Developing SGV to Study the ILD's Vertex Detector in the Very Forward Region

Leonardo Badurina, University College London, United Kingdom

September 6, 2017

Abstract

In this work, we present the fast detector simulation SGV 3.0 and underline the need to continually update fast detector simulations for physical analysis and hardware R&D. Firstly, we explain the features of SGV, identify the program's dependence on legacy code (`CERNLIB`) and highlight the work done to remove these dependencies. Secondly, we provide the results of a study, using SGV, to determine the optimal cable configuration in the ILD's Vertex Detector.

Contents

1 Introduction

In HEP experiments, detector simulation is of fundamental importance for both physics analysis and hardware-software development. Traditionally HEP simulations have been either exclusively physics simulations, which do not include information about the detector response, or complete detatiled simulations, which are very demanding in terms of CPU-cycles and data storage [1]. Fast detector simulations, like SGV - *La Simulation à Grande Vitesse* [2], have been developed since the early nineties to address the need of software at an intermediate level of detail, which allows users to generate significant amounts of simulated data on a limited number of CPUs.

Recently, the scope of fast simulation programs has shifted. As stated by M. Berggren, when referring to current research in the e^-e^+ International Linear Collider (ILC) [3]: "the latest years of development has brought forward very performant and complete full simulation packages, both in SiD [Silicon Detector] and ILD [International Large Detector]" [2]. Because of their speed and accuracy, efficient fast detector simulations like SGV are paramount to accurately validate detector concepts and simulate a large variety of processes predicted by SM, BSM and SUSY.

In accordance with the work conducted at DESY between mid-July and September 2017, this report is structured in two sections: *Developing SGV*, in which we will present the features of this fast detector simulation program and the recent work conducted to remove dependencies on legacy code; and *Very Forward Vertexing at the ILD*, in which we will present our findings regarding the optimisation of the cable configuration within the ILD's vertex detector to maximise the resolution of the impact parameter.

2 Developing SGV

2.1 Overview

SGV [2], *La Simulation à Grande Vitesse*, is a fast detector simulation developed in the early nineties to study the compatibility of the DELPHI detector in the upgrade of LEP I to LEP II, at CERN. The latest release, SGV 3.0, dates back to 2012 and its main author is M. Berggren. The program is organised into a SVN-managed Fortran95 package that developed out of the well-tested SGV 2.0 series, which was written in Fortran77[1]. SGV is supported by Linux and Unix distributions and the assumed Fortran95 compiler is gfortran (GNU Fortran compiler).

SGV belongs to the class of sophisticated, fast and efficient detector simulations. Like all efficient detector simulations, the detector simulation time is comparable to the time required to generate events with an efficient event generator, like PYTHIA [4] or Whizard [5]. Unlike less sophisticated fast detector simulations - such as paramteric simulations and those in which generated four-vectors are simply smeared with assumed global properties - SGV calculates the total covariance matrix from the generated particles and detector geometry. Please see Appendix A for more information regarding the program structure of SGV.

2.1.1 Detector Description

In SGV, the detector geometry is described using cylinders, with a common axis parallel to the magnetic (B-) field, and planes perpendicular to the aforementioned axis [1]. Whilst cylinders are described by their radius and minimum-maximum height, planes are described by their position along the common axis and their minimum and maximum radius (because of rotational symmetry around the common axis). The thickness of the detector layers is given in terms of the radiation length and the precision of the detector components can also be set.

As will be discussed in subsequent sections, SGV cannot handle cones; thus, if cones need to be simulated, a combination of planes and cylinders covering the same angle must be employed.

2.1.2 Tracking Detectors

In SGV, the helix of a particle in a magnetic field is tracked to determine the number and type of detector layers intersected by the particle's helical trajectory, until the innermost surface of the outermost detector is reached. Depending on the geometry of the layer, cylindrical or Cartesian coordinates are used (see Fig. 1).

[1] This change resulted in a 15% increase in the program's speed.

The total covariance matrix at the perigee is then determined by using the Kalman filter method (written in elemental form), which is iterated from the last trajectory-layer intersection to the first hit[2].

The calculations are conducted in five-dimensional helix space using the following helix parameters [6]:

- ϕ_0, the azimuthal angle of the particle's linear momentum at the point of neareast approach (perigee);
- Ω, the curvature, where $|\Omega| = \frac{1}{R}$, R is the curvature of the track and the magnitude depends on the direction of motion relative to the B-field in the detector;
- d_0, the impact parameter in the $R\phi$ plane relative to a reference point (chosen to be $(0, 0, 0)$ in Cartesian coordinates);
- z_0, the z position of the track at the perigee with respect to the chosen reference point;
- $\tan\lambda = \frac{dz}{ds}$, where s is the path length of the helix in the $R\phi$ plane.

The perigee parameters are then smeared according to the calculated covariance matrix by Cholesky-decomposition of the matrix, such that the lower-triangular matrix multiplied by a vector filled with uncorrelated random variables returns the correlations of the calculated covariance matrix.

