00 SOFTWARE FOR THE ALIGNMENT OF THE CMS EXPERIMENT AT CERN

Pedro Arce, Instituto de Fisica de Cantabria

1. Description of the CMS alignment system

The CMS experiment design emphasizes the accurate measurement of the muon momentum in the range 5 GeV to 1 TeV (see [1]). To reach the desired accuracy, the position of the muon chambers has to be known with precisions of the order of 100 μm. With this aim a complex optical alignment system has been designed (see [2]), which is composed of three parts that correspond to the main parts of the detector: the alignment of the barre1 muon chambers, the alignment of the end cap muon chambers, and the link between both chambers and the inner tracker subdetector. The total number of elements in the three systems is around seven thousand.

The purpose of the CMS optical alignment software is to analyze the data taken by all these elements and reconstruct the position of the muon chambers with respect to each other and with respect to the inner tracker reference system and to propagate the errors of the measurements to the errors in the positions of the chambers.

The software for data taking will be agreed by the collaboration and will probably be a commercial one like Labview© or Bridgeview©; therefore it is not a subject for the software described here. The alignment software has nevertheless to read this data, probably written to an Objectivity© Object Oriented database and will have to write its results in the same format so that they can be used by the reconstruction software. If will be convenient that the software has the capability to run on-line, as it may be needed to detect a misplacement of the chambers for triggering purposes ; in any case, the alignment system will only take data every few seconds or minutes.

2. Basics of the software

As the rest of the software for the CMS experiment, the core of the optical alignment software has to be written in the langauge C++ following the object oriendted paradigm. The GUI has though been written in Java, taking advantages of the portability and distributed properties of this langauge. As mentioned above, the choice for the database will be an object oriented one and the Objectivity Inc. firm will probably be the final choice (see [3]). The interface with this database can be written in C++ or Java and given the current tendencies of the market we will likely write the final version in Java, although the current implementation in the alignment software is written in C++.

A first version of the software for optical alignment has already been released under the name *Simulgeo++*. The basic philosophy is similar to the Simulgeo software [4]: making a geometrical approximation to the light propagation and solving the system through a non-linear chi square fit, where the derivatives of the positions and angles with respect to the measurements are obtained with a computer-based numerical method propagating the light ray(s) through the system elements.

In its current phase this software has been designed as a versatile tool able of studying any optical system through a geometrical approximation. *Simulgeo++* converts the measurements of the optical system into the positions and angles of the objects that compose the optical system (plus other interna1 parameters of these objects), propagating also the errors from the measurements to the positions and angles. At present it is able to treat optical systems with any combination of the following basic optical objects:

laser, point source, lens, mirror, splitter, rhomboid_prism, pseudo_penta_prism, Risley_prism, 2Ddetector, camera-detector', distancemeter[2], tiltmeter

In its phase II, it will be tailored for the data taking of CMS, optimizing its speed for the system and interacting with the on-line data, the slow control system and the reconstruction software.

2.1 Software engineering process

The software is being written as a software engineering process: the first step was writing the list of User Requirements in a close interaction with the potential users. The list of User Requirements can be seen in Table 1. The User Requirements were then traslated into Use Cases, the first of which were the basis for the first iteration of analysis and design.

Table 1 – User Requirements of the software

No	REQUIREMENT
1	Simulgeo++ shall convert the measurements of any optical system into the positions and angles with errors of the objects that compose the optical system
2	Simulgeo++ shall read the description of the optical system from a text file or from an Objectivity DB file
3	Simulgeo++ shall read the measurements from a text file or from an Objectivity DB file. The format of this file will be taken from the system description file
4	User shall be able to set if a geometrical parameter is unknown (and choose an starting value) or oreviously calibrated, and if he wants the oarameter to be calculated or be kept fixed
5	Simulgeo++ shall detect any error in the format of the system configuration file and send a clear message (including line number)
6	Simulgeo++ shall calculate the propagation of calibration and measurement errors, make fit of positions and angles and calculate regression errors
7	Simulgeo++ shall be able to identify any measurement that seems to be wrong with respect to the expected value from redundant measurements

[1] Camera consisting of lens plus CCD
[2] Device measuring distance by detecting the light after hitting a reflective surface

8	Simulgeo++ shall be extremely optimised for speed
9	Simulgeo++ shall permit to choose which parameters to fit (only unknown or all)
10	Simulgeo++ shall convert from text to Objectivity/DB file
11	Simulgeo++ shall convert from Objectivity/DB to text file
12	Simulgeo++ shall analyse the system and decide if calculation is suitable
13	Simulgeo++ shall permit to set correlations between parameters
14	Simulgeo++ shall provide an extensive user and technical documentation
15	GUI shall permit to choose which calculations to do: error propagation or parameter fit
16	GUI shall permit to edit, change and print the system description file and the measurements file
17	GUI shall permit to read another system description file (and implicitly a measurements file)
18	GUI shall provide the standard editing options (cut, paste, ...)
19	GUI shall permit to show the system description file with line numbering to check for errors
20	GUI shall show graphically the internal model and navigate through it
21	GUI shall permit to show and save a report with the results, choosing different levels of detail
22	GUI shall show the properties and the images of the fit sparse matrices
23	GUI shall permit to save the fit matrices
24	GUI shall permit to plan a systematic study for one or two parameters

