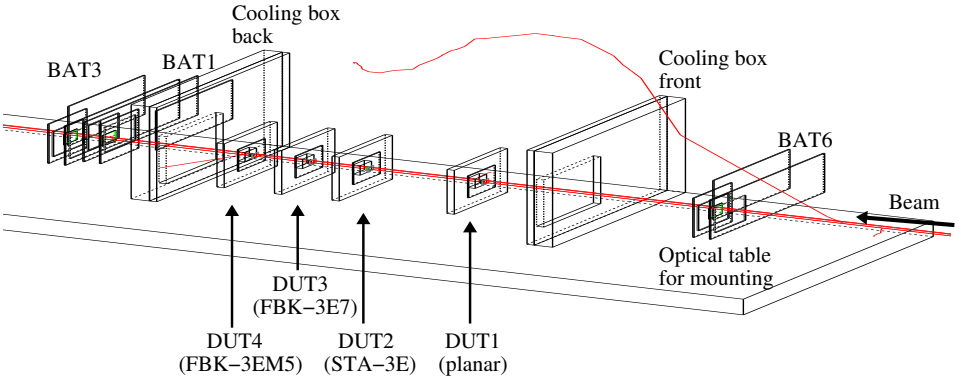
Full simulation of a testbeam experiment including modeling of the Bonn Atlas Telescope and Atlas 3D pixel silicon sensors

Kyrre Ness Sjøbæk
k.n.sjobak@fys.uio.no
Department of Physics, University of Oslo



Thesis presented for the Master of Science degree
in Experimental Particle Physics

September 2010

Abstract

3D silicon pixel sensors are a strong candidate for the sensor component of a new B-layer in the ATLAS detector, and for the ATLAS sLHC tracker, as these sensors can be highly radiation hard, fast, and sensitive to the edge. In order to characterize the sensors before large-scale application, samples are mounted in small fixed-target *testbeam* experiments. Here the samples are exposed to high-energy charged hadrons, and the response to this radiation is measured. The hit position in the sensor is estimated using a beam telescope, which measures the position of the particle while in flight up- and downstream of the sample. The hit position is then estimated by assuming that particle flies in a straight line between the telescope measurements and the sample.

This thesis presents a full Geant4 simulation of the interaction between the beam particles and the material in the testbeam, including but not limited to sensors. The output from the simulation is then used for detailed modeling of the signal formation and electronics response for both the 3D pixel sensor samples and the beam telescope sensor. Predictions from these models are compared to experimental data, indicating that the telescope models developed have a very high accuracy.

This work has made it possible to estimate the telescope tracking resolution in the sensor samples to approximately $6\ [\mu\mathrm{m}]$. An off-line method that may reduce the telescope hit position measurement uncertainty by subtracting common mode noise is also described.

Having this simulation system enables experimenting with sensor sample response models while monitoring how they behave from the perspective of data analysis. The results so far, still inconclusive, indicates that polysilicon-filled electrodes in full 3D sensors retain a non-zero efficiency.

Acknowledgments

In the two years of work and courses leading up to this thesis, I have been fortunate enough to be a part of the ATLAS 3D pixel R&D collaboration. The people forming this collaboration have given me a great learning environment, not only teaching me a lot about detectors, silicon sensors, experiment operation, and data analysis, but also taught me the process of research itself. I especially want thank my supervisors Ole Røhne, Steinar Stapnes, and PhD student Håvard Gjersdal for bringing me on board, answering lots of questions, sending me numerous times to CERN and elsewhere, helping me write this thesis, and generally guiding me through my adventures in this field of research.

Being a student at the Oslo experimental particle physics group have been also been very important for me. Working, and at times living, together with you have been a great experience, both scientifically and socially.

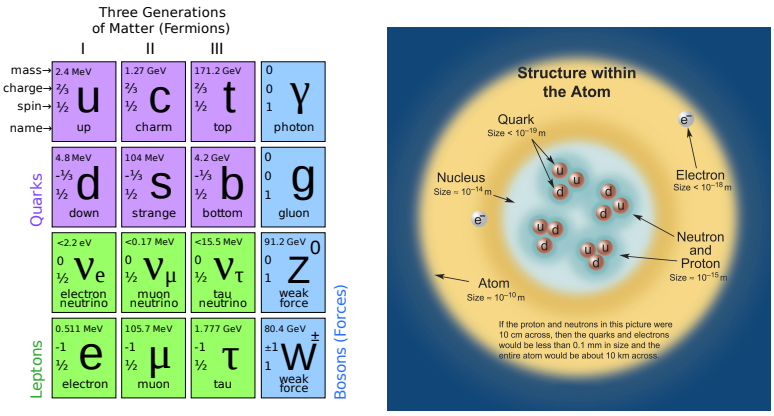
Last but not least, I want to thank my family, friends, and my girlfriend Helga Holmestad, who have supported and motivated me through this project.

Contents

Chapter 1

Introduction

Through the history of natural science, mankind has always sought the most fundamental way possible for describing the world around us. This has taken us from theories describing the world as composed of four elements with different properties, and into the modern understanding of a universe built up from a few elementary particles (figure 1.1(a)). These particles build up everything we know, including ourselves. By describing how they interact with each other, it is in principle possible to describe how they assemble larger and more complex systems such as atoms, chemical molecules, rocks and organisms, planets and stars, galaxies, the entire universe we see, and the light that makes it possible for us to see it.



(a) The particles of the standard model (Figure from Wikipedia)

(b) The structure of an atom (Figure from "The Particle Adventure")

Figure 1.1: The building blocks of matter

The way we figure out how things are made usually involves taking them apart, either carefully as if examining the delicate mechanism of a watch, or with force if examining the insides of a stone. This is also the general metaphor used when examining the particles building up the world; we build giant machines like the CERN accelerator complex and the LHC (figure 1.2) for smashing particles to pieces, and

incredibly complicated detectors like ATLAS (see figure 1.3) for examining the shards that get scattered from the collisions – all in hope to get a glimpse into the secret life of the smallest parts.

But why is it not possible to carefully pick the atoms apart, laying the parts out on a table, and examine them one at a time under a powerful microscope? It turns out that there are several reasons why this does not work. The first one is that it is not possible to make mechanical tools that are small enough – after all our tools are also made of atoms, which are much larger than many of the particles we want to study. Further, the light used in a normal microscope can be described as waves, which has a certain size or wavelength. And just like it is not possible to detect the presence of a narrow pole at sea by looking at long sea-waves that has passed it, it is not possible to detect a single atom by looking at waves of light which has passed it, as the waves used in ordinary light-microscopes are thousands of times longer than the size of a single atom, and will thus not be changed. With the particles composing the atoms, the differences in size are even larger. This could in principle be solved by using shorter wavelengths, but when doing this, it turns out that light itself is composed of particles known as *photons* – and light with shorter wavelength means photons with higher energy. This energy will not just "wiggle" the atoms like the longer wavelength light, but break them apart – just like when smashing atoms together. Another problem is that some particles, such as the neutrinos and gluons, do not even interact (directly) with photons – instead the photons just pass by as if these particles where not there at all. The third and last reason why this method does not work is that we do not only want to study the particles composing normal matter; that is the electrons and nuclei composing atoms, with the nuclei in turn composed of neutrons and protons, which in turn is composed of "up" and "down" quarks bound together with gluons (see figure 1.1(b)). It turns out that when colliding particles at high enough energy, more particles can be created. This way, heavy particles – such as the rest of the quarks and leptons shown in figure 1.1(a), or heavy "force particles" such as $\mathrm{W}^{\pm}$ or Z – can be created and studied. This is due to that, as described by Einstein in the Theory of Relativity, mass is a form of energy, stated in the iconic formula $E = mc^2$. It is thus possible to create new mass by supplying enough energy to the single particles colliding – and this is exactly what a particle accelerator such as the LHC does.

The particles accelerated up to energies of 3.5 [TeV] by the LHC are protons, which brings them up to 99.99 99 96 % of lightspeed. In the LHC there are two such beams circulating in opposite directions, which are steered into collision at four points along the machine. At these *interaction points* the protons collide at a total energy of $3.5 + 3.5 = 7.0$ [TeV][1]. This is also the position of the detectors,

[1] This is half its design energy; it is planned to go to $7 + 7 = 14$ [TeV] after the 2012 winter shutdown.

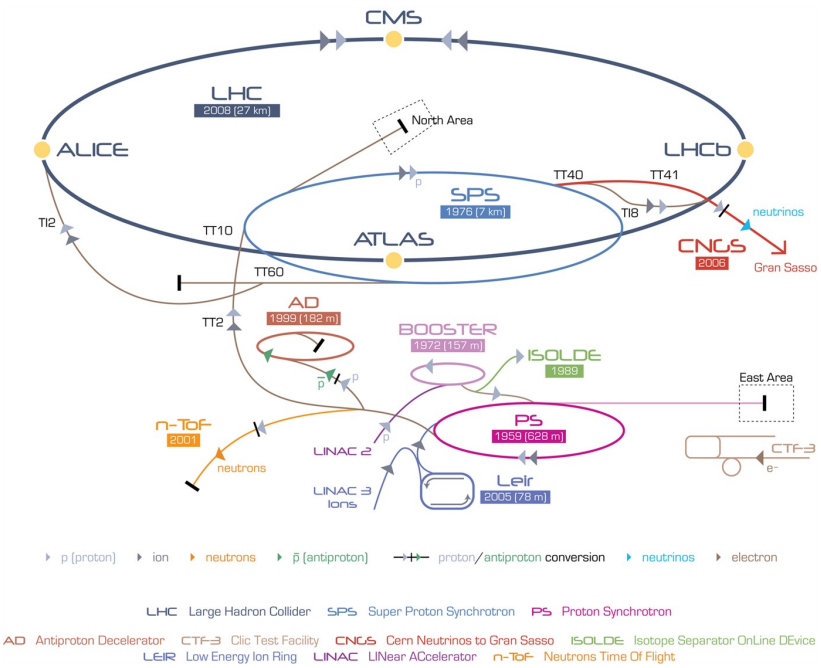


Figure 1.2: CERN particle accelerator complex, including transfer lines connecting the different accelerators, and important experiments. (Figure by Christiane Lefèvre)

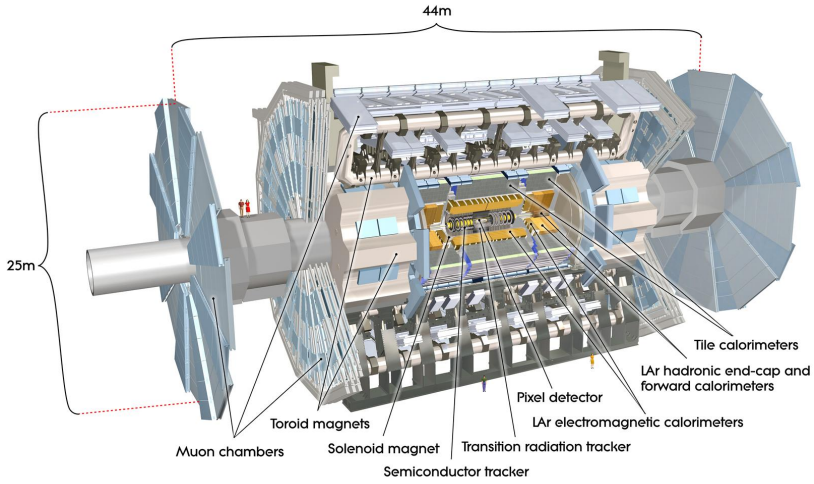


Figure 1.3: The ATLAS particle detector, subdetectors named and placed. This detector is sitting around one of the interaction points of the LHC accelerator at CERN, where protons are collided head-on. The detector measures the remains from these collisions. (Figure by Joao Pequenao)

which captures data from "interesting" collisions[2] – collisions that might give an insight to unknown physics.

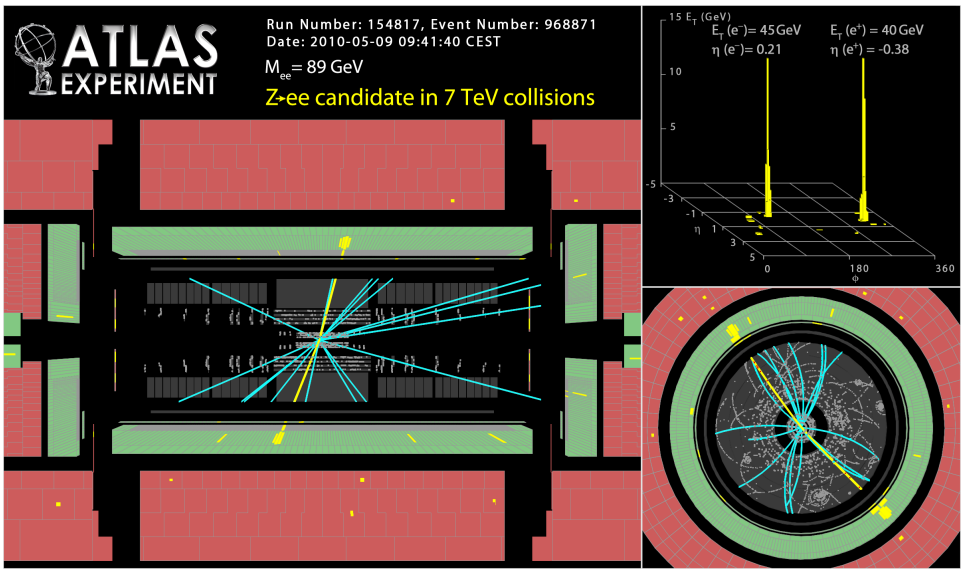
On a sidenote, one should know that even if the combined energy of the protons is 7 $[\mathrm{TeV}]$, this is not the energy available for production of new particles, as the interacting particles are usually not protons. As described above, protons are composed of quarks and gluons (often reffered to as *partons*), and it is they who interact, forming new particles. A proton is (to first approximation) consisting of two up and one down quark, and these particles have to share the total momentum and energy of the proton. To make matters even more complicated, the content of the proton resolves into even more particles – quark- anti-quark pairs and gluons – as the relative momentum of the probing particle (a parton contained within another proton) increases. This results in that the momentum of each single quark or gluon no longer have a certain value. Instead we get a probability distribution to find a certain parton at a certain energy, so called "parton distribution functions". This does not just result in a lowering of the effective energy available in the collision, but also in that the energy is different from one collision to the next. This makes proton machines such as the LHC good tools for discovering new particles with unknown properties, but as the collision energy and the identity of the colliding particles has to be identified after the fact, not so good for precision measurements of already discovered particles.

