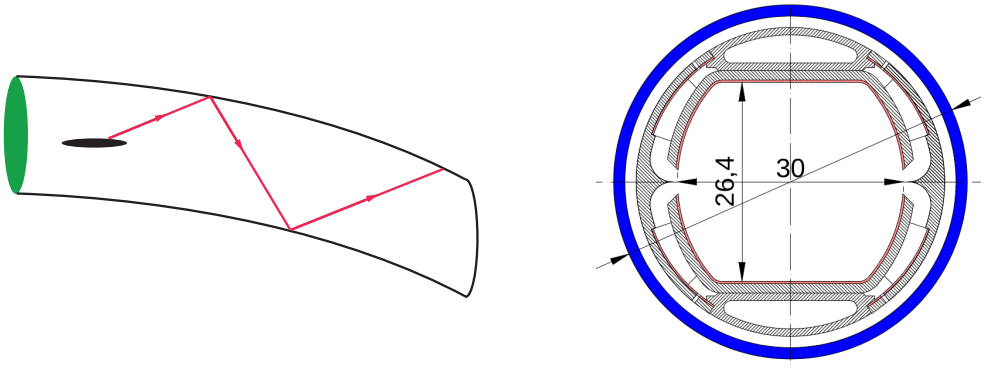

Synrad3D Photon Tracking Program

David Sagan and Gerry Dugan

August 8, 2018

Contents

Chapter 1

Introduction

Synrad3D is a program which simulates the production and scattering of synchrotron radiation generated by an electron beam in a high energy machine. The program was written by David Sagan and the photon scattering model was developed by Gerry Dugan, both of Cornell University.

Synrad3D is built atop the the Bmad software library [**?**]. The Bmad library, developed at Cornell, has been developed for modeling relativistic charged particles in storage rings and Linacs, as well as modeling photons in x-ray beam lines.

The motivation for developing Synrad3D was to estimate the energy and position distribution of where SR photons are absorbed. This is a critical input to codes which model the growth of electron clouds. Synrad3D has also been used to model radiation mask efficiency. Synrad3D includes scattering from the vacuum chamber walls, based on X-ray data from an LBNL database [**?**] for the smooth-surface reflectivity, and an analytical model [**?**, **?**] for diffuse scattering from a surface with finite roughness. Synrad3D can handle a wide variety of vacuum chamber profiles. Synrad3D also handles a wide variety of machines including machines whose geometry is not planar and machines using electrons or positrons as well as machines using protons or antiprotons.

Synrad3D is not to be confused with an older program called Synrad. The Synrad program is used for calculating wall heating from the primary beam. Synrad only tracks photons in the horizontal plane and does not simulate the scattering of the photons from the wall. The advantage of Synrad is that it is fast and directly gives wall heating numbers. The advantage of Synrad3D is that it tracks photons in three dimensions and it does simulate scattering.

Chapter 2

Running Synrad3D

2.1 Syntax

Syntax for invoking Synrad3D:

```
synrad3d {options} {<master_input_file_name>}
```

The <master_input_file_name> optional argument is used to set the master input file name. The default is "synrad3d.init".

The command line options for Synrad3D are:

```
-out <index>  ! For creating a file of the starting position of the
              !   photon with index <index> (§??.
-plot <what>  ! For viewing diagnostic plots (§??).
-test <what>  ! For generating diagnostic files (§??).
```

Examples:

```
synrad3d my_input_file.init
synrad3d -plot xy  my_input_file.init
```

2.2 -plot Option

The -plot <what> option is used for diagnostic purposes. When the -plot option is used, no photon tracking is done and instead a plot window is displayed and Synrad3D will ask for the appropriate parameters for making a plot. One of four types of plots is chosen by the setting of <what>:

```
-plot reflect ! Plot photon reflection probability curves.
-plot xy      ! Plot cross-sections in the x-y plane.
-plot xs      ! Plot cross-section in the x-s plane.
-plot ys      ! Plot cross-section in the y-s plane.
```

The `-plot reflect` option is for viewing photon reflection probability curves.

The `-plot xy` option is for viewing of wall cross-sections in the x-y plane at any point in the machine. The `-plot xs` option plots the wall outline in the x-s plane at $y = 0$. The `-plot ys` option plots the wall outline in the y-s plane at $x = 0$.

When plotting x-s or y-s cross-sections, if the Master Input file specifies a `photon_track_file`, this file will be read and the photon trajectories plotted. If the `photon_track_file` does not exist, and a `wall_hit_file` file does, the `wall_hit_file` will be read and the photon trajectories plotted. Since photon tracking is *not* done when there is plotting done, the `wall_hit_file` must be generated in a previous run of `Synrad3D` prior to using the `-plot` option.

When plotting x-y cross-sections, symbols will be drawn at the vertex points `v(i)` that define the cross-section.

`plot_param` structure, which can be set in the `synrad3d_parameters` namelist (§**??**), can be used to to set the size of the plot window, or to vary the number of points used to construct curves. The components of the `plot_param` structure are:

```
plot_param%n_pt          = 1000            ! Num points used for plot curves.
plot_param%window_width  =  800            ! Plot window width in pixels.
plot_param%window_height =  400            ! Plot window height in pixels.
```

The default values are shown.

2.3 -test Option

The `-test <what>` option is for generating diagnostic data. Tests are also useful for generating data for plotting reflection statistics. Possible tests are:

```
-test monte_carlo_reflection ! §??
-test specular_reflection    ! §??
-test diffuse_probability    ! §??
```

See Chapter §**??** for more details.

Chapter 3

Simulation Technique and Physics Models

3.1 Overview

The diffuse scattering theory used in `Synrad3D` is discussed by Dugan and Sagan[**?**] modified as discussed in §**??**.

The `Synrad3D` program uses the Monte Carlo method for photon generation, scattering, and absorption calculations.

For photon generation, a section of the machine (which may be the complete machine) is designated for photon generation. The total number of photons generated is set by the user. `Synrad3D` calculates how many photons need to be generated within each machine element. Both circular and "linear" (that is, non-circular) machines can be simulated. In both cases, the orbit of the charged particle beam, which may be non-zero, sets the centroid position and angular orientation of the generated photons. The local bending field at the beam orbit is used to determine the photon spectrum. Thus, for example, radiation for an offset beam in a quadrupole is included.

The photon is tracked from the point of origin to the point at which it hits the vacuum chamber wall. The angle of incidence relative to the local normal to the vacuum chamber is computed. A scattering probability is computed, based on this angle and the photon's energy. Depending on the value of this probability, the photon is either absorbed at this location, or scattered. If it is scattered, the scattering is taken to be elastic. That is, photon energy does not change. This ignores any florescence. Surface roughness, on the other hand is taken into account so there is a diffuse component to the scattering.

The photon is then tracked to the next encounter with the vacuum chamber wall, and the probability of scattering is again computed. This process continues until the photon is absorbed.

3.2 Photon Generation

Photon generation is based on the standard synchrotron radiation formulas, applicable for dipoles quadrupoles, and wigglers. The radiation is assumed to be incoherent, so this program cannot treat undulator radiation. Polarization of the photon is ignored.

`Synrad3D` slices up each element longitudinally and generates photons from each slice. The number of photons generated in a slice weighted by the local probability of photon emission which depends on the local orbit curvature.

Both circular and "linear" (that is, non-circular) machines can be simulated. In both cases, the orbit of the charged particle beam, which may be non-zero, sets the centroid position and angular orientation of the generated photons. Photon generation is based upon the local field along the beam orbit. Thus, for example, off-axis photons in a quadrupole will produce radiation. The beam orbit is calculated from such things as the settings of steering elements, element misalignments, etc. as given in the lattice file. For circular machines, the beam orbit is the closed orbit. For linear machines, the starting beam position as set in the lattice file is used in the orbit calculation.

When a photon is generated at a given longitudinal position, the beam's emittances and centroid are used so that the resulting photon distribution mirrors the Gaussian positional distribution of the beam. Horizontal/vertical coupling is taken into account in this calculation. The photon energy distribution will be the standard energy spectrum of photons generated in a bend.

A photon's initial angular orientation is generated by first using a random number generator to generate an angular orientation using a probability function that corresponds to the beam's angular distribution. To this orientation, an angular offset out of the plane of the bend is added where the offset is calculated using a random number generator with a probability distribution based on the standard angular spectrum of photons generated in a bend. There is no angular offset added to the angle in the plane of the bend. The generated photons will have the proper correlation between photon energy and photon angle. Note that the bending plane of the charged particle beam need not be horizontal. For example, the bend plane orientation for an offset beam in a quadrupole will depend upon the offset.

3.3 Photon Scattering

Simulated photons are tracked until they hit the wall, where the probability of being scattered, and the scattering angle, are determined by their energy and angle of incidence. This section describes the scattering model.

Generally, the probability of specular reflection of a photon from a rough surface depends on the the rms surface roughness σ, the photon wavelength λ, and the grazing angle. An explicit formula for this probability is [**?**]

$$P_{\text{spec}} = \mathrm{e}^{-g(x,x)}, \tag{3.1}$$

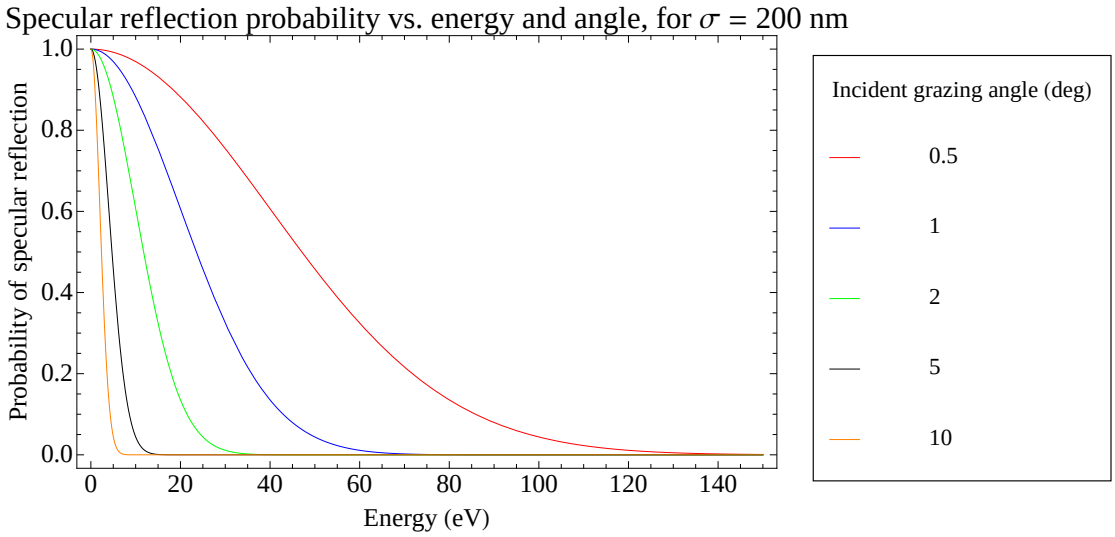

Figure 3.1: Specular reflection probability [**?**], vs. photon energy and angle, for an rms surface roughness of 200 nm.

in which

$$g(x,y) = \frac{4\pi^2\sigma^2(x+y)^2}{\lambda^2} \tag{3.2}$$

where x is the cosine of the incident polar angle, and y is the cosine of the scattered polar angle. For a typical technical vacuum chamber surface, the rms surface roughness $\sigma \gtrsim 200$ nm is greater than most of the X-ray wavelengths of interest, for all except the lowest energy photons. In this regime, except at very small grazing angles, diffuse scattering from the surface dominates over specular reflection. This is illustrated in Fig. **??**. The theory of diffuse scattering of electromagnetic waves from random rough surfaces is a well-developed subject, and is covered in detail in references [**?**] and [**?**]. The model we use assumes a Gaussian distribution for both the surface height variations (rms σ) and for the transverse distribution (equal in both transverse directions, with autocorrelation coefficient T).

The most general expression for the diffusely scattered power is complex, and involves an infinite sum. However, the expression simplifies substantially in the limit $g(x,y) \gg 1$. For very rough surfaces corresponding to technical vacuum chambers, for which typically $\sigma \gg \lambda$, this condition is satisfied over much of the region of interest. In this limit, the diffusely scattered power per unit solid angle is given by

$$\frac{dP_{\text{diff}}}{d\Omega} = P_0 \frac{\langle R\rangle}{4\pi y} \frac{(1+xy)^2}{(x+y)^4} \tau^2 \mathrm{e}^{-\frac{(2-x^2-y^2)\tau^2}{4(x+y)^2}} (1 - a\cos\phi)^2 \mathrm{e}^{b\cos\phi}, \tag{3.3}$$

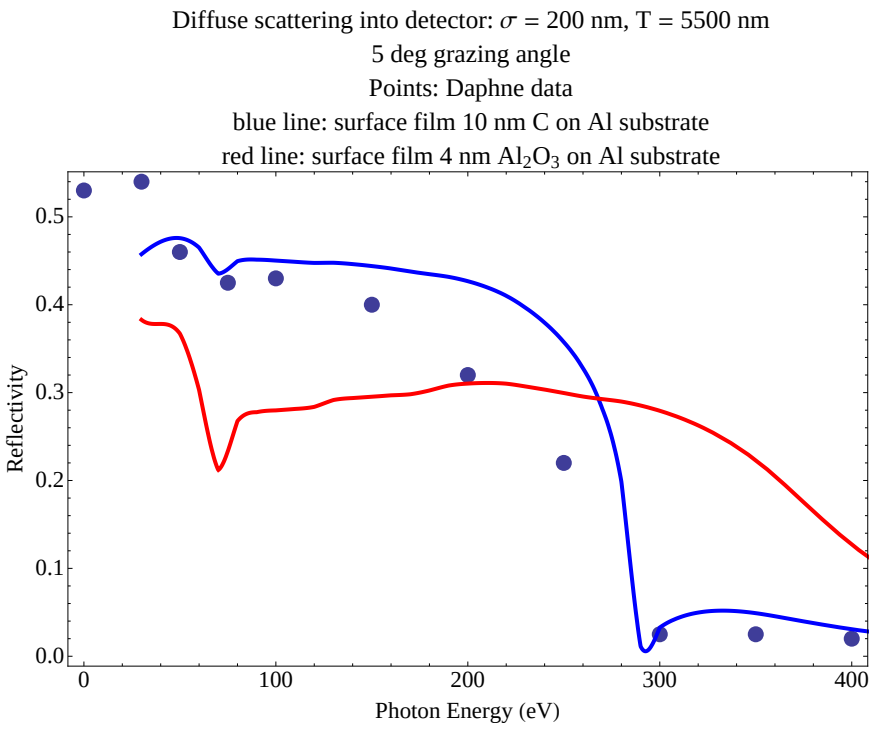

Figure 3.2: Diffuse scattering at 5 deg from a surface layer on an aluminum substrate: comparison of data and model