A first set of static and dynamic diagrams were written with the help of the Rational Rose© CASE (Computer Aided Software Engineering) tool in the UML [5] language. These diagrams were then traslated in a first version of the C++ code. Since then many iterations have lead to a refined set of diagrams as well as to a more advanced code that in its final version will satisfy each of the Use Cases. The current version of the class diagrams for the two main packages can be seen in Figures 1 and 2. The whole file including all the diagrams can be found at [6].

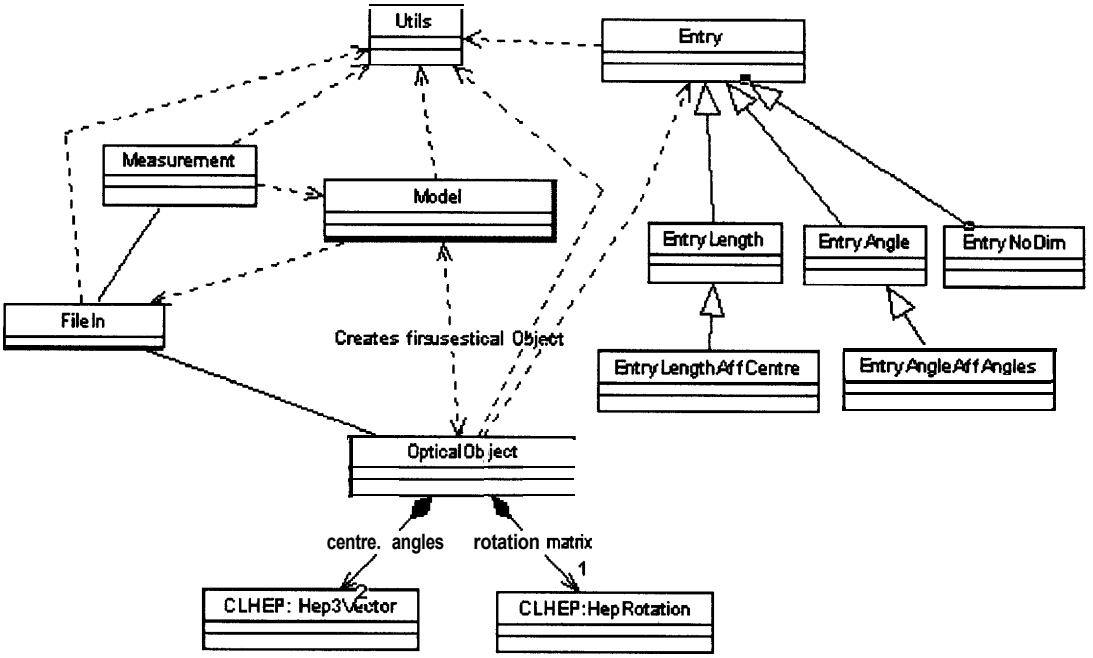

Figure 1 – *Simulgeo*++ class diagram for Mode1 category

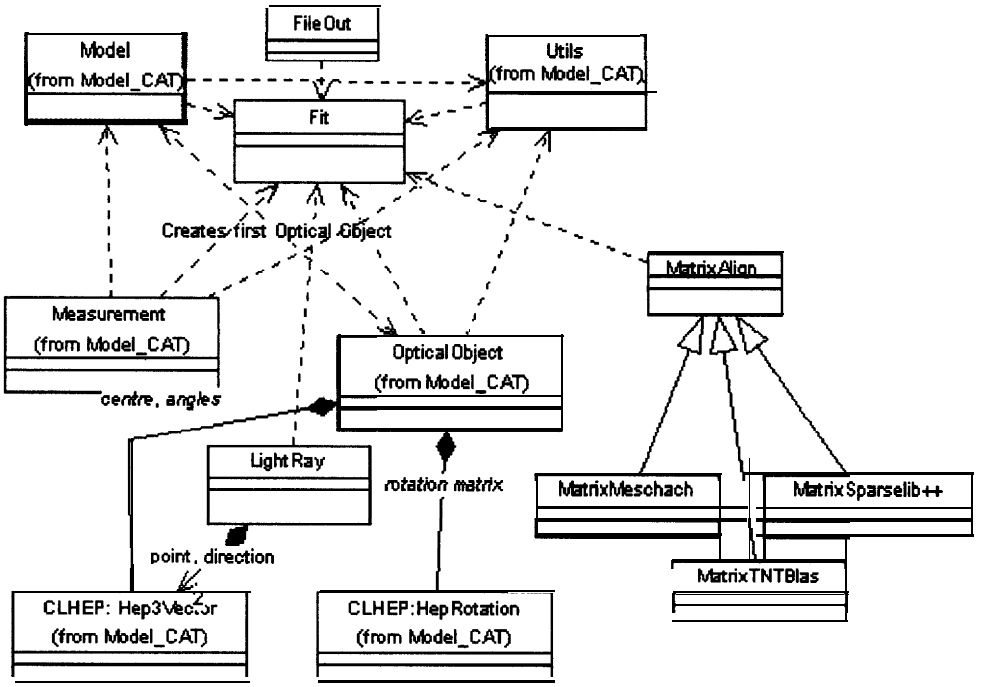

Figure 2 – *Simulgeo*++ class diagram for Fit category

To assure the quality of the software, a set of test input files and the corresponding output files from simple to complicated systems is provided and the results are tested every new version of the software. An automatic procedure for these testings is under study.

3. Structure of the input and output

To calculate the positions and angles of the elements of a system (we will call these positions and angles the system entries), you have first to give simulgeo++ the description of the system. This has to be done through the System Description File, a text file with a special format. It can also be input from an Objectivity© Object Oriented file (see section below).

The system description file contains five sections. The first one serves to set some global options while the second one defines the values of some parameters that can be used many times when filling the optical object tree data or the measurements.

In the third section the user must define the structure of the optical object tree, that is, enumerate every optical object with its components in a recursive manner. Actually, what is enumerated in this section are the object types, while the object names will be defined in the next section. In this fourth section, the user gives to simulgeo++ the name and the position and angles of every optical object defined in the previous section. The fact that you have to write twice the system objects is a redundancy check to avoid user errors in writing the desired system. Each entry has to be assigned a 'quality' that can be: *fix*, *cal* or *unk*. It is clear that if the system has only a few measurements you will only be able to fit but a few system entries. Therefore, when you provide the entries to simulgeo++ you have to tell it which ones you don't know at all (*'unk'*) and are going to be calculated and which are known already. Among these known entries you can further