1.1 Particle physics detectors

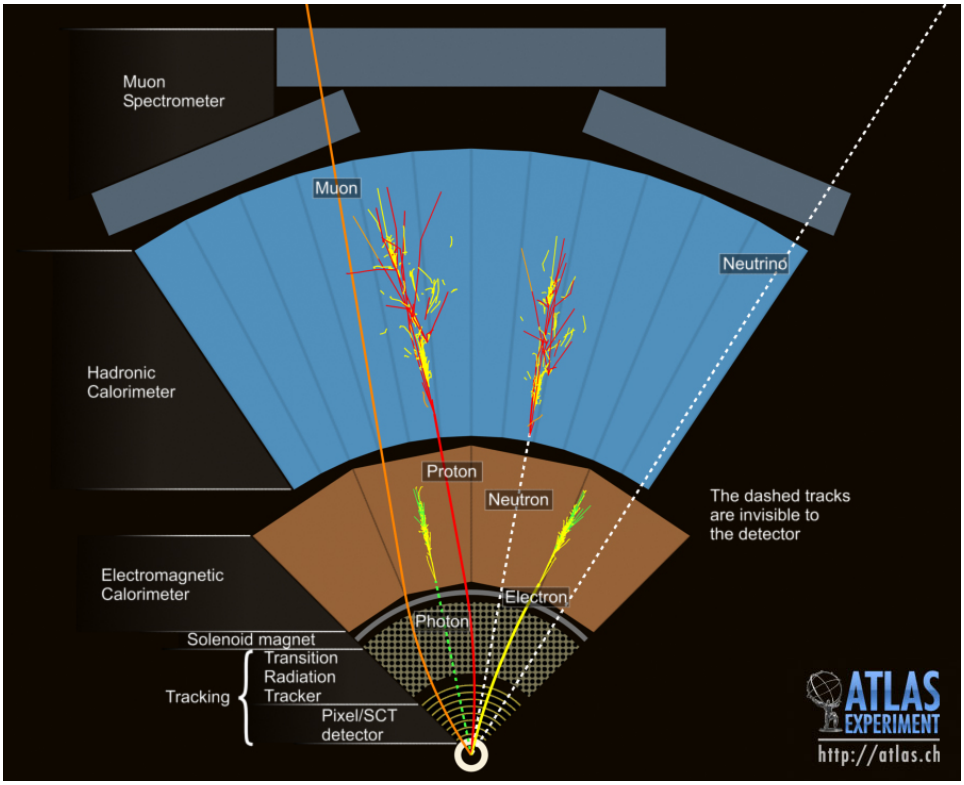
When an "interesting" collision occurs, this usually means that some heavy and therefore short-lived particle or resonance is created. This object quickly decays into two or more new particles, and these continues into the detector as shown in figure 1.4. These new particles carry information from the original *underlying event* which produced them, and the purpose of the detector is to record as much information as possible about the secondary particles. The recorded information is later used in order to enable an offline reconstruction and analysis, which tries to piece together as much information as possible about the original underlying event.

The detector itself consists of many sub-detectors, as seen in figure 1.4. These sub-detectors have different purposes, and are therefore built using different technologies. There are two main types of detectors: *Tracking* detectors, which measures the position of particles where they traverse the sensitive parts of the detector, and *calorimeters*, which stop particles in order to measure their energy. As the detector is structured as an onion, the particles first have to go through the inner tracking detector. Here their trajectory is bent into a helix by a magnetic field parallel to the beam axis. By fitting a helix to the measured points, it is possible to

[2]The detectors really capture data every 25 $[\mathrm{ns}]$, but a system known as *trigger* acts as a filter which throws away most "uninteresting" collisions. This is done as it is impossible to download from the detector, store, reconstruct, and analyze the full stream of approximatly $1.5\ [\mathrm{MB/event}] \times 40 \cdot 10^6\ [\mathrm{events/sec}] \approx 60\ [\mathrm{TB/sec}]$.



(a) Candidate $Z \rightarrow e^+e^-$ event in ATLAS 7 [TeV] run



(b) Event Cross Section in a computer generated image of the ATLAS detector (Figure by Joao Pequenao)

Figure 1.4: Examples of events in ATLAS

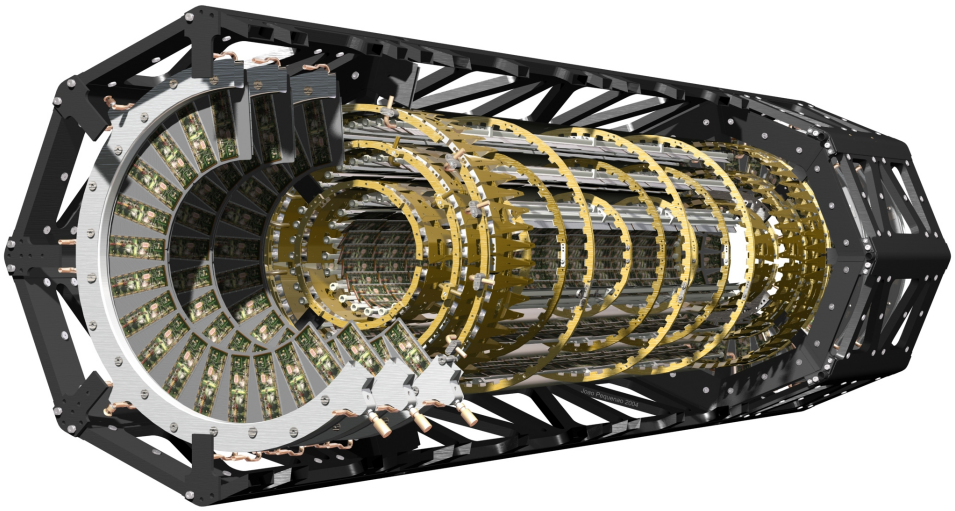


Figure 1.5: ATLAS pixel detector

extract the momentum component transverse to the beam axis:

$$|P_\perp| = qBr$$

Here q is the charge of the particle, B is the magnetic field, and r the radius of the fitted helix. Further, the total momentum can be calculated by using the angle to the beam axis (z-axis):

$$\tan(\theta) = \frac{|dz|}{dr} = \frac{|\vec{P}_\parallel|}{|\vec{P}_\perp|} \Rightarrow |\vec{P}| = |\vec{P}_\perp| \cdot \sqrt{\left(\frac{|dz|}{dr}\right)^2 + 1}$$

These measurements are one of the main goals of a tracking detector, such as the ATLAS pixel detector shown in figure 1.5. Another important goal is *vertex reconstruction*, which means finding the point(s) where the tracks cross each other. This is likely to be point where the particles was created, either the collision point or the decay position of a short-lived particle such as a B-meson.

In the real reconstructed ATLAS event shown in figure 1.4(a), the bending of tracks can be clearly seen for some of the low-energy tracks. We see that there are many tracks in the tracking detector – some bent, and some very straight. The two tracks highlighted in yellow is probably high-energy electrons, as they are very straight and is stopped in the electromagnetic calorimeter. By using the calculated momentum from the tracker together with the energy measurement from the calorimeter, the mass of the decayed original particle in its rest-frame can be reconstructed. This is done by calculating the *invariant mass* of the two electrons,

using the following relativistic formula:

$$\begin{aligned} M^2c^2 =& (P_1^\mu + P_2^\mu)^2 = P_1^\mu P_{1\mu} + P_2^\mu P_{2\mu} + 2P_1^\mu P_{2\mu} \\ =& \frac{E_1^2}{c^2} - \vec{p_1}^2 + \frac{E_2^2}{c^2} - \vec{p_2}^2 + 2\left(\frac{E_1 E_2}{c^2} - 2\vec{p}_1 \cdot \vec{p}_2\right) \\ =& m_1^2c^2 + m_2^2c^2 + 2\left(\frac{E_1 E_2}{c^2} - 2\vec{p}_1 \cdot \vec{p}_2\right) \end{aligned}$$

From the text included in figure 1.4(a), the measured invariant mass of the two electrons is $M_{\text{ee}} \approx 89$ [GeV], which is close to the mass of the Z boson ($M_Z = 91.19$ [GeV]) [14]. As the Z boson is a very unstable particle, it does not have a specific mass, but rather a distribution of possible masses. Thus this measurement is compatible with being a Z created and decaying such as shown in the Feynman diagram of figure 1.6. Here two constituent quarks from the two colliding protons – one quark and one anti-quark[3] – combine with the right energy to create a Z. This Z then immediately decays, producing a pair of electrons, which is the measured final-state.

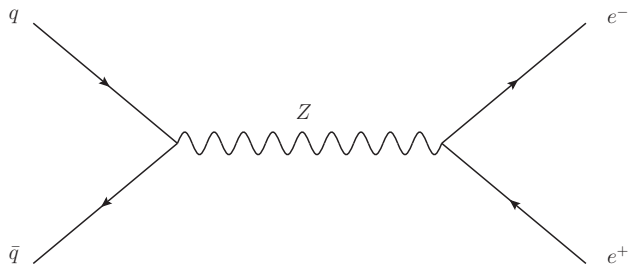


Figure 1.6: Feynman diagram of $q\bar{q} \to Z \to e^+e^-$

However, this is not the only possibility of producing such a signal in a detector; for example one could replace the Z with a photon with the same invariant mass. This then constitutes *background* to our measurement, which means that one measurement with invariant mass of ≈ 91 [GeV] is not enough to claim discovery of the Z. Instead one needs (at least) a histogram of the invariant masses across many events, which should show a significant peak around the Z mass.

[3] Even if the proton to first approximation only consists of normal quarks (two up and one down), and no anti-quarks, it resolves into a more complicated structure also containing pairs of quarks and anti-quarks when probing at higher energies, as discussed above.

1.1.1 Detector output and reconstruction

The output of a detector such as ATLAS is divided up in *events*, where each event is roughly[4] corresponding to one collision. One event is then further subdivided into many detector *digits*, which is the smallest piece of information coming from the detector. A digit contain one piece of information such as "channel XX of subdetector YY read a value of ZZ".

This information makes it possible to reconstruct the event, which means piecing the digits together to make tracks and calorimeter energy measurements. To do the reconstruction, a one needs general knowledge of the detector (position of channels, detector response to different stimuli, how to decode digits, etc.), the behavior of the particles (equations of motion, scattering in material, etc.), and the event data (the digits). This is necessary before one can do physics analysis, as described above.

1.1.2 Insertable B-layer

The *B-layer* of the ATLAS pixel detector (figure 1.5) is the third and innermost layer of pixel sensors. This layer is crucial for ATLAS physics performance, enabling accurate vertex reconstruction and jet quark flavor identification [43, 6].

The *Insertable B-layer* (IBL) is a proposed fourth layer of pixel sensors, which is to be placed inside the current B-layer. This will increase the overall robustness and performance of track reconstruction, helping dealing with both high occupancy in the current B-layer due to high luminocities expected in the future by providing an additional measurement, and also mitigating damage to the current B-layer from radiation, beam losses etc. Additionally, as it is placed very close to the interaction point, it will improve the tracking precision, boosting accuracy in determination of the vertex point. This will thus increase sensitivity to physics channels, especially those involving b-quark jets [6].

There are several tough constraints that have to be obeyed by the IBL design, the main ones described below. Firstly, the space available is very tight, with an outside envelope radius of only $43.5\ [\mathrm{mm}]$. This space needs to fit the beam pipe (with an outer envelope radius of $30\ [\mathrm{mm}]$), sensors, electronics, services, and support structures [6]. Secondly, the material budget should be kept as low as possible, in order to minimize interference with detector layers further from the interaction point. Third, the sensors are be positioned very close to the interaction point, increasing the radiation load and particle density. This means that radiation hard detectors able to cope with high particle multiplicities are needed. One candidate for the sensor component are 3D pixel sensors, described in Chapter 2.2.5.

[4] If the luminocity of the accelerator is large, there might be more than one collision each time two proton bunches passes each other. This phenomenon is known as "pile-up", and must be dealt with in offline reconstruction and analysis. Generally this happens if there is more than one underlying event inside the detectors sensitive time-window.

1.2 Testbeam and testbeam simulation

In order to test the response and performance of new detector components, we mount them in small fixed-target experiments known as testbeams. A picture of one such setup is shown in figure 1.7, and it is sketched in figure 1.8.

Such experiments has been performed in the SPS H6 and H8 beamlines, situated at the CERN Prevessin site ("North Area" in figure 1.2). Here we get a beam of reasonably monochromatic high-energy pions by illuminating a target material with a proton beam from the SPS accelerator, and sorting the secondary particles produced. The sensors samples under test is placed in this beam. Since the pions have a high kinetic energy, they produce a rather low amount of ionization along their track, and can thus be taken as minimum ionizing particles (MIPs), as described in Chapter 2.1.

The H8 beamline is equipped with a superconducting magnet with a 1.6 meter bore [23], measured to a maximum field of 1.6 [T]. This makes it possible to test sensor response in presence of a significant magnetic field, such as the conditions they will operate in inside a HEP experiment. In order to provide trigger for our experiment, we used scintillators: Two scintillators in coincidence mode was placed in front of the devices under test in order to define the effective size of the beam, and one large scintillator with a hole through it was placed behind the setup in order to suppress showers and restrict the experiment's angular acceptance. The Bonn Atlas Telescope (BAT), described in Chapter 4, was used for tracking the pions. With this we are able to estimate where the pions hit our device under test with an accuracy of approximately 6 [μm] (see Chapter 5).

Using this setup we measured how the devices respond to MIPs as a function of hit position, incident angle, and magnetic field strength, and thus measure hit efficiencies, charge-sharing properties etc., as a function of these parameters. Some results for this is discussed in Chapter 2.2.5 and Chapter 6.

With these quantities measured, we can make models predicting the sensor response, and compare their output to the experimental data. One way of doing this to making a full simulation of the testbeam, and comparing the simulated and real data. If our models are good, they should be able to reproduce the testbeam results. Such as simulation also enables testing hypothesis about sensor response against the real data, excluding or strengthening them.

Having a good model for how sensors respond is necessary for making good simulations of larger experiments such as ATLAS. This is necessary to predict their sensitivity for different physics scenarios. The rest of this thesis will detail the development and testing of a testbeam simulation system, including such device models.