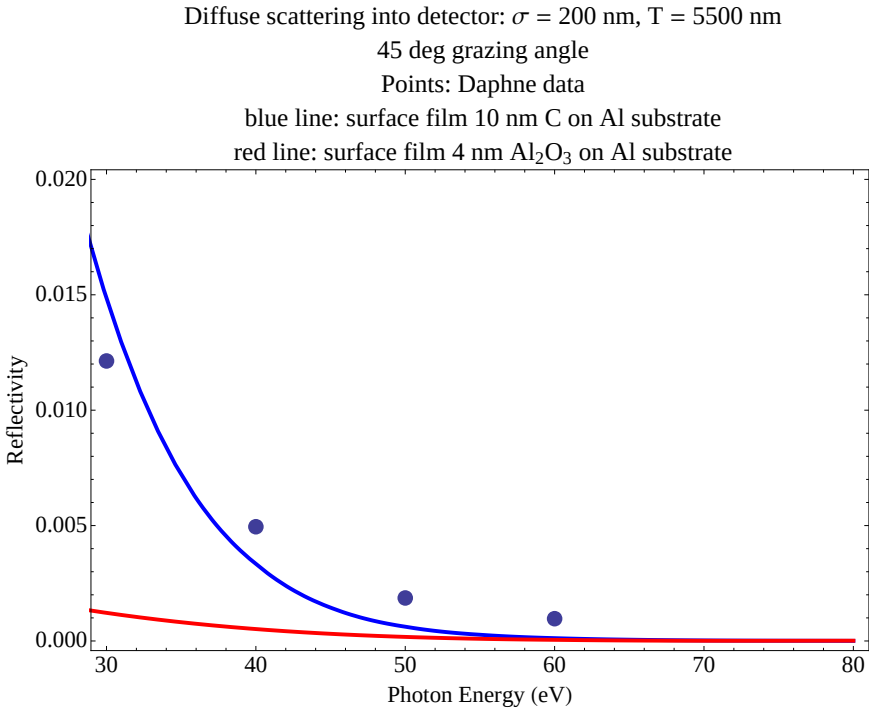

Figure 3.3: Diffuse scattering at 45 deg from a surface layer on an aluminum substrate: comparison of data and model

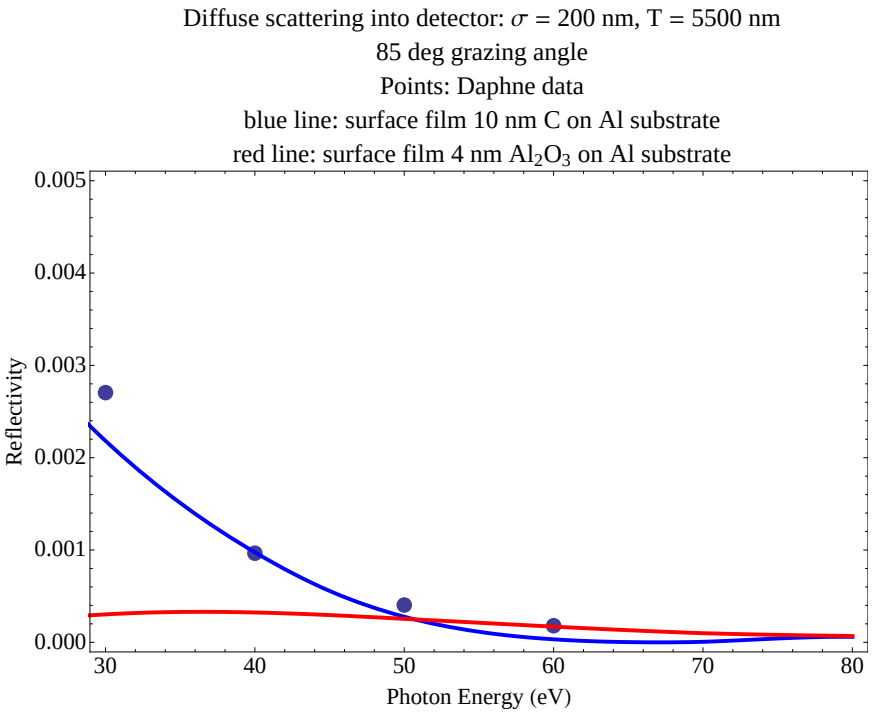

Figure 3.4: Diffuse scattering at 85 deg from a surface layer on an aluminum substrate: comparison of data and model

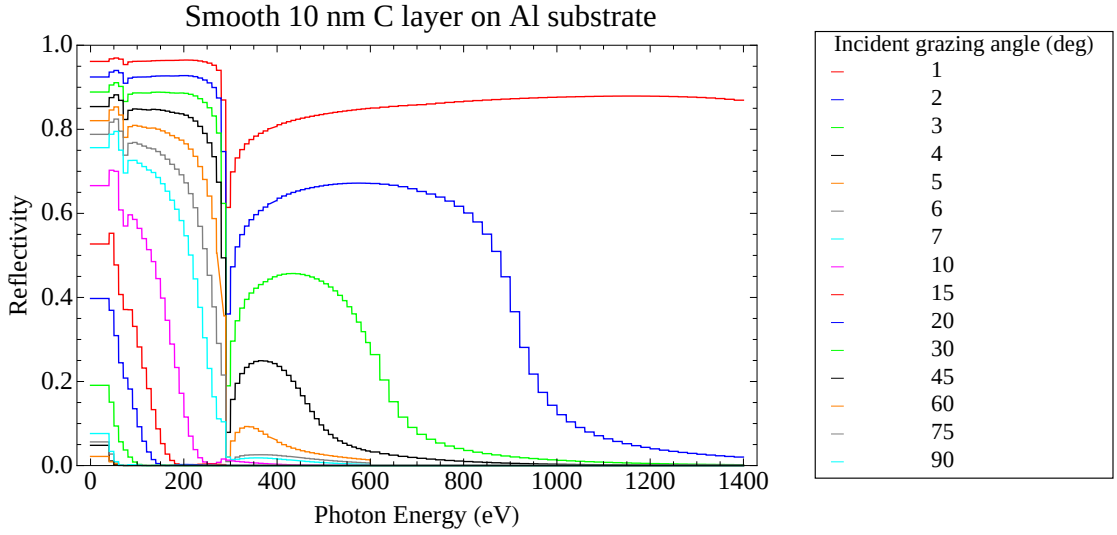

Figure 3.5: Smooth surface reflectivity for a 10 nm C film on Al substrate: from [**?**]

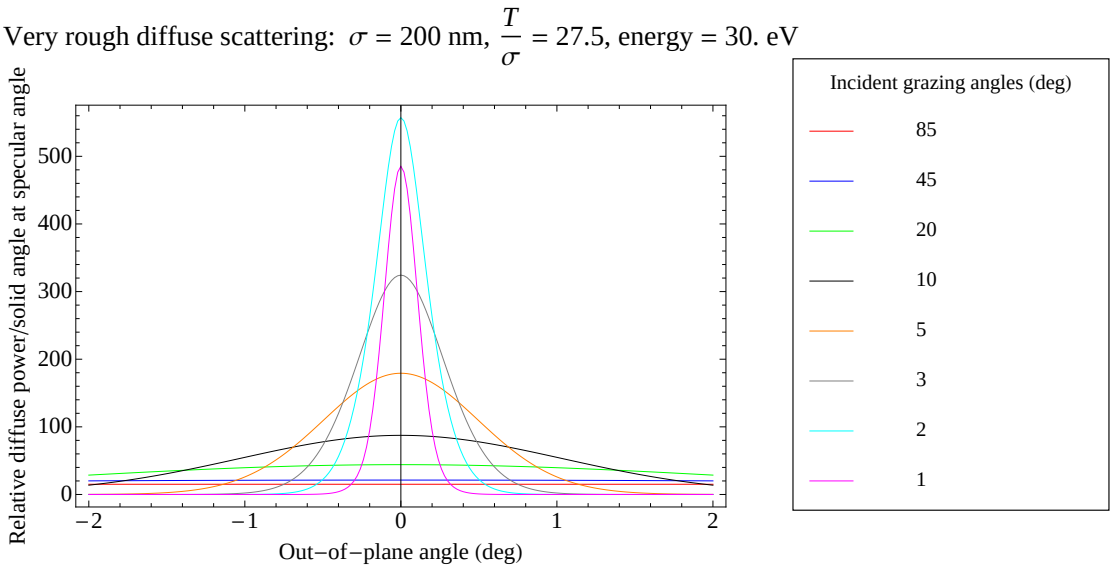

Figure 3.6: Diffuse scattering out-of-plane angular distributions for 30 eV photons

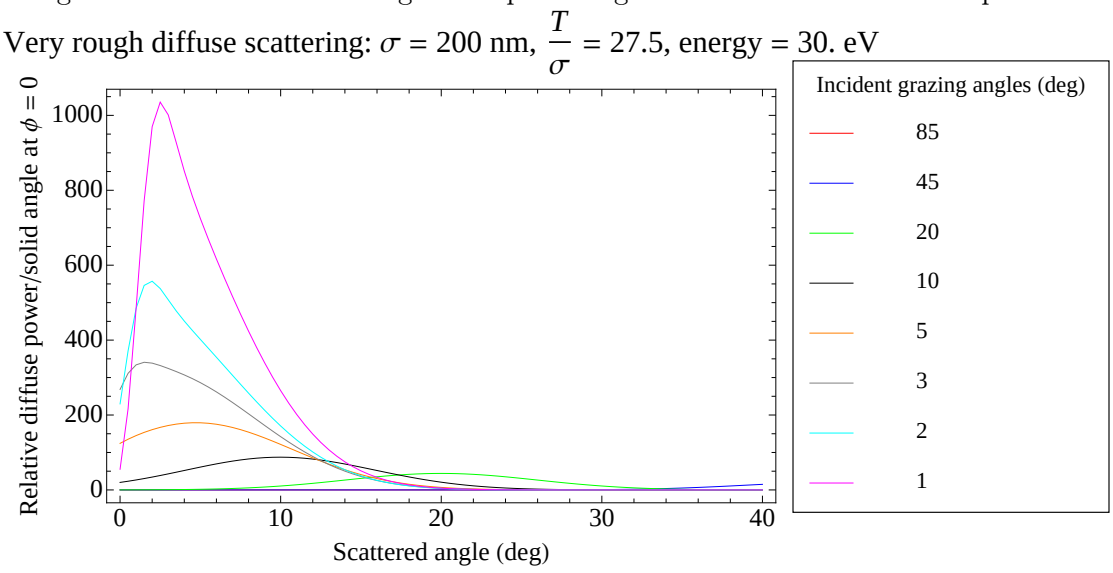

Figure 3.7: Diffuse scattering polar angular distributions for 30 eV photons

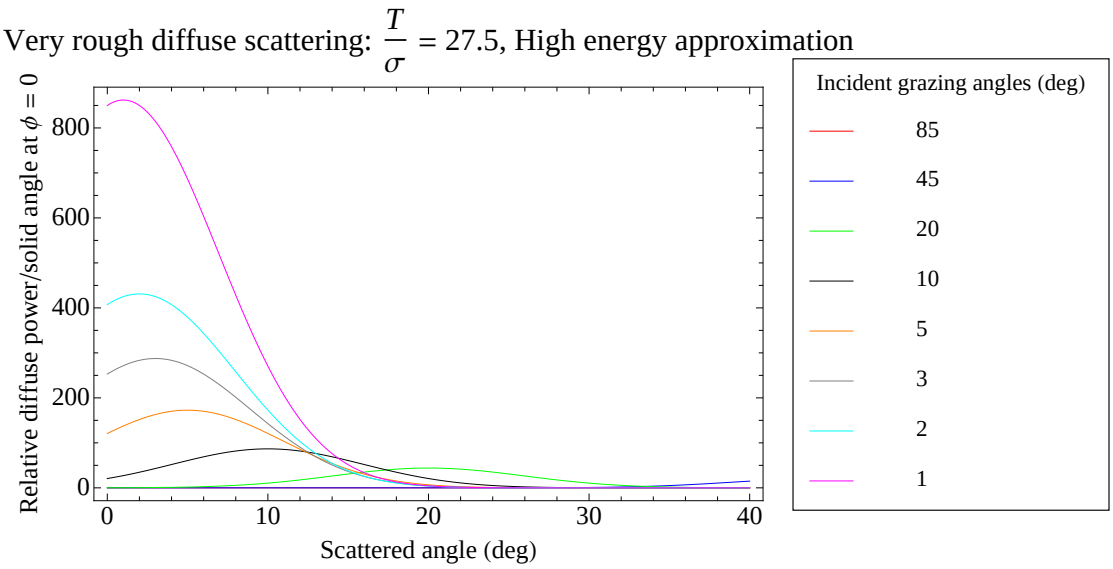

Figure 3.8: Diffuse scattering polar angular distributions for high energy photons

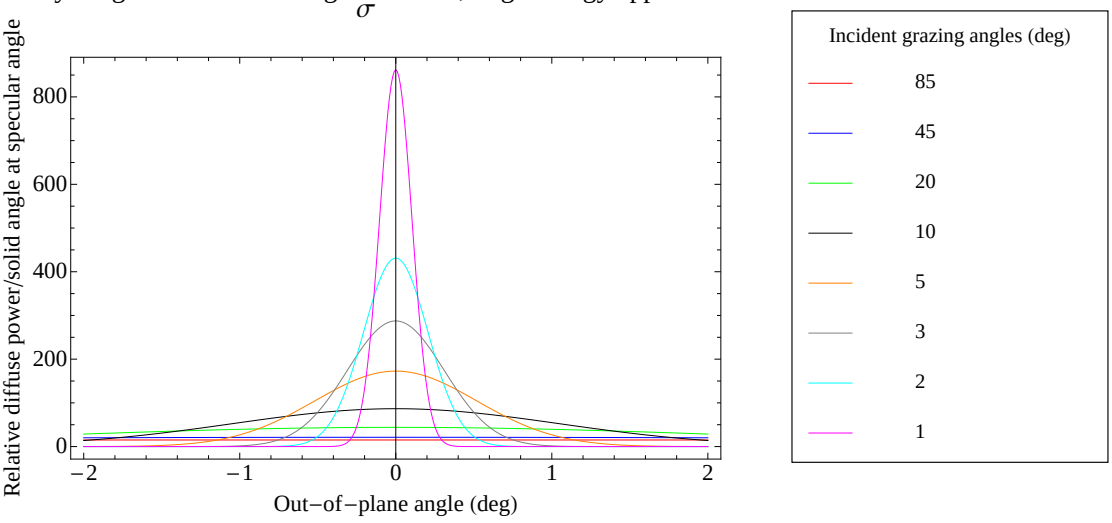

Figure 3.9: Diffuse scattering out-of-plane angular distributions for high energy photons

with

$$a = \frac{h(x,y)}{1+xy}, \tag{3.4}$$

$$b = \frac{2h(x,y)\tau^2}{4(x+y)^2}, \tag{3.5}$$

$$h(x,y) = \sqrt{1-(x^2+y^2)+x^2y^2}. \tag{3.6}$$

In this expression, P_0 is the incident power, and $\langle R \rangle$ is the smooth-surface reflectivity, which is determined by the atomic structure of the surface material. ϕ is the scattering angle out of the plane of incidence. Note that the relative power depends on the ratio $\tau = T/\sigma$, and not on the T or σ separately.

The smooth-surface reflectivity $\langle R \rangle$ depends on the atomic structure of the surface materials (including any thin layers which may be deposited on the surface). The surface roughness parameters σ and T depend on the geometry of the surface deviations from a perfect plane. These parameters may be determined from inspection of the vacuum chamber surface, for example, using an atomic force microscope.