distinguish two kinds: those for which you have some knowledge, with the corresponding error, obtained by construction or calibration (*'cal'*) and those, generally few, that are fixed (*'fix'*). Usuallly you set as *'fix'* entries from which no measurement depend upon and entries that are taken as reference. Apart from the centre co-ordinates and angles of each optical object, some object types have a certain number of extra entries (focal length, shift, wedge, . ..) that affect the propagation of light in the system. Some of these extra entries are mandatory, and the software will tell you if you don't write them and some are optional.

The last section, is the measurements section, which can also be read from another file. Until now the measurement types implemented are 2DDETECTOR, CAMERA-DETECTOR, DISTANCEMETER and TILTMETER. For each measurement, you have to tell the software which are the objects that take part on the measurement ; for example for a 2DDETECTOR it means all the objects that the light ray touches, from its creation to its hitting the 2D-detector. The name of the objects may end with ':' plus a key letter to distinguish how the object should behave (for example if when hitting a splitter the light that drives to this measurement has traversed it or has been reflected).

The output may be written to text files or to Objectivity© files. The global option 'debug_verbosity' have five possible values to control how much output you want in each method called, from nothing to a huge amount of output. The global option 'output_verbosity' controls the amount of output that fills the output files. There are two kind of output files : the

first one contains the calculated positions, angles and extra entries of the system objects; the verbosity flag allows you to print also the original values and the correlation between each pair of parameters. The other file contains the matrices needed to solve the chi square problem. The first file can also be written to an Objectivity file.

A detailed explanation of each of the sections of the System Description File can be found at [7].

4. Graphical User Interface

The Graphical User Interface is an important part of every software as it is the one that makes the difference for the user. The simulgeo++ GUI has several components, all managed through a unique pane1 with several labels. The first subpanel helps the user to fill each of the sections of the System Description File (see Figure 3). Each button in it opens a new dialog and the 'SYSTEM TREE' button opens the pane1 shown in Figure 4, that allows you to fill the systerm tree of objects operating on a tree view.