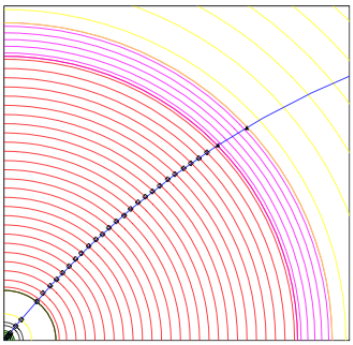


Figure 1: Cross-section of a detector with cylindrical geometry [2]. The trajectory of the particle is projected in the $R\phi$ plane. The intersections between the helical trajectory and the detector layers are shown as white crosses. Please note that the arrow shows the direction of the tracking method employed by the program.

[2] At each intersection, the surface measurement contributions to the covariance matrix are added in quadrature. The effects of multiple scattering at the surface are added in quadrature to the relevant elements of the inverted (weight) matrix, the matrix is inverted and tranlsated along the helix to the subsequent intersection, where the calculations are repeated. [2]

2.1.3 Calorimeters

In the calorimeter simulation, independently of the chosen process, the program extrapolates the charged or neutral particles from the intersection with the calorimeters. The parameters of the calorimeters - i.e. precision, particle-type to detect, shower type (hadronic, electromagnetic or none) - can be chosen by the user. The detector response is then simulated from the values contained in the geometry description input file. Since the code simulating the tracks, which lies at the core of SGV, is separate from the code simulating the calorimeter response, user routines can be used instead. Please note, however, that these must be substituted at compile-time.

SGV also allows for the simulation of electromagnetic interactions, such as pair-creation and bremsstrahlung, in the detector material, tracking efficiencies and scintillators or taggers. By default, the random error associated with the detected energy, shape and position of showers is included. However, association errors, due to the incorrect assignment of clusters to tracks, or the splitting or merging of clusters, are not considered. These can lead to errors in the total reconstructed energy.

2.2 CERNLIB Dependencies

As clearly stated in [2], although most previous `CERNLIB` dependencies have been removed from SGV 2.0, a number of dependencies remains in SGV 3.0. In effect, as of July 2017, `FFREAD` [7], `HBOOK` and a number of general purpose routines [8] were still used despite interrupted development and support of the `CERNLIB` package. In line with previous work, we proceeded to further reducing SGV's dependence on `CERNLIB`. Please note that `HBOOK` outputs data (tuples, plots, etc.) on all graphic devices. Despite being called in SGV, it is not a core feature of this fast detector simulation.

2.2.1 FFREAD

`FFREAD` is a set of Fortran-77 subroutines that allows format-free input processing. The correct usage of `FFREAD` is as follows:

1. The variables or arrays to set-up are stored in a Fortran `COMMON` block, which defines a block of main memory storage that allows different program units to share the same data [9].

2. After calling the initialisation subroutine `FFINIT(NW)`, previously defined keys are erased and the integer argument `NW` specifies the number of words the user has allocated to the `COMMON` block. Optional values (input logical unit to given value, output logical unit to given value and significant size of keys) can be set by calling `FFSET`.

3. The keys are defined by a call to `FFKEY`, which gives the location, type and length (only for arrays) of the key.

4. A call to the main subroutine of FFREAD, FFGO, reads in the data cards and modifies the variables in memory accordingly.[3]

In SGV, the subroutines of FFREAD are called in a number of user and non-user routines[4] [1]. The data cards are read from steering files containing information about the event-type, centre-of-mass energy, number of events to simulate, etc. If no variables are given in the steering files, or if all variables are commented out, then the values of the defined keys are not modified in memory (default values). See Appendix A to understand, with an example, how FFREAD is called in SGV.

2.3 General Purpose Routines

2.3.1 SORTZV & INTSOR

SORTZV and INTSOR [8] are sorting algorithms extensively used in SGV. While SORTZV (A,INDEX,N) sorts the first N elements of a one-dimensional array A containing characters, integers or real data and outputs an integer array INDEX containing the ordered indices indicating the order of the original array, INTSOR (A, N) sorts the first N elements of a one-dimensional array A and returns the sorted array. These routines are particularly efficient because they are based on the QuickerSort algorithm.

2.3.2 CROSS & CLTOU

As suggested by the name, CROSS(a,b,c) computes the vector (cross) product c of two three-vectors a and b. Letting $c = (c_1, c_2, c_3)$, we obtain the following relations:

$$c_1 = a_2 b_3 - a_3 b_2$$
$$c_2 = a_3 b_1 - a_1 b_3$$
$$c_3 = a_1 b_2 - a_2 b_1$$

The CLTOU subroutine converts lower case letters in a character string to upper case.

2.4 Proposed & Implemented Changes

2.4.1 Namelists

As clearly stated in [7], Fortran namelists [9] allow input of unformatted variables, much like FFREAD. We therefore decided to explore the possibilty of substituting the FFREAD routines with Fortran namelists.

[3] In FFREAD, the term *data card* refers to a variable with an associated value. This nomenclature is a relic of the punch-card era, when programs and numbers were called *cards* and *data*, respectively.

[4] The SGV routines which call FFREAD routines are: zxini, zxgord, zaord, zegord and zdord. For more information about their usage, see Appendix B and [1].

In Fortran, NAMELIST is declared as a non-executable statement in the main program and the values in it are listed. Differently from FFREAD, the input file of a namelist must begin with an ampersand followed by the name of the namelist and ends with a forward-slash. FFREAD and Fortran namelists also differ in that the latter uses equal signs between the name of a variable and its associated.