To derive a working model for the smooth surface reflectivity and the surface parameters for a typical vacuum chamber surface, we have relied on measurements [**?**] of X-ray scattering from an aluminum vacuum chamber surface made at `DAPHNE`. For these measurements, the rms surface roughness of the sample was reported to be 200 nm.

The theory of diffuse scattering discussed above has been used, together with smooth surface reflectivity results taken from an X-ray database [**?**], to predict the scattering and compare with the measurements. From these comparisons, the best-fit value for the transverse autocorrelation parameter, T, was found to be 5500 nm. As discussed by Dugan and Sagan[**?**], it was found that the smooth-surface reflectivity corresponding to a 10 nm carbon film on an aluminum substrate was needed to fit the data. The assumption of an aluminum oxide surface film was not consistent with the data. The data and the corresponding fits are shown in Fig. **??**, **??**, and **??**.

With the smooth-surface reflectivity determined, and the surface parameters established, the scattering model in `Synrad3D` is completely determined. The model currently in use has a smooth-surface reflectivity illustrated in Fig. **??**. Diffuse scattering distributions for 30 eV photons are shown in Fig. **??** and Fig. **??**. At this low photon energy, the approximation $g(x,y) \gg 1$ does not hold in general, and the full diffuse scattering formalism is used to compute these distributions. Diffuse scattering distributions for high energy photons, for which $g(x,y) \gg 1$ are shown in Fig. **??** and Fig. **??**. These distributions have been computed from Eq. **??**.

Chapter 4

Master Input File

4.1 Fortran Namelist

Fortran namelist syntax is used for parameter input by `Synrad3D`. The general form of a namelist is

```
&<namelist_name>
  <var1> = ...
  <var2> = ...
  ...
/
```

The tag `"&<namelist_name>"` starts the namelist where `<namelist_name>` is the name of the namelist. The namelist ends with the slash `"/"` tag. Anything outside of this is ignored. Within the namelist, anything after an exclamation mark `"!"` is ignored including the exclamation mark. `<var1>`, `<var2>`, etc. are variable names. Example:

```
&place section =   0.0, "arc_std", "elliptical", 0.045, 0.025 /
```

here `place` is the namelist name and `section` is a variable name. Notice that here `section` is a "structure" which has five components – a real number, followed by two strings, followed by two real numbers.

Everything is case insensitive except for quoted strings.

Logical values are specified by `True` or `False` or can be abbreviated `T` or `F`. Avoid using the dots (periods) that one needs in Fortran code.

4.2 Example Master Input File

The master input file can be specified on the command line invoking `Synrad3D`. If not given, the default name for the master input file is "`synrad3d.init`".

Fortran namelist syntax is used (§??). A `synrad3d_parameters` namelist holds parameters for `Synrad3D`. Example:

```
&synrad3d_parameters
  ix_ele_track_start   = 1      ! Radiation region start lattice element.
  ix_ele_track_end     = 912    ! Radiation region end lattice element.
  photon_direction     = 1      ! 1 = Forward generation, -1 = Backward generation.
  num_photons          = 50000  ! Target number of surviving photons.
  num_photons_per_pass = -1     ! photons generated per pass. -1 => num_photons/5.
  ds_step_min          = 0.01   ! Photons are generated at discrete points.
                                ! Multiple photons can be generated at each point.
                                ! This is minimum distance between points.
  emit_a       = -1         ! Horizontal emit. Meters. If < 0 -> Calc from lattice.
  emit_b       = 7.52E-11   ! Vertical emit.  Meters. If < 0 -> Calc from lattice.
  sig_pz       = -1         ! Sigma pz. If < 0 -> Calc from lattice.

  lattice_file = "../lattice/cesr/bmad/bmad_6wig_8nm_2085.lat"
  wall_file    = "synrad3d.wall"   ! Vacuum chamber wall file.
  dat_file     = "synrad3d.dat"    ! Output data file.
  wall_hit_file = ""               ! Photon wall hit output data file.
  photon_track_file = ""           ! Photon track output data file
  lat_ele_file  = ""               ! Write lattice element data file.
  photon_start_input_file  = ""    ! File for initializing the photons
  photon_start_output_file = ""    ! File recording photon start positions

  random_seed = 123456                ! 0 -> Use sys clock.
  e_init_filter_min = -1              ! Min initial energy filter param.
  e_init_filter_max = -1              ! Max initial energy filter param.
  vert_angle_init_filter_min = -2     ! Min initial vertical angle filter.
  vert_angle_init_filter_max = 2      ! Max initial vertical angle filter.
  vert_angle_symmetric_init_filter = F ! Symmetrize vertical angle filter?
  e_filter_min = -1                   ! Min final energy filter param.
  e_filter_max = -1                   ! Max final energy filter param.
  s_filter_min = -1                   ! Min S position filter param.
  s_filter_max = -1                   ! Max S position filter param.
  filter_phantom_photons = T          ! Filter photons striking a phantom wall?
  num_ignore_generated_outside_wall = 0    !
  turn_off_kickers_in_lattice = F          ! Zero the closed orbit?
  surface_roughness_rms       = -1         ! Roughness for diffuse scattering.
  roughness_correlation_len   = -1         ! Roughness correlation length.
  surface_reflection_file     = ""         ! File for default reflection table.
  chamber_end_geometry = ""                   ! Wall at the lattice ends.
  sr3d_params%ds_track_step_max        = 3    ! Photon propagation step size
  sr3d_params%dr_track_step_max        = 0.1  ! Photon propagation step size
  sr3d_params%specular_reflection_only = F    ! For testing diffuse scattering.
  sr3d_params%allow_reflections        = T    ! For testing purposes.
```

```
    sr3d_params%allow_absorption         = T    ! For testing purposes.
    sr3d_params%max_reflections  = 10000        ! For avoiding endless loop problems.
    plot_param%n_pt              = 1000         ! Num points used for plot curves.
    plot_param%window_width      =  800         ! Plot window width in pixels.
    plot_param%window_height     =  400         ! Plot window height in pixels.
  /
```

4.3 Master Input File Parameters

`chamber_end_geometry`

The `chamber_end_geometry` sets how the chamber ends at the ends of the lattice are treated. `chamber_end_geometry` may be set to one of:

```
  ""            ! Use lattice geometry (default).
  "closed"      ! The chamber ends connect together.
  "open"        ! The chamber end are not connected together.
```

If `chamber_end_geometry` is set to blank (""), the setting of the lattice geometry will be used as the chamber geometry (the lattice geometry is set in the lattice file, see the Bmad manual[**?**] for more details). For a `"closed"` geometry, the chamber ends at the ends of the lattice are connected together so a photon hitting one end will reappear at the other end. For an `"open"` geometry, the camber ends at the ends of the lattice are not connected so a photon hitting an end will be reflected or absorbed.

Note: the setting of `chamber_end_geometry` only affects branch 0 (the root branch) of the lattice. If The lattice has other branches, the chamber end geometry for chambers in these branches is always determined by the `geometry` setting for the branch.

`dat_file`

This string gives the name of the output data file (§**??**). See below for more details.

`ds_step_min`

This parameter establishes the minimum distance to track the particle beam between emission points. The thought was that if `Synrad3D` decided to make very small steps between emission points, this might slow the calculation down. This has not been tested. This parameter does not influence photon tracking. Default is 0.001 meters.

`e_init_filter_min`, `e_init_filter_max`

Minimum and maximum filter values for a photon's initial energy (§**??**). A negative filter value (which is the default) indicates that the particular filter is not used.

`e_filter_min`, `e_filter_max`

Minimum and maximum filter values for a photon's final energy (§**??**). A negative filter value (which is the defaults) indicates that the particular filter is not used.

`emit_a`, `emit_b`, `sig_pz`

These parameters set the particle beam size and so will affect the starting coordinates

of the photons. A negative value (which is the default) of any of these parameters will result in `Synrad3D` using the value for the parameter from a calculation of the synchrotron radiation integrals.

`filter_phantom_photons`
: Phantom photons are photons that have hit a "phantom" wall (§**??**). That is, a wall whose surface name is set to "PHANTOM". If `filter_phantom_photons` is set the `True` (the default) a filter test is used to exclude phantom photons from the output. If set to `False`, the filter will not be applied. Default is True.

`ix_ele_track_start, ix_ele_track_end`
: The parameters `ix_ele_track_start` and `ix_ele_track_end` establish the region where radiation is produced. These are the index numbers of elements in the lattice. The radiation region boundary is taken to be at the exit end of the elements so no radiation is produced in the element with index `ix_ele_track_start`. If `ix_ele_track_end` is negative, the end of the radiation region is taken to be end of the lattice. If `ix_ele_track_end` is positive and less than `ix_ele_track_start`, the track region will be from `ix_ele_track_start` through the end of the lattice and from the beginning of the lattice to `ix_ele_track_end`. Defaults are 0 for `ix_ele_track_start` and -1 for `ix_ele_track_end`.

`lat_ele_file`
: This string, if not blank `""` (and blank is the default), will create a data file listing the lattice elements within the emission region. For each element the following are given.

  1. The element name
  2. The element type (quadrupole, etc.)
  3. Longitudinal position at the exit end of the element
  4. Element length.
  5. I_0 radiation integral through the element.
  6. The nominal (not actual) number of photons to be generated based upon the I_0 integral, the total I_0 integral, and the setting of `num_photons`
  7. The longitudinal step size between photon emission points.

`lattice_file`
: This file defines the machine optics. See the Bmad manual for more details.

`num_ignore_generated_outside_wall`
: Photons may be generated outside of the beam chamber for various reasons. For example, the beam chamber can be too small or the closed orbit may lie near or outside the chamber. Another possibility is that the beam emittance is large enough so that, from time-to-time, a photon generated at large amplitude will be generated outside the wall. `Synrad3D` will ignore photons generated outside the wall, and generate another one at the same longitudinal position, up to the number set by `num_ignore_generated_outside_wall`. If the number of photons generated outside the wall exceeds this number, `Synrad3D` will print an error message and stop. Default is 0.

num_photons

num_photons establishes the minimum number of "surviving" photons that need to be generated before Synrad3D will stop the simulation. The actual number of surviving photons will be between num_photons and num_photons+num_photons_per_pass. See §??. [A surviving photon is a photon that passes all filter requirements (§??) and is recorded in the output statistics.]

num_photons_per_pass

num_photons_per_pass sets the number of photons generated per "pass". A "pass" is the act of generating photons throughout the radiation production region. If num_photon_per_pass is negative (the default), the number of photons generated per pass is taken to be num_photons/5. The actual number of surviving photons will be between num_photons and num_photons+num_photons_per See below for more details.

photon_direction

The photon_direction parameter determines in what direction the photons are traveling when initially created. A value of +1 (the default) indicates the photons are created traveling in the $+s$ direction and a value of -1 indicates that the photons are created in the $-s$ direction.

photon_start_output_file

This string, if not blank (and blank is the default), will cause Synrad3D to create an output file, with a name given by the value of photon_start_output_file, of the photon starting positions along with the state of the random number generator. The format for this file is compatible with the format of the photon_start_input_file. This file is generally used for debugging purposes and is not of general interest. Note: If photon_start_input_file is set, photon_start_output_file will be ignored (no output file is generated). Also note that the file will have the starting coordinates of all photons generated, not just the photons that pass any filtering tests. Thus, if any of the filter parameters are set, the size of the file may be very large. See §?? for the syntax of this file. Alternatively, the -out command line option (§??) may be used to create a starting position file for a particular photon.

photon_start_input_file

If not blank (and blank is the default), the file named by photon_start_input_file will be read by Synrad3D and used to initialize photon starting positions. See §?? for the syntax of this file. The state of the random number generator at the time of a photon's initialization can also be specified in the file. In this case, the random number generator state will also be set along with the state of the photon. This is useful for diagnostic purposes when one wishes to compare the results of different versions of the Synrad3D program. Note: Starting position files may be generated by setting photon_start_output_file or by using the -out command line option (§??).

If photon_start_input_file is set to "CUSTOM", instead of reading from a file, custom code is used to generate the initial photon position. Details of how this works is given in Sec. §??.

photon_track_file

This string, if not blank (and blank is the default), will create a data file with a table of

points along the photon tracks. This file is useful for plotting photon trajectories since it records intermediate points between points where the photon hits a wall. See §?? for more details

`plot_param%...`
: The `plot_param` structure contains parameters used with the `-plot` option when starting `Synrad3D`. See Section §?? for more details.

`random_seed`
: Random number seed used in by the random number generator. If set to 0 (the default), the system clock will be used. That is, if set to 0, the output results will vary from run to run and if non-zero the results will be the same from run to run.

`roughness_correlation_len`
: This parameter sets the surface roughness correlation length in meters for the default surface (§??). This is used for diffuse scattering. If negative (the default), the value of 5.5 μm will be used.

`s_filter_min`, `s_filter_max`
: Minimum and maximum longitudinal position filter values of the final photon position. A negative filter value (the default) indicates that the particular filter is not used. See §??.

`sr3d_params%allow_absorption`
: This parameter, if set False, will cause `Synrad3D` suppress absorption and to always reflect photons when they hit the chamber wall. Default is True. Setting to False is only used for testing purposes.

`sr3d_params%allow_reflections`
: This parameter, if set False, will cause `Synrad3D` to stop tracking a given photon once it hits the chamber wall. This can be used to generate data on where the primary photons are striking. Default is True.

`sr3d_params%ds_track_step_max`, `sr3d_params%dr_track_step_max`
: These parameters determine the maximum distance a photon is tracked in a given "step" see §?? for more details. Defaults are 3 meters for `sr3d_params%ds_track_step_max` and 0.1 meter for `sr3d_params%dr_track_step_max`.

`sr3d_params%max_reflections`
: This parameter sets the maximum number of reflections that a photon can have. If a photon is reflected more times than this parameter, `Synrad3D` will stop tracking the photon, print an error message, and then go on to the next photon. The offending photon will not be included in the output data file statistics. The default value for `sr3d_params%max_reflections` is 10000. Normally no photon will ever have so many reflections. `sr3d_params%max_reflections` serves as a "sanity checker" just in case, for example, some inaccuracy in the tracking causes photons not to be absorbed.

`sr3d_params%specular_reflection_only`
: This parameter can be used to test whether diffuse scattering is important. If set to `True`, photons will always be specularly reflected. Default is `False`.

`surface_reflection_file`
: This parameter is used to change the default reflection probability curves used in the simulation. See §**??** for details. To define multiple surfaces with differing reflectivities, see §**??**.

`surface_roughness_rms`
: This parameter sets the surface roughness RMS in meters for the default surface (§**??**). This is used for diffuse scattering. If negative (the default), the value 200 nm will be used.

`turn_off_kickers_in_lattice`
: If `turn_off_kickers_in_lattice` is set to True (the default is False), then all kicks from steering elements in the lattice will be zeroed.

`vert_angle_init_filter_min, vert_angle_init_filter_max`
: Minimum and maximum filter values for a photon's initial vertical angle (§**??**). Without filtering photon angles will be in the range $[-\pi/2, \pi/2]$. If `vert_angle_init_filter_min` is less than $-\pi/2$ (which is the default) a value of $-\pi/2$ is used for it. If `vert_angle_init_filter_max` is greater than $\pi/2$ (which is the default) a value of $\pi/2$ is used for it. Also see `vert_angle_symmetric_ini`
Note: If vertical angle filters are used, `e_init_filter_min` must be set. This is mandated to avoid problems that occur at zero photon energy where the inital photon distribution from a bend has infinite probability density per unit of energy and an infinite vertical angle width.