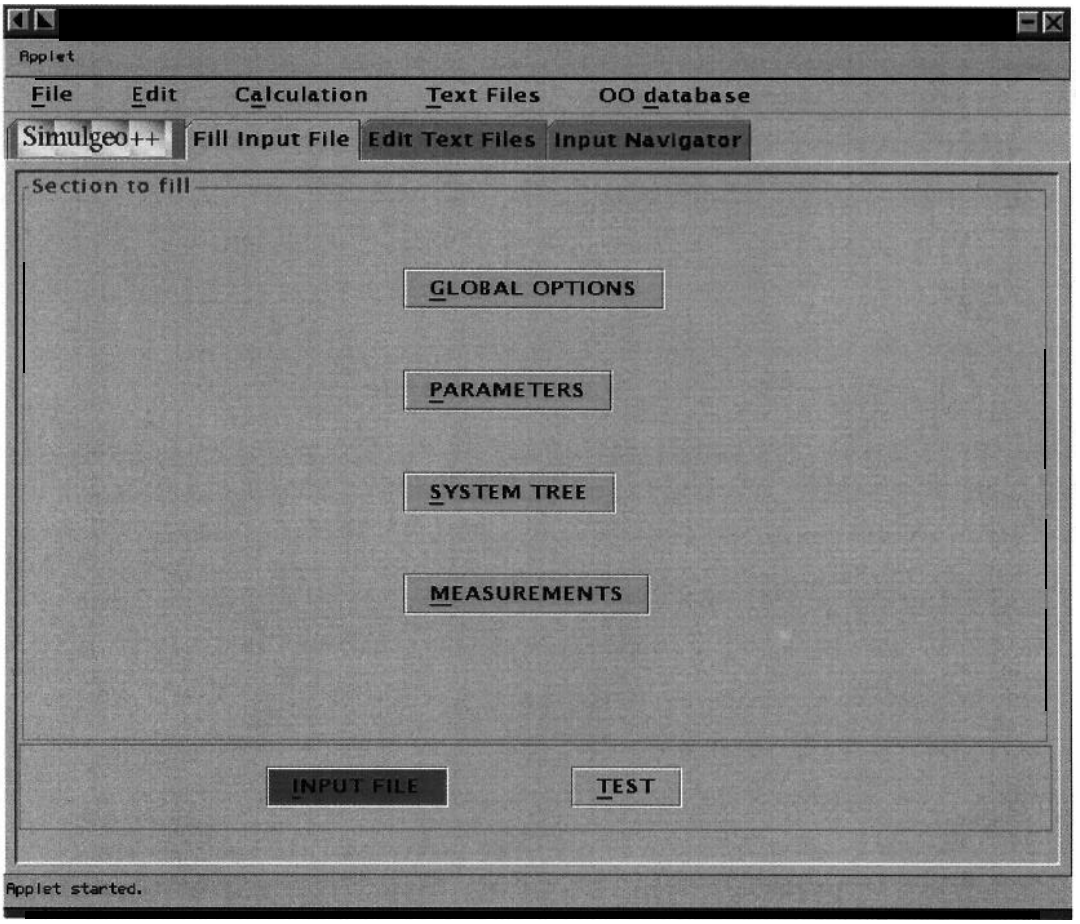

Figure 3 – GUI dialog for filling the System Tree of Optical Objects

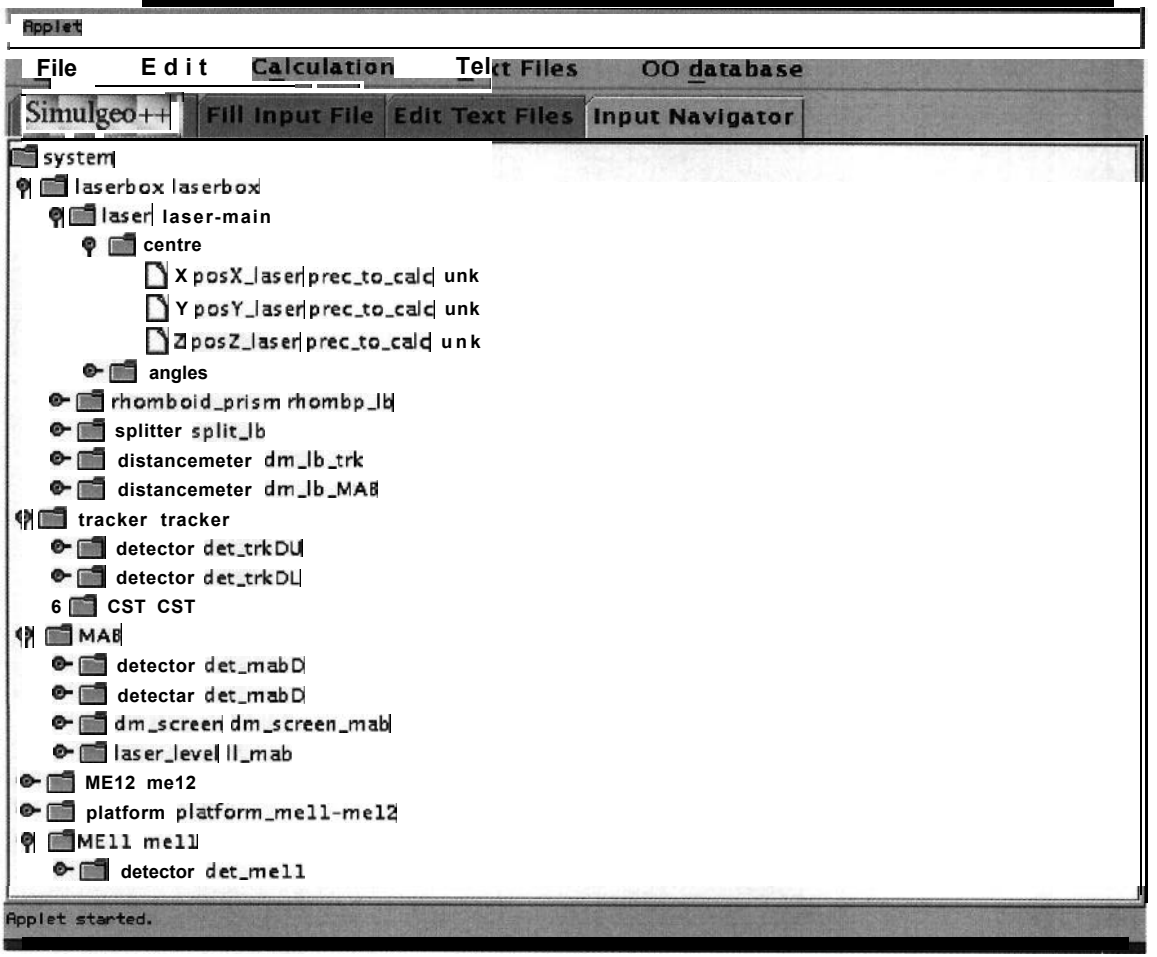

Figure 4 – GUI panels to navigate and edit the system tree of optical objects

The third label in the GUI allows the user to see and edit the text files: the input file, the output file and the matrices file (see Figure 5). The Objectivity input and output text files can be seen through an option in the File Menu.

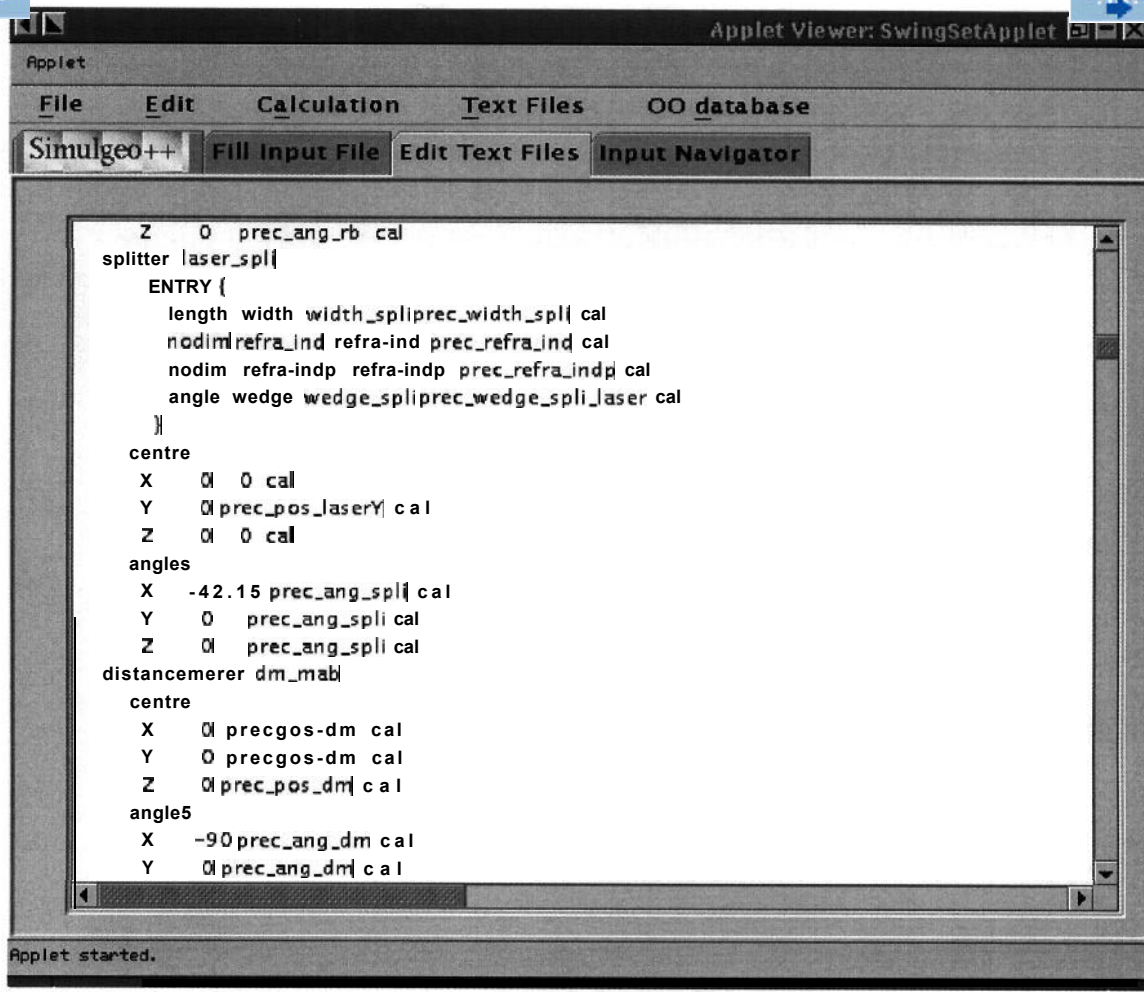

Figure 5 – GUI panels to edit the text files

5. Object Oriented database

As mentioned above, the need to interact at one side with the on-line software and at the other with the reconstruction software, forces the alignment software to use as data format an Objectivity© Object Oriented database. As of today the format of the on-line database is not defined and that of the reconstructed database is just starting to take shape. Therefore the actual database format of *Simulgeo++* is but an exercise. We have simply traslated the structure of the System Description File to an Objectivity schema.