After extensive testing using example steering files, we determined that the optimal code structure when using NAMELIST largely maintains the ordering of calls in the SGV initialisation routine (zxini). Whilst the order of calls to zxgord, zegord and zaord is unchanged, the number of calls is halved. Since keys are no longer declared when using namelists, default values can be set in the variable declaration section of the aforementioned subroutines. Therefore, by reading the namelists relevant to each subroutine (zxgord, zegord and zaord) in each subroutine, the variables are modified in memory accordingly, without needing to read the variable values twice (mode = 0 and mode = 1). Please see Appendices B and C for an example of how namelists have replaced FFREAD. Because of such changes, the steering file is no longer explicitly opened or read in zxini; rather, the steering file is divided into a number of namelists which are read in the respective subroutines.

2.4.2 General Purpose Routines

The subroutines CLTOU and CROSS were easily rewritten in Fortran95 and placed in the src/sgvlib directory. To rewrite SORTZV and INTSOR, we used a Fortran95 QuickerSort algorithm (qsor) [10] which took the following arguments: an array of real numbers and, optionally, an empty array, of equal length. If no optional argument was stated, the algorithm returns the sorted array; else, the algorithm returns both the sorted array and the array of ordered indices indicating the order of the original array. This routine is particularly compatible with SGV because SORTZ is called exclusively to sort arrays containing integers or reals numbers. To avoid compilation errors due to ambiguous variable declarations, we created two routines: SORTZV, to sort arrays of real numbers, and SORTZVN, to sort arrays of integers. To allow each subroutine to use the sorting algorithm, we placed qsor in a Fortran module.

2.5 Conclusions & Outlook

In conclusion, we successfully implemented Fortran namelists and completely removed FFREAD. This implementation, however, impedes backward compatibility on non-geometry steering files. Consequently, the rewriting of test- and sample-steering files is planned for future releases. Additionally, we also found solutions to the usage of general purpose and mathematical routines from the CERNLIB package (CROSS, SORTZV, INTSOR and CLTOU), by rewriting these in Fortran95.

Currently, work is being conducted to substitute more elaborate routines, like TRCHLU and EISRS2 [8], with ones contained in LAPACK 3.2 (Linear Algebra PACKage) [11].

Finally, future work will also necessarily have to concern the removal of the dependence on `HBOOK`, in order to fully achieve the task of removing SGV's dependence on the `CERNLIB` package.

3 Very Forward Vertexing at the ILD

3.1 Overview

3.1.1 The ILC and ILD

The International Large Detector (ILD) is a concept for one of the two planned detectors of the future International Linear Collider (ILC), which will collide electron-positron pairs at an initial centre-of-mass energy (CM) of 250 GeV [3].

By requiring that all particles in an event, charged and neutral, are individually reconstructed (particle flow), the ILD design stresses the importance of topological event reconstruction. This is particularly relevant when considering the objectives of the ILC physics programme: investigating the mechanisms of electroweak symmetry breaking conducting parameter scans of BSM and SUSY theories.

3.1.2 The Vertex Detector

The Vertex Detector (VTX) [3], see Fig. 2, is key to achieving very high performance flavour tagging by reconstructing displaced vertices. To guarantee the identification of heavy quarks and tau leptons, which are short-lived particles, the VTX of the ILD should be light and precise. It is also important in the reconstruction of low momentum particles which barely penetrate the detector, due to the strength of the magnetic field in the detector or the small production (polar) angle θ. We therefore assess the performance of a VTX by studying the resolution of the impact parameter (IP) of charged particles.

3.1.3 The Forward Region

The forward and very forward regions in the ILD refer to the polar angular θ range from approximately 5° to 30° , or equivalently $0.87 < \cos\theta < 0.996$. As will be shown later, the region of interest is $6° < \theta < 16°$, which we define as the *very* forward region of the vertex detecor. While angles less than 5° are still of much importance for achieving high statistics in a variety of SM processes, such as $e^+e^- \rightarrow Z/\gamma* \rightarrow \mu^+\mu^-$ and the higgs' various decay modes, accurate detection in this region is limited by the large densities of charged particle incident on the pixel strips; effectively, precision tracking is impossible in this region [12]. It is well known that the uncertainty on the IP, σ_{D0}, increases by one order of magnitude for $\theta < 6°$: from sub 40μm at $\theta = 20°$ to 500μm at $\theta = 4°$.

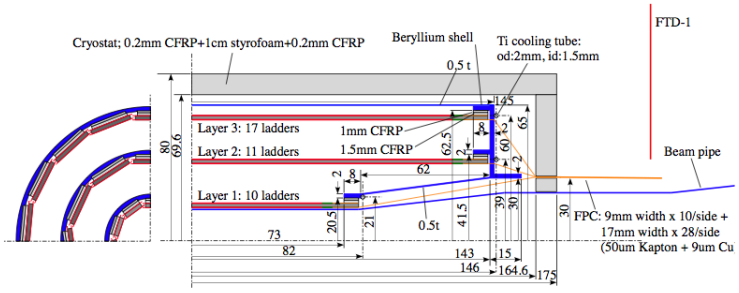


Figure 2: Cross-section of the ILD's vertex detector as seen in the Rz plane, where R is along the vertical direction and z is along the horizontal direction (along the B-field). This diagram shows the beryllium mechanical support structure of the detector (blue), the pixel disks (red), the readout components (green), the cryostat (light grey) and the cables (orange). Please note that the diagram shows a quarter of the detector's cross-section because the detector exhibits cylindrical, or barrel, symmetry along the axis parallel to the B-field, and reflective symmetry in the $R\phi$ plane - i.e. the forward and backward regions are identical.