`vert_angle_symmetric_init_filter`
: Sets if the photon's initial vertical angle vertical angle (§**??**) filtering is symmetric about 0. If set to False (the default), the filter angular range is

```
[vert_angle_init_filter_min, vert_angle_init_filter_max]
```

If set to True, photons will be accepted for tracking if they are in one of the ranges

```
[-vert_angle_init_filter_max, -vert_angle_init_filter_min], or
[vert_angle_init_filter_min, vert_angle_init_filter_max]
```

`wall_file`
: This string gives the name of the vacuum chamber wall definition file. See §**??** for more details.

`wall_hit_file`
: This string, if not blank (which is the default), will create a data file listing the points where the photons hit the wall including all the reflection points. This will be in addition to the regular output file. See §**??** for more details.

Chapter 5

Vacuum Chamber

5.1 Subchambers

The shape of the vacuum chamber is described by a number of "subchambers" (the reason for dividing the vacuum chamber into subchambers is discussed in §**??**). This is illustrated in Figure **??**. The entire vacuum chamber is the union of all the subchambers. That is, a photon is considered within the vacuum chamber if, and only if, it is inside at least one of the subchambers. It is perfectly fine for subchambers to overlap one another. In fact, some overlap is desirable to prevent abutting subchambers from appearing to be separated due to round-off errors in the calculation.

5.2 Wall_file Syntax

The vacuum chamber is defined in the `wall_file`. The name of the `wall_file` is specified in the master input file §**??**. The syntax for specifying the vacuum chamber follows Fortran namelist input as explained in §**??**.

Each subchamber is defined by specifying the subchamber cross-section at a number of longitudinal positions as illustrated in Figure **??**. Each cross-section is specified using a namelist named `place`. The general form of a `place` namelist is:

```
&place
  section = <s>, "<section_name>", "<section_id>"
  surface = "<surface_name>", <is_local>
/
```

The optional `surface` parameter, which is used to specify a particular surface, is explained in §**??**. Example:

```
&place section =   0.0, "arc_std", "beam:dipole_shape" /
&place section =  74.3, "Near_IR"  "left_ante:shape-A@START" /
&place section =  82.9, "wig1",    "left_ante:shape-B@END"
```

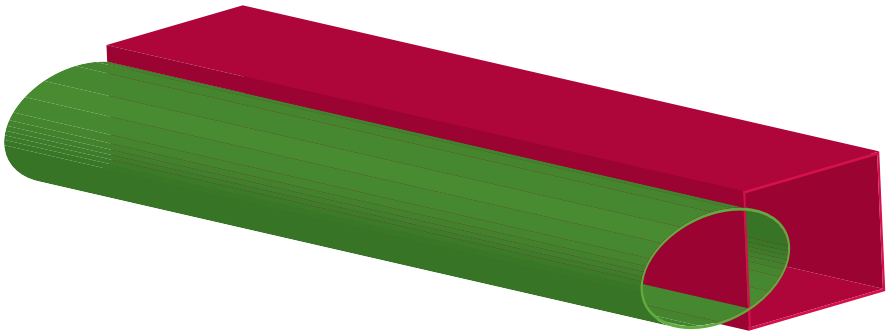

Figure 5.1: The vacuum chamber is the union of a number of subchambers. This figure illustrates this showing two subchambers – one colored green and the other colored red. A photon is considered within the vacuum chamber if, and only if, it is inside at least one of the subchambers.

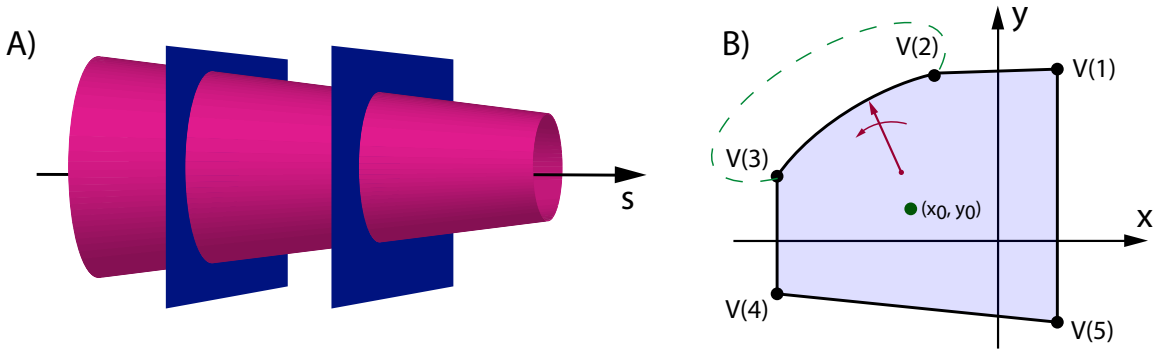

Figure 5.2: A subchamber is defined by a number of cross-sectional slices. A) A subchamber (red) and two cross-sectional slices (blue). B) A given cross-section is defined by a number of vertices.

```
&place section =  91.1, "IR1",     "rectangle" /
```

The <s> values for the cross-sections of a given subchamber must be in increasing order.

The <section_name> is a descriptive name that can be, for example, used in plotting. The <section_name> is ignored for any calculation. The <section_name> may be blank.

The <section_id> identifies the section. The general format for <section_id> is:

```
<sub_chamber_id>:<shape_id>@<edge-sec>
```

For example, if a section is placed like:

```
&place section =  74.3, "Near_IR"  "left_ante:shape-A@START" /
```

the <sub_chamber_id> component is "left_ante", the <shape_id> component is "shape-A", and the <edge-sec> component is "START". The <shape_id> must always be present but <sub_chamber_id> and <edge-sec> may be omitted. If <sub_chamber_id> is omitted, the

colon after the <sub_chamber_id> may be omitted too. If <edge-sec> is omitted, the ampersand before the <edge-sec> may be omitted too.

The <sub_chamber_id> component specifies which subchamber is being specified. If <sub_chamber_id> is omitted, the subchamber name is blank. For example, ":chamC@", which is equivalent to "chamC", is associated with the subchamber whose name is blank. Omitting the <sub_chamber_id> can always be done if there is only one subchamber.

The <edge-sec> component specifies if a subchamber is beginning or ending at a certain longitudinal position. If present, <edge-sec> must be:

```
"START" or
"END"
```

For every "START" there must be an "END". A subchamber begins with the section with <edge-sec> set to "START" and ends with the section with <edge-sec> set to "END". Example:

```
&place section =     5.0, "", "left_ante:shape-B@END" /
&place section =   174.0, ""  "left_ante:shape-A@START" /
&place section =   182.0, "", "left_ante:shape-B@END"
&place section =   936.0, ""  "left_ante:shape-A@START" /
```

With a closed geometry (set by chamber_end_geometry (§??), there are two subchambers, both with the same same: left_ante. One subchamber begins at $s = 174$ meters and ends at 182 meters. The other subchamber that begins at $s = 936$ meters, wraps around the end of the lattice back to the beginning, and ends at 5 meters. With a open geometry the above example would produce and error since subchambers are not wrapped around the ends of the lattice.

Subchambers with the same name may not overlap longitudinally. That is, "nested" start/end groups are not allowed since there is an ambiguity as to how to pair the cross-sections. For example, the following is not allowed:

```
&place section =  173.0, ""  "left_ante:shape-A@START" /
&place section =  174.0, ""  "left_ante:shape-A@START" /
&place section =  182.0, "", "left_ante:shape-B@END" /
&place section =  183.0, "", "left_ante:shape-B@END" /
```

A subchamber does not have to have START and END end sections. Such subchambers are called "open-ended". Open-ended subchambers extend along the entire length of the machine. For a lattice with an open geometry, an open-ended subchamber must have the longitudinal s-position of its first and last sections be at the ends of the lattice. For a lattice with a closed geometry, Synrad3D will naturally extend the wall from the last section around the ends of the lattice to the fist section. In fact, it is perfectly legitimate to define an open-ended subchamber in an closed geometry with just one section.

The <shape_name> component of <section_id> identifies the exact shape of the cross-section. The <shape_name> is matched with a shape_def namelist of the same name. The shape_def namelist has the format:

```
&shape_def
```

```
  name = <shape_name>           ! Name of this shape.
  r0 = <x_center>, <y_center>   ! Center of section.
  absolute_vertices = <T/F>     ! Vertex nums relative to r0 or abs? Default = F.
  v(1) = <x1> <y1> <radius_x1> <radius_y1> <tilt1>
  v(2) = <x2> <y2> <radius_x2> <radius_y2> <tilt2>
  v(3) = <x3> <y3> <radius_x3> <radius_y3> <tilt3>
  ... etc ...
/
```

Example:

```
! Anything outside the namelists is ignored.
&place section =  65.7, "Near_IR" "rectangular1" /
&place section =  74.3, "Arc"     "dipole_shape" /
&place section = 100.3, "Arc"     "dipole_shape" /

&shape_def
  name = "rectangular1"
  v(1) =  0.045,  0.025
/
&shape_def
  name = "dipole_shape"
  v(1) =  0.045,  0.025, 0.3, 0.9
  v(2) = -0.045,  0.025
  v(3) = -0.045,  0.025, 0.3, 0.9
  v(4) = -0.045,  0.025
/
```

There are three `place` namelists in this example. The first `place` namelist refers to the first `shape_def` namelist whose name is `"rectangular1"`. The other two `place` namelists refer to the second `shape_def` namelist named `"dipole_shape"`. As seen in this example, multiple `place`s may refer to the same `shape_def`.

For a `shape_def` namelist, the `name` labels the shape for use by a `place` namelist. A `shape_def` is specified by a "central point" r_0 and a list of connected vertices. Each vertex is specified by it's (x, y) coordinates that are with respect to r_0 except if `absolute_vertices` is set to `True` in which case the vertex numbers are absolute. The vertices must "wind" counter-clockwise around the central point. That is, if (x, y) coordinates of the vertices are expressed in terms of polar (r, θ) coordinates with respect to the central point, the vertices must be in order of increasing θ. If all the vertex points have non-negative y values, it is assumed that the gen_shape is symmetric with respect to the x-axis and that only half the vertex points are being specified. If all the vertex points have both non-negative x and y values, it is assumed that the gen_shape is symmetric with respect to both the x and y axes.

Between vertices, the wall is assumed to be a straight line except if a `<radius_x>` is given. If `<radius_x>` is set to a non-zero value for a given vertex, the wall segment between that vertex and the vertex before it is the arc of a circle with the given radius. If, in addition, `<radius_y>`

is given, the arc will be a section of an ellipse. If, in addition to <radius_x> and <radius_y>, <tilt> is given, then the ellipse will be tilted.

If <radius_x> is positive, the arc of the wall is convex. If it is negative, the arc is concave. Note that when the wall cross-section is concave, there can be problems with the photon tracking. See §?? for more details.

In the example above, the cross-section at $s = 65.7$ meters is a gen_shape whose definition is given by the shape_def with name "rectangular1". This shape_def defines a rectangle with half width 0.045 meters and half height 0.025 meters.

In the above example, "the dipole_shape" shape is similar to the rectangular1 shape except the right and left sides are sections of an ellipse.

5.3 Multi-Section Cross-Section

The multi-section cross-section is not actually a single cross-section but a repeating pattern of cross-sections. The basic repeat pattern is specified by a multi_place namelist of the form

```
&multi_place
  name = "<multi_name>"
  section(1) = 0.0,  "<section0_name>", "<section0_id>", ...
  section(2) = <s1>, "<section1_name>", "<section1_id>", ...
  section(3) = <s2>, "<section2_name>", "<section2_id>", ...
  ...
/
```

The syntax for any of the section(n) sections is the same as described above for rectangular, elliptical, and gen_shape. shapes. The first section, section(1), must have <s0> set to 0 and the <s> of the other cross-sections represent an offset from this 0^{th} section. The last section(N) section must have <sectionN_name> set to one of

```
open
closed
```

and the last section(N) must have <sectionN_id> set to

```
end_marker
```

This last "end_marker" section is not a real cross-section but simply serves as a marker for the end of the basic repeat pattern. The significance of the <sectionN_name> of the end_marker section is discussed below. Example:

```
&multi_place
  name = "zig_zag"
  section(1) = 0.000, "in_zig",  "ellipse1",
  section(2) = 0.002, "out_zag", "ellipse2",
  section(3) = 0.003, "open",  "end_marker"
/
```

Here the basic repeat pattern is 0.003 meters long and is made up of two cross-sections called `in_zig` and `out_zag`.

To position a series of cross-sections, a `place` namelist is used general syntax for this is:

```
&place
  section = <s>, "<section_name>", "<section_id>", <repeat_count>
  surface = "<surface_name>", <is_local>
/
```

The optional `surface` variable, which is used to specify a particular surface, is explained in §**??**.

`<s>` is the longitudinal starting position of the series of cross-sections, `<section_name>` is not used but must be present to keep the format the same as other `place`s. The cross-sections in the `multi_place` namelist will be repeated `<repeat_count>` times when the subchamber is constructed. Example:

```
&place section = 10.0, "", "zig_zag", 1000 /
```

With this `place`, coupled with the `multi_place` example above, an `in_zig` cross-section will be placed at $s = 10$ meters at the starting point. The basic pattern will repeat 1000 times and the series of cross-sections will be:

```
    S       Name
10.000      in_zig
10.002      out_zag
10.003      in_zig
10.005      out_zag
...
12.997      in_zig
12.999      out_zag
```

Since the `end_marker` section is "open", the basic pattern is repeated exactly 1000 times and there will be 2 * 1000 cross-sections. If the `end_marker` section is "closed", an extra `section(1)` cross-section would be added at the end to make the ending cross-section the same as the beginning cross-section. In this case, if the `end_marker` section is set to "closed", a `in_zig` cross-section would be added at $s = 13.000$ meters.

5.4 Fast and Slow Subchambers

When a subchamber has many cross-sections (which can easily happen when a multi-section (§**??**) is used), the computation time for photon tracking may become large due to the necessity of stopping the photon at each cross-section to check whether the photon has crossed the subchamber boundary. This can be ameliorated by defining an associated "fast" subchamber with only a few cross-sections that is contained withing the "slow" subchamber with the large number of cross-sections as illustrated in Figure **??**.