6. Sparse matrix libraries

The big amount of optical objects and measurements that the software has to deal with, and the low cor-relation among them, results in a very big sparse matirx in the solution of the chi square fit.. As it will shown in the next chapter, the big number of parameters of the system

demands an optimization of the matrix calculations, and therefore this subject must receive a dedicated effort. As the main part of the simulgeo++ software is written in C++, our first approach is to use one of the sparse matrices libraries written in C++ or C library. Most of the libraries are though written in FORTRAN and therefore we are also including these libraries in the search for the one that is fastest for our system.

The comparison of different libraries is under study. The chosen libraries to start this comparison are meschach [8] (in C), Sparselib++[9] (in C++) and FORTRAN BLAS[10] with C++ templates through TNT [11].

7. Application to the CMS link alignment system

We have applied the above described software to the full CMS link alignment system (see [12]), the system that is in charge of positioning the barre1 and end cap muon chamber with respect to the inner tracker reference system. In Figure 6 we show an sketch of this system showing one of the six planes.

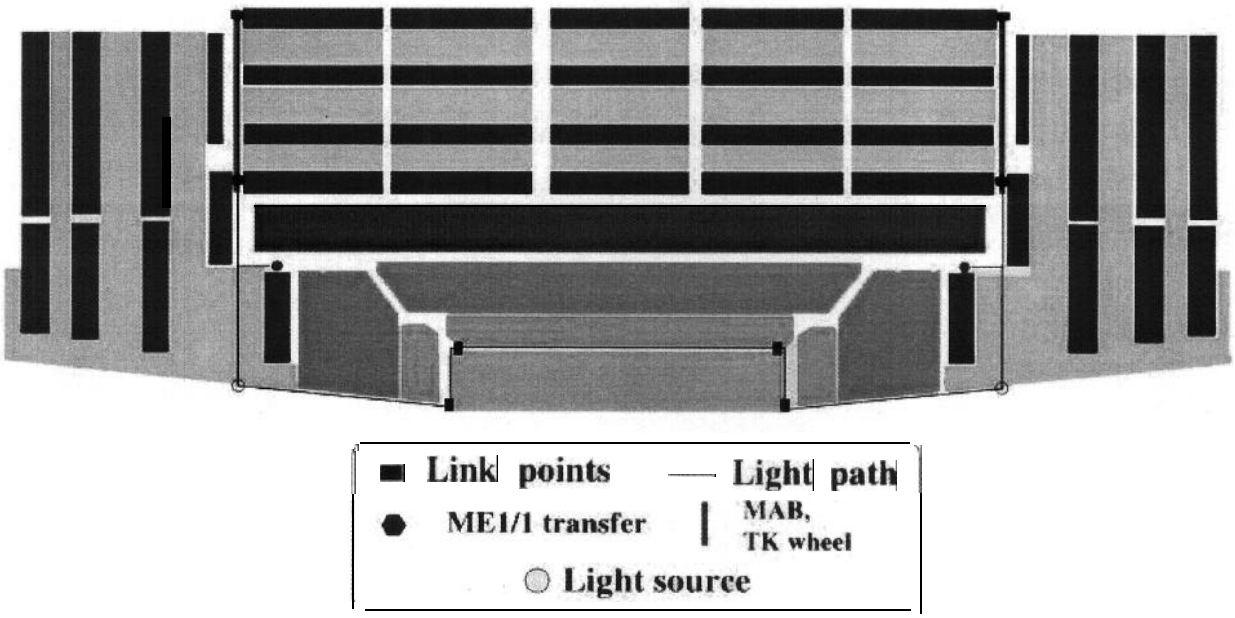

Figure 6 – Sketch of the a plane of the CMS link alignment system

We have first checked that the results of the simulation agree with previous calculations for half a plane (see [2]). The agreement found is quite good, apart from the fact that the implemented simulation is more detailed than the one used in [2]. We have then simulated one, two, three and six full planes to study possible correlations and to get an estimate of the time and memory needed for a full simulation of the whole link system. The machine used was a SUN E450 ULTRASparc- at 300 MHz. Figure 7a shows the time spent on this machine for the five simulations. The time is splitted in the three: the time to read the input file, the time to fill the matrix of derivatives and the time to solve the system equations (matrix manipulations).

Only the time for one iteration is plotted in this figure, although, as for any non-linear chi square problem, several iterations may be needed to get a final solution. Figure 7b shows the memory consumption for the five input files.