3.1.4 Motivation

Following previous unpublished studies conducted by M. Berggren using SGV, we set out to investigate the minimisation of σ_{D0} in the very forward region by simulating the detector response for different cable configurations within the VTX's structure. M. Berggren has previously shown that the resolution of the IP can be improved by re-routing the cables outwards from the VTX, towards the inner Silicon Tracker (SiT) layer, past the first two disks and down to the beam-pipe. Also, preliminary investigation were conducted to determine the possibility of using thin, instead of thick cables, thus reducing the thickness of the copper cables, in units of radiation lengths X_0, from $5\%X_0$ to $0.3\%X_0$.

3.2 Method

To simulate the detector reponse we used SGV 3.0 in *scan* mode to calculate the covariance matrix at a constant momentum of 2.5 GeV for $0°<\theta<40°$. In *scan* mode, single particles are shot through the detector at given polar angles and momenta and the detector response is subsequently simulated. In this study, we used electrons (JETSET code 11) and set the IP at the origin.

The geometry of the detector was modelled using the parameters defined in the Detailed Baseline Design (DBD) [12] and the cable thickness estimates were determined from previous work by M. Berggren. Importantly, since SGV cannot handle cones - such as parts of the beam-pipe, beryllium support shell and cables - we approximated

the material distribution using planes perpendicular to the detector's axis. Using basic trigonometry, we determined that the effective radiation length X_0' could be estimated as $X_0|\cot(\theta - \theta_{cone})|$, where θ_{cone} is the angle between the z-axis and the lateral area of the cone (see Fig. 3). For shallow angles, this correction led to an 25-fold increase in the radiation length, as seen by the particles travelling through the detector layers.

Differently from previous studies, we increased the accuracy of the simulation by dividing the disks into rings, each ring accounting for a different effective material thickness.

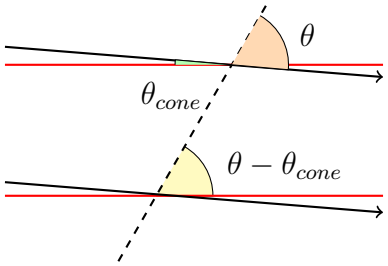


Figure 3: Schematic diagram showing the intersection between a particle trajectory (dotted line) and a detector layer (delimited by the two parallel red lines). The direction of the magnetic field, parallel to the z-direction, is shown using two parallel arrows. The length of the dotted line within the material is clearly given by the quotient of the detector thickness and $\sin(\theta - \theta_{cone})$, where θ is the polar angle describing the particle trajectory and θ_{cone} is the angle describing the lateral surface of the cone. When using discs to approximate cones, the distance travelled by a particle inside the perpendicular disk is equal to the thickness of the disk multiplied by a factor of $\frac{1}{\cos\theta'}$, where $\theta' = \theta - \theta_{cone}$. Therefore, to correctly account for the thickness of the conical surface, *as seen by the particle*, the thickness of the disk is given by the product of the conical thickness and $\frac{\cos\theta'}{\sin\theta'} = \cot\theta'$.

3.2.1 Cable Configurations

As shown in Fig. 4, the angular region of interests is between 9° to 15°, which corresponds to the Forward Tracking Detector (FTD-1) hit with smallest θ and the first VXT layer hit with smallest θ. To minimise the total radiation length in this region, we developed three possible cable configurations (see Fig.5):

- the cables run along the beryllium (Be) shell or support structure and are routed upwards;
- the cables run along the beryllium shell but are tunneled through the component parallel to the beampipe at `Z` = 14.6 cm and `R` = 3.0 cm (Be-cut);

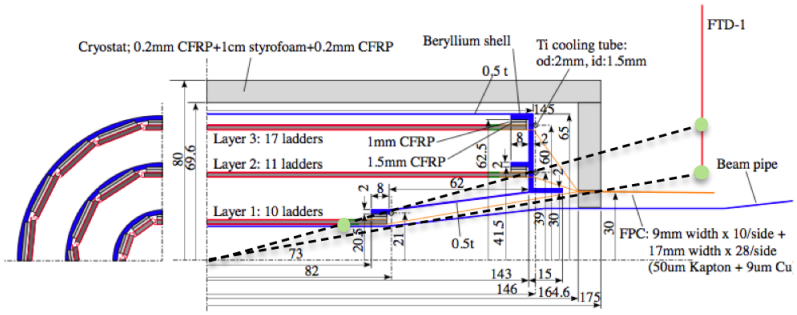


Figure 4: Cross-section of the ILD's vertex detector as seen in the Rz plane, showing the trajectories of the particle at the limits of the region of interest. Please note that the green dots indicate the intersections between the shown trajectories (black dotted lines starting from the origin) and the VTX layers.

- the cables from layer 1 are routed along the beam pipe and exit through the cryostat layer at Z = 16.46 cm and R = 3.0 cm and cables from layer 2 and 3 are routed upwards along the Be shell (split).
- the cables ables from layer 1 are routed inside the beryllium shell, which acts as a Faraday cage, at an angle of approximately 14.6° (F-cage). This configuration minimises the amount of material present in the angular region of interest.

