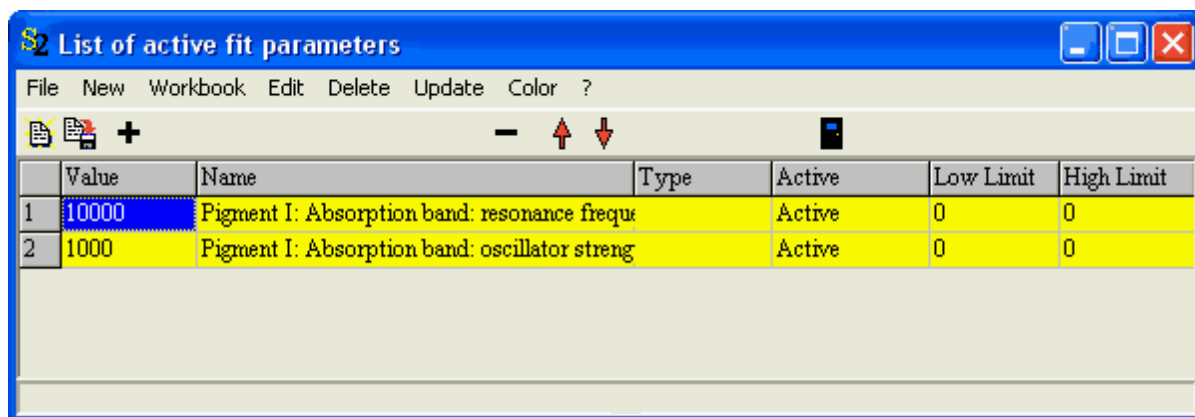
3. Move to line 68 and set the resonance frequency parameter to 'Active' as well. All other parameters should be set to 'Inactive'.



List of all possible fit parameters

	Value	Name	Type	Active	Low Limit	High Limit
57	0.100000001	Pigments volume fraction		Inactive	0	0
58	50	Pigments number of points		Inactive	0	0
59	30000.300781	Pigments spectral maximum		Inactive	0	0
60	10000	Pigments spectral minimum		Inactive	0	0
61	1309.544677	Pigments scatter		Inactive	0	0
62	188.3412933	Pigments absorption		Inactive	0	0
63	1000	Pigments spectral position		Inactive	0	0
64	1.700000047	Pigment I: n		Inactive	0	0
65	0.100000001	Pigment I: Absorption band: Gauss-Lorentz-		Inactive	0	0
66	2000	Pigment I: Absorption band: damping		Inactive	0	0
67	1000	Pigment I: Absorption band: oscillator streng		Active	0	0
68	10000	Pigment I: Absorption band: resonance frequ		Active	0	0
69	1.5	Binder: n		Inactive	0	0
70	0.100000001	Binder: UV: Gauss-Lorentz-switch		Inactive	0	0
71	5000	Binder: UV: damping		Inactive	0	0
72	10000	Binder: UV: oscillator strength		Inactive	0	0
73	40000	Binder: UV: resonance frequency		Inactive	0	0

4. Close the list. Another window should popup (if not, push the button '**Show active fit parameters**'). This is a list which displays the parameters that are varied in the fit:

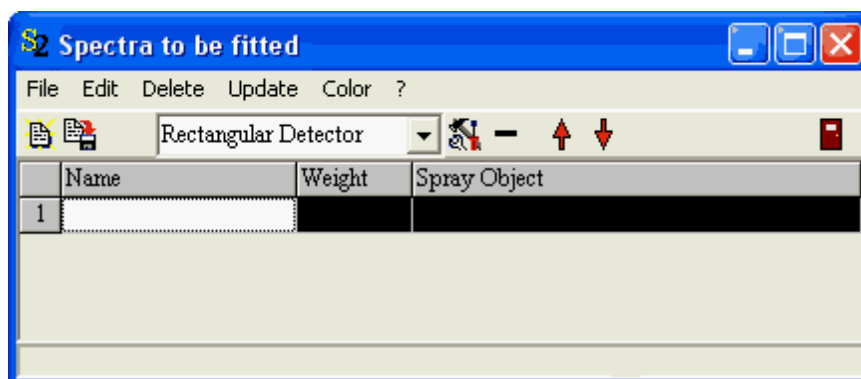


List of active fit parameters

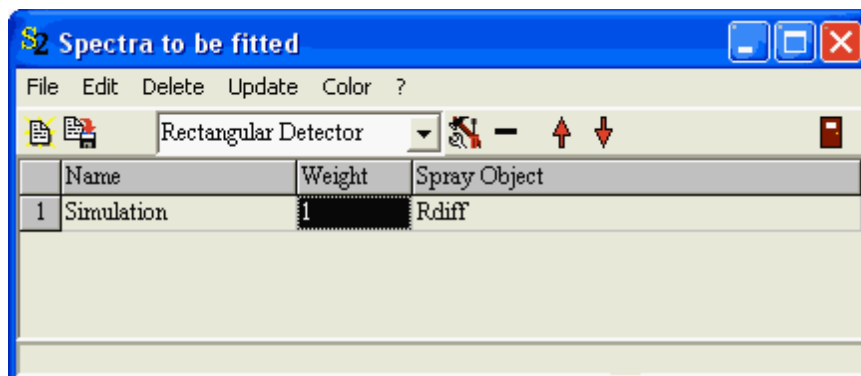
	Value	Name	Type	Active	Low Limit	High Limit
1	10000	Pigment I: Absorption band: resonance frequ		Active	0	0
2	1000	Pigment I: Absorption band: oscillator streng		Active	0	0

Keep this window open in order to watch the changes of the fit parameters during the optimization.

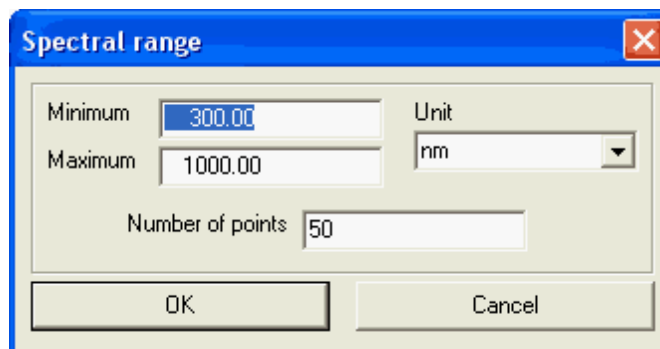
5. Now press the '**Spectra**' button in the SPRAY optimizer window. This opens a list of spectra that are used to define the target of the fit:



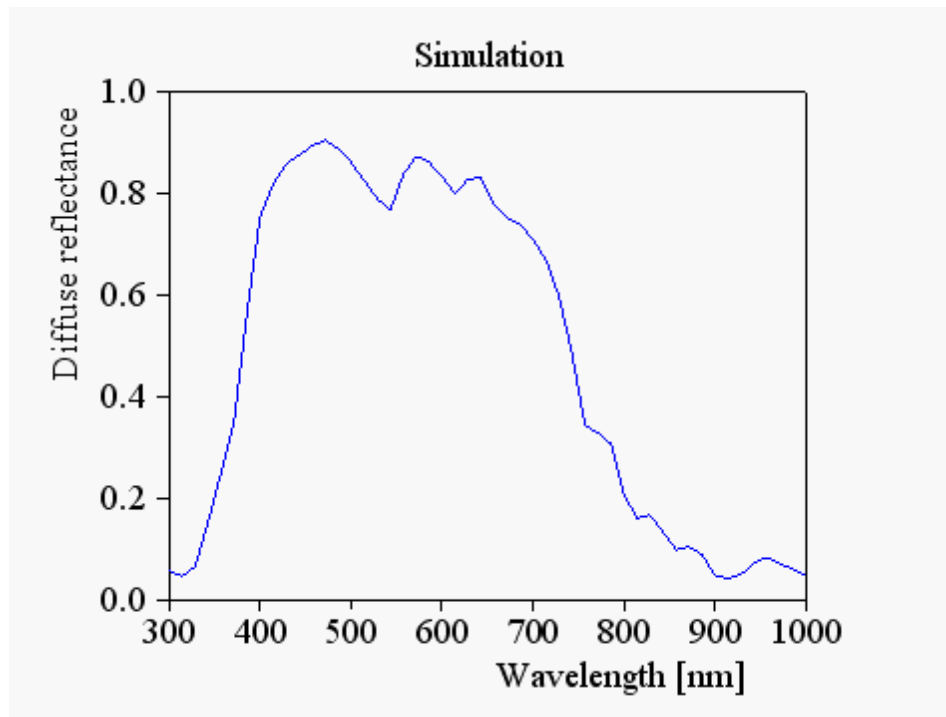
Up to now, you can only use spectra computed by 'Rectangular detector' objects in SPRAY for a comparison to measured spectra. **In order to fit the spectrum of a rectangular detector just drag it from the list of geometric objects in SPRAY to the list 'Spectra to be fitted'.** In our case, the window looks like this after drag&drop of the detector object:



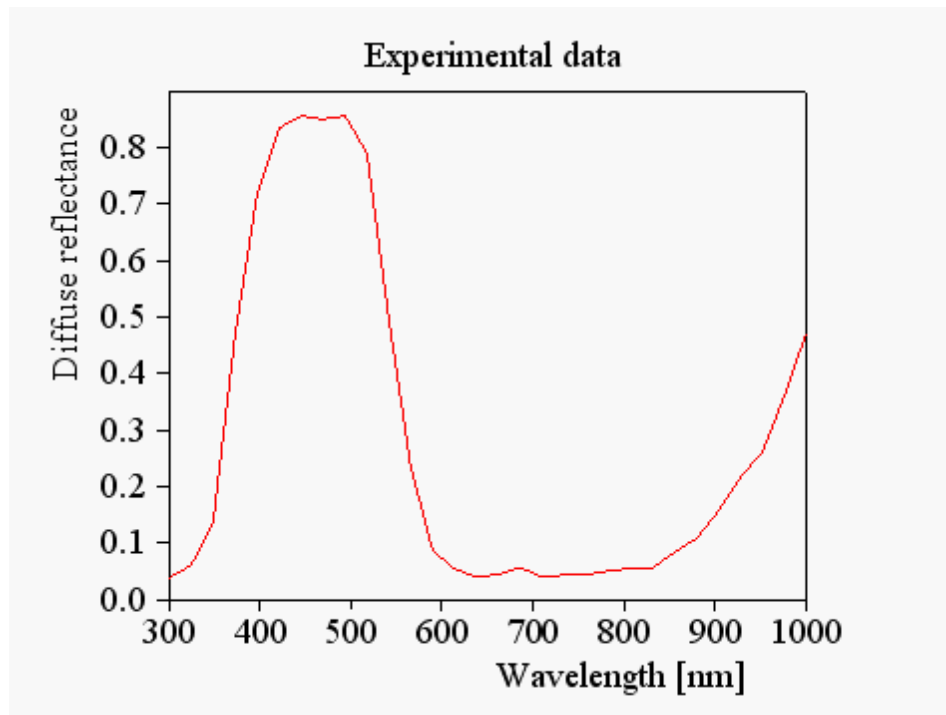
6. Use the **Edit** command to open the spectrum window. With the **Range** command you have to select the spectral range for the comparison of measured and simulated spectra:



With the command **Receive data** you can transfer the current simulated spectrum of the SPRAY detector object to the spectrum which is used for optimization:

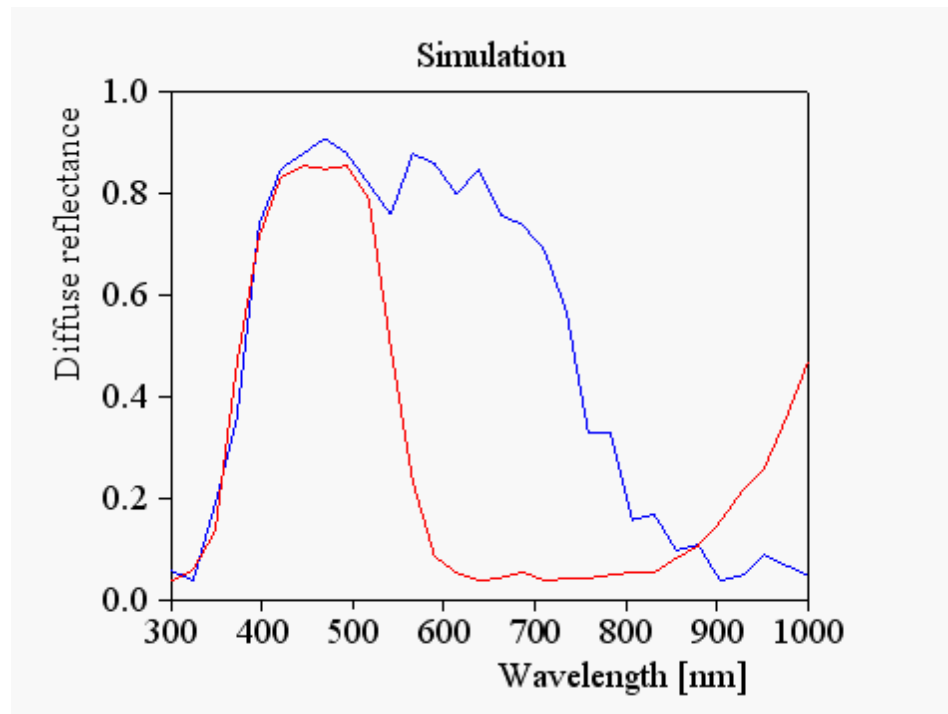


7. Now use the **Experiment** command to open the subwindow which is used to import a measured spectrum. In this window, use the **Import** command and load the spectrum from the file `measured_spectrum.std` (standard file format):



8. Close the window with the experimental spectrum, and verify that the spectrum window now

shows both the simulated and the measured spectrum:



The configuration work is done now and we can proceed to the final fit action.

10.2.4 Running the fit

Finally we can start the fit pressing the 'Start fit' button in the optimization window:

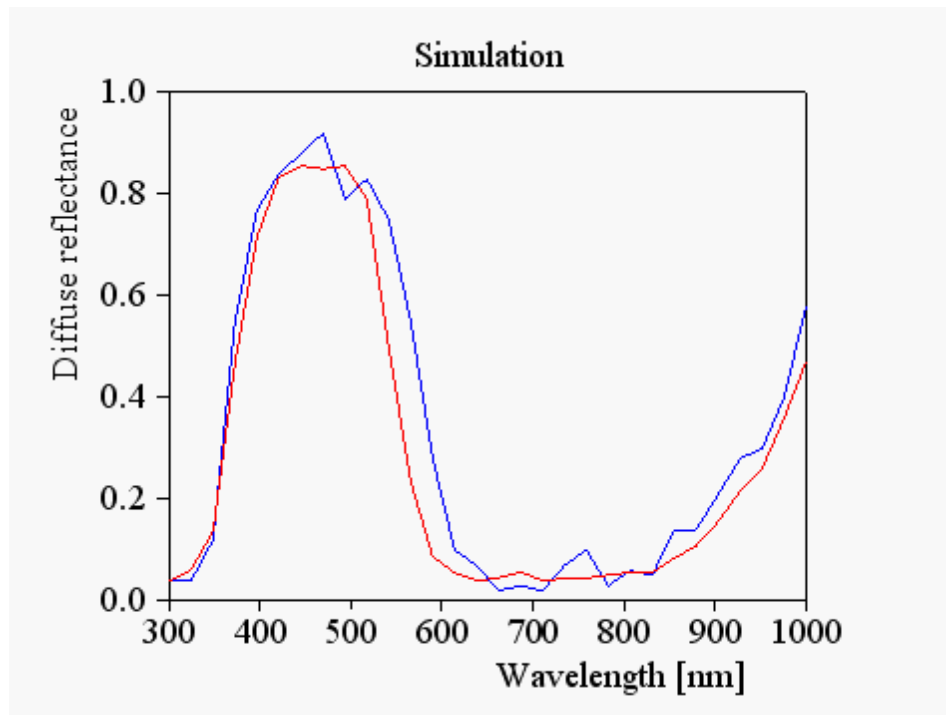


It will be useful to keep the comparison of the spectra and the values of the fit parameters visible on the screen.

While doing the fit, your PC will be blocked almost completely, and you should patiently do something else while the computer works for you.

If one of the parameters changes the properties of one of the extended Mie scatterers, the Mie computation is done automatically before the SPRAY simulation is started. This can cause a significant amount of additional computational effort - you can see the Mie program window flashing up from time to time in the background.

After a long time, the fit algorithm will stop by itself. However, you can press the 'Stop fit' button anytime. In this case, SPRAY will finish the current iteration of the fit, load the optimized fit parameters and re-do the simulation a last time with the best parameters. The fit is finished when you get the message 'Fit finished!' in the main window.



Note that the quality and the progress of the fit depends on the 'noise level' of the simulation: If you take too many rays per spectral point, the optimization will take a very long time. If the number of rays is too small the noise level is too high, and the optimization will be difficult in the final stages, because the best values of the parameters may not lead to the smallest deviation between simulation and experiment. You have to experiment a little in order to find the most efficient balance between speed and quality.

Finally you should not forget to save the SPRAY configuration with the optimized parameters.

11 References

11.1 References

Here are some useful WWW links:

SCOUT technical manual: www.mtheiss.com/docs/scout2/index.html

CODE (Coating Designer) manual: www.mtheiss.com/docs/code/index.html

Data factory manual: www.mtheiss.com/docs/data_fac/index.html

Digit manual: www.mtheiss.com/docs/digit/index.html

Graphics course: This document describes handling and features of 2D and 3D graphics used to display data in SCOUT, CODE, SPRAY and all other programs developed by *M.Theiss Hard- and Software*. It is distributed with printed program manuals.

Platypus Animator: www.c-point.com

This shareware tool made by C Point Pty Ltd creates video sequences from a list of input pictures.