As explained in §**??**, when a photon is tracked that is contained in both a slow and an associated fast subchamber, `Synrad3D` can ignore the slow subchamber and track the photon through the

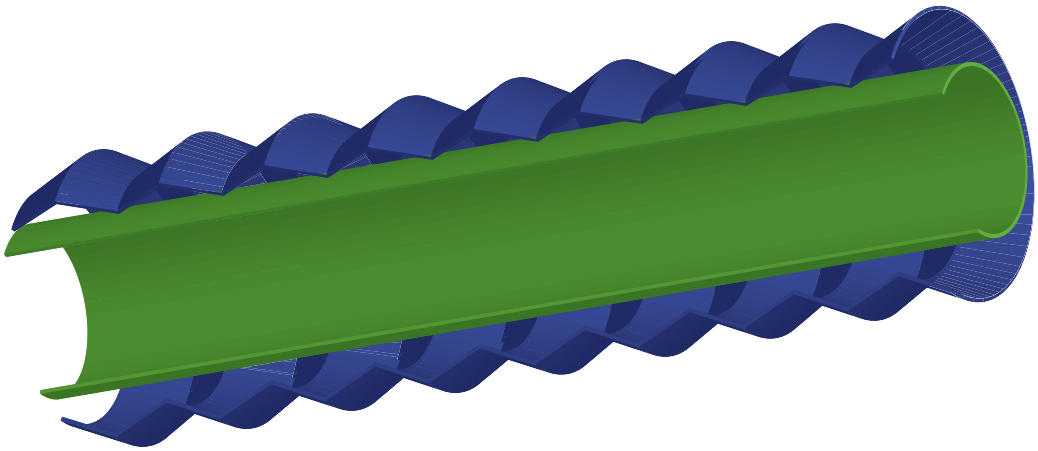

Figure 5.3: A "fast" subchamber (green), which uses fewer cross-sections to describe it, resides within another "slow" (blue) subchamber with many cross-sections. The fast subchamber enables `Synrad3D` to speed up tracking. The two subchambers have been cut in the illustration to better show the geometry.

fast subchamber until the photon exits the fast subchamber. Tracking will be quicker since there will be fewer cross-sections to stop the photon at.

To associate two subchambers as a fast/slow pair, a `fast_slow` namelist instance is used. The syntax of this namelist is:

```
&fast_slow
  fast = "<fast_subchamber_name>"
  slow = "<slow_subchamber_name>"
/
```

A `fast_slow` namelist instance establishes a single association. So to make multiple associations, multiple `fast_slow` namelist instances need to be used.

A subchamber can have multiple associated slow or fast subchambers. A subchamber can also have an associated fast subchamber and an associated slow subchamber although in practice this is to be avoided since it will, if anything, slow down tracking.

A fast subchamber may extend beyond the slow subchamber it is associated with. The entire vacuum chamber is the union of all subchambers including the fast ones. That is, in terms of defining the vacuum chamber, fast subchambers are no different than the slow ones or the "normal" subchambers that do not have any fast or slow associations.

5.5 Lattices with Branches

To describe things like injection lines or a ring attached to a X-ray beam line, a lattice will have multiple "branches". [See the Bmad manual[**?**] for more details on lattice branches.] By default,

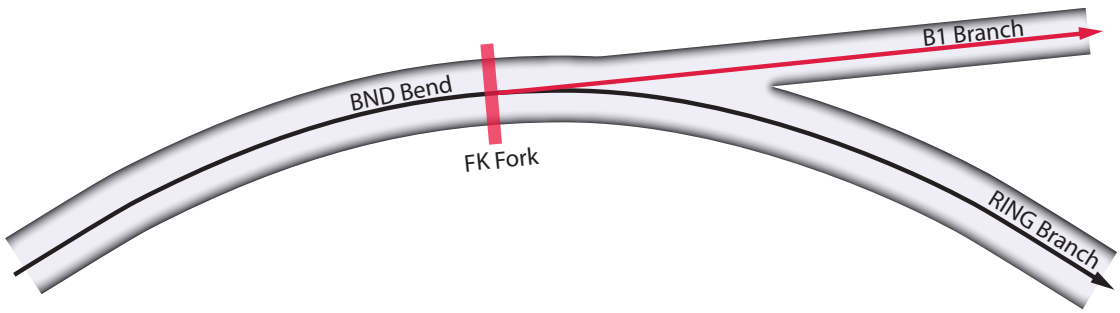

Figure 5.4: Branch line illustration. Here a branch line named `b1` is connected to the root branch, named `ring`, via a fork element named `fk`.

subchambers will be associated with the "root" branch (branch 0). A `subchamber_branch` namelist is used to associate a subchamber with a given branch. The form of this namelist is:

```
&subchamber_branch
  subchamber_name = <name>
  branch_name = <name-or-index>
/
```

The `subchamber_name` is the name of the subchamber to be associated and `branch_name` is name or index of the associated branch.

As an example, consider the following lattice:

```
parameter[E_tot] = 1e9
bnd: sbend, l = 2.0, g = 0.1
fk: fork, superimpose, ref = bnd, offset = -0.2, to_line = b1
dd: drift, l = 3
ring: line = (..., b, ...)
b1: line = (dd, ...)
b1[geometry] = open
use, ring
```

This lattice is illustrated in Figure **??**. This lattice has two branches. The root branch is named `ring` and this branch contains a bend called `bnd`. Superimposed upon `bnd` is a `fork` element called `fk` which forks to the second branch called `b1`.

The wall file for this example is:

```
&place section = 0.0,  "", "main:sdef" /
&place section = 2.0,  "", "main:sdef" /
&place section = 0.0,  "", "x_wall:sdef" /
&place section = 3.0,  "", "x_wall:sdef" /

&shape_def
  name = "sdef"
```

```
  v(1) =    0.01,    0.01
/

&subchamber_branch
  subchamber_name = "x_wall"
  branch_name = "b1"
/
```

There are two subchambers. One subchamber named `main` is, by default, associated with the root branch – `ring`. The other subchamber, named `x_wall`, is associated with the `b1` branch.

5.6 Subchamber Surface Interpolation

Once a set of subchamber cross-sections have been defined, the cross-section of the subchamber at a given longitudinal position is computed using linear interpolation.

For a given subchamber, and at a given s position, the r, θ coordinate system in the transverse x, y plane is defined with respect to an origin $\mathbf{r}_O(s)$ given by a linear interpolation of the origins of the cross-sections to either side of the given s position. Let s_1 denote the position of the cross-section just before s and s_2 denote the position of the cross-section just after s. Let $\mathbf{r}_{01}$ be the (x_0, y_0) origin defined for the cross section at s_1 and $\mathbf{r}_{02}$ be the (x_0, y_0) origin defined for the cross section at s_2. Then

$$\mathbf{r}_O(s) = (1 - \widetilde{s})\, \mathbf{r}_{01} + \widetilde{s}\, \mathbf{r}_{02} \tag{5.1}$$

where

$$\widetilde{s} \equiv \frac{s - s_1}{s_2 - s_1} \tag{5.2}$$

Let $r_{c1}(\theta)$ and $r_{c2}(\theta)$ be the radiusus of the sub-section boundary as a function of θ for the cross-sections at $s = s_1$ and $s = s_2$ respectively. The sub-section boundary $r_c(\theta, s)$ at any point s between s_1 and s_2 is then defined by the equation

$$r_c(\theta, s) = (1 - \widetilde{s})\, r_{c1}(\theta) + \widetilde{s}\, r_{c2}(\theta) \tag{5.3}$$

A photon at (r, θ, s) is inside the subchamber if

$$r < r_w(\theta, s) \tag{5.4}$$

5.7 Subchamber Surface and Patch Elements

A `patch` element, which shifts the reference coordinate system complicates the calculation of where a photon hits the wall. The Bmad manual has a full explanation but the essential factor is that, since the reference orbit is discontinuous in a patch, to avoid a discontinuity in the subchamber surface, wall interpolation is done using the global reference system. This can lead to non-intuitive behavior when the region between two wall sections contains both a bend and

a patch since, without the patch, the center of the subchamber follows the curved reference orbit but with the patch the subchamber surface center follows a straight line between the wall sections.

To avoid this non-intuitive behavior, if a region between wall sections contains both a patch and a bend, extra sections will be added to eliminate this situation.

To simplify matters, `Synrad3D` does not allow wall cross-sections to be defined whose longitudinal position overlaps any `patch` element.

5.8 Chamber Surface Reflectivity

By default, the photon reflectivity of the vacuum chamber wall is based on the reflectivity of a 10 nm C film on an Al substrate. This default can be changed by setting the following parameters in the master input file (§**??**):

```
surface_reflection_file
surface_roughness_rms
roughness_correlation_len
```

If the chamber wall is made up of sections of differing material, additional surface types can be defined using `surface_def` namelists in the chamber wall file. The syntax for this namelist is

```
&surface_def
  reflectivity_file = "<file_name>"
/
```

`reflectivity_file` is a file that specifies the surface reflectivity as explained in (§**??**).

To associate a particular material with a particular region of the chamber wall, the optional `surface` variable is set for the `place` namelist (§**??**), at the downstream (maximal s) end of the region of interest. The syntax is:

```
&place
  section = ...
  surface  = "<name>", <is_local>
/
```

The `<name>` here must match the `name` in the surface reflectivity file The two exceptions are if `<name>` is set to

```
ABSORBER              ! Case sensitive
PHANTOM               ! Case sensitive
```

The `ABSORBER` setting makes the surface a perfect absorber. The `PHANTOM` setting is for surfaces where, if `filter_phantom_photons` (§**??**) is True, if a photon hits the phantom surface, it is not counted in the final statistics. This is useful for situations like simulating an X-ray beam pipe where, for the purposes of the simulation, the pipe is terminated at some point and photons intersecting the end of the pipe are not to be considered to have hit a wall but rather to have

safely gone through to the experimental end station. Effectively, photons intersecting a phantom wall is treated as any other photon that fails a filter test (§??). Note that, independent of the setting of `filter_phantom_photons`, a phantom surface acts like a perfect absorber. That is, there are never any reflections from a phantom surface.

If `<is_local>` is `True`, then the surface material only spans the region from the previous cross-section to this one. If `<is_local>` is `False` or not present, the surface material spans from the closest previous cross-section where there is a `surface` variable set to this cross-section. If a `place` instance refers to a `multi_section` (§??), and `<is_local>` is `True` then the material only spans the multi_section region.

Note: For any given region with finite area of the vacuum chamber wall, if there are multiple subchambers whose surfaces overlap the region, it is important to make sure that the surface types of these subchambers are all the same since, for a photon striking this region, which subchamber is used to compute the surface properties is ill-defined. As an example, consider Figure `f:vac-chamber`. The overlap between the two subchamber surfaces are the two lines where the subchamber surfaces intersect. Since the overlap has zero area, it is OK to have the two subchambers have different surface types.

If a region of the vacuum chamber wall is defined by multiple subchambers (that is, multipole subchamber surfaces coincide in this region), which subchamber is used to select the surface type can vary from photon to photon. Therefore, it is important to make sure that for any given point on the vacuum chamber wall, that the local surface type is the same for all subchambers that have walls that coincide with the point in question.

Example:

```
&place section = 0.0, "elliptical", ... /
&place section = 1.0, "elliptical", ... /
&place
  section = 3.0, "elliptical", ... /
  surface = "ss", True
/
&place
  section = 5.0, "multi_section:ms", ...
  section = "ni", True
/
&place
  section = 7.0 "elliptical", ...
  surface = "cu"
/
&place 10.0 section = ... /

&surface_def
  name = "ss"
  reflectivity_file = "ss_burnished.reflect"
/
```

Assuming the multi_section spans the region from 5 meters to 6 meters, the materials associated with different regions of the chamber are:

S_begin	Surface	S_end
0.0	cu	1.0
1.0	ss	3.0
3.0	cu	5.0
5.0	ni	6.0
6.0	cu	7.0
7.0	default	10.0

5.9 Old Wall Format

Originally, `Synrad3D` did not have the concept of subchambers. That is, there was only one chamber and that chamber was allowed to be concave. This was problematical, as discussed above, and motivated the development of the subchamber concept. This section discusses the old format to facillitate conversion from the old format to the new. Example of the old format:

```
&section_def section =   0.0, "arc_std", "elliptical", 0.045, 0.025 /
&section_def section =  74.3, "Near_IR"  "gen_shape:dipole_shape",  /
&section_def section =  82.9, "wig1",    "rectangular", 0.045, 0.025 /
&section_def section =  91.1, "IR1",     "rectangular", 0.045, 0.025,
                                                        -1, -1, 0.08, 0.01 /
```

The `<section_id>` can have one of five values:

```
elliptical
rectangular
gen_shape:<shape_name>
multi_section:<shape_name>
```

Historically, the `elliptical` and `rectangular` shapes where developed first. The `gen_shape` was developed later and is a generalization of the original two shapes. The `elliptical` and `rectangular` cross-sections are explained in the next section. `Gen_shape` is explained in the section after and `multi_section` is explained in the section after that.

An example `elliptical` cross-section is shown in Figure **??**. The format for an `elliptical` or `rectangular` cross-section is

```
&section_def
  section = <s>, <section_name> <section_id>, <width2>, <height2>, <width2_plus>,
            <ante_height2_plus>, <width2_minus>, <ante_height2_minus> /
  surface = "<surface_name>", <is_local>
```

The first five parameters – `<s>`, `<section_name> <section_id>`, `<width2>`, and `<height2>` – must be specified. Values for the other parameters are optional and default to the "unset" value of -1.

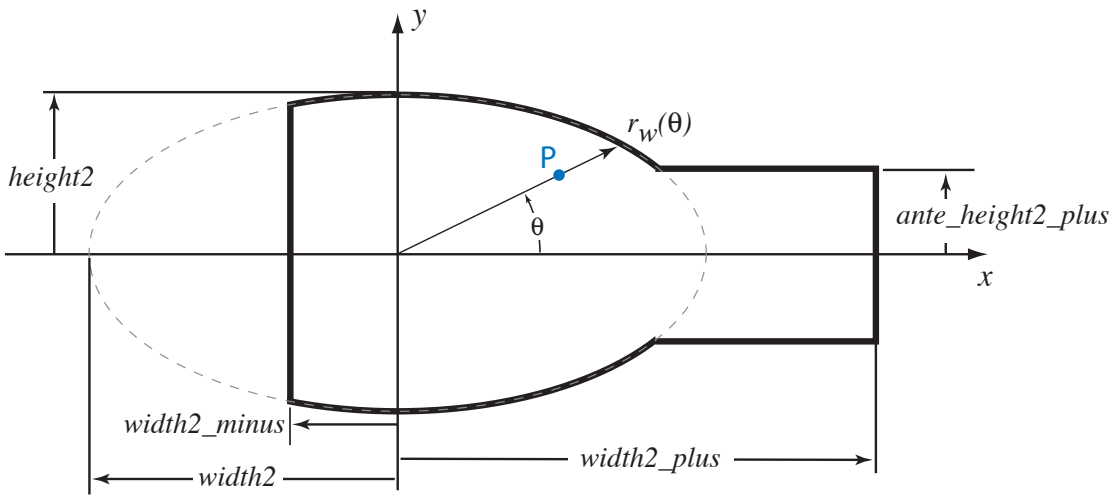

Figure 5.5: Example vacuum chamber cross-section for an `elliptical` chamber with an antechamber on the $+x$ side of the chamber and an aperture on the $-x$ side.

The optional `surface` variable is explained in §**??**.

For `elliptical` or `rectangular` shapes, the parameters needed to specify the chamber are:

```
<s>                  ! Longitudinal position
<section_name>       ! Descriptive Name of the cross-section.
<section_id>         ! Either "elliptical" or "rectangular"
<width2>             ! Half width ignoring antechamber.
<height2>            ! Half height ignoring antechamber.
<width2_plus>        ! Distance from pipe center to +x side edge.
<ante_height2_plus>  ! Antechamber half height on +x side of the wall
<width2_minus>       ! Distance from pipe center -x side edge.
<ante_height2_minus> ! Antechamber half height on -x side of the wall
```

For both `"rectangular"` and `"elliptical"` shapes, `<width2>` and `<height2>` define the half-height and half-width of the shape.