Please note that each cable configuration was simulated for thin and thick cables and the cable configurations with highest IP precision were further simulated using an additional FTD-like pixel disk in the cryostat at Z = 8.15 cm and R = 1.6 cm.

3.3 Results

From the plots shown in Fig. 6, we observe that σ_{D0} increases sharply at approximately $\theta = 7°$. This feature is independent of cable configuration, as previously discussed. From baseline measurements, where no cables were introduced in the model, we also find that, in agreement with previous studies, σ_{D0} increases by a factor of five. For polar angles less than 15°, the particle trajectory firstly intersects the beryllium support structure and cryostat, and subsequently intersects FTD-1, where a hit is recorded by the layer's readout electronics. This simulation is of particular importance because it serves as a benchmark for optimal cable routing.

Other features of interest in the plot include the impact of additional cables running around the beryllium shell and the resolution increase that can be achieved for $10°<\theta<12°$ by simply running the cables through the beryllium extension at Z = 14.6 cm and R = 3.0 cm.

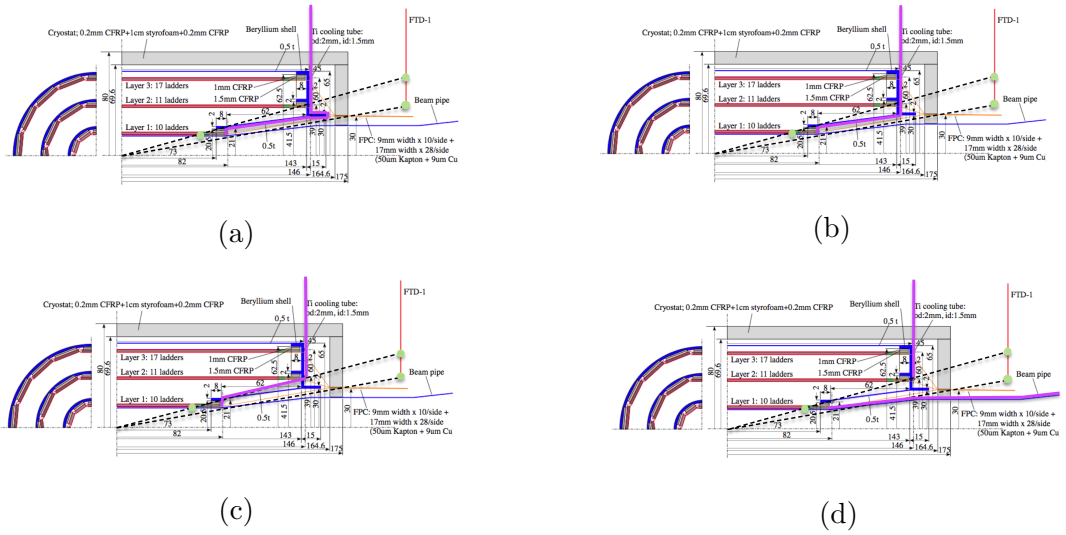


Figure 5: Diagrams showing the investigated cable configurations: (a) cables run along the outer beryllium shell; (b) cables run along the berylliu shell and are tunnelled through the beryllium kink; (c) cables run inside the beryllium shell; (d) cables from layer 1 run along the beam-pipe, whilst cables from layers 2 and 3 are routed upwards. Please note that the cables are drawn in magenta and that the orange lines should be ignored.

The split cable routing and the configuration in which the cables run inside the beryllium shell, within the Faraday cage show the highest level of precision across the range of interest. Despite almost replicating the simulation plot obtained for no-cable scenario, the latter arrangement is possibly unfeasible. The cables could interfere electromagnetically with the VTX layers, thus affecting the accuracy of the VTX across all angular ranges and undermining the purpose of the Faraday cage.

To study the possibility of using an additional FTD-like pixel disc, we used the split-cable and F-cage configurations, with thin and thick cables. As shown in Fig. 7, σ_{D0} increases by a factor of two for $\theta<11°$, when using thick cables. Because of the impossibility of fully covering the angular region of interest with an additional pixel disk, no improvement is seen when simulating a FTD-type pixel disk in cryostat at Z = 8.15 cm and R = 1.6 cm.

3.4 Conclusion & Outlook

Without additional FTD discs in the cryostat, we conclude that the resolution of the IP depends largely on cable thickness and routing, within the critical region $9°<\theta<15°$. The maximum resolution, for the optimal cable configuration, is approximately 120 μm for thick cables. With an additional FTD-like disk in the cryostat, we conclude that the resolution can be doubled, allowing for relatively constant resolution levels for $\theta>8°$. In both cases, however, we note that the optimal cable configuration should limit the

radiation length along the beam pipe, independently of the number of FTD-like discs placed within the cryostat.

Future work will necessarily have to consider detector assembly, if one of the above configurations is adopted, and further analysis using the complete ILD simulation package. Also, we recommend conducting this study at different low momentum values.