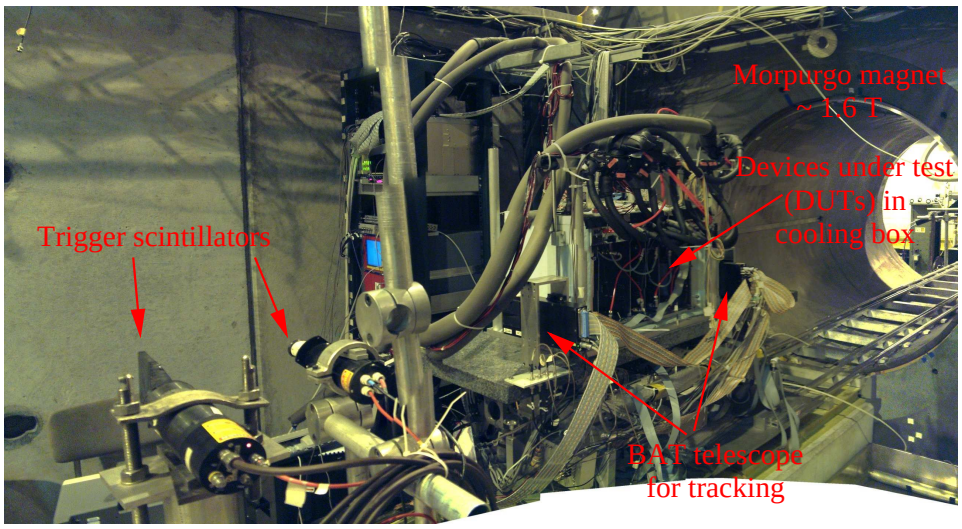


Figure 1.7: Setup for SPS H8 testbeam, May 2009

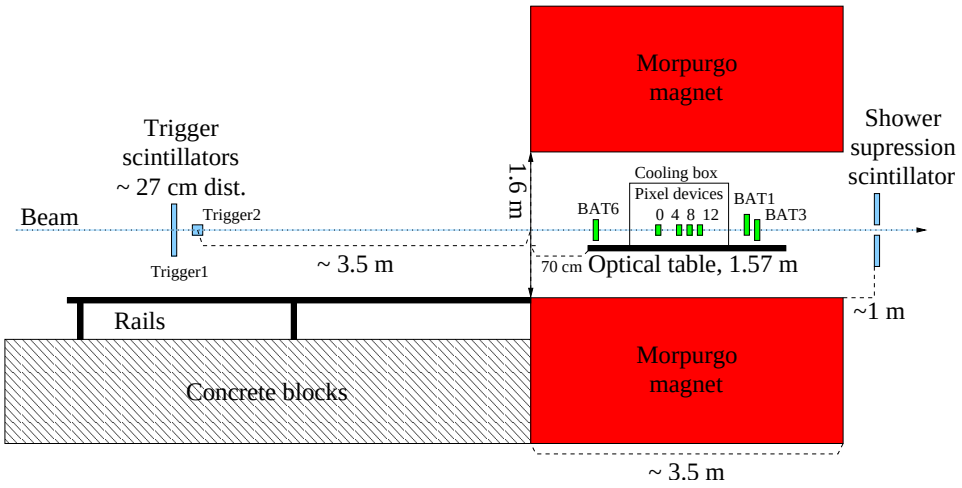


Figure 1.8: Sketch of setup for SPS H8 testbeam (not to any scale)

1.3 Organization of material in thesis

The main part of this thesis describes the methods and results from a full simulation of the May 2009 H8 testbeam. This is organized in several chapters as follows: Chapter 3 discusses the experimental setup, and how this is described in the simulation. Chapter 4 and 6 describes simulation of the sensor response to stimuli presented by energy deposits from the simulated particles. Chapter 5 describes characterization of the hit resolution in the plane of the sensor samples under test, as reconstructed with the Bonn Atlas Telescope (BAT, see Chapter 1.2 or 4), using simulated data. Additionally, Chapter 2 discusses some background material necessary for understanding semiconductor radiation sensors and Monte-Carlo simulations in general.

Technical aspects of the software written is documented in Appendix A, B, C and D. In addition to this, a short dictionary of many terms and abbreviations is given in Appendix E. When a terms are used for the first time, this is usually marked with *italics* text. Technical names referring to computer code, files, and code examples are written in `typewriter typeface`.

Some of the material presented in this thesis also form the foundation of several presentations given at ATLAS 3D Pixel R&D collaboration meetings, as well as conference talks at Spåtind 2010 [32] and the 2009 annual meeting of the Norwegian Physical Society [30]. Further, work not included in this thesis, such as shifting, rigging, and data analysis at several test beam experiments using BAT and EUDET telescopes has been discussed in several publications and conference talks [13, 16, 18, 5, 28].

Chapter 2

Semiconductor radiation sensors

This chapter introduces the basic concepts and theory necessary to understand how semiconductor radiation sensors work, and how they are used. 3D pixel sensors are specially introduced, as this is a major motivation and topic for this thesis. Additionally, the Monte Carlo method is briefly introduced at the end of the chapter.

2.1 Interactions between radiation and matter

When a charged particle passes through a material, it will interact with the electrons and nuclei in the matter. This interaction mostly happens via the electromagnetic force, but for hadronic projectiles "hard" interactions through the strong force also sometimes happen. This results in that the impinging particles will loose energy and get deflected.

The main mechanism for energy loss of charged particles passing through a material is ionization and excitation of electrons in the material. As this is quantum-mechanical in nature, the amount of energy lost by a traversing particle is a stochastic quantity. For relatively thin slabs of material, such as a silicon tracking detector, the probability for a particle to loose an amount of energy Δ can be described by the Landau distribution, which according to [21] is given by equation (2.1), containing two parameters λ and ξ.

$$P(\Delta) = \phi(\lambda)/\xi \quad \text{where} \quad \phi(\lambda) = \frac{1}{\pi}\int_0^\infty \exp(-u\log u - u\lambda)\sin(\pi u)\,\mathrm{d}u \quad (2.1)$$

The parameter ξ is the approximate mean[1] energy loss, taken from the Bethe-Bloch formula. It is defined by equation (2.2), where x is the thickness of material, ρ is the material density, Z and A is the atomic -number and -weight of the absorbing material, z is the charge of the projectile particle, N_a is Avogadro's number, $r_e = 2.817 \cdot 10^{-13}[\mathrm{cm}]$ the classical electron radius, and m_e the electron mass.

$$\xi = 2\pi N_a r_e^2 m_e c^2 \rho \frac{Z}{A}\left(\frac{z}{\beta}\right)^2 x \quad (2.2)$$

[1] ξ is referred to as σ by the software package `ROOT`.

Further, the parameter λ, which occurs in the integral of equation (2.1), is given by equation (2.3). Here $\gamma = 0.577$ is Eulers constant, and the parameter ε is defined by equation (2.4). This uses the projectile speed β, the projectile mass m, and the "mean excitation potential" I, which is depending on the quantum-mechanical structure of the target atoms, and is in practice tabulated from experimental data. The constant Δ_0 is a location parameter for the Landau distribution[2].

$$\lambda = \frac{1}{\xi}\left[\Delta - \xi(\log\xi - \log\varepsilon + 1 - \gamma)\right] \equiv \frac{\Delta - \Delta_0}{\xi} \tag{2.3}$$

$$\log\varepsilon = \log\frac{(1-\beta^2)I^2}{2mc^2\beta^2} + \beta^2 \tag{2.4}$$

The Landau distribution, defined by equation (2.1) with parameters λ and ξ from equations (2.3) and (2.2), is plotted in figure 2.1. This figure also shows data from a full simulation, including hadronic interactions etc., using the simulation system discussed in Chapter 3. Here we see that while the Landau distribution is in shape very similar to both the distribution of lost kinetic energy ΔE_k[3], and also the distribution of deposited charge converted to units of energy[4], the Landau distribution is misplaced towards higher energy. One possible reason for this is that equation (2.3), which describes the position parameter Δ_0, is somewhat simplified; for example, the density correction is not included. The density correction takes into account that the material becomes polarized by the electric field of the traversing particle, shielding electrons at larger distances from the full electric field intensity [21]. Including this decreases the amount of energy lost by high-energy particles quite a bit (see figure 2.2), especially in dense materials. Still, the similarity in shape makes the Landau distribution a good choice when fitting both the energy loss and charge deposit, as seen in the bottom panels of figure 2.1.

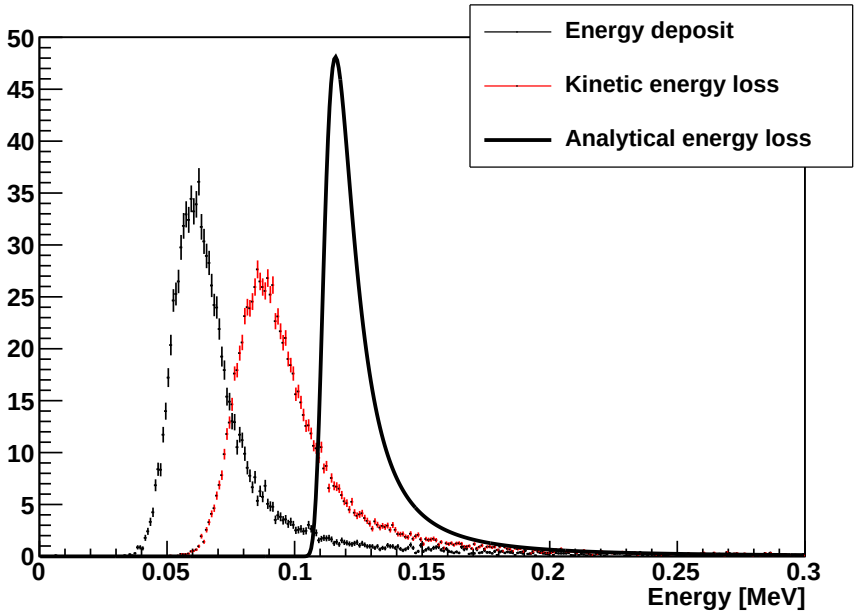
Figure 2.1 also shows that there is a considerable difference between the ionizing energy deposit and the amount of kinetic energy lost by the particle. This is to be expected, as some of the energy lost by the particle leaves the detector block in the form of high-energy electrons ("δ-rays") or photons, especially in the case of a thin detector with little material just in front of it in which the particle can produce secondary radiation (δ-rays etc.) that later hits the detector.

The shape of the Landau distribution can be understood by considering the physical mechanisms at work when a high-energy charged particle traverses a block of material. This particle will interact many times with the electrons in the material, transferring different amounts of energy to them. Usually there will be

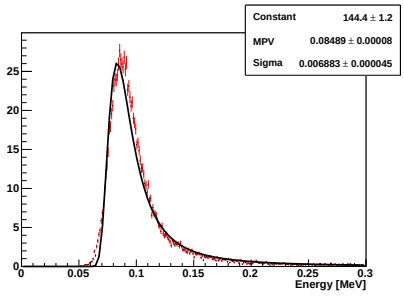
[2] Δ_0 is referred to as *MPV* (Most Probable Value) by the software package `ROOT`, used for fitting Landau distributions to histograms through this thesis. Note that the true most probable value of the Landau distribution is given as $\Delta_{\text{MP}} = \Delta_0 - 0.225\,\xi$ [21].

[3] The Landau distribution is supposed to describe ΔE_k.

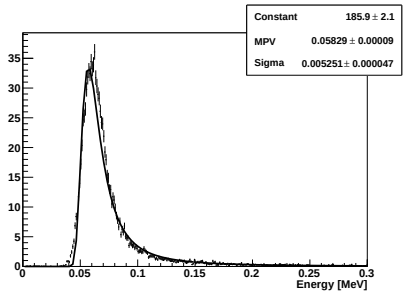
[4] The distribution describing the number of electron-hole pairs N_0 created is converted into units of energy by multiplying N_0 with the mean energy w needed to produce an electron hole pair. This equals the amount of energy used to produce the electron-hole pairs, up to a Gaussian smearing (with standard deviation a function of energy) from the Fano factor (see Chapter 2.2.3).



(a) Comparison of charge deposition, kinetic energy loss, and analytical Landau distribution. All graphs normalized to unit area.



(b) Kinetic energy lost by incident particle, fitted with a Landau



(c) Deposited charge (in units of energy), including Fano factor, fitted with a Landau

Figure 2.1: Simulated energy deposits from 180 [GeV] π^- in 210 [μm] thick slab of Si. Done with detector geometry as shown in figure 6.2.

a set of many relatively small energy transfers, but sometimes a more energetic interactions happens, resulting in a high-energy electron. This produces the tail of the Landau distribution of energy loss. These high-energy electrons (δ-rays) can move through several μm of material, and some examples of this can be seen in figure 4.4.

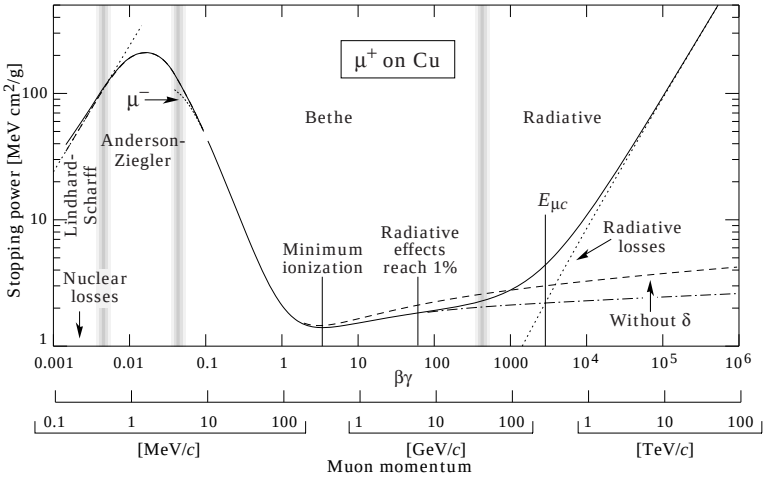


Figure 2.2: Stopping power $-\langle dE/dx\rangle$ for positive muons on copper as function of $\beta\gamma = p/m_\mu c$. Figure from PDG review of particle physics [14].

If one rather than looking at single particles looks at a large collection of particles, the most interesting number is the mean energy loss. This is well described for relativistic particles by the Bethe-Block formula, plotted in figure 2.2. For ultrarelativistic particles radiation losses (Bremsstrahlung) becomes significant, while for lower energies the mechanisms gets more complicated. Note that bremsstrahlung becomes important much earlier for electrons than for heavier particles such as muons or pions, as the bremsstrahlung cross-section is proportional to $1/m^2$ [21].

Looking at the $\left\langle \frac{-dE}{dx} \right\rangle$ graph of figure 2.2, one notices a minimum in energy loss for particles of medium-high energy. Particles with energy around this minima are known as *Minimum Ionizing Particles* (or just MIPs), and most interesting particles in a tracker that are not electrons, can be viewed as MIPs.