Index

- 2 -

2D 39, 76

- 3 -

3D 39, 76

- A -

a* 67

About SPRAY 11

absorber 43

Absorbing 62, 67, 71, 76

absorption 19, 23, 25, 32, 36, 38, 44, 46, 85

absorption coefficient 27, 47

absorption spectrum 85, 85

Activation 56

activity 11

air/water surface 159

algorithm 19, 143

amplitude 115, 141

angle 38, 90, 99, 151

angle dependence 47

angle distribution 79

angle resolution 47, 79, 140

angles of incidence 9

angular dependence 79, 140

aperture 106

array 71, 76, 85

Array detector I 76

Array detector III 79

Array detectors (OLE) 155

array_detector_value 155

assignment 122

ATR 99

ATR crystal 99

automation 150

automation clients 150

automation server 150

average 141

average volume 38

averaging 9

axis vector 92, 125

- B -

b* 67

background 154

bitmap 134

bitmaps 155

black 67

blue light 79

bottom halfspace 27, 43, 47

box 64, 96, 101, 103

brightness 71

Buttons 11

- C -

C++ 150

camera 134, 134

camera views 56

Cameras 11

CCD camera 71

center 125

charts 150

Circle 89, 113, 122, 125

Circular aperture 106

Circular interface 89

circular light source 32, 62, 79

client PC 143, 145

clients 150

clipboard 67

Closed cylinder 92

Closed volumes 85

clouds 23

cm 21

coating 27, 79

Coating Designer 67

CODE 67

Collect 11, 39, 67, 76

collections of spheres 25

color 11, 67, 134

color coordinates 67

color dialog 56

colors 56

Comments 11

Complex objects 122, 125, 127

Composite scatterers 36

composition 36
 computational speed 143
 computational time 122
 concentrator 94
 Cone 9, 62, 94, 115, 125
 cone angle 60, 62, 79
 configuration 147, 150
 Connect 145
 Continuous scattering media 23
 continuum 23
 continuum scatterers 43
 control operation 150
 Converging lens 106
 coordinates 122, 125, 127
 Copy 56, 127
 corner 125
 CreateObject 150
 critical angle 9
 cross product 60, 88
 cross section 38
 Cylinder 9, 32, 92, 94, 108, 113, 125
 cylinders 94
 cylindrical surface 92

- D -

D65 67
 Data Factory 11, 32, 44, 67, 111
 database 11, 22
 daytime 143
 De-activation 56
 Declaration 150
 delay 147
 Delphi 150
 Demo video 159
 density 23
 design 9
 destroy 150
 detected fraction 141
 detected fraction (OLE) 155
 detector 67, 76, 79, 155, 155
 detector array 79
 detector signal (OLE) 155
 detector spectrum 155
 detectors 32
 dialog 88
 dielectric constant 22
 dielectric function 22

dielectric functions 25
 diffuse 46
 Diffuse reflection 44
 diffusion 44
 diffusor 44
 Digit 44
 dimensions 85
 direction 19, 79
 display 67
 distance 19, 56, 111, 151
 distributed computing 11, 143, 144, 147
 distribution 25
 distribution of radiation 9
 Diverging lens 108
 Doing the simulation (OLE) 154
 DOS box 27
 drag 44
 drag&drop 22, 36, 44, 58, 67
 Duplication of objects 56
 dust 23

- E -

edges 44
 efficiency 25
 Ellipsoid 101, 125
 emission cone 62
 energy 32, 140
 energy shift 32
 examples 11
 Excel 127, 150, 154
 Export 67
 Extended Mie scatterers 27

- F -

F4 122
 F5 122
 F6 122
 F7 122
 fiber 113
 file formats 67
 filename 150
 flight 155
 fluorescence 19, 32, 36
 fluorescent Mie scatterers 36
 Fluorescent scatterers 32

focal point 101
focus 103
folder 143
foreground 154
formula 127
forward scattering 79
frame 155
frequency 19, 140
Frequency scan 21
function 111
function key 122

- G -

General scatterer 24, 79
geometric objects 11, 56
Geometric objects (OLE) 151
geometric scenery 134
geometrical objects 85
geometry information 122
geometry objects 85
glass 115
glass brick 96
glass prism 97
graph 76
graphics 11
graphics course 11
grave 85, 85
gray level 71

- H -

halfspace 22, 27
halfspaces 86
Hardware protection 11
Height vector 97
Hide 150
hit 19
hit point 19
host 36
host material 25, 38
How many rays 141

- I -

identification 11
illumination 18

import 24, 122
import RT data (OLE) 151
infinity 140
Information panel 11
infrared spectroscopy 99
inhomogeneities 44
ini 155
initialization 19
inside 85
installation 143, 145
intensity distribution 71
intensity needles 18
interactions 140
interface 9, 22, 43, 43, 44, 46, 85, 89, 122
interface assignment 122
interface objects 43, 140
interfaces 85, 86
Interfacing 21
interference 79
internal reflection 159
inverted 115
isotropic scattering 32

- K -

K 24, 36

- L -

L* 67
LabView 150
Lambertian 44, 46
law of reflection 44
layer 47
layer stack 22, 27, 106
Layer stacks 11, 47
layered spheres 27
LayeredSphere.exe 27
length
 unit 21
lens 106, 108
licence information 11
light beam 159
light scattering 36, 58
Light source 58, 60, 64
light sources 58
Linear arrays 76

list 36, 56
 list of dielectric functions 22
 list of interfaces 27
 list of materials 85
 list of scatterers 23, 36, 43
 lists 11
 literature data 22
 Loading a configuration (OLE) 150
 Location 60, 62, 64