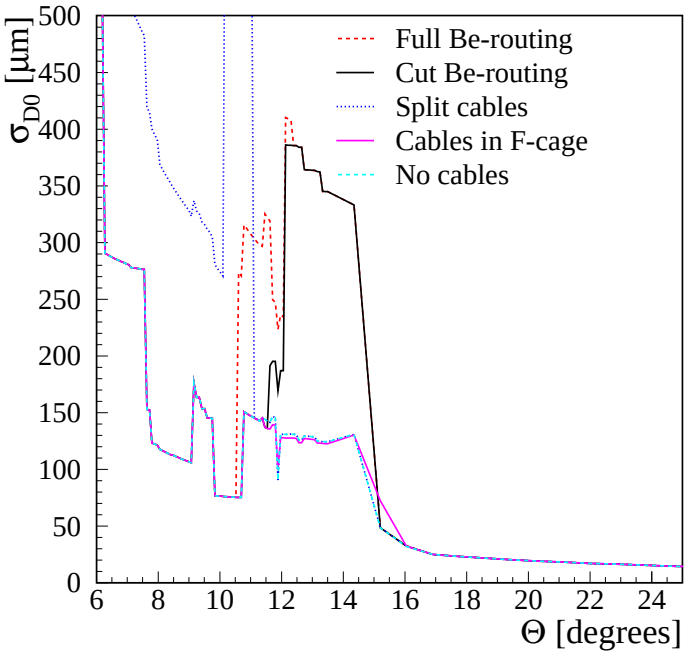
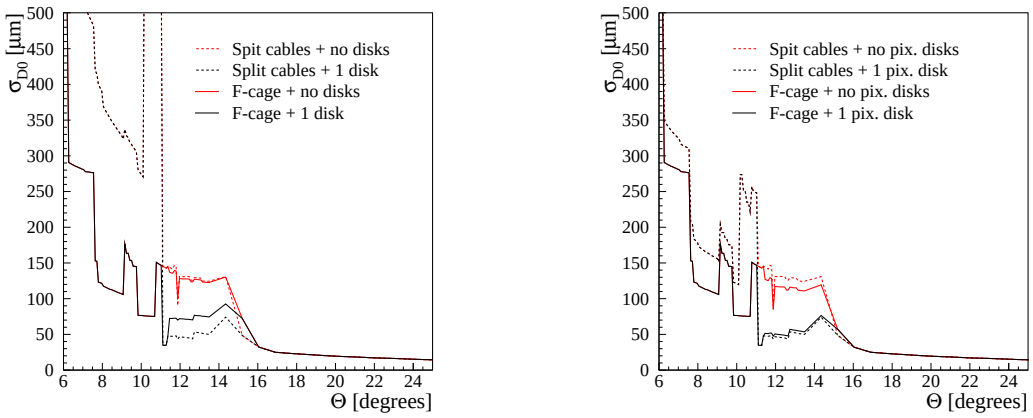


Figure 6: Plot showing the behaviour of σ_{D0} for $6°<\theta<24°$ for all cable configurations, where only thick cables have been simulated and no additional FTD pixel-discs have been introduced in the model.



(a) (b)

Figure 7: Plot showing the behaviour of σ_{D0} for $6°<\theta<24°$ when simulating F-cage and split-cable configurations, with and without additional FTD disks. (a) Thin cables (b) Thick cables.

4 Acknowledgements

I would like to take this opportunity to thank all of the people who have supported me during my stay at DESY. Firstly, I would like to thank the organisers of the the DESY Summer Student Programme for giving me the opportunity to work and study in a highly stimulating physics environment. Secondly, I would like to thank my supervisor, Mikael Berggren, for giving me the opportunity to work with him on SGV and detector optimisation. Thirdly, I would like to thank the FLC group at DESY for hosting me over the summer. Finally, I would like to thank my family and friends, old and new, for their support and encouragement.

Appendix A SGV Program structure

The overall core structure of SGV after installation is as follows, as detailed in the `README` file:

- **Top directory**, containing the installation script (*install*), the script to recompile libraries (*makesgvlibs*), the script to compile and link the user's program (*cresgvexe*) and template user routines and steering files.
- **Library subdirectory**, containing four subdirectories:
  - **sgvlib**, containing the compiled core routines;
  - **sgvdeflib**, with compiled defaults for some routines the user might wish to alter;
  - **sgvdumlib**, containing no-op defaults for some routines the user might wish to alter;
  - **sgvmodule**, containing binaries of interface-blocks;
- **Source-code subdirectory**, containing eight subdirectories:
  - **sgvdeflib, sgvdumlib, sgvlib, sgvmodule**, containing the source code of the libraries;
  - **sgvevsim**, containing the source-code to interface to different event generators (Pythia, Whizard, etc.);
  - **sgvinclude**, containig .h files to interface to external Fortran77 code, such as CERNLIB routines;
  - **sgvnews**, which is an empty directory for upgrades;
  - **sgvuser**, containing the source code of user routines.

Appendix B FFREAD in SGV

In this section, we will give an example of how `FFREAD` is called in SGV. The first program portion (B.1) is from the SGV initilisation routine `zxini`, which is a subroutine of the steering initialisation routine `zx_steering`. The second portion of code (B.2) is taken from the routine `zdord`, which sets the keys of the detector simulation and initialises the tracking.