A convenient unit when quoting the thickness of material is the *radiation length*, which is defined as the distance over which the energy of an electron traversing the material is reduced by a factor $1/e$. This is convenient, as it will indicate how effective a given amount of the material in question is for stopping charged particle radiation.

2.2 Silicon radiation sensors

When a charged particle passes through a material, it will deposit energy in the material as described in Chapter 2.1. This will result in ionization of the material, which means that electrons in the material will gain energy and be exited to a higher energy state (see figure 2.3). In semiconductor and gas detectors, this enables

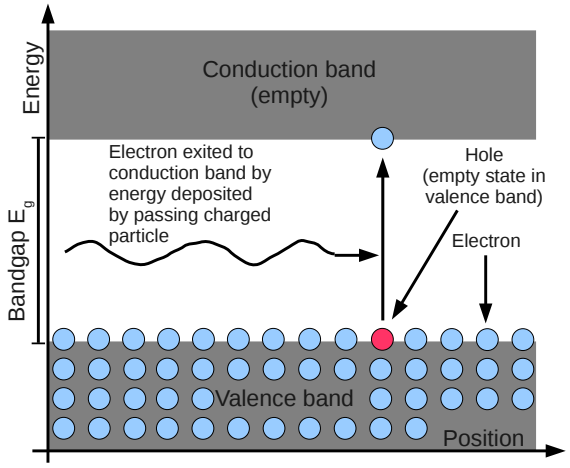


Figure 2.3: Excitation of an electron from the valence band into the conduction band, leaving behind an unoccupied state (a hole) in the valence band.

the liberated charges to move around in the material, making it go from a non-conducting state and into a conducting state. This is measured by the electronics connected to the detector.

The main difference between semiconductor detectors and other detector materials such as gas, is that semiconductors are (as their name indicates) somewhat conductive in their natural state, while most other detector materials are isolating. The reason for this is related to the size of the materials bandgap[5], shown in figure 2.3. For an *intrinsic* semiconductor (a semiconductor without impurities) at a temperature of $T = 0^\circ$ [K] in a radiation-free environment, all the electrons are sitting at the lowest free quantum-mechanical energy state. Further, the energy states of a semiconductor material is split into two[6] "bands", and the number of states in the lower-lying band (the valence band) is exactly equal to the number of electrons in the material. Thus an intrinsic semiconductor at $T = 0^\circ$ [K] have no electrons in

[5]Gases does not really have a bandgap, as there are no higher energy state available in the material in which the charges are mobile. Instead the gas atoms has to be completely ionized, which means that the electrons has to be into the vacuum (and a bit beyond to prevent immediate recapture).

[6]There may be more than one split, but the bands below the valence band are almost always completely full. Further, there is usually more than one conduction band, but these are usually overlapping in energy.

the conduction band, all states in the valence band filled, resulting in that there are no charges which is free to move, rendering the material nonconducting.

However if the temperature is increased, some of the electrons in the valence band gain enough energy to "jump" across the bandgap and into the conduction band. This results in a small equilibrium concentration[7] n_0 of electrons in the conduction band ($n_0 = 10^{-10}$ [electrons/cm^3] at $T = 300^\circ$ [K] for silicon), and an equally small concentration p_0 of empty states in the valence band, as described by equation (2.5) [35]. Here E_c is the energy of the conduction band edge, and N_c the effective density of states in the conduction band[8]. Note that N_c is also a function of temperature, $N_c \propto T^{3/2}$. Further E_F is the Fermi level of the semiconductor, which is the energy of a hypothetical quantum state that is 50% filled. This is approximately $E_F = E_c - E_g/2$ in an intrinsic semiconductor.

$$p_0 = n_0 = N_c e^{-(E_c - E_F)/kT} \tag{2.5}$$

Now electrons can move around in both bands, as there are empty states available for moving into. This produces two conduction mechanisms, the first one being just electrons (negative charges) moving around in the conduction band. The second mechanism is that an electron in the valence band can move into an unoccupied state, which effectively results in a movement of the unoccupied state. This can be treated as a "particle" with positive charge moving in the opposite direction of the electron, and is known as a *hole*.

The main difference between an isolator and a (semi)conductor is then the size of the bandgap E_g, isolating materials having large bandgaps, while conducting materials having small or no bandgaps. In a material with a large bandgap fewer electrons gets thermally exited across the bandgap, and this makes for a lower density of movable charges, which in turn results in a lower conductivity σ as indicated by equation (2.6) [35].

$$\sigma = q(n\mu_n + p\mu_p) \tag{2.6}$$

This equation is dependent on the density of electrons n and holes p in the semiconductor, as well as the electron charge q, and the mobilities $\mu_{\text{n/p}}$. The mobility is (for a current with only one vector component) defined as $\mu_{\text{n/p}} = \mp\frac{\langle v \rangle}{E}$, where $\langle v \rangle$ is the mean velocity of the charges, and E the magnitude of the electric field component in the direction of conduction. This takes advantage of that the velocity of the charges in a semiconductor is almost[9] only dependent on the magnitude of the electric field.

[7] This is a dynamic equilibrium, as electron-hole pairs (EHPs) are constantly created and recombined. The equilibrium concentration is determined by the point of balance between the probability of EHP creation from a large concentration of valence electrons and empty conduction band states, and the probability that the much less concentrated EHPs recombine.

[8] N_c is defined from taking the actual electron density found in the material, as N_c is the number of states per volume unit if they where all located at E_c, and distributed the electrons distributed according to the Fermi distribution.

[9] Valid up to a certain limit in field strength, where velocity saturation becomes important and $\langle v \rangle$ as a function of E levels off.

2.2.1 PN-junctions

Even if a semiconductor is naturally conductive, it can be made non-conductive without having to cool it to 0° [K]. This is done by making a *PN-junction* in the material, making the device into a diode.

An ideal diode is a device that lets current through one way, but blocks any current going the other way. In the real world this is not the case (see figure 2.4), but not too far from it. This figure shows the current through the device as a function of bias voltage when in the *reverse bias* domain, which means the applied voltage is in the blocking direction. This is the domain most semiconductor sensors are operated in. We see that the current is very small to begin with, and that it is not really increasing as a function applied voltage. But after increasing the voltage above a certain point, a process known as *breakdown* occurs, suddenly rendering the device conductive. This may damage the device, as the current going through can be significant while the applied voltage across the device remains large, resulting in a significant power dissipation.

Semiconductors used in detectors – such as silicon, germanium, or diamond – all live in group IV of the periodic table, and thus have four valence electrons. This means they can form closed diamond lattices where each atom bonds to four nearby atoms, and this structure also provides the band-structure discussed above. To make a PN-junction, small[10] concentrations of impurities (also called *dopants*) are introduced into the material, changing its properties. For example atoms from group V in the periodic table, such as phosphorus or arsen, have five valence electrons. When introduced into a silicon crystal, this extra electron will be given up to the crystal, and the resulting ion bonded into the lattice. Likewise atoms from group III, such as boron, will also be bonded into the lattice. But as these atoms only have three valence electrons, an electron from the top of the crystal's valence band will be "eaten" into the bond, creating a hole and an ionized atom. Creating either extra conduction-band electrons or extra holes results in a shift in the dynamic balance between electron- and hole-concentrations, according to equation (2.7) [35], where n_i is the concentration of electrons and holes in intrinsic silicon at the same temperature.

$$n_0 \cdot p_0 = n_i^2 \tag{2.7}$$

Semiconductor crystals containing electron-donating dopants (*donors*) are known as *N-type* material, while crystals containing electron-eating (hole-creating) dopants (*acceptors*) are known as P-type. Putting P- and N-type material together creates a PN-junction at the interface, as shown in figure 2.5. What happens is that electrons and holes created by the dopants close to the interface diffuse over to the other side and recombine, and this process removes movable charges in this region. This results in an empty conduction band and a completely filled valence band, leaving the charges of the ionized dopants exposed. This region of the device, where there

[10] Typical dopant concentrations in a doped semiconductor material is in the order of $10^{14} - 10^{17}$ $[\frac{\text{atoms}}{\text{cm}^3}]$ [35], which means approximately 1 dopant atom per $10^8 - 10^5$ silicon atoms.

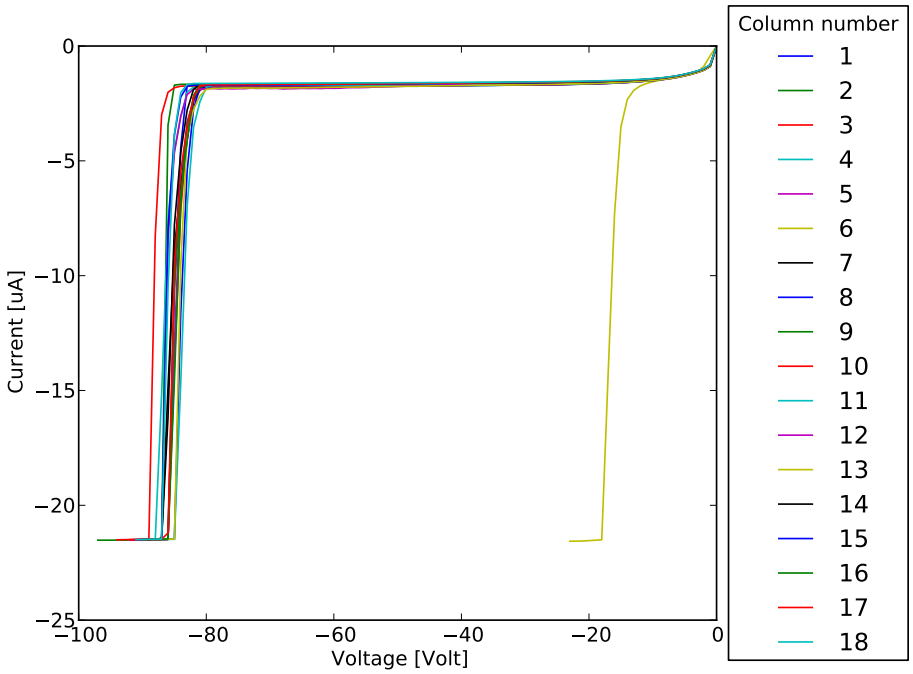


Figure 2.4: I-V characteristic from Sintef 2E first-generation full ATLAS 3D pixel sensor device in the reverse bias regime, measured at one pixel in every column (see Chapter 3.2 and6.1 for description of ATLAS pixel sensor geometry) One pixel measured per column, all showing typical diode IV characteristic until breakdown (Column 6 breaks down early). Measurements made in the Oslo EPF “clean-room" lab by Håvar Gjersdal. Flat part of plot after breakdown due to sourcemeter compliance at $I \approx 20$ [μA].

are no mobile charges, but space-charge due to the exposed dopant atoms, is known as the *depletion region*. The space-charge also results in an electric field inside the depletion region. This field is also present when there are no external influences on the device, and is known as the built-in or equilibrium field. The electric potential due to this field is known as the built-in potential.

The width of the depletion region is controlled by a balance of two currents, the first one being the diffusion current of electrons and holes “emitted” from the edges of the undepleted areas at the side where they are in majority, and diffusing across the depletion region against the electric field. This current is said to be in the “forward” or conduction direction of the diode. The other current is the drift current due to electrons and holes originating from the side where they are in minority, and drifting in the electric field across the depletion region. This current is said to be in the “reverse” or non-conducting direction of the diode. Under

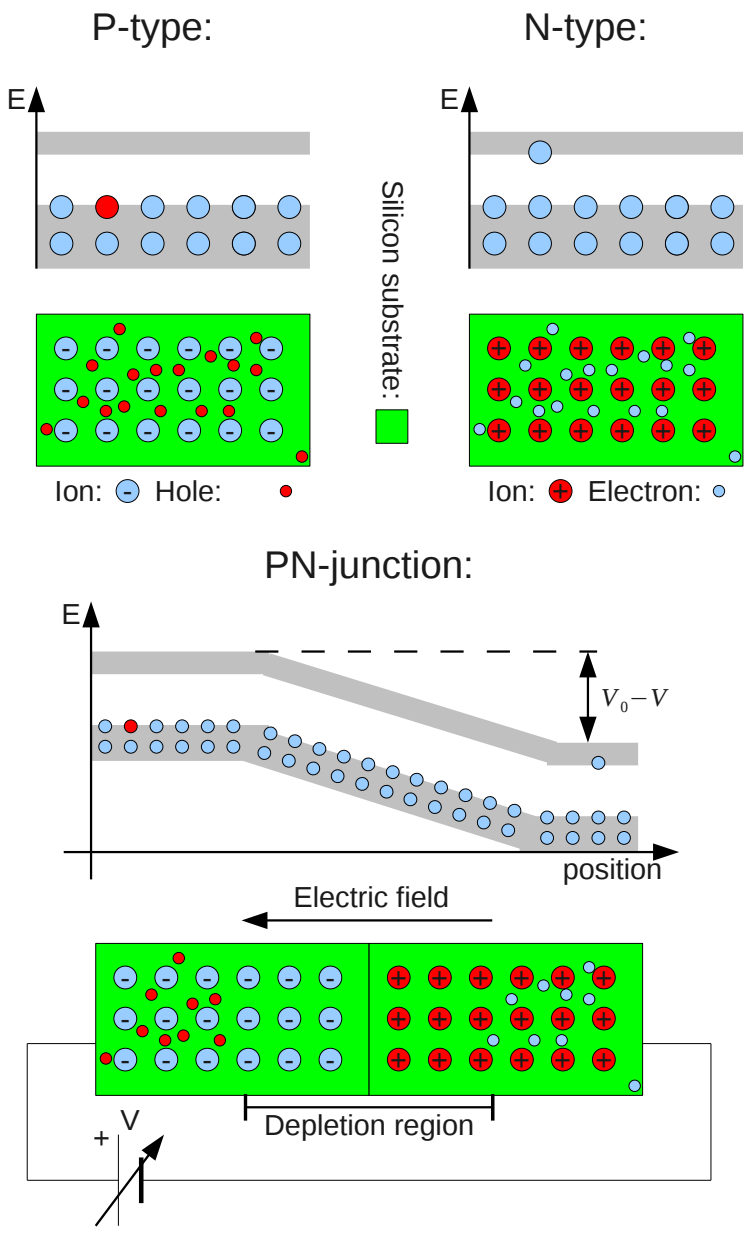


Figure 2.5: Bringing two pieces of N- and P-type semiconductor together, forming an (abrupt or "metallurgical") PN-junction. Energy levels and populations indicated, together with charge distributions. External voltage supply shown to illustrate external bias voltage V in the forward/positive direction. The built-in potential is denoted as V_0.

equilibrium conditions these currents balance each other, so that the net current is zero and the width of the depletion region is constant. In addition to these currents, there is a small contribution to the drift current from electron-hole pairs (*EHP*s) being thermally excited in the depletion region, and electrons and holes may also recombine in this region.