Antechambers on the $+x$ side and/or $-x$ side of the chamber can be added to the basic shape. On the $+x$ side, an antechamber is formed if the `<ante_height2_plus>` parameter is set to a positive value. If `<ante_height2_plus>` is set to a positive value, as shown in Figure **??**, the parameter `<width2_plus>` specifies the horizontal distance from the chamber center to the far end of the $+x$ side antechamber. In this case, the value of `<width2_plus>` must be large enough so that the antechamber far wall does not stick back into the basic shape. For a rectangular shape, this translates to `<width2_plus>` being larger than `<width2>`. Similarly, if `<ante_height2_minus>` is set to a positive value, an antechamber is formed on the $-x$ side. In this case, `<width2_minus>` is the (positive) horizontal distance from the chamber center to the far end of the $-x$ antechamber.

If <ante_height2_plus> is not set, a set of <width2_plus> defines an aperture on the $+x$ side of the chamber. In this case, the value of <width2_plus> must be less than the value of <width2>. Similarly, if <ante_height2_minus> is not set, a set of <width2_minus> defines an aperture on the $-x$ side of the chamber. A $-x$ aperture is shown in Figure **??**.

Example:

```
! Anything outside the namelists is ignored.
&section_def section =   0.0, "arc_std", "elliptical", 0.045, 0.025 /
&section_def section =  82.9, "wig1",    "rectangular", 0.045, 0.025 /
&section_def section =  91.1, "IR1",     "rectangular", 0.045, 0.025,
                                                        -1, -1, 0.08, 0.01 /
```

Prescription for converting:

1. If there are &section_def namelists that use rectangular or elliptical shapes. That is, no "gen_shape:..." is present then the appropriate &shape_def namelists will have to be created. If the shape is not convex, the shape will need to be split into subchambers.

2. rename: "gen_shape:..." to "..." in &section_def namelists. If the shape is not convex, the shape will need to be split into subchambers.

3. If present, remove "ix_vertex_ante1" and "ix_vertex_ante2" lines from &gen_shape_def. To get the correct antechamber absorbtion, use a separate subchamber for the antechamber regions.'

4. rename: "&section_def" to "&place".

5. rename: "&gen_shape_def" to "&shape_def".

6. If a shape_def has a name that has a colon ":", replace this by some other character.

Chapter 6

Photon Tracking & Reflections

6.1 Photon Tracking

[Note: Parameters like `num_photons_per_pass` mentioned in here are set in the master input file (§**??**).

`Synrad3D` will generate a set of approximately `num_photons_per_pass` photons in the radiation production region. This is called a "pass". After any filtering, `Synrad3D` will check to see if the number of surviving photons is at least `num_photons`. If so, photon generation will stop. If not enough photons have been generated, `Synrad3D` will generate another `num_photons_per_pass`, and so on, until at least `num_photons` surviving photons have been generated. Thus the actual number of surviving photons will be between `num_photons` and `num_photons_per_pass`. [A surviving photon is a photon that passes all filter requirements (§**??**) and is recorded in the output statistics.]

When a photon is created, `Synrad3D` checks to see if the photon is within any subchamber. If the photon is not within any subchamber, that is, if it is outside the vacuum chamber wall, `Synrad3D` will generate an error message if the number of photons that have been generated up to this point exceed `num_ignore_generated_outside_wall`. If the generated photon is within the vacuum chamber wall, the phton is "associated" with one of the subchambers that it is within. The photon will be tracked through the associated subchamber until it reaches the edge of the subchamber.

When the photon reaches the edge of the associated subchamber it is going through, `Synrad3D` checks if there is a suitable alternative subchamber to switch association to. A suitable subchamber is any subchamber that the photon is within or a subchamber where the photon is at the edge and has velocity directed inward.

If there is a suitable alternative, `Synrad3D` will associate the photon with the alternative subchamber and keep tracking until the photon gets to the edge of this subchamber. This process of reassociating the photon to a second subchamber when the photon reaches the edge of the current associated subchamber will continue until there is no suitable alternative. At this point

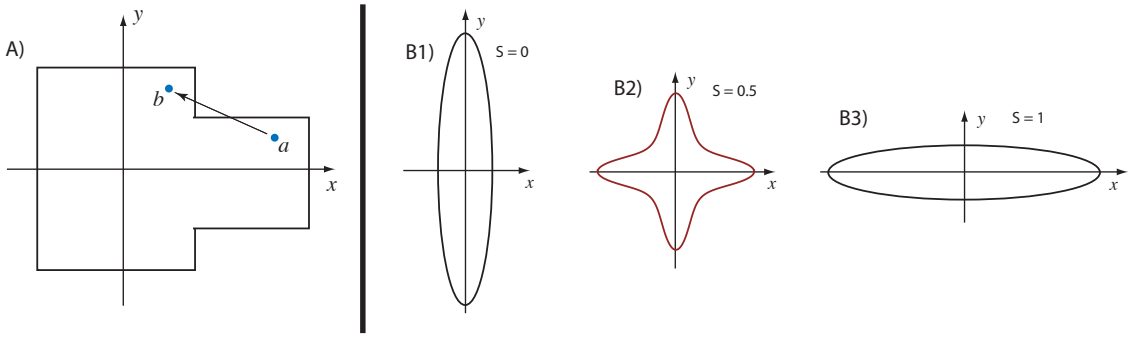

Figure 6.1: A) A convex cross-section can lead to problems with linear interpolation since it is assumed that if the beginning and ending points of a step are inside the subchamber the entire track is inside the subchamber. The track shown violates this assumption. B) With linear interpolation, convex cross-sections are not a guarantee that intermediate cross-sections are convex. B1 and B3 are the defined cross sections at $s = 0$ and $s = 1$ respectively. These cross sections are ellipses with 5:1 aspect ratio. The interpolated cross-section B2 at $s = 0.5$ is seen to be concave.

the photon has hit the vacuum chamber wall.

When tracking a photon through its associated subchamber, `Synrad3D` can ignore all other subchambers since, by definition, if the phton is inside its associated subchamber, it is inside the vacuum chamber.

The exception to the above process happens when the subchamber associated with a photon itself has an associated "fast" chamber (§**??**). In this case, `Synrad3D`, while tracking through the slow subchamber, will periodically check to see if the photon is also within the fast subchamber, if it is, the photon's associated subchamber will be switched to the fast subchamber.

Photons are tracked in "steps". A step consists of propagating a photon from it's current position to some new position that is a certain distance away. The length to propagate the photon in a given step is determined by a number of factors. The maximum length for a step is set by the input parameters

```
sr3d_params%ds_track_step_max
sr3d_params%dr_track_step_max
```

These set longitudinal (`%ds_track_step_max`) and transverse (`%dr_track_step_max`) distances. Additionally, a single step will always be terminated at, and never cross over, a defined cross-section plane of the subsection associated with the photon. Steps will also be terminated at the boundaries of bend magnets and at the point in a bend where a trajectory is closest to the center of the bending radius.

After a step, the new photon position is checked to see if it is still inside the associated subchamber. If it is not, the point where it has crossed the subchamber surface is calculated via a root finding algorithm.

If the new photon position is inside the associated subchamber, there is still the question as to whether the photon trajectory between the beginning step point and the end step point is inside the subchamber. Synrad3D assumes that the subchamber is "locally convex". That is, given the two end points of a step with points inside the subchamber, it is assumed that the line drawn between these two points never touches or goes outside the subchamber. This locally convex assumption will be true if every cross-section $r_w(\theta, s)$ for fixed s is convex where s is in the range of longitudinal s positions covered by the step in question.

Synrad3D assumes that if the beginning and ending points of a step are inside the subchamber, the entire track is inside the subchamber. This assumption my be violated if the cross-section of a subchamber is concave as illustrated in Fig. **??**A. Furthermore, even if two defined cross-sections are themselves convex, an intermediate cross-section can be concave as shown in Fig. **??**B.

A subchamber "region" is the volume of the subchamber between two adjacent cross-sections. The total subchamber volume is thus the sum of all the regions of the subchamber. Since Synrad3D always limits steps so that the beginning and ending points of a step cannot be in different regions, there is no problem with Synrad3D failing to detect photons crossing the chamber wall as long as each region indivdually is convex in shape. This is true even if the subchamber considered as a whole is not convex.

How much of a problem the possibility that Synrad3D will not detect some photons leaving and then entering the chamber will, of course, depend upon the specifics of the chamber geometry and what is being calculated. One way to test if this is a problem this is to reduce the track step length limits:

```
sr3d_params%ds_track_step_max
sr3d_params%dr_track_step_max
```

This will reduce the possibility of mis-tracking at the cost of increased computation time. If the results do not change significantly when the track limits are reduced, this is good evidence that the results are valid.

Notice that if all the subchambers are convex there will be no problem even if the union of the subchambers – which defines the entire vacuum chamber – is concave. To put it another way, given a concave vacuum chamber, if the vacuum chamber can be constructed using a number of convex subchambers, there will be no problem with Synrad3D not detecting photons leaving and entering the vacuum chamber in a single step.

6.2 Photon Filtering

Filters (set in the main parameter file §**??**) are used to discard uninteresting photons from being recorded in output files. There are two types of filters. Initialization filters are applied when a photon is being generated and end filters are applied at the end of photon tracking. The initialization filters are

```
e_init_filter_min               ! Min initial energy filter.
e_init_filter_max               ! Max initial energy filter.
```

```
vert_angle_init_filter_min        ! Min vertical angle filter.
vert_angle_init_filter_max        ! Max vertical angle filter.
vert_angle_symmetric_init_filter  ! Logical: Symmetric angle filtering?
```

The use of initialization filters will also speed up the simulation since it eliminates tracking of uninteresting photons.

Note: If vertical angle filters are used, `e_init_filter_min` must be set. This is mandated to avoid problems that occur at zero photon energy where the inital photon distribution from a bend has infinite probability density per unit of energy and an infinite vertical angle width.

The four end of photon tracking filters are:

```
e_filter_min        ! Min Energy filter.
e_filter_max        ! Max energy filter.
s_filter_min        ! Min longitudinal position filter.
s_filter_max        ! Max longitudinal position filter.
```

The s-filters can wrap around $s = 0$: That is, if both `s_filter_min` and `s_filter_max` have been set, and if `s_filter_min` is *greater* than `s_filter_max`, the region for keeping a photon is from `s_filter_min` through the end of the lattice along with the region from the start of the lattice to `s_filter_max`.

Note: Processes like florescence can create photons with an energy that is different from the energy of the photon that initiates the process. Currently, `Synrad3D` does not model such processes so the final photon energy will be equal to the initial photon energy. This being the case, `e_init_filter_min` and `e_init_filter_max` are equivalent to `e_filter_min` and `e_filter_max` respectively except that the inital filters will speed up the simulation.

See §**??** for more details.

6.3 photon_start_input_file

Instead of having `Synrad3D` generate photons randomly, starting photon positions may be read in from a file. This is done typically for debugging purposes.

The name of this file of starting positions is given by the `photon_start_input_file` parameter in the `synrad3d_parameters` namelist (§**??**).

The `photon_start_input_file` file contains a number of `&start` namelists. One for each photon starting position. The format for a `&start` namelist is:

```
&start
  orbit%vec = <x>, <vx/c>, <y>, <vy/c>, 0.0, <vz/c>  ! Photon position
  orbit%s   = <s>   ! Longitudinal distance from start of lattice.
  orbit%p0c = <ev>  ! Photon energy
  ran_state = <random_state_struct>
  random_seed = <num>
  ix_branch = <num>  ! Lattice branch index to start photon in. Default is 0.
```

/

The fifth component of `orbit%vec` is not used and should be set to zero. The `orbit%s` component is the longitudinal distance from the start of the lattice. If needed, The `ran_state` and `random_seed` parameters can be used to initialize the random number generator.

The magnitude of the velocity, $\sqrt{v_x^2 + v_y^2 + v_z^2}/c$ must be 1.

Note: Files of photon starting positions may be generated by setting the `photon_start_output_file` parameter in the `synrad3d_parameters` namelist or by using the `-out` command line option (§**??**).

6.4 Surface Reflection File

A surface reflectivity file is used to redefine the default chamber wall surface reflectivity (By setting `surface_reflection_file` in the master input file (§**??**)) or to define the surface reflectivity for a particular wall material (§**??**).

There is a script for downloading surface reflectivity data from the LBNL x-ray data base at

`http://henke.lbl.gov/optical_constants/layer2.html`

and creating reflectivity files for use with `Synrad3D`. The script is in the directory

`bsim/synrad3d/lbl_reflectivity_downloader`

See the `README` file in this directory for more instructions.

The probability for a photon reflecting from a wall is dependent upon the grazing angle and the photon energy. For a given graze angle and photon energy, there are two reflection probabilities called `p_reflect` and `rel_p_specular` These determine the probabilities that the photon will be reflected either diffusely or specularly, or that the photon will be absorbed. In particular

```
probability of absorption          = 1 - p_reflect
probability of reflection          = p_reflect
probability of specular reflection = p_reflect * rel_p_specular
probability of diffuse reflection  = p_reflect * (1 - rel_p_specular)
```

Generally, `p_reflect` will be near unity at small grazing angles and fall off with increasing angle. At the smaller photon energies ($E \lesssim 100\,\text{eV}$), the reflection probabilities will still be substantial at grazing angles of 10 degrees or more. At the larger photon energies ($E \gtrsim 1000\,\text{eV}$), The reflection probabilities are quite peaked and are small above a few degrees.

`rel_p_specular` is given by Equations (1) and (2) in Reference [**?**] and so `rel_p_specular` is not specified in a reflection file.

In the reflection probability file, the reflection probability `p_reflect` is specified at a number of different angles and energies. `Synrad3D` will interpolate as needed. Reflections probabilities are specified over some energy range from E_{min} to E_{max}. If a photon has an energy below E_{min}, reflection probability is taken to be the same as the probability at E_{min}. If a photon has an

energy E_p that is above E_{max}, it is assumed that the reflection probability is the same as the reflection probability at energy E_{max} and angle $\theta_g(\text{eff}) = \theta_g * E_p/E_{max}$ where θ_g is the photon grazing angle. If θ_g(eff) is grater than 90 degrees, 90 degrees is used for θ_g(eff).

For specifying the reflection probabilities, the energy range from E_{min} to E_{max} is broken up into a number of "tables" each covering some energy interval. The tables are ordered in increasing energy so that table 1 starts from E_{min} and the last table goes up to E_{max}. The energy intervals for the tables abut one another so that the upper energy of one table is the lower energy of the next. Each table specifies the reflection probabilities at a number of energies $E_i, i = 1, \ldots, N_E$ and a number of grazing angles $\theta_j, j = 1, \ldots, N_\theta$. The energies E_i must be equally spaced and in ascending order but the grazing angles θ_j do not only have to be in ascending order with $\theta_1 = 0$ and $\theta_{N_\theta} = 90$. Except for the restriction that the energy intervals of the tables abut one another, the tables are independent in the sense that, for example, the spacing between the E_i, the particular θ_j chosen, etc. can vary from table to table. The reason for having multiple tables is for compactness. That is, the particular choice of the spacing between the E_i and the particular $theta_j$ chosen can be optimized for each energy interval to give the maximum accuracy with the least amount of input data.