B.1 zxini

```
          !.
          !. skipped beginning of program
          !.
USE zxste
           IMPLICIT NONE

    REAL, INTENT(INOUT)                          :: steer(4,0:*)

    !         FFREAD space
    REAL :: ffspa
    COMMON /cfread/ ffspa(1000000)

    CALL ffinit(1000000)

#if (defined(LINUX))
    OPEN(17,FILE='fort.17',STATUS='OLD',ERR=99)
#endif


    !-- Set FFREAD options

    CALL ffset('LINP', 17)
    CALL ffset('LOUT', 6 )
    CALL ffset('SIZE', 32    )
    CALL ffset('LENG', 132)

    !       Define cards for...
    !       general stuff
    CALL zxgord(0,steer)
    !       event generator
    CALL zeord(0,steer)
    !       detector simulation
    CALL zdord(0,steer)
```

```
!       and analysis.
CALL zaord(0,steer)
!       Read cards.
CALL ffgo
!       Do initalization using the values just read.
CALL zxgord(1,steer)
IF ( steer(1,1) == 0. ) THEN
  CALL zeord(1,steer)
END IF
CALL zdord(1,steer)
CALL zaord(1,steer)

CLOSE(17)
 ! .
 ! .  code continues...
 ! .
```

B.2 zdord

```
USE zxsiz

      IMPLICIT NONE

INTEGER, INTENT(IN)                          :: mode
REAL, INTENT(OUT)                            :: steer(4,0:*)

INTEGER :: iii
LOGICAL :: vdhits,mtkr,chshow,genbc,prdet,pldet
INTEGER :: vdlays,ndets,usesvp
REAL*4 :: pminbr,pminpa,ptloslim
COMMON /dsivari/ vdhits,mtkr,chshow,genbc,prdet,vdlays,ndets,&
    usesvp(maxste-17),pldet
COMMON /dsivarr/ pminbr,pminpa,ptloslim


IF ( mode == 0 ) THEN

!          Generate hits in the vertex detector ? If so, the VD will
!          not be included in the track parameters generated by ZDETSI.
!          Instead, the user must supply code to use the hit-pattern in
!          her analyising routines.
  vdhits = .false.
  CALL ffkey( 'VDHITS',vdhits,1,'LOGICAL')
```

```
!          Number of layers in the vertex detector (if hits generated)
  vdlays = 1
  CALL ffkey( 'VDLAYS',vdlays,1,'INTEGER')

!          Generate full track parameters (w/ error-matrix) or just
!          pxpypz and production points.
  mtkr = .true.
  CALL ffkey( 'MTKR',mtkr,1,'LOGICAL')

!          Number of detector geometries to simulate.
  ndets = 1
  CALL ffkey( 'NDETS',ndets,1,'INTEGER')


!          Generate showers in calorimeters also for charged
!          particles (else only for neutrals).
  chshow = .true.
  CALL ffkey( 'CHSHOW',chshow,1,'LOGICAL')

!          Generate brems and photon conversions in the detector
!          material
  genbc = .false.
  CALL ffkey( 'GENBC',genbc,1,'LOGICAL')

!          Minimum electron momentum to generate brems.
  pminbr = 0.3
  CALL ffkey( 'PMINBR',pminbr,1,'REAL')

!          Minimum photon momentum to generate pair-production.
  pminpa = 0.3
  CALL ffkey( 'PMINPA',pminpa,1,'REAL')

!          Minimum fraction of pt after to pt before the brems for the
!          original electron to be kept for the tracking
  ptloslim = 0.9
  CALL ffkey( 'PTLOSLIM',ptloslim,1,'REAL')

!          Send particles with these codes to analysis
!          By default, all particles with non-zero lifetime
!          are sent and need not be specified here.
!          Use the LUND partcle codes.
  usesvp=0.
  CALL ffkey('SAVE_PARTICLES',usesvp,100,'INTEGER')
```

```
!          Print the geometry of the dectector after loading.
  prdet = .false.
  CALL ffkey( 'PRDET',prdet,1,'LOGICAL')
!          Print the geometry of the in a form suitable for
!          drawing it in PAW
  pldet = .false.
  CALL ffkey( 'PLDET',pldet,1,'LOGICAL')
ELSE

  steer(3,0) = ndets
  IF ( vdhits ) THEN
    steer(3,1) = vdlays
  ELSE
    steer(3,1) = 0
  END IF

  IF ( mtkr ) THEN
    steer(3,2) = 1
  ELSE
    PRINT *, ' Sorry, right now You must tranfere the full'
    PRINT *, ' track-parameters and errors to the analysis.'
    PRINT *, ' The program will ignore that You set MTKR to'
    PRINT *, ' .FALSE.'
    steer(3,2) = 1
  END IF

  IF ( chshow ) THEN
    steer(3,3) = 1
  ELSE
    steer(3,3) = 0
  END IF

  IF ( pldet ) THEN
    prdet = .true.
  END IF

  IF ( genbc ) THEN
    steer(3,4) = 1
    steer(3,5) = pminbr
    steer(3,6) = pminpa
    steer(3,7) = ptloslim
  ELSE
    steer(3,4) = 0
    steer(3,5) = 0.0
```

```
    steer(3,6) = 0.0
    steer(3,6) = 0.0
  END IF

  steer(3,17) = count(( usesvp(1:maxste-17) /= 0 ))
  steer(3,17+1:maxste) = usesvp(1:maxste-17)


  IF ( steer(1,1) == 0.0 ) THEN
!             Initialise the tracking.
    CALL ztini(steer,3.0,prdet,pldet)

  END IF
END IF
END SUBROUTINE zdord
```

Appendix C Namelists in SGV

In this section, we will give the same examples from Appendix B, namely `zxini` and `zdord`. The major difference, however, is the use of namelists instead of `FFREAD`.