Applying or increasing the reverse bias voltage will have the effect of increasing the electric field in the sensor. As the field in the depletion region and the voltage drop across it is fixed by the concentration of ions in this regions (Gauss's law), the rest of the voltage drop has to be in the undepleted regions. The resulting electric field then removes majority carriers from these regions by collecting then at the power supply, thus increasing the size of the depletion region. This continues until enough space-charge has been "uncovered" in the depletion region to "absorb" almost[11] all the applied voltage. Applying a large-enough reverse bias will remove the mobile charges from the whole sensor interior volume (*sensor bulk*), which puts the sensor into a *fully depleted* state. Increasing the reverse bias voltage beyond this point (*overbiasing* the sensor) does not result in increased depleted volume, but increases the magnitude of the electric field.

If the sensor is biased in the reverse direction, the forward current component due to diffusion becomes smaller as fewer charges are able to diffuse across the wider depletion region and "taller" potential, while the reverse current stays constant. With large-enough applied voltage, this results in that the reverse current as a function of bias voltage becomes approximately constant, as seen in figure 2.4. This current is equal in magnitude to the drift current described above.

When biased in the forward direction, the electric potential across the depletion region is decreased. Thus a larger fraction of the majority carriers in the undepleted regions have enough energy to travel "uphill" across the depletion region, resulting in a forward current that grows exponentially as a function of the forward bias voltage.

Putting this into a mathematical form, one can show that the current as a function of bias voltage is given as equation 2.8 [35]. This equation is valid as long as the bias voltage is not positively large enough for removing the depletion region, or negatively large enough to cause reverse breakdown, and the rate of generation and recombination in the depletion region can be neglected.

$$I = I_0 \left(e^{qV/kT} - 1 \right) \tag{2.8}$$

Overbiasing the sensor too far results in a large electric field magnitude inside the device. When the field becomes large enough, reverse breakdown occurs, leading to a rapid increase of the current through the sensor, as seen in figure 2.4. Reverse breakdown is a complex phenomenon, and outside the scope of this thesis. It should however be noted that as it is initiated by a strong electric field, it is dependent on the electric field geometry. Further, breakdowns should normally be avoided, as they may damage the sensor.

[11] There still has to be a tiny field in the undepleted regions to keep the reverse current going.

The dopant concentration profiles portrayed in figure 2.5 are not particularly realistic, as this is not the normal way to create semiconductor devices[12]. What is normally done is to have a substrate (a *wafer*) which is "pre-doped" with a low concentration of N- or P-type dopant. The PN-junction is then created by introducing a dopant at the surface of the wafer by the use of techniques such as diffusion (where the sample is heated in an atmosphere with high concentration of dopant ions) or ion implantation (where the wafer is bombarded by dopant ions from a small particle accelerator). The device is then *annealed*, which means that it is heated to several hundred degrees Celsius in order to restore lattice damage created by the doping process, and also for integrating the dopant atoms into the crystal lattice. When this happens the dopant atoms become electrically active (meaning that they can exchange electrons with the rest of the lattice atoms), and this is why annealing is often referred to as "activation". The dopant is thus distributed with a continuous distribution in the device bulk.

2.2.2 Signal creation

As described in the beginning of this section, a charged particle passing through a semiconductor will excite electrons from the valence band into the conduction band, thus creating electron-hole-pairs (EHPs). When this happens inside the depletion region of the sensor, the charges will start drifting and diffusing under the influence of electromagnetic fields.

In order to collect the charges and transport it to electronics that can amplify and measure the signal, conductive electrodes are "inserted" into the sensor bulk and connected to the electronics. In practice these electrodes are just the non-depleted part of the sensor, which means that the field lines in an overbiased sensor ends on them. As the electrons and holes are more-or-less transported along the electric field lines, this means that electrons and holes created in the sensor bulk will be transported to electrodes, where they will be collected.

If one wants to calculate the time-dependence of the signal charge $Q_i(t)$ induced on electrode i by a charge q in the bulk of the sensor, it should be noted that $Q_i(t)$ is not equal to the sum of the charge that has arrived at the electrode at time t. Instead one should apply Gauss's law as shown in equation (2.9), where $\vec{E}$ is the electric field from the charge q somewhere in the sensor bulk, and the integral closing around the volume containing the induced charge $Q_i(t)$ (the electrode) which we want to calculate.

$$Q_i(t) = \epsilon \oint \vec{E}(t) \cdot \mathrm{d}\vec{A} \tag{2.9}$$

This approach is problematic because it demands a detailed calculation of the field $\vec{E}(t)$ for each of the timesteps we are interested in. However the problem can be solved in a simpler way through the Shockley-Ramo theorem (SRT) [29, 26, 17],

[12] Although a similar effect can be achieved by growing an epitaxial layer of doped silicon on top of a substrate with another doping.

as long as the charges are moving slowly enough that magnetic effects can be ignored.

The SRT introduces a *weighting potential* φ_0, and states that the charge $Q_i(t)$ and current $I_i(t)$ on electrode i due to a charge q at position $\vec{x}(t)$ can be calculated according to equations (2.10) and (2.11).

$$Q_i(t) = -q\varphi_0(\vec{x}(t)) \tag{2.10}$$

$$I_i(t) = q\dot{\vec{x}} \cdot \vec{E}_0(\vec{x}(t)) \tag{2.11}$$

Here the *weighting field* $\vec{E}_0$ is simply given as $\vec{\nabla} \cdot \vec{E}_0 = -\varphi_0$, and $\dot{\vec{x}}$ is the time-derivative of $\vec{x}$. The weighting potential φ_0 itself is defined as the electric potential that would exist if the electrode of interest is at unit potential, while all other electrodes are at zero potential, and all charges are removed. The weighting potential is also taken to go to zero at infinity.

Thus to calculate the time-dependence of a induced signal Q_i on electrode i, we first need to calculate the weighting field and the true electric field inside the device. When this is done, the drift-diffusion equations have to be solved for the charges created by the energy deposit[13]. This yields the charges position $\vec{x}(t)$ as a function of time, which can be inserted into equation (2.10) to get the induced charge on the electrode of interest.

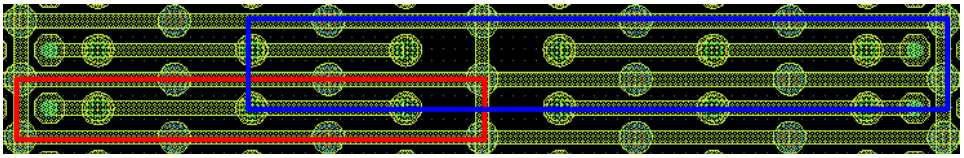
The number we are mostly interested in when studying the detector response is not the charge on the electrode as a function of time, but rather the input to the amplifier connected to the electrode. If the amplifier is ohmically coupled to the electrode, the amplifier will act as a source of charges, supplying the induced charge Q_i to the electrode. This means the input of the amplifier gets a charge $Q_{\text{input}} = -Q_i = -(-q\varphi_0(\vec{x})) = +q\varphi_0(\vec{x})$, which means it is of the same sign as the charge (eventually) collected by the electrode. On the other hand, if the amplifier is capacitively coupled to the electrode the sign is inverted.

HEP position sensing detectors normally use charge-integrating amplifiers that react slower than the sensors charge-collection time. This means that time-dependence of the signal is normally not so much of interest. Therefore the input to the amplifier from an electron-hole-pair (EHP) is given by equation (2.12). Here the carrier type collected on the electrode of interest is at position $\vec{x}_1(t_{\text{end}})$, and the opposite polarity carrier at position $\vec{x}_2(t_{\text{end}})$, where t_{end} is the end of the amplifier integration time.

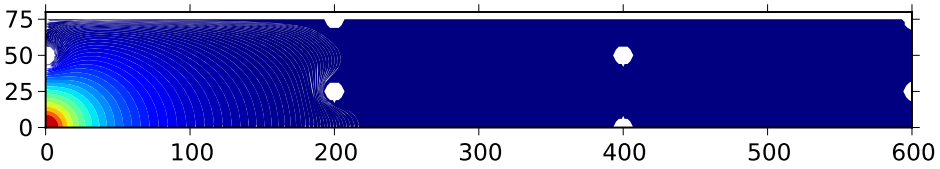
$$Q_{\text{input}} = \pm q\left[\varphi_0(\vec{x}_1(t_{\text{end}})) - \varphi_0(\vec{x}_2(t_{\text{end}}))\right] \tag{2.12}$$

If all charges are collected, the number inside the square brackets of equation (2.12) equals one. However, if some of the charges are trapped at a position $\vec{x}_{\text{trap}}$, this means that the amount of charge collected will be lowered, but not completely

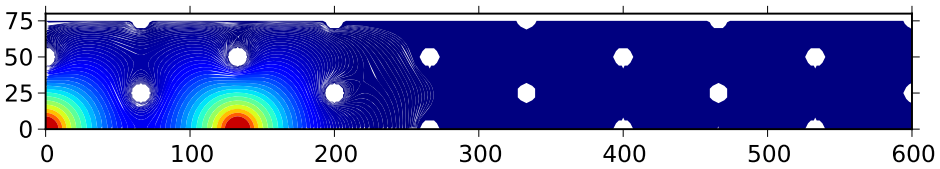
[13] This could be done in several ways, either treating the electrons and holes as particles or as distributions (charge-clouds) containing many electrons or holes. The underlying equations is anyway the diffusion equation and the Lorentz force, taking into account that carrier is (in most cases) given by the product of the electric field and the carrier mobility. A treatment of this is given in Chapter 4 and 6.2.



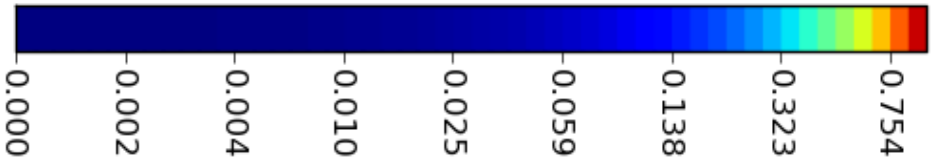
(a) Metal mask of 2x2 pixel cell of 3E device. Red box: One single pixel, covering 400×50 [μm^2]; Blue box: Quartercell off 3x3 cell used for calculation



(b) Weigthing field of 1E device 3x3 pixel quartercell. Units on x,y axis are in μm



(c) Weigthing field of 3E device 3x3 pixel quartercell. Units on x,y axis are in μm



(d) Colormap used in plots above. Additionally: White=0.0, Dark red=1.0

Figure 2.6: Two-dimensional numerical calculation of the weighting field of two 3D pixel sensor geometries. Plane of calculation parallel to sensor plane, perpendicular to electrodes. Calculated using relaxation method, program code inspired by [19], but extended to handle complicated boundaries and Von Neuman boundary conditions at axis of symmetry. 1×1 [μm^2] calculation grid used. Electrodes taken to have a radius of 7 [μm].

gone. Trapping may happen in the case of radiation-damaged sensors, which can contain additional energy levels inside the bandgap. These levels are localized around the point of the defect, and there may be a high probability that some of the created charges will transition into this *trap*. The charges then sit in the trap with a decay-time that is significantly longer than the integration time of the amplifier, resulting in an incomplete collection of the signal.

2.2.3 The Fano factor

When a charged particle deposits energy in a semiconductor, this energy has to be transfered to the electrons in the valence band for exiting them into the conduction band, thus creating EHPs as shown in figure 2.3. When this happens, one must conserve both energy and momentum. This is made possible by the emission of *phonons*, quanta of lattice vibrations, in addition to the excitation of electrons from the "top" of the valence band into the conduction band. The result is that the mean energy required to create one single EHP ($w = 3.6$ [eV] in silicon [8]) is larger than the bandgap ($E_g = 1.12$ [eV]).

This sharing of energy between excitation of electrons across the bandgap, and creation of phonons results in a a distribution in the number of EHPs created for a given energy deposit Δ. For $\Delta \gg E_{ionization} = E_g$ one can with good approximation assume that the number of EHPs created follows a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with mean value μ and variance σ^2, and it can be shown [33] that the parameters are given by equation (2.13).

$$P(N_{\text{EHPs}}) = \mathcal{N}(\mu, \sigma^2) \quad \text{where} \quad \mu = \frac{\Delta}{w} \quad \text{and} \quad \sigma^2 = F\mu \tag{2.13}$$

Here F is the *Fano factor*, which is measured to be $F = 0.1$ in Silicon [8].

The Fano factor thus presents an intrinsic energy limit to the energy resolution of semiconductor photon detectors, avoidable only using materials with smaller F.

2.2.4 Use of semiconductor sensors for position measurement

Since their introduction for precise photon energy measurements the 1960s [21], semiconductor sensors are today used in large areas of nuclear and particle physics, usually connected to measuring the position and/or the energy of particles.

Measuring where a charged particle hit a silicon sensor is mostly done with segmented sensors – sensors that have many electrodes, each electrode connected to separate amplifier channels. Thus the charges created by a traversing particle are collected on contacts close to the particles path, and by knowing the position of the contacts, this yields an estimate for the hit position.

As an example, consider a silicon strip detector, such as shown in figure 2.8(a). A charged particle passing through this detector leaves charges on nearby strips, so a simple estimate for the hit position is the charge-weighted mean (equation (2.14))

where x_k is the position of strip k, and $Q^{(k)}$ is the signal on this strip.

$$\hat{x}_c = \frac{\sum_k x_k Q^{(k)}}{\sum_k Q^{(k)}} \tag{2.14}$$

However this estimator has a problem if an unproportionally large amount of the signal is collected by the strip closest to the path of the particle, as this biases the estimator away from the centerline between the strips.