The probability file must start with a namelist named `general` that specifies the number of tables and roughness numbers. Example:

```
&general
  name = "ConCu"      ! Case sensitive
  description = "50um C layer on Cu substrate"
  n_table = 5    ! 5 tables
  surface_roughness_rms      = 200e-9   ! meters
  roughness_correlation_len  = 5.5e-6   ! meters
/
```

The `name` parameter is used to identify the surface in the vacuum chamber wall file (§**??**). The `description` parameter is an optional description string that `Synrad3D` prints when plotting reflectivity curves.

Next, each table is specified in turn starting from the table for the lowest energy interval. A table is specified starting with a `table` namelist. There are two different syntaxes that can be used here An example of the first syntax is:

```
&table
  energy_min    =  600 ! eV
  energy_max    = 1400 ! eV
  energy_delta  =   20 ! eV
  angles =  0.0, 0.4, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 90.0
/
```

In this example, the energy range is from 600 eV to 1400 eV in steps of 20 eV and the reflection probability is specified at 9 different grazing angles (in degrees). Angles must start at 0 and end at 90 degrees.

An example of the second `table` namelist syntax is:

```
&table
  energies = 7, 10, 31, 45
  angles =  0.0, 0.4, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 90.0
/
```

In this example, the angle points are the same but the energy points are specified point by point. This second syntax is useful when the reflection data is not evenly spaced.

After the `table` namelist, for each energy "row", `p_reflect` is specified by a `row` namelist. Example:

```
&row
  ix_row = 2   ! 620 eV
  p_reflect =  1.00, 0.941, 0.882, 0.852, 0.771, 0.669, 0.514, 0.056, 0.0
/
```

There must one `row` namelist for each energy value in the table. For this example there would be 41 `row` namelists. The `row` namelists must be in order of increasing energy and each `row` namelist has a `ix_row` component which must be set to `1` for the first `row` namelist, `2` for the second, etc. `Synrad3D` uses `ix_row` as a sanity check when reading in the table. The `p_reflect` component of the `row` namelist give the reflection probabilities at the N_θ angle points.

There must be no gaps between the energy ranges of the tables. That is, the lowest energy of one table must be at most the highest energy of the previous table. On the other hand, it is permitted for the energy ranges of the tables to overlap.

Chapter 7

Output Files

7.1 Main Output File

The `dat_file` (whose name is specified in the master input file (§**??**)) is divided into two parts. the top part essentially echos the information provided in the main input file. It looks like:

```
# photon_number_factor      =  8.155E-01
# num_photons               = 10     ! Target number of unfiltered photons.
# num_photons_generated     = 123    ! Total including filtered photons.
# num_photons_generated_eff = 123    ! Total including filtered photons.
# num_photons_passed_test   = -1     ! As set in the input file.
# I0_tot_ring               = 5.6e7  ! I0 radiation integral for the ring
# I0_tot                    = 5.6e7  ! I0 radiation integral for emit region
# I0_tot_eff                = 2.3e7  ! Effective I0 including filters
# ix_ele_track_start        = 103
# ix_ele_track_end          = 104
# photon_direction          = 1
# random_seed               = 123456
# lattice_file              = "../tao/examples/cesr/lat.bmad"
# photon_start_input_file   = ""
# wall_file                 = "synrad3d.wall"
# dat_file                  = "synrad3d.dat"
# chamber_end_geometry      = ""
# ds_step_min               =  1.000E-02
... etc ...
```

The first line gives the `photon_number_factor` which is computed via the formula

$$\text{photon\_number\_factor} = \frac{5\sqrt{3}\,e^2\,I_{0,eff}}{4\,\pi\,\epsilon_0\,\hbar\,c\,N_{sim}} \tag{7.1}$$

where N_{sim} is the number of photons generated in the simulation run, and $I_{0,eff}$ is the effective synchrotron radiation integral. If there is no initial filtering (§**??**), $I_{0,eff}$ is given by the standard

radiation integral for I_0

$$I_0 = \int_{emit} ds\, \gamma_0\, g(s) \tag{7.2}$$

with $g(s)$ being the bending strength (equal to $1/\rho$, with ρ being the bending radius), and the integral is done over the region where photons are emitted in the simulaiton. If there is inital filtering applied, $I_{0,eff}$ is

$$I_{0,eff} = f_p * I_0 \tag{7.3}$$

where f_p is the fraction of photons that will pass the initial filter tests. If the filtering is too stringent, and the value of f_p falls below 10^{-16} or so, round-off error will make the calculated value of $I_{0,eff}$ zero. In this case, `Synrad3D` will not be able to run.

The `photon_number_factor` is related to N_γ, the total number of photons per beam particle emitted in one revolution in the actual machine via

$$\text{photon\_number\_factor} = \frac{N_\gamma}{N_{sim}} \tag{7.4}$$

Thus, if $N_r(\text{sim})$ is the number of simulated photons absorbed in a particular region, the actual number of photons $N_r(\text{actual})$ absorbed in this region per beam particle per turn is

$$N_r(\text{actual}) = \text{photon\_number\_factor} \cdot N_r(\text{sim}) \tag{7.5}$$

The second part of the file is a table. Each row is the data for one photon. Only photons that pass the filter tests are shown. The table looks like:

```
 1       2  6.981783E+02   -0.000221    0.001255    0.000052   ... etc.
 2       1  4.705803E+03    0.000605    0.001255    0.000050   ... etc.
 3       2  6.883773E+02    0.001430    0.001255    0.000048   ... etc.
 4       2  3.463312E+02    0.002255    0.001255    0.000046   ... etc.
 5       1  2.104224E+01    0.003080    0.001255    0.000043   ... etc.
 6       2  1.393605E+02    0.003904    0.001254    0.000041   ... etc.
 7       5  1.979004E+01    0.004729    0.001254    0.000039   ... etc.
 8       2  4.288703E+02    0.005553    0.001253    0.000037   ... etc.
 9      12  1.493411E+01    0.006377    0.001253    0.000034   ... etc.
10       2  1.297799E+02    0.007200    0.001252    0.000032   ... etc.
```

The columns of the file are:

```
1:      Photon index number.
2:      The number of times the photon has struck the vacuum
            chamber wall including the final hit.
3:      The photon energy (in eV).
4-9:    Initial photon position (x, Vx/c, y, Vy/c, s, Vs/c).
10      Index of the lattice branch where the photon was created
11-16:  Final photon position (x, Vx/c, y, Vy/c, s, Vs/c).
17:     sin(graze_angle). 0 = Grazing incidence, 1 = perpendicular incidence.
18:     Photon travel length
```

```
19:     Photon longitudinal travel length - beam travel length
                                                    in the same time period.
20:     Index of the lattice branch where the photon is absorbed.
21:     Lattice element index where photon is absorbed.
22:     Lattice element type (Eg: quadrupole, etc.) where photon is absorbed.
23:     sub-chamber name where photon is absorbed.
```

x is the horizontal position (direction along the local normal to the closed orbit, in the bend plane, zero on the closed orbit, positive to the outside of the machine,), y is the vertical position (direction perpendicular to the bend plane, zero on the closed orbit, positive up), and s is the longitudinal position (direction tangent to the closed orbit, zero at the beginning of the lattice, positive in the direction of motion of the beam).

7.2 Wall Hit Output File

If the switch `wall_hit_file` is non-blank, an additional output file is generated, which contains more detailed information on where photons are hitting the vacuum chamber wall. Example:

```
1     0       0.0     -0.0008    0.0000     67.7857       0.0000   ... etc.
1     1     698.2      0.0450   -0.0002     70.5150       0.0323   ... etc.
1     2     698.2     -0.0393   -0.0138     71.2168      -0.1152   ... etc.
2     0       0.0     -0.0007    0.0000     67.8515       0.0000   ... etc.
2     1    4705.8      0.0450    0.0001     70.5782       0.0322   ... etc.
3     0       0.0     -0.0006    0.0000     67.9172       0.0000   ... etc.
3     1     688.4      0.0450    0.0004     70.6414       0.0322   ... etc.
3     2     688.4     -0.0450   -0.0020     71.0368      -0.2197   ... etc.
4     0       0.0     -0.0005    0.0000     67.9829       0.0000   ... etc.
4     1     346.3      0.0450   -0.0003     70.7046       0.0322   ... etc.
4     2     346.3     -0.0450   -0.0014     71.2347      -0.1643   ... etc.
5     0       0.0     -0.0004    0.0000     68.0487       0.0000   ... etc.
5     1      21.0      0.0450    0.0003     70.7678       0.0321   ... etc.
```

The columns of this file are:

```
1:     photon_index
2:     wall_hit_index
3:     photon_energy              ! When wall_hit_index = 0 this is 0]
4-6:   (x, y, s)                  ! Coordinates of photon at wall
7:     ix_branch                  ! Index of lattice branch where photon is.
8-10:  (vx, vy, vs)               ! Velocity before bounce.
11-13: (vx, vy, vs)               ! Velocity after bounce.
14-16: (perp_x, perp_y, perp_z)   ! Wall perpendicular.
17:    cos_perp_in                ! Cos of photon incoming direction wrt wall.
18:    cos_perp_out               ! Cos of photon outgoing direction wrt wall.
19:    reflectivity               ! Reflectivity coef.
20:    sub-chamber                ! Subchamber name where photon hits.
```

The `photon_index` is the same index as in the main output file. The `wall_hit_index` starts at 0 for the emission point and increases up to the value for `n_wall_hits` in the main file. The `photon_energy` will be the same as in the main file.

Columns 4 through 6 give the coordinates of the photon where it strikes the wall. except that the entry for wall_hit_index = 0 gives the emission point coordinates.

Columns 8 through 10 gives the velocity of the photon just before it bounces and columns 11 through 13 gives the velocity just after it bounces (or what would be its velocity if it does not get absorbed).

Columns 14 through 16 give the perpendicular vector to the wall at the point of photon impact.

Columns 17 and 18 give the cosine of the angle between the wall perpendicular and the photon velocity just before and just after the bounce. 0 = photon velocity parallel to wall, ± 1 = photon velocity perpendicular to wall.

Column 18 gives reflectivity which is a function of the type of surface and the photon orientation with respect to the wall.

If the photon is "adsorbed" due to reflections being disallowed, columns 10 and higher will be zero.

In the example given above, the first photon is absorbed after reflecting once. and the second photon is not reflected at all being absorbed on the first hit.

Note: If the beam emittances are zero, photons will be generated on the central orbit. If there are no steerings powered, the central orbit will be the zero orbit and in this case all photons will start with $x = vx = y = 0$.

Note: Since the (x, y, s) coordinates are curved in bend elements, the photon trajectory in (x, y, s) coordinates is not, in general, a straight line between hit points. For more accurate plotting, a `photon_track_file` (§**??**) should be generated. Caution: `photon_track_file`s are over an order of magnitude larger than `wall_hit_file`s.

7.3 Photon Track Output File

A photon track file records the photon position after each propagation step. A photon track file is generated if the `photon_track_file` parameter is set to something that is not blank in the master input file (§**??**).

This file is useful for plotting photon trajectories since it records intermediate points between points where the photon hits a wall. However, if a large number of photons are generated, this file can be very large.

Example track file output:

```
  1    1  -0.00082    0.00005    67.78576    0      0.00000    0.00000    0.00000
  1    1   0.04500   -0.00022    70.51507    0      0.03230   -0.00010    0.99947
  1    1  -0.03930   -0.01381    71.21681    0     -0.11528   -0.01921    0.99314
```

```
2   4  -0.00074    0.00005    67.85150   0      0.00000    0.00000    0.00000
2   4   0.04500    0.00012    70.57826   0      0.03227    0.00002    0.99947
3   5  -0.00065    0.00005    67.91725   0      0.00000    0.00000    0.00000
3   5   0.04500    0.00046    70.64145   0      0.03224    0.00015    0.99948
3   5  -0.04500   -0.00206    71.03689   0     -0.21972   -0.00623    0.97554
4   7  -0.00057    0.00005    67.98299   0      0.00000    0.00000    0.00000
4   7   0.04500   -0.00037    70.70463   0      0.03221   -0.00015    0.99948
4   7  -0.04500   -0.00149    71.23479   0     -0.16439   -0.00208    0.98639
5   9  -0.00049    0.00005    68.04873   0      0.00000    0.00000    0.00000
5   9   0.04500    0.00039    70.76781   0      0.03218    0.00012    0.99948
```

The columns of this file are

```
1:     Photon index
2:     Generated photon index.
3-5:   (x, y, s)                ! Coordinates of photon
7:     ix_branch                ! Index of lattice branch photon is in.
7-9:   (vx, vy, vs)             ! Velocity of the photon
```

Column 1 gives the photon index which gives a count of the photons that have been tracked and passed the filter restrictions. Only photons that have passed the filter restriction have their tracks recorded in the file. Thus the photon index numbers will be consecutive.

Column 2 gives the photon generation index. Each tracked photon is given a unique generation index starting from 1. Thus for the example above, photons with generation index 1, 4, 5, 7 and 9 passed the filter tests and thus are present in the file. Photons with generation index 2, 3, 6, and 8 did not pass the filter tests and are not shown in the file.

Columns 3 through 5 give the position of the photon.

Columns 7 through 9 gives the velocity of the photon.

Chapter 8

Test Modes

Test modes are used to generate diagnostic data files. tests are also useful for generating data for plotting reflection statistics.Test modes are specified using the `-test <what>` option on the command line (§??).

```
-test monte_carlo_reflection ! §??
-test specular_reflection    ! §??
-test diffuse_probability    ! §??
```

These test modes are explained below. In all cases, the `synrad3d_parameters` namelist in the master input file (§??) is ignored and instead another namelist is read in from the master input file as specified in the documentation below.

8.1 Monte Carlo Reflection Test

The `monte_carlo_reflection` test mode, invoked by using the `-test monte_carlo_refleciton` on the command line (§??). The output of this test are a number of lines recording the direction of the outgoing scattered photon under the conditions where the photon energy and incoming graze angle are fixed.

The parameters used for the test are read in from the master input file specified on the command line (§??). The namelist used for the input parameters for the reflection test is called `reflection_test`. Example:

```
&reflection_test
  surface_reflection_file      = ""      ! Reflection probability file
  graze_angle_in               = 0.12    ! Incident grazing angle in radians.
  energy                       = 100     ! Photon energy in eV
  surface_roughness_rms        = -1      ! Roughness for diffuse scattering.
  roughness_correlation_len    = -1      ! Roughness correlation length.
  n_photons                    = 1000    ! Number of reflections simulated
  random_seed                  = 0       ! Random number seed.
```

```
  include_specular_reflections = F       ! Diffuse reflections only?
  output_file                  = ""      ! Blank => Use default name.
/
```

The `surface_reflection_file` specifies the reflection probability file to be used. If no file is specified then the default surface is used (§**??**).

The `graze_angle_in` is the incident grazing angle in radians and `energy` in the photon energy in eV. `n_photons` are the number of reflections to simulate, and `output_file` is the name of the output data file. The default data file name is "test_monte_carlo_reflection.dat". The other parameters are explained in §**??**.