C.1 zxini

```
          !.
          !. skipped beginning of program
          !.
USE zxste
           IMPLICIT NONE

    REAL, INTENT(INOUT)                          :: steer(4,0:*)

    CALL zxgord(1,steer)
    IF ( steer(1,1) == 0. ) THEN
      CALL zeord(1,steer)
    END IF
    CALL zdord(1,steer)
    CALL zaord(1,steer)

    CLOSE(17)
        !.
        !. code continues
        !.
```

C.2 zxdord

```
USE zxsiz

      IMPLICIT NONE

INTEGER, INTENT(IN)                          :: mode
REAL, INTENT(OUT)                            :: steer(4,0:*)

       INTEGER :: iii
       LOGICAL :: vdhits =.FALSE.,&
                  mtkr = .TRUE.,&
                  chshow = .TRUE.,&
                  genbc = .FALSE.,&
                  prdet = .FALSE.,&
                  pldet = .FALSE.
       INTEGER :: vdlays = 1,&
```

```
                  ndets = 1
       INTEGER :: usesvp(100) = 0
       REAL*4 :: pminbr = 0.3,&
                 pminpa = 0.0,&
                 ptloslim = 0.9
COMMON /dsivari/ vdhits,mtkr,chshow,genbc,prdet,vdlays,ndets,&
    pldet,usesvp!(maxste-17)
COMMON /dsivarr/ pminbr,pminpa,ptloslim

NAMELIST /detste/ vdhits,vdlays,vdlays,mtkr,ndets,chshow,chshow,genbc,&
         pminbr,pminpa,ptloslim,usesvp,prdet,pldet

IF ( mode == 1 ) THEN

#if (defined(LINUX))
  OPEN(10, FILE = 'fort.10') ! input file is opened
#endif
  READ(10, NML = detste) ! namelist from opened file is read

  steer(3,0) = ndets
  IF ( vdhits ) THEN
    steer(3,1) = vdlays
  ELSE
    steer(3,1) = 0
  END IF

  IF ( mtkr ) THEN
    steer(3,2) = 1
  ELSE
    PRINT *, ' Sorry, right now You must tranfere the full'
    PRINT *, ' track-parameters and errors to the analysis.'
    PRINT *, ' The program will ignore that You set MTKR to'
    PRINT *, ' .FALSE.'
    steer(3,2) = 1
  END IF

  IF ( chshow ) THEN
    steer(3,3) = 1
  ELSE
    steer(3,3) = 0
  END IF

  IF ( pldet ) THEN
    prdet = .true.
```

```
  END IF

  IF ( genbc ) THEN
    steer(3,4) = 1
    steer(3,5) = pminbr
    steer(3,6) = pminpa
    steer(3,7) = ptloslim
  ELSE
    steer(3,4) = 0
    steer(3,5) = 0.0
    steer(3,6) = 0.0
    steer(3,6) = 0.0
  END IF

  steer(3,17) = count(( usesvp(1:maxste-17) /= 0 ))
  steer(3,17+1:maxste) = usesvp(1:maxste-17)


!  IF ( steer(1,1) == 0.0 ) THEN
!             Initialise the tracking.
    CALL ztini(steer,3.0,prdet,pldet)

!  END IF
  WRITE(18, NML = detste)
  CLOSE(10)
END IF
END SUBROUTINE zdord
```

References

[1] M. Berggren, *SGV 2.31-User's Guide*.

[2] M. Berggren, "SGV 3.0 - a fast detector simulation", arXiv:1203.0217 (2012).

[3] T. Behnke et al., *The International Linear Collider technical design report Volume 4: detectors*, arXiv:1306.6329.

[4] T. Sjstrand, S. Mrenna and P.Z. Skands, "A brief introduction to PYTHIA 8.1", *Comput. Phys. Commun.* **178** (2008) 852, arXiv:hep-ph/0603175.

[5] J. Reuter et al., "Modern particle physics event generation with WHIZARD", *J. Phys. Conf. Ser.* **608** (2015) 012063, arXiv:1410.4505.

[6] Thomas Kraemer, "Track Parameters in LCIO", LC-DET-2006-004 (2006).

[7] "FFREAD: Format Free Input Processing", *CERN Program Library Long Writeups*, I302/Q123 (1993).

[8] *CERN Program Library Short Writeup* (1992).

[9] K. Holcomb, "SCIENTIFIC PROGRAMMING IN FORTRAN 2003" (2012).

[10] A. Ramos, A FORTRAN 90 Numerical Library (AFNL), https://sourceforge.net/projects/afnl/.

[11] E. Anderson *et al.*, *Lapack's Users' Guide*, (Third ed.), Philadelphia, PA: Society for Industrial and Applied Mathematics (1999).

[12] S. Aplin et al., "Forward tracking at the next e^+e^- collider part II: experimental challenges and detector design", *JINST* **8** T06001 (2013).