A way of correcting for this is the so-called η-correction described by Belau et. al [3]. This can be understood by considering the case of sharing between two strips "L" and "R" with a pitch δ, and wanting to estimate where in the space between them the particle hit. This can be done by defining a variable η as equation (2.15), similarly to $\hat{x}_c$ for two strips only.

$$\eta \equiv \frac{Q_L}{Q_L + Q_R} \tag{2.15}$$

The goal is to find a mapping from η to a hit position estimator $\hat{x}_\eta$. This can be done experimentally by using the distribution $\mathrm{d}N/\mathrm{d}\eta$ (such as shown in figure 2.7(a)) for the detector, where N is the number of hits. Using the chain rule and integrating yields:

$$\frac{\mathrm{d}N}{\mathrm{d}\eta} = \frac{\mathrm{d}N}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}\eta} \Rightarrow \int_0^\eta \frac{\mathrm{d}N}{\mathrm{d}\eta'}\mathrm{d}\eta' = \int_0^\eta \frac{\mathrm{d}N}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}\eta'}\mathrm{d}\eta'$$

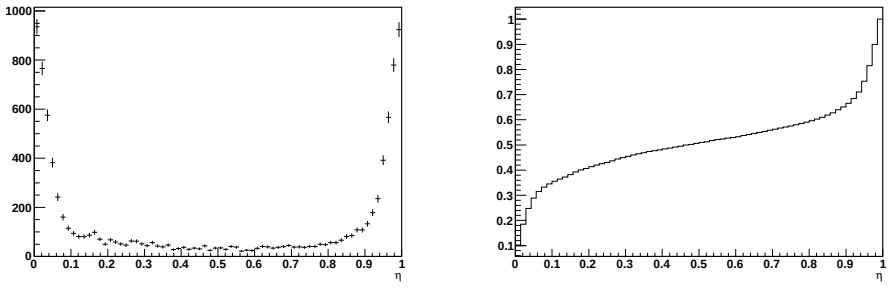
This can be simplified by assuming that the beam intensity changes little over the distance δ, which yields that $\mathrm{d}N/\mathrm{d}x = N_0/\delta$, where N_0 is the number of tracks intersecting the detector. Inserting this into the expression above and performing the integration of the right hand side then yield an expression for $\hat{x}_\eta$ as a function of η, which is was we wanted.

$$x_\eta(\eta, x(\eta = 0)) = \frac{\delta}{N_0}\int_0^\eta \frac{\mathrm{d}N}{\mathrm{d}\eta'}\mathrm{d}\eta' + x(\eta = 0) \tag{2.16}$$

Thus the distribution $\mathrm{d}N/\mathrm{d}\eta$ can be measured experimentally for the sensor of interest, and the cumulative distribution function (CDF) $\int_0^\eta \frac{\mathrm{d}N}{\mathrm{d}\eta'}\mathrm{d}\eta'$ found by numerical integration. This is then inserted into (2.16), which for a given η yields a position estimate.

2.2.5 3D pixel sensors

A pixel sensor used in high-energy physics is a device that measures the (x, y) position of a passing particle. This is done by segmenting the electrodes of the detector into a rectangular grid. The charges created by a passing particle are then collected on a nearby electrode, and the hit position can then be inferred by techniques such as discussed in Chapter 2.2.4.



(a) $\mathrm{d}N/\mathrm{d}\eta$ distribution

(b) CDF of normalized $\mathrm{d}N/\mathrm{d}\eta$ distribution

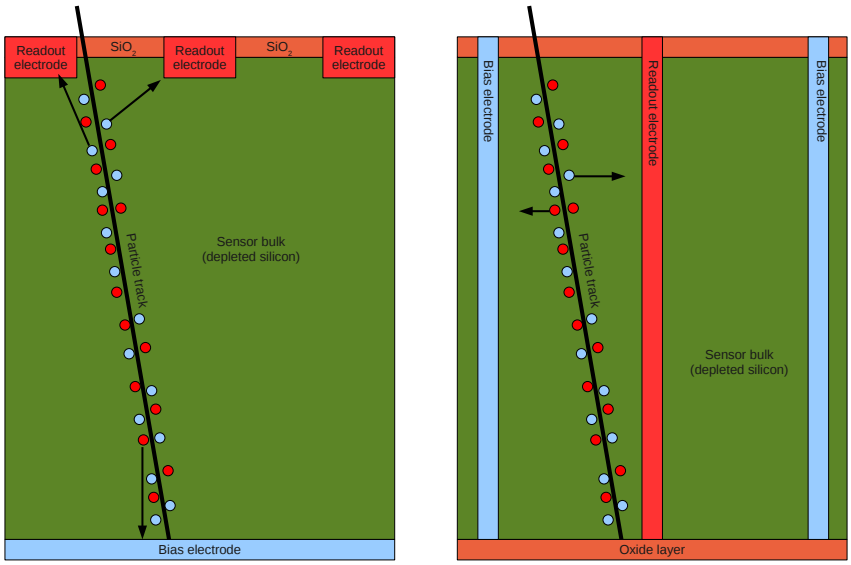
Figure 2.7: $\mathrm{d}N/\mathrm{d}\eta$ distribution and its cumulative distribution function, used for η-correction. Data from BAT1 p-side, run705 May 2009, using common-mode corrections described in Chapter 4.2.3.)

What differentiates a "3D" pixel sensor (described in [24, 20]) from a planar pixel sensor is the geometry of the electrodes, illustrated in figure 2.8. In a planar geometry, the collecting electrodes are created by heavily doping areas of the surface, while the bulk has a low doping concentration. This is changed in a "3D" geometry, where the electrodes extend into the bulk of the sensor. This is achieved by DRIE etching holes in the sensor with close to vertical walls, and doping the walls of these electrodes to create a PN-junction around the holes. The electrodes can then optionally be filled with a conducting material such as polysilicon.

There are currently two kinds of 3D pixel sensors being made, the *full 3D* design (figure 2.9(a)), and the *Double-side Double Type Column* design (figure 2.9(b)). The full 3D design is processed on one side, with the electrodes fully penetrating the sensor substrate. This was the original design proposed by S. Parker and C. Kenny, and currently fabricated at Stanford Nanofabrication Facility and Sintef. The DDTC design proposed and used by research organizations FBK and CNM is processed from two sides (one bias- and one readout-side), the etched electrode holes only partially overlapping. This provides a hybrid between planar and fully 3D structures.

Additionally, the sensors with different number of readout electrodes per pixel are being tested. An example of the layout of a "3E" sensor, which has three readout electrodes per pixel, is shown in figure 2.6(a).

There are two main advantages of using this more complex production technique; radiation hardness [41] and the possibility of active edges. As mentioned in Chapter 2.2.2, the main mechanism of radiation damage in a silicon sensor is the creation of "trap" levels inside the bandgap that captures moving charges created by the passing particle. In a planar sensor much of the charge has to travel through a significant part of the wafer thickness, which leaves it more time to get caught in a trap. The 3D sensor geometry helps this by making the inter-electrode distance



(a) Charge collection in a 2D (planar) semiconductor sensor

(b) Charge collection in a 3D semiconductor sensor

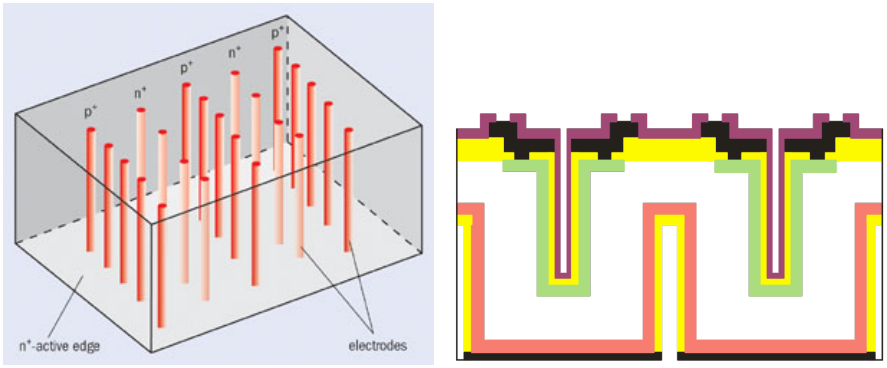
Figure 2.8: Schematic comparison of ohmically coupled 2D- and 3D-sensor geometries

smaller, especially in the layouts with many electrodes. Further, radiation damage also result in inversion of the effective bulk doping from n- to p-type, with the p-type doping concentration gradually increasing over time [44]. This results in an increasing space-charge, increasing the voltage needed to deplete the sensor until a point where it no longer can be fully depleted is reached. This is a smaller problem for 3D sensors, as they have a lower depletion voltage than planar sensors.

The second big advantage is the possibility of *active edges*, which means that the sensor is active all the way to its edge. This is not possible in a conventional planar sensor, as one needs to step down the bias voltage before reaching the edge of the wafer. This is necessary due to defects in the silicon close to the sawcut made when separating the sensors on a wafer, making this volume conducting. The stepping down is done with the help of *guard rings*, which are consecutive rings of implants [4, 43] running around the active area of the device, with stepwise lower potential, as shown in figure 2.10.

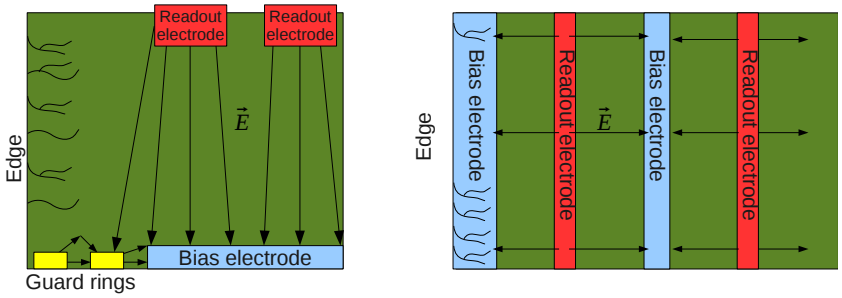
One should also notice that 3D sensors collect the charges very quickly, which with the right amplifiers could enable very high event rates. Further, no intrinsic limit on the thickness of 3D sensors are imposed by charge collection. This could be an advantage when building high-energy photon detectors with high efficiency.

Due to the electric field in a full 3D sensor being parallel to the device plane, and the caging of bias electrodes around the active electrodes in a pixel, charges



(a) Full 3D (b) Double-side Double Type Column

Figure 2.9: Geometry concepts for 3D pixel sensors



(a) Planar sensor with guard rings (b) 3D sensor with guard electrode at edge

Figure 2.10: Cutthrough view of the edge of sensors, showing microcracks that make the edges conductive, and how this mitigated in different sensor technologies.

are mostly collected at the pixel being hit by a particle. This means lower charge sharing than in planar geometries, especially for perpendicular tracks (see figure 2.11 and [13]). Lower charge sharing is a double edge sword; On one side there is less information available for sub-pitch position estimation techniques such as described in Chapter 2.2.4, but on the other hand the charges are not spread over several amplifier channels, something that can result in failure to detect the particle.

Another special property of 3D sensors is that the the material inside the holes are mostly[14] insensitive to energy deposits, as it it not depleted. This effect is clearly present when the track is perpendicular to the device, as the particle may then only pass through dead material. However, if the track passes through the device at an angle, it will also pass through some active material. This can be seen in figure 2.12, which shows that the sensors have little or no dead areas with angled tracks[15], and is also discussed in Chapter 6.2.2.1. This effect is naturally more present in the full 3D sensors than in the DDTC sensors, as the particle will always pass through some active material in a DDTC (see figure 2.9), and this is also visible in the data (figure 2.12).

2.3 The Monte-Carlo method

The Monte-Carlo method is a class of numerical techniques that use pseudo-random[16] numbers to solve complicated mathematical problems. With the use of such methods it is often possible to approximate the solution of very complicated problems, which is not possible or very hard to solve analytically or with ordinary deterministic numerical techniques.

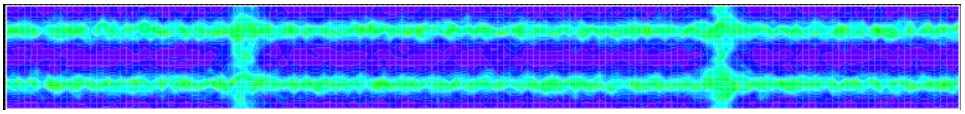
The Monte-Carlo method is often referred to as a method of doing "computer experiments". This is due to the similarity of applying the method and the performance of actual experiment. In an actual experiment you will usually repeat the same measurement many times, and record the outcome. After doing this, you can analyze the data to find probability distributions of the measured value(s), or simply the mean and RMS.

As an example, take an experiment for measuring the distribution of the sum of two dices. This example is possible to calculate analytically, but nevertheless offers an opportunity to demonstrate the Monte-Carlo method. The probability of

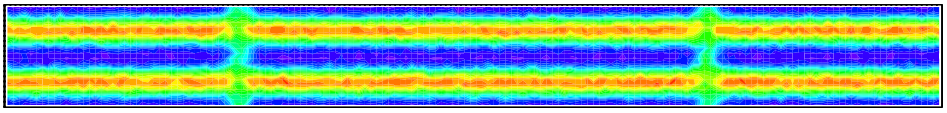
[14] Some sensitivity may still be present due to diffusion of created minority carriers into the depleted region

[15] It has been indicated in [13] that 3D silicon sensors respond very weakly to a magnetic field, while the Lorentz force leads to a focusing of the charge-clouds in an angled planar sensor. Comparisons are therefore shown with the magnetic field turned on.

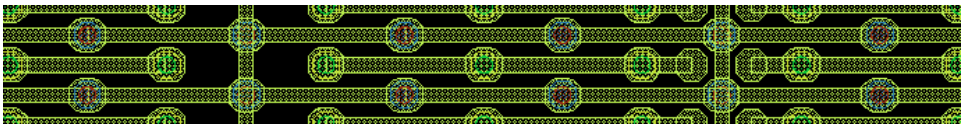
[16] As a computer is a deterministic machine following a set algorithm, it cannot generate true random numbers. However numerical algorithms exist that generate number sequences with the wanted statistical properties, such as being distributed according to a (usually flat) distribution, and which have a very low correlation between numbers for most "patterns" below a certain length.



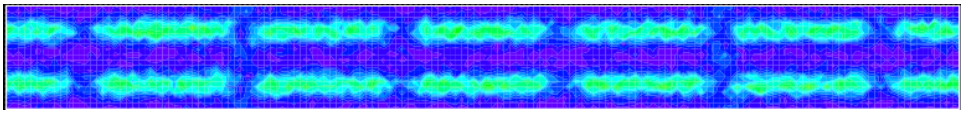
(a) Planar sensor, $\approx 0^{\circ}$, magnet off.



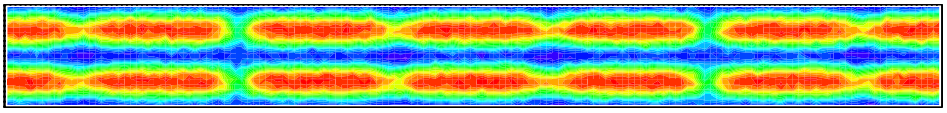
(b) Planar sensor, $\approx 15^{\circ}$, magnet on.