- M -

macro language 150
 macros 150
 macroscopic volume 85
 Main window 11
 master PC 11, 143, 144
 material 22, 85
 Materials 11
 maximum 140
 Maximum (OLE) 154
 metafile 67
 Mie 25, 27
 Mie scatterer 36
 Minimum (OLE) 154
 minimum 140
 mirror 43, 103
 mixture 36
 Modifying pictures 155
 multilayer stack 79
 multiple scattering 27, 44, 79

- N -

n(host) 25
 nanometer 39
 network 143, 144, 145, 147
 night 143
 NIGHTSHIFT 143, 145
 noise 141
 noise amplitude 141
 noise level 67
 Normal Vector 89, 113
 normalization 9
 Number of rays (OLE) 154

- O -

Object 150
 object colors 56
 object generation 11
 object oriented programming 18
 Object parameters (OLE) 151
 object_parameter 151
 object_parameter_string 151
 objects 11, 21, 56, 85
 observation point 134
 observation window 134
 Observer 134
 observer_x 151
 observer_y 151
 observer_z 151
 off-axis paraboloid 103
 Offset 122
 OLE 150, 151, 154, 155, 155, 159
 OLE automation 127, 143, 144, 145, 150
 OLE commands 154
 OLE server 150
 Open cylinder 94
 open structures 85, 86
 optical constant 22, 58
 optical constants 23, 25, 47
 optical setups 9
 optimize 151
 Orientation 60, 62, 64, 106, 108

- P -

paint 36
 paints 23
 Paraboloid 103
 parallel beam 103
 parallel light 79
 Parameters 11, 140
 parameters (OLE) 151
 particles 25
 Paste 127
 peak 32, 79
 pending task 143, 145
 Perfect absorber 43
 Perfect mirror 43
 periodic 115

photograph 134
pictures 18, 155
pixel 18, 134
pixel (OLE) 155
pixel array 134
pixels 71, 76
plane 103
plastic 115
Platypus Animator 155
Point light source 58
polarization 19, 47, 133
polarizations 9
Polarizer 133
position 19, 58
positioning text 155
pre-defined interfaces 43, 43, 85
Principle 18
print 67
priority 19, 56
Prism 97
probabilities 36
probability 23, 27
progress bar 11, 142
projection 133
propagation direction 18
Properties 44
pyramid 115

- Q -

quantum efficiency 36
queue 19

- R -

Radiation Transfer 24
radius 27, 89, 90, 90, 92, 94, 106, 106, 108, 113, 125
radius distribution 25
radius vector 125
rainbow 79, 159
random number 133
randomness 141
ray 23, 47
ray tracing 18
Ray vector 62
rays 9, 23, 134, 141, 154

Rays per task 144
Ray-tracing 21
rectangle 60, 71, 86, 96, 99, 106, 111, 122, 125, 133
Rectangle points 125
rectangular box 85, 96, 101, 103
Rectangular detector 67
rectangular detector (OLE) 155
Rectangular interface 86
Rectangular light source 60
red light 79
re-emission 32
reflectance 9, 44, 46, 47, 140
reflection 23, 46
refraction 23, 79, 86
refractive index 25, 38, 79, 99
Registration 150
Rendered view 134
Rendered views 134
resolution 71, 79
results 143
Retrieving results 155
RGB 67
Rotation 134
RT data 39, 151
RT file 24
RT file format 38
RT files 79
rt-files 25
RTMIE 25