If the `include_specular_reflections` parameter is set to `True`, specular reflection is allowed. Otherwise only diffuse reflections are simulated. The default is `False`.

The `surface_roughness_rms` parameter sets the surface roughness. If negative then the roughness specified by the surface is used.

The `roughness_correlation_len` parameter sets the surface roughness correlation. If negative then the correlation specified by the surface is used.

`random_seed` is the random number seed used in by the random number generator. If set to 0, the system clock will be used. That is, if set to 0, the output results will vary from run to run.

The output contains a header with the input parameters followed by `n_photon` lines. Each line is of the form:

```
theta_out  phi_out
```

where `theta_out` is the grazing angle of the reflected photon in radians. `theta_out` = 0 means the photon is traveling parallel to the surface. The `phi_out` angle is the azimuthal angle in radians with zero `phi_out` indicating that the incident ray, the surface normal, and reflected rays are in the same plane.

8.2 Specular Reflection Test

Specular reflection test mode (§**??**), invoked using the `-test specular_reflection` option on the command line, is used for studying specular (angle in = angle out) reflections from a given set of photon initial conditions using the chamber wall specified by the `wall_file` parameter in the Master Input file. Each photon is allowed a single specular reflection and the initial and final photon position and velocity are recorded in the output file.

The parameters used for the test are read in from the master input file specified on the command line (§**??**). The namelist used for the input parameters for the reflection test is called `specular_reflection_test`. Example:

```
&specular_reflection_test
  photon_start_input_file = "photon.start"  ! Photon starting positions
  lattice_file            = "lat.bmad"      ! Lattice file
```

```
  wall_file               = "synrad3d.wall" ! Wall definition file
  output_file             = ""     ! Default is "test_specular_reflection.dat"
/
```

The `photon_start_input_file` is the file of initial photon positions. The format of this file is given in `s:photon.start`. Note that for the specular test, values for the photon energy and the random number generator state in the initial photon position file do not affect the results and do not have to be set. Also: The if the magnitude of the velocity $\sqrt{v_x^2 + v_y^2 + v_z^2}/c$ is not too far off from 1 then `Synrad3D` will renormalize so that $\sqrt{v_x^2 + v_y^2 + v_z^2}/c$ will be 1.

`lattice_file` give the name of the lattice. See the Bmad manual for more details.

The `wall_file` gives the name of the vacuum chamber wall definition file. See §**??** for more details.

The `output_file` gives the name of the output data file. See §**??** for details of the syntax for this file.

8.3 Diffuse Probability Test

The `diffuse_probability` test mode, invoked using the `-test diffuse_probability` option on the command line, creates a table of diffuse reflection probabilities.

The parameters used for the test are read in from the master input file specified on the command line (§**??**). The namelist used for the input parameters for the reflection test is called `diffuse_probability_test`. Example:

```
&diffuse_probability_test
  surface_reflection_file   = ""      ! Reflection probability file
  graze_angle_in            = 0.12    ! Incident grazing angle in radians.
  energy                    = 100     ! eV
  surface_roughness_rms     = -1      ! Roughness for diffuse scattering.
  roughness_correlation_len = -1      ! Roughness correlation length.
  prob_normalization        = "1"     ! Probability normalization.
  output_file               = ""      ! Blank => Use default name.

  row_type                 = "energy" ! Table row type.
  row_min                  = 10       ! Min row value
  row_max                  = 1000     ! Max row value
  row_n_pts                = 101      ! Number of row lines.
  row_log_scale            = T        ! Use log scale?
  row_values               = 0.001, 0.003, 0.01 ! Alternative way to
                                      ! specify row values.

  graze_angle_out_min      = 0.001    ! Minimal value in radians
  graze_angle_out_max      = 0.010    ! Maximal value
```

```
  graze_angle_out_n_pts    = 10       ! Number of graze angle columns.
  graze_angles_out         = 0.001, 0.003, 0.01 ! Alternative way to
                                      ! specify outgoing graze angles.
/
```

The output is a table. Each row shows the reflection probability for different values of the parameter specified by `row_type`. Possible `row_type` settings are:

```
"azimuth_angle_out" ! Outgoing azimuth angle of the photon.
"energy"            ! Photon energy rows.
"roughness"         ! Surface roughness.
"correlation"       ! Surface correlation length.
"graze_angle_in"    ! Incoming photon graze angle.
```

The output table has a number of columns. The first column is an index number from 1 to the number of rows in the table. The second column is the value of what `row_type` is being used. The other columns will be the diffuse reflection probability for a certain value of outgoing graze angle.

If the `row_type` is set to `"azimuth_angle_out"`, the rows will be the outgoing azimuth angle ϕ of the photon. In this case, the probabilities shown in the table will be the probability $P(x, \phi)$ of scattering a photon with outgoing orientation x and ϕ where $x = \sin(\theta_{\text{g,out}})$ with $\theta_{\text{g,out}}$ is the outgoing graze angle (see Equation A138 of [**?**]).

If `row_type` is set to anything else but `"azimuth_angle_out"`, the probabilities shown in the table will be the probability $P_x(x)$ which is the scattering probability integrated over ϕ (see Equation A140 of [**?**]).

The values used for the parameter specified by `row_type` can be set in one of two ways. One way is to specify an explicit list of values using the `row_values` parameter. Example:

```
row_values = 0.001, 0.003, 0.01
```

The alternative way is to set `row_min`, `row_max`, `row_n_pts` and `row_log_scale`. This will create `row_n_pts` number of rows with the `row_type` parameter ranging from `row_min` to `row_max`. The points will be evenly spaced if `row_log_scale` is set to `False` and will be evenly spaced on a log scale if `row_log_scale` is set to `True`. Note: If `row_n_pts` is positive then this alternative is used. Otherwise the values from `row_values` are used.

Similar to specifying row values, the column values for the outgoing graze angle can be set either by setting `graze_angles_out` to a list of values or by setting `graze_angle_out_min`, `graze_angle_out_max`, and `graze_angle_out_n_pts`.

The `surface_reflection_file` specifies the reflection probability file to be used. If no file is specified then the default surface is used (§**??**).

The `graze_angle_in` parameter sets the incoming graze angle but is ignored if `row_type` is set to "graze_angle_in".

The `energy` parameter sets the photon energy but is ignored if `row_type` is set to "energy".

The `surface_roughness_rms` parameter sets the surface roughness. If negative then the roughness specified by the surface is used. This parameter is ignored if `row_type` is set to "roughness".

The `roughness_correlation_len` parameter sets the surface roughness correlation. If negative then the correlation specified by the surface is used. This parameter is ignored if `row_type` is set to "correlation".

The `prob_normalization` parameter sets how the displayed probabilities are normalized. Possible settings are:

```
"1"
"reflect"
"all"
```

A setting of "1" means that the displayed probability integrated over all outgoing angles (and keeping everything else fixed) is 1. That is, absorption and specular reflection probabilities are ignored. A setting of "reflect" means that the integrated probability will be the probability of being diffusely reflected relative to the probability of being reflected and not absorbed. That is, absorption is ignored. Finally, a setting of "all" means that the integrated probability will be the probability of being diffusely reflected taking into account both absorption and specular reflection probabilities.

The default file name for the output data file is `test_diffuse_probability.dat`.

Appendix A

Spline Fit of the Diffuse Probability Function

The diffuse scattering theory used in `Synrad3D` is discussed by Dugan and Sagan[**?**] (D&S) with one change pertaining to how the integrated diffuse probability function $C_x(x)$ (D&S Eq. A144) is evaluated.

In D&S, C_x was evaluated using a Chebyshev fit to the probability function $P_x(x)$ in the interval $[0, 1]$. While this gives good results with relatively rough surfaces where P_x is fairly broad smooth function, the fit will be poor when the surface roughness is small and P_x is a peaked function with the peak centered at $x = y$. [That is, the diffuse reflection becomes specular like (angle out = angle in) in the limit where the surface is smooth.] The reason why the Chebyshev fit becomes poor when P_x becomes peaked is due to the fact that a Chebyshev fit is a polynomial fit and it takes a polynomial of large degree to fit a peaked function well.

As a replacement, a spline fit with adaptive point selection was implemented. The Akima spline used[**?**] has the advantage of locality (the calculated slope at a knot point is only affected by the local knot points around it) and the ability to better handle large changes in slope since an Akima spline does not try to maintain a continuous second derivative. The adaptive point selection is used to better fit peaked functions. The basic idea here is to find at what knot point the estimated integrated area error is largest and then to add knot points to either side of this knot point. The integrated area error at a given knot point is estimated by the difference between the integrated area from the previous knot point to the next knot point using the spline fit and the integrated area as calculated from a cubic fit to the three knot points (previous point, point under consideration and the next point). Adding points is stopped when the estimated error for the total integral is below some fraction of the total integral.

Appendix B

Default Reflectivity Tables.

The default reflectivity used by `Synrad3D` is based on a calculation for a 10 nm C film on an Al substrate. Figure **??** shows the parameters used to generate the reflectivity tables. The tables are from the LBNL X-ray data base found at:

`http://henke.lbl.gov/optical_constants/layer2.html`

The polarization choice, $P = -1$, was made since this corresponds to the polarization direction of the direct synchrotron radiation striking the chamber surface, on the first encounter. It is probably not really correct for subsequent scatterings. However the polarization dependence of the reflectivity is not very strong.

The default surface roughness for diffuse scattering is 200 nm and the default surface roughness correlation length is 5.5 μm. These values can be modified by setting `surface_roughness_rms` and `roughness_correlation_len` in the master input file (§**??**).

Note: There is a script for downloading surface reflectivity data and creating reflectivity files for use with `Synrad3D`. See section §**??**.

Layered Mirror Reflectivity

- Layer Material: C (enter chemical formula).
- Layer Density: -1 gm/cm^3 (enter negative value to use tabulated values.)
- Layer Thickness: 10 nm
- Top Surface Roughness: 0 nm (Sigma).
- Substrate Material: Al (enter chemical formula).
- Substrate Density: -1 gm/cm^3 (enter negative value to use tabulated values.)
- Substrate Roughness: 0 nm (Sigma).
- Polarization: -1 (-1 < pol < 1) where s=1, p=-1 and unpolarized=0.
- Scan Photon Energy (eV) from 30 to 300 in 27 steps (< 500).
  (NOTE: Energies must be in the range 30 eV < E < 30,000 eV, Wavelength between 0.041 nm < Wavelength < 41 nm, and Angles between 0 & 90 degrees.)
- At fixed Angle (deg) = 1

To request a Linear Plot press this button: Submit Request

To reset to default values, press this button: Reset .

Explanation of Tables

Material
: The chemical formula is required here. Note that this is case sensitive (e.g. CO for Carbon Monoxide vs Co for Cobalt).

Density
: If a negative value is entered, the chemical formula is checked against a list of some common materials. If no match is found then the density of the first element in the formula is used.

Grazing Angle
: In keeping with the standard notation for the x-ray region the incidence angle is measured relative to the surface (NOT the surface normal).

Polarization
: Pol = 1 corresponds to s-polarization (electric field perpendicular to the plane of incidence). Pol=-1 corresponds to p-polarization (electric field in the plane of incidence). Pol=0 for unpolarized radiation.

Output
: A GIF plot is generated for viewing the results. For numerical values, follow the link above the GIF plot to an ASCII text file. For a nice looking printed copy, you might try using the EPS file.

Figure B.1: Input parameters for the default reflectivity curves used by `Synrad3D`.

Appendix C

Customizing Synrad3D

Synrad3D is designed so that custom code, created by you, can be used for generating photons. This custom code is compiled with the Synrad3D library to create a custom executable.

C.1 Setup

Creating a custom version of Synrad3D involves creating custom code that is put in a directory that is distinct from the bsim directory which is the directory that contains the standard Synrad3D code files.

It is important to remember that the code in the bsim directory is not to be modified. This ensures that, as time goes on, and as Synrad3D is developed, changes to the code in the bsim directories will have a minimal chance to break your custom code. If you do feel you need to change something in the bsim directory, please seek help first.

To setup a custom Synrad3D version do the following:

1. Establish a base directory in which things will be built. This directory can have any name. Here we will call this directory ROOT. This base directory can be located anywhere. It is recommended that if you have a local Bmad Distribution, that you keep the base directory separate from the Distribution

2. Make a subdirectory of ROOT that will contain the custom code. This directory can have any name. Here this directory will be called synrad3d_custom.

3. Copy the files from the directory bsim/synrad3d/custom to ROOT/synrad_custom. The bsim directory is part of the Bmad Distribution package. If you do not know where to find it, ask your local Guru where it is.

   There are three files in the bsim/synrad3d/custom directory. One file is a template file for creating custom code called

   ```
   photon_init_custom.f90
   ```

There is documentation in this file on how to customize it. To have this custom routine called, the `photon_start_input_file` (§**??**) must be set to "`CUSTOM`".

The other two files are CMake[1] script files which do not have to be modified. These files are

```
CMakeLists.txt
cmake.custom_synrad3d
```

These scripts are setup to make an executable called `custom_synrad3d`. This name can be changed by modifying the `cmake.custom_synrad3d` file.

4. Copy the file `bsim/synrad3d/synrad3d.f90` to `ROOT/synrad3d_custom`. This file should not be modified.

5. Go to the `ROOT/synrad3d_custom` directory and use the command `mk` to create the executable

   `ROOT/production/bin/custom_synrad3d.`

   Similarly, the command `mkd` will create a debug executable

   `ROOT/debug/bin/custom_synrad3d`

A debug executable only needs to be created if you a debugging the code.

[1] CMake is a program used for compiling code

Bibliography

[1] D. Sagan, "Bmad: A Relativistic Charged Particle Simulation Library" Nuc. Instrum. & Methods Phys. Res. A, **558**, pp 356-59 (2006).

[2] B.L. Henke, E.M. Gullikson, and J.C. Davis. X-ray interactions: photoabsorption, scattering, transmission, and reflection at E=50-30000 eV,=1-92, Atomic Data and Nuclear Data Tables Vol. 54 (no.2), 181-342 (July 1993) `http://henke.lbl.gov/optical_constants/`

[3] P. Beckmann & A. Spizzichino, *The Scattering of Electromagnetic Waves from Rough Surfaces*, Pergamon Press, New York (1963).

[4] J. A. Ogilvy, *Theory of Wave Scattering from Random Rough Surfaces*, Hilger, Bristol (1993).

[5] G. Dugan and D. Sagan, "Simulating Synchrotron Radiation in Accelerators Including Diffuse and Specular Reflections", Phys. Rev. Accel. Beams, **20**, 020708, (2017).

[6] G.Dugan and D.Sagan, "SYNRAD3D: Photon Propagation and Scattering Simulations," " INFN-LNF/INFN-Pisa/CERN-LER/EuCARD-AccNet Joint Workshop on Electron-Cloud Effects (ECLOUD12)", (2012).

[7] N. Mahne et. al., "Experimental Determination of ECLOUD Simulation Input Parameters for DAΦNE", EuroTev-Report-2005-013 (2005).

[8] J. D. Jackson, *Classical Electromagnetism*, third edition, Wiley, New York (1999)

[9] H. Akima, "A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures," J.ACM, vol. 17, no. 4, pp. 589-602, (1970).