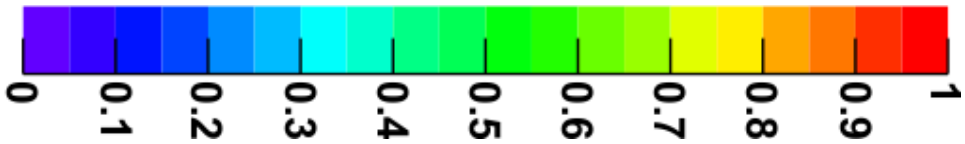
(c) Mask detail from full 3D sensor with 3 electrodes, 2x2 pixels centered on one pixel



(d) Full 3D sensor, $\approx 0^{\circ}$, magnet off.

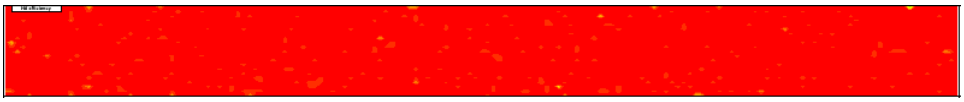


(e) Full 3D sensor, $\approx 15^{\circ}$, magnet on.

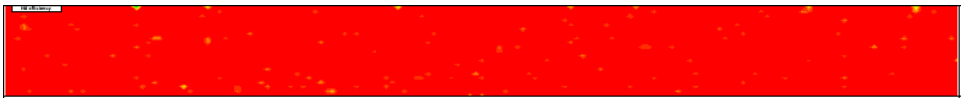


(f) Palette used for plots

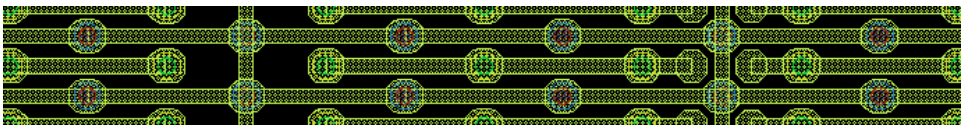
Figure 2.11: Measured sensor charge sharing probabilities as function of estimated track position in 2x2 pixel cell, with tracks normal to sensor plane and no magnetic field, and tracks tilted with respect to sensor plane and ≈ 1.6 [T] magnetic field. Figures from [13], using data from May 2009 testbeam.



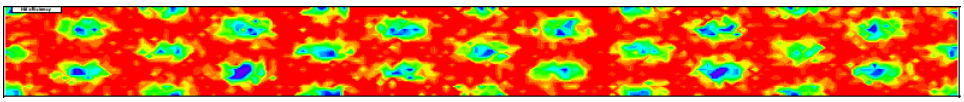
(a) Planar sensor, $\approx 0^{\circ}$, magnet off. 99.9% efficiency



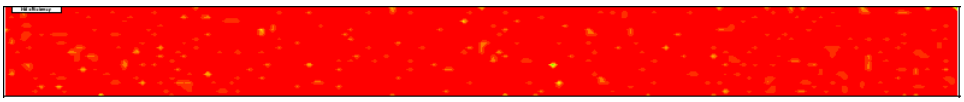
(b) Planar sensor, $\approx 15^{\circ}$, magnet on. 100.0% efficiency



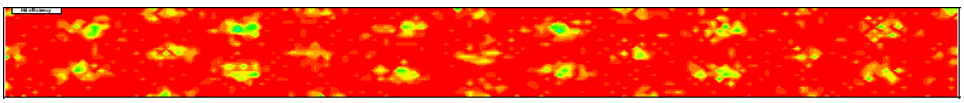
(c) Mask detail from full 3D sensor with 3 electrodes, 2x2 pixels centered on one pixel



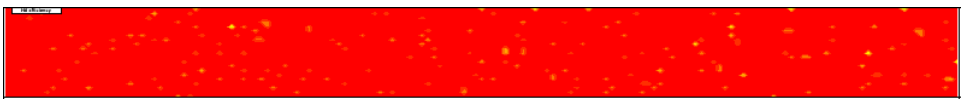
(d) Full 3D sensor, $\approx 0^{\circ}$, magnet off. 96.7% efficiency



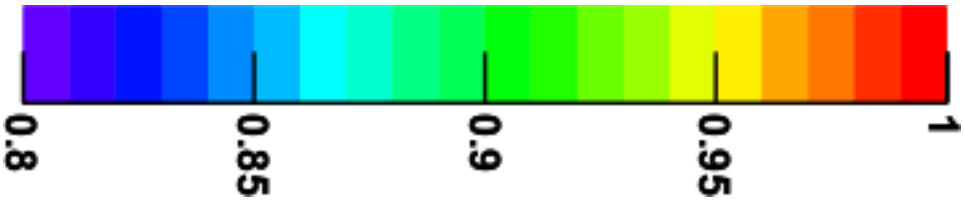
(e) Full 3D sensor, $\approx 15^{\circ}$, magnet on. 99.9% efficiency



(f) DDTC sensor, $\approx 0^{\circ}$, magnet off. 99.2% efficiency



(g) DDTC sensor, $\approx 15^{\circ}$, magnet on. 99.8% efficiency



(h) Palette used for plots

Figure 2.12: Measured sensor hit efficiencies as function of estimated track position folded into a 2x2 pixel cell. Plots for tracks normal to sensor plane and no magnetic field, and tracks tilted with respect to sensor plane and ≈ 1.6 [T] magnetic field. Hit efficiency defined as the probability of at least one non-noisy central region pixel which is matching track position going above threshold. Figures from [13], using data from May 2009 testbeam.

Listing 2.1: Python/pyROOT code for simulating
the probability distribution $P(K)$ as defined in equation (2.17)

```
1 def DoMC(trials,histo):
2     for trial in xrange(trials):
3         k1 = random.randint(1,6)
4         k2 = random.randint(1,6)
5         K = k1+k2
6         histo.Fill(K)
7     #Normalize total probability to unity
8     histo.Scale(1.0/histo.Integral())
```

a 6-sided dice giving a result k is:

$$P_0(k) = \begin{cases} \frac{1}{6} & \text{for} \quad k \in [1, \ldots, 6] \\ 0 & \text{else} \end{cases}$$

Further, we can define the probability of rolling two 6-sided dices such that their eyes sum up to K as $P(K) = P(k_1, k_2)$, where $K = k_1 + k_2$. This probability can be calculated by noting that $P(K)$ is equal to the sum of the probabilities of all possible ways of getting the dices to sum up K, which can be written as a convolution of $P_0(K)$ with itself:

$$P(K) = \sum_{s=-\infty}^{\infty} P_0(s)P_0(K-s) \tag{2.17}$$

A Monte-Carlo method for calculating this distribution is very analogous to how one would do find it experimentally by throwing two real dices. A code fragment is shown in listing 2.1. Here the variable `trials` is the number of virtual dice-pairs to throw, and `histo` the histogram used to tally the results. In figure 2.13 the resulting histogram is shown for different numbers of trials.

One can also do Monte-Carlo methods for generating continuous probability distributions. Here one start with a pseudo-random number generator which generates random numbers with a flat probability distribution in the interval $[0, 1)$, and transform these into the distributions wanted. There are several methods for doing this transformation, where using the inverse cumulative distribution function, and acceptance-rejection sampling are the most common [31, 19].

Using the Monte-Carlo method, more complicated distributions can be simulated by convolution or repeated application of several sampled distributions. If one for example wants to study the probability distribution describing the final state of a particle after traversing several layers of material, this can be effectively done as long as the probability distributions describing what happens in each layer is known. Assuming only one particle and no emission of secondaries or absorption of the particle under study, one would start with an initial state described by the particle's energy-momentum four-vector. The particle will then be scattered by the

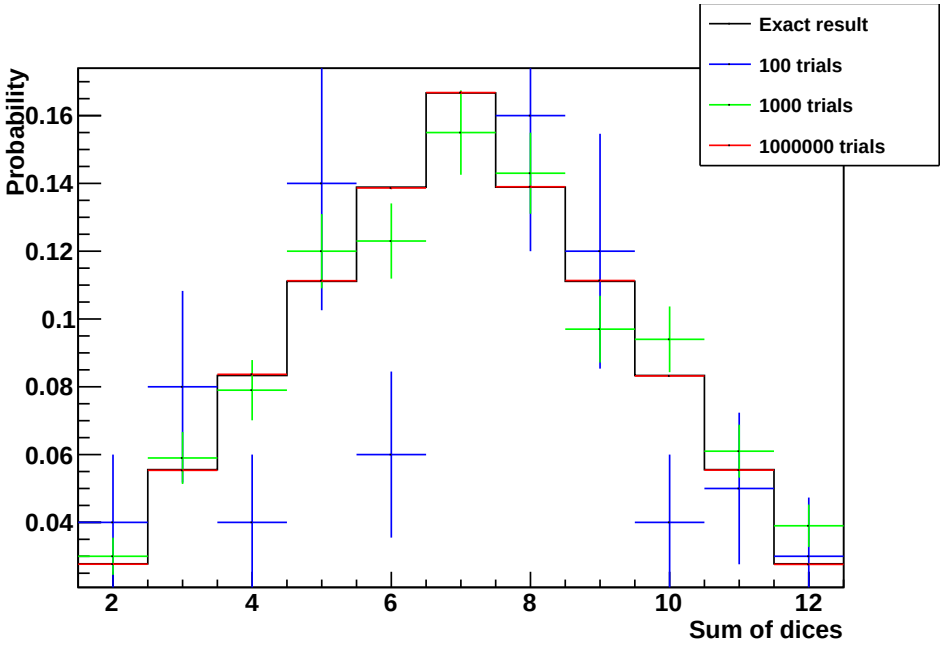


Figure 2.13: Resulting distribution from a Monte-Carlo simulation of an experiment to measure the distribution of the sum of two dices. PDFs estimated with a different number of trials, and compared to the exact result. Note that different estimated distributions are generated for each run of the simulation, as it is based on pseudo-random numbers. Error bars are $\pm\sqrt{N}$. The “core" simulation code is shown in listing 2.1. All distributions normalized to unity.

first layer of material, and the scattering process is described by a PDF which is depending on the particle's state before the scattering. Sampling this PDF will yield new state, which is then further changed by the PDF describing the second process etc. This is repeated until all layers of material is traversed, and the final state reached. This is then a method of sampling the final state, and it is again similar to the experiment one would perform in order to measure the final-state PDF.

Chapter 3

Simulation of testbeam experimental setup

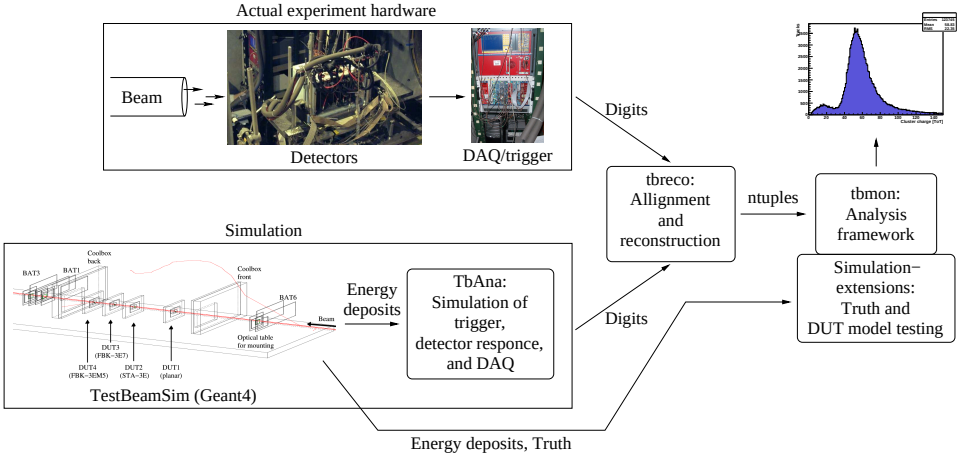


Figure 3.1: Data flow from experiment or simulation to analysis, showing the steps taken from data production to analysis. Points of data creation and processing also showed.

A simulation system (figure 3.1) has been written in order to enable benchmarking of the testbeam setup (described in Chapter 1.2), and to test models describing sensor response. This framework is consisting of three main parts:

1. Simulation of interactions between beam particles and the material in the experiment (sensors, mechanics, air, etc.).

2. Simulation and tuning of the telescope sensor and electronics response, and simulation of the trigger/DAQ system.

3. Simulation and development of models describing ATLAS pixel device under test (DUT) response.

The first part is done in a Geant4-based program called `TestBeamSim`, and is the main topic of this chapter. In addition to this chapter, the technical aspects of the program is documented in depth in Appendix A.

The second part is the domain of the program package `TbAna`. The telescope response model used is described in Chapter 4, the file format used by the DAQ system to store the digits (BDT) is described in Appendix D, and the trigger system simulation is described in Chapter 3.4. Technical aspects of the `TbAna` software package is described in Appendix B.

The third part of the simulation framework is the simulation support which has been added to the testbeam data analysis framework `TbMon`. The technical parts of this is described in Appendix C, and the model development in Chapter 6. In addition to pixel device simulation, this analysis framework (with extensions) was used to do the telescope performance analysis described in Chapter 5.

3.1 Physics models used in the simulation

The part of the simulation dealing with interaction between the beam particles and material is handled by the program `TestBeamSim`. This program is based on the Geant4 framework, and the code itself is documented in Appendix A.

Geant4 [2, 10, 12] is a `C++` library for simulating the passage of high-energy[1] particles through matter. To use it, one must specify a geometry built up from volumes (shape, position, and material), the incident radiation, and the physics describing the behavior of the radiation when passing through material. When this has been specified, the Geant4 kernel can be instructed to simulate a set number of events.

For each event simulated, the particles in the event are tracked through the geometry in a series of steps. For each step taken, the physics processes applied to the tracked particle is allowed to change its state (energy, momentum direction, spin, etc.), and can also to create new secondary particles. Internally in the processes, the new state is selected with a Monte-Carlo method (see Chapter 2.3), which means that two particles with identical initial states will usually have different final states.

As there are a large range of processes that can be applied to the particles, many of which take multiple parameters, setting up the list of which processes (including model parameters settings) applies to which particles can be quite complicated. Therefore pre-made and validated physics lists are included in the Geant4 distribution. A comparison of some of these physics lists is found in Chapter 3.1.2.

The focus of Geant4 is the behavior of the simulated particles, not what happens to the material traversed by them. However at each step, Geant4 calculates

[1] The energy range for which the simulation is valid is depending on the physics model(s) used.

the energy deposit in the material. This is the part of the energy lost in the step by the tracked particle, which has not been transfered to new secondary particles.