Figure 7a shows that the main time bottleneck lies on the operations done with the matrices to solve the equation system. The library used for this exercise has been the Meschach C library. As mentioned in the previous chapter, the choice of the final sparse matrix library is under study with the aim of optimizing its speed.

The time to calculate the numerical derivatives is quite smaller than the time for matrices operations and has been already optimised by studying which is the smaller number for the dependent variable to start the derivative and which is the smaller acceptable precision in the derivative. If, after optimizing the time for matrices operations, it becomes a major contributor to the total time, a farther sutdy will have to be done, mainly focused to the propagation of the light rays thorugh the objects.

Conceming the memory consumption, it is clear that the tested library is not a good candidate, as the total CMS alignment system will need a hundred times bigger memory than the already excesive 1.6 Gb needed for the CMS link alignment system. As it can be seen from Figure 8, the matrix for the full CMS link alignment system is very sparse. Therefore, the use of a library with provides a compressed data format could reduce the size of the memory needed for the full system by a very big factor.

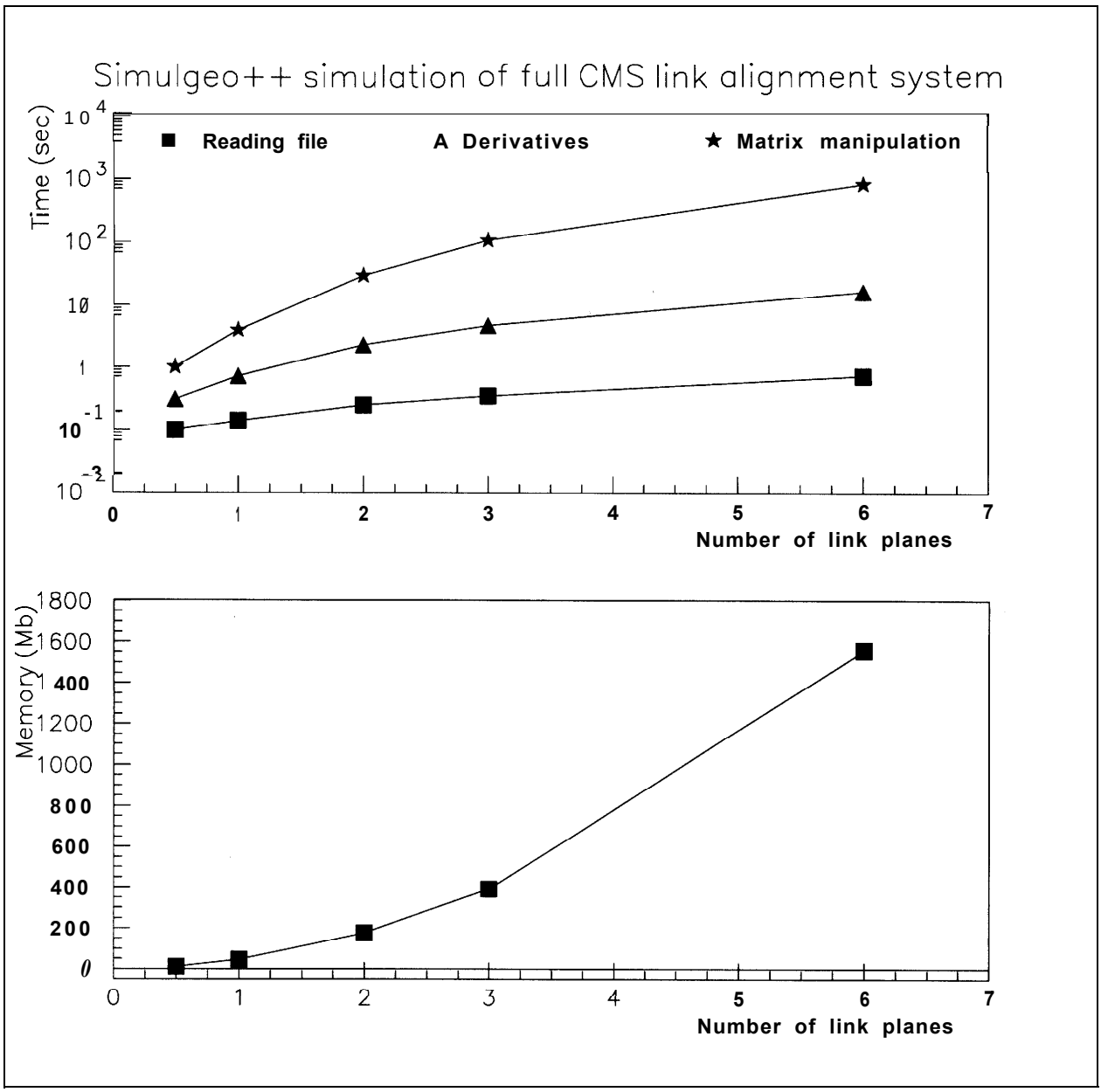

Figure 7 – Time and memory consumption simulation of the CMS link alignment system