- S -

S 24, 36
save_bitmap 155
save_bitmap_auto 155
save_detector_bitmap 155
Saving a configuration (OLE) 150
Saving pictures (OLE) 155
scattered radiation 79
scatterer 23, 24, 25, 32, 36, 58, 58
scatterer_load_rt 151
scatterer_parameter 151
scatterers 27, 43, 79
Scatterers (OLE) 151
scattering 23, 25, 32, 36, 38, 44, 79
scattering angle 23, 44
scattering coefficient 27

scattering event 23
 scattering media 23
 scattering particles 38
 scattering probabilities 39
 scenery 134
 SCOUT 9, 11, 21, 22, 47
 scratches 44
 screen 9, 71, 150
 screen fraction 155
 screen_value 155
 Screens (OLE) 155
 scripts 150
 server 143, 150
 shape 23, 111, 115, 122
 shift key 56
 Show 150
 silicon prism 99
 silicon wafer 85
 simple_detector_value 155
 simulation of optical devices 18
 simulation of pictures 18
 simulation parameters 11, 140
 Simulation parameters (OLE) 154
 simulation thread 154
 Simulaton 11
 sine 115
 single scattering 38, 79
 sinusoidal surface 111
 size 25
 size distribution 27
 sky 79
 slider 39, 67, 71
 small particles 44
 smooth interface 44
 Source DF 25
 spectral distribution 36
 spectral points 79, 140
 spectral points (OLE) 154
 spectral position 85, 155
 Spectral range 36, 67, 134, 140
 spectral range (OLE) 154
 spectral unit 25
 spectrum 67
 Specular 44, 46
 Specular and diffuse 46
 speed 143
 Sphere 23, 25, 43, 79, 90, 90, 115, 122, 125
 sphere segment 79, 90, 106, 108

spheres 27
 Spherical detector arrays 79
 SPRAY algorithm 32
 SPRAY tutorial 11
 Spray_Remote 150
 spreadsheet program 150
 standard deviation 141
 Start options 142
 Start simulation 11
 start_grafx 155
 status 154
 Stop 11
 Strategy 147
 subfolder 143
 Subobject 125
 subobjects 122
 sun 79
 sunlight 79
 surface 43, 47, 85, 113, 115
 surface normal 60, 62, 67, 71, 76, 88, 96, 97, 125
 surface_formula 151
 surrounding material 58
 surrounding medium 79
 symmetry axis 92, 108
 symmetry axis 90, 103, 106

- T -

tables 150
 Target 134
 target_x 151
 target_y 151
 target_z 151
 tasks 143
 test rays 134, 134
 test_ray 155
 text 155
 text files 122
 texture 115
 textured surface 85
 Thickness 47, 108
 thin film 85
 thread 142, 154
 tolerance 19
 Tools 11
 top halfspace 27, 43, 47
 total reflection 9
 transmittance 9, 44, 46, 47, 140

transparent 58, 79
Triangle 88, 97, 122, 125
tutorial 11

- U -

unit 140
USB port 11
user dialog 21
user-defined interfaces 43, 85, 99
User-defined surface 111, 113
user-defined surface shape 151

- V -

vector 125
vector 1 60
vector 2 60
vector graphics 67
vector_1 151
vector_2 151
version 2 11
video generation 155, 159
video sequence 155
view 134
View Data 67
view pictures 155
view_parameter 151
View_RT 24, 38, 39
views 11, 39, 56, 134
Views (OLE) 151
virtual reality 18
visible 67
VisualBasic 127, 150, 150
volume 38
volume fraction 24, 25, 27
Volume light source 64
volume_fraction (OLE) 151
voxel 85

- W -

W(Q) 24
waiting queue 19
water 79
water film 27
water sphere 159

wavelength 79, 140
wavenumber 38, 39
wavenumber range 25
wavenumbers 140, 155
white 71
Windows metafile 67
Windows Scripting Host 150
Workbook 25, 27, 32, 67, 76, 122, 127
write_ini_file 155
WSH 150

- X -

X 67
x-coordinate 58

- Y -

Y 67
y-coordinate 58

- Z -

Z 67
z-coordinate 58

Endnotes 2... (after index)

Back Cover