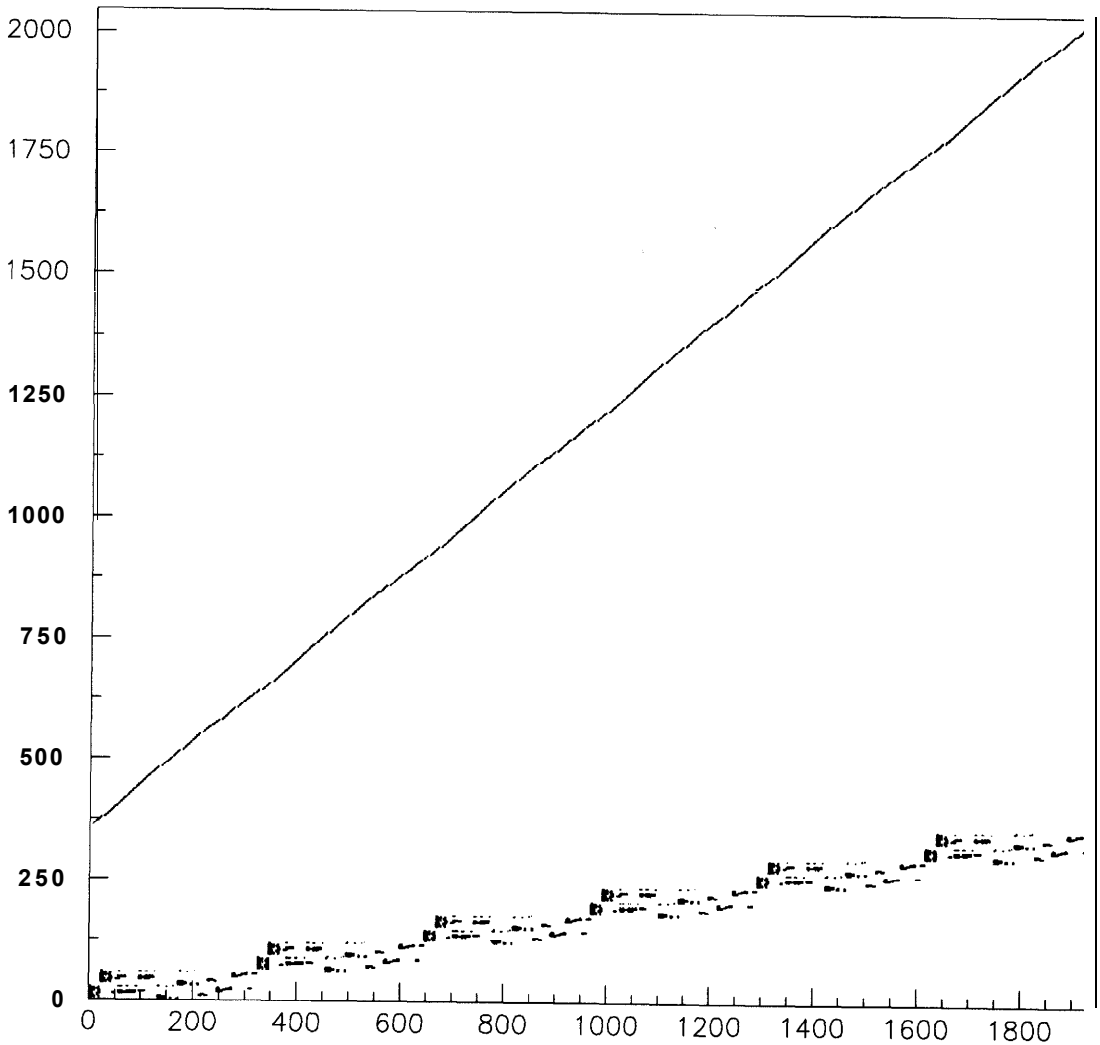

Figure 8 - Image of the matrix of derivatives for the full CMS link alignment system

8. References

[1] **The CMS Collaboration, "The Compact Muon Solenoid Technical Proposal.** CERN/LHCC 94-38, LHCC/P 1, December 1994

[2] **The CMS Collaboration, "The Muon Project" Technical Design Report.** CERN/LHCC 97-32, CMS TDR 3, December 1997

[3] **http://www.cern.ch/asd/cernlib/rd45/index.html**

[4] **L.Brunel. SIMULGEO Simulation and Reconstruction Software for Opto-Geometrical Systems.** CMS NOTE-1998/079. December 1998

[5] **http://www.rational.com/uml**

[6] **http://cmsdoc.cern.ch/~arce/simulgeo++/Version1.0/user_doc.html**.

[7] **http://cmsdoc.cern.ch/~arce/simulgeo++/Version1.0/tech_doc/simulgeo++.mdl**.

[8] **http://www.meschach.com**

[9] **http://www.nist.gov/sparselib++**

[10] **http://www.nist.gov/blas**

[11] **http://www.nist.gov/tnt**

[12] I.Vila, , **"The CMS link system".** IWAA 99 Conference, Grenoble, October 1999 .