As the number of low-energy secondaries (mostly low-energy δ-electrons) that are created when a charged particle traverses a block of material can be very large due to the large number of low-energy interactions, it is not computationally feasible to simulate all of them in detail. A cutoff is therefore introduced to remove some of these particles, implemented in a way that does not ruin accuracy. In Geant4 this is done by a range cutoff at the production of new particles – a particle is only simulated in detail if its energy is high enough so that its expected range in the material is longer than a set range-cut, or if it can reach a new volume (which may have a less dense material). The energy of the particles that are "killed" by this cut is added to the energy deposit of the current step. This way of doing production cuts ensures that the simulation is spatially accurate down to the selected cutoff range, while avoiding divergence in the number of particles produced.

3.1.1 Fano sampling

To bridge the gap between the energy deposited by the tracked particles and the effect this has on the material (liberation of electron-hole pairs), a routine for generating "charge clusters" is used for silicon detectors, and is documented in Appendix A.4.5. This routine takes the ionizing energy deposit in the step, and the position of the step's start- and end-points as input, and produces a number of smaller energy deposits which are uniformly sampled along the step. Here the approximate size of the smallest energy deposits wanted can be specified as an input parameter, in units of number of electron-hole pairs. The routine then does a random sampling of the size of each energy deposit, taking into account the Fano factor described in Chapter 2.2.3. It is based on `G4ElectronIonPair`, but changed to produce clusters of charges instead of single electron-hole pairs. This is done because it would be too computationally expensive to handle $O(10^4)$ EHPs separately in each sensor and event, both for file I/O and also later when calculating sensor response.

The derivation of the method is based on the summation and linear transformation properties of Gaussian distributions, described in [42]. We start with assuming an energy deposit E_0, which we want to split across N charge clusters. The energy deposit E_0 produces a total number of N_0 electron-hole pairs, where N_0 is a stochastic variable following the distribution given by equation (2.13):

$$P(N_0) = \mathcal{N}\left(\frac{E_0}{w}, \frac{E_0}{w} \cdot F\right) \equiv \mathcal{N}(\mu, \sigma^2) \tag{3.1}$$

Here $\mathcal{N}(\mu, \sigma^2)$ is a Gaussian distribution with mean μ and variance σ^2, w the mean ionization energy of the material, and F the Fano factor.

We then assume that the total energy E_0 is split equally across the N charge-clusters, and name the number of electron-hole pairs in cluster k as N_k ($k \geq 1$). N_k is then another stochastic variable, which is distributed from the unknown distribution $P(N_k)$. This distribution can then be found by requiring that the probability

distribution of the sum of all charges is equal to the probability distribution of N_0:

$$P\left(\sum_{k=1}^{N} N_k\right) = P(N_0)$$

Due to the central limit theorem (CLT), this is possible as long as the N_ks are independent random variables with finite mean and variance. We then assume that each of the N_ks are distributed from Gaussian distributions, which is valid as long as its mean $\mu_k \gg 1$ (fractional electrons don't make sense), and also the number of charge clusters $N \gg 1$ (validity of CLT).

From the general summation properties of Gaussian distributions, we know that if $Z_1, Z_2, \ldots, Z_N$ is distributed following a Gaussian distribution with mean μ and variance σ^2, then $Z' = Z_1 + Z_2 + \ldots + Z_N$ is also distributed following a Gaussian distribution with mean $N \cdot \mu$ and variance $N \cdot \sigma^2$. Assuming that the energy is split equally, it we can assume that each N_k is also identical and independently distributed with a Gaussian probability distribution $P(N_k)$. This distribution is then given as:

$$P(N_k) = \mathcal{N}\left(\frac{N_0}{N}, \frac{\sigma_0^2}{N}\right) = \mathcal{N}\left(\frac{E_0}{wN}, F\frac{E_0}{wN}\right) \equiv \mathcal{N}\left(\mu_k, \sigma_k^2\right) \tag{3.2}$$

Due to the fact that Fano sampling was introduced after much of the device simulation software for the BAT telescope was already mostly finished, output is not in units of electron-hole pairs, but in units of energy. The effective energy of the cluster is then given as $E'_k \equiv w \cdot N_k$, which by use of the linear transformation properties of Gaussian distributions can be shown to be distributed as:

$$P(E'_k) = \mathcal{N}\left(w\mu_k, w^2\sigma_k^2\right) = \mathcal{N}\left(\frac{E_0}{N}, \frac{FE_0w}{N}\right) \equiv \mathcal{N}\left(\mu_{E_k}, \sigma_{E_k}^2\right) \tag{3.3}$$

The number of sampled clusters N is calculated from the size of the total energy deposit in the step E_0, so that that the mean size of the clusters stays approximately constant and below a limit which is a parameter to the simulation. This parameter is the maximum mean cluster size m, which is in units of electron-hole pairs. Thus the smallest integer number of clusters N which yields a mean cluster size below m is given by:

$$N = \text{ceil}\left(\frac{E_0}{wm}\right) \tag{3.4}$$

While uniform sampling along the step is used for charged particles, uncharged particles such as photons are taken to have all its deposit at the end of the step. Fano "smearing" of the deposit is still applied, in units of energy. This is done by scaling N_0 as defined in equation (3.1) by w, the mean energy necessary to produce an electron-hole pair:

$$P(E'_\gamma) \equiv P(w \cdot N_0) = \mathcal{N}\left(E_0, FE_0w\right) \tag{3.5}$$

3.1.1.1 Implementation errors in Fano sampling routine

Unfortunately a mistake was made in the first derivation of the Fano sampling equations (3.3) and (3.5). This error was discovered too late for redoing the Monte Carlo production, device simulation, BAT model tuning, track reconstruction, and analysis. However, as seen in figure 3.2, the resulting total charge deposit spectra does not seem to be affected. A comparison of the spread in charge cluster size as a function of total charge deposit is shown in figure 3.3, showing the effect of the error. Here the points below the horizontal "line" at $400\ [\mathrm{e}^-]$ is due to that the sampling is done per-step, while the total energy deposit is summed up over all clusters in the step. This means that if there are more than one simulation step inside the sensor, which happens if there are energy deposits from secondary particles, or if the primary particle take two steps through the sensor, these "extra" steps may have a mean cluster size below $400\ [\mathrm{e}^-]$ if their energy deposit are small.

The mistake has since been corrected in the code, in a way that in the future will enable new geometries and DUTs to use the correct Fano factor, while keeping the old and technically wrong sampler for BAT planes in order not to invalidate device simulation tunings[2]. This updated code has been used to produce some of the plots in this thesis which are not depending on track reconstruction, such as figure 2.1 and the figures in Chapter 3.1.2.

For charged particles, this resulted in the effective Fano factor being $F_{\text{eff}} = F^2$:

$$\sigma_{E'_k}^{(\text{implemented})} = F\sqrt{\frac{E_0 w}{N}} \neq \sigma_{E'_k}^{(\text{correct})} = \sqrt{F\frac{E_0 w}{N}}$$

Since $F = 0.1 \Rightarrow F_{\text{eff}} = 0.01$ in Silicon, this effectively removes most effects of smearing due to the Fano factor for charged particles. Note that the random sampling of cluster position along the track and the mean cluster size remains correct, which should be the most important part of this routine for device simulation. This error makes that all charge clusters created from a step are of approximately the same size, when there should have been a larger spread, as seen in figure 3.3. For the overall charge distribution the effect should be small, as deposits in the high energy tail is most affected. Here the distribution is relatively flat, so the effect of not including this should average out after many events. This is seen in figure 3.2.

The output of the device models, such as discussed in Chapter 4 and 6, are still valid, as their output are benchmarked and tuned against experimental data. Thus conclusions based on simulation data, such as telescope resolution estimates, are not invalidated by this error.

For uncharged particles, the effective Fano factor is $F_{\text{eff}} = \frac{F^2}{w} \approx 0.0031\ [\mathrm{eV}^{-1}]$, as the mean energy per EHP w also dropped out of the expression. With the `CLHEP` system of units, $1\ [\mathrm{eV}]$ is numerically represented as 10^{-6}, as the base unit of energy used is MeV. This means that the numerical value of the effective Fano factor becomes $F_{\text{effective}} \approx 3 \cdot 10^3$, which is way off. Luckily this only affects photons,

[2] For details, see Appendix A.4.5

and as there are relatively few high-energy photons incident on the sensor, and even fewer stopping in it, the effect should be negligible.

It should be noted that `G4ElectronIonPair`, part of the Geant4 distribution, implements $\sigma_{N_0} = 0.2\sqrt{E_0/w}$, which is also incorrect. This yields $F_{\text{eff}} = 0.2^2 = 0.04$.

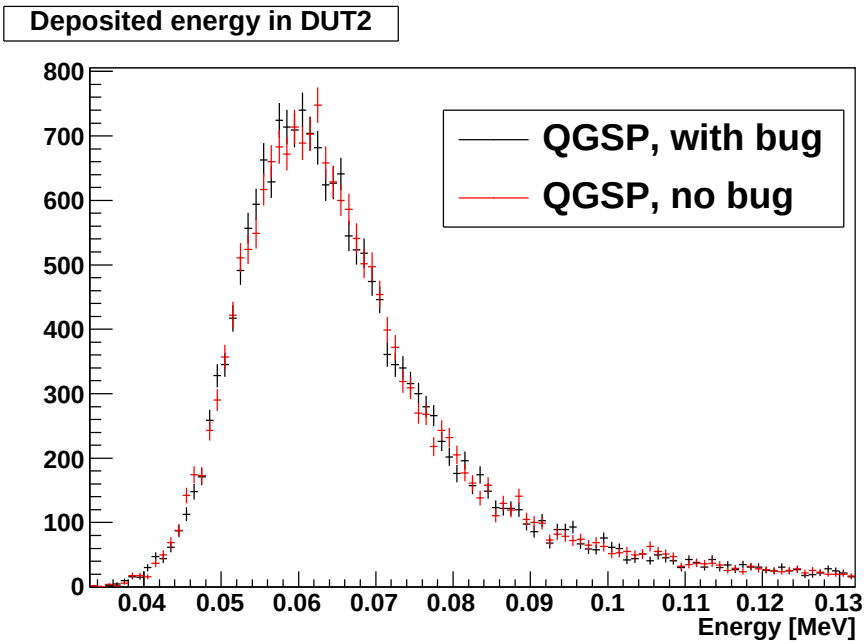
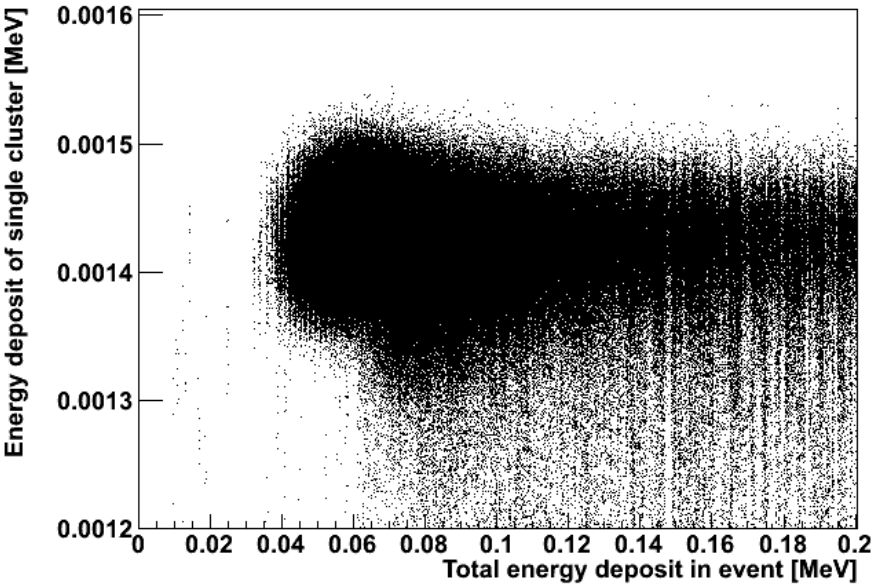


Figure 3.2: Comparison of total charge deposition spectrum in DUT2 with and without error in Fano sampling code (see Chapter 3.1.1.1).

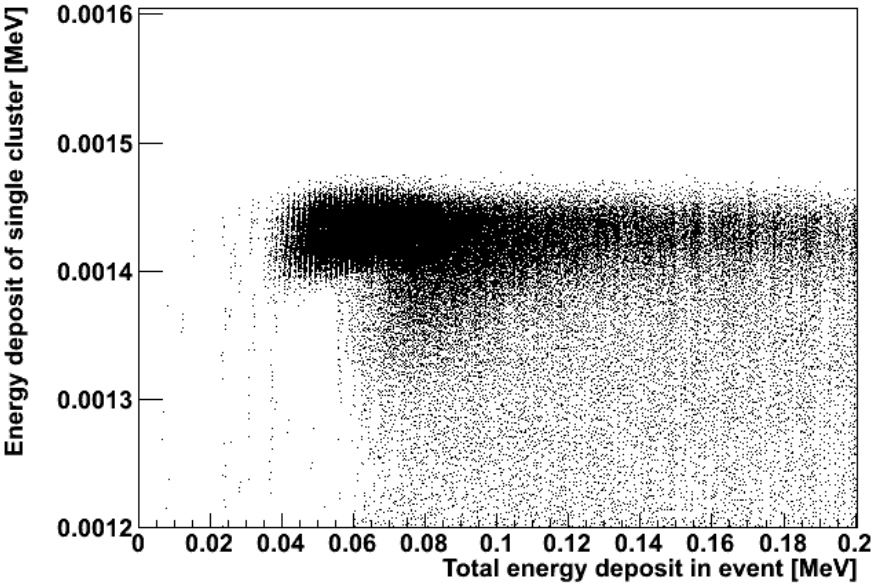
3.1.2 Comparison of Geant4 physics models

As stated above, setting up all the processes applying to all particles is a tedious and error-prone process. The Geant4 distribution therefore come with a set of ready-made physics lists [7], and a subset of these can be easily used with `TestBeamSim`: QGSP, QGSC, and LHEP. These physics list all share the same standard electromagnetic physics, but implements different hadronic physics. In addition to these lists, `_EMW` variants of all the physics lists listed above are also available, where the electromagnetic physics has been tuned to yield a better CPU performance at the cost of slightly lower precision.

Both QGSP and QGSC use "string models" for modeling the interaction between high-energy hadrons. For low energy hadrons, they use a parameterized model. They differ in how they treat de-excitation of excited nuclei – QGSP uses a "precompound model for modeling nuclear de-excitation", while QGSC treats this with a "Chiral Invariant Phase Space" model [7]. The LHEP physics lists use



(a) Fano sampling working correctly



(b) Fano sampling not working correctly

Figure 3.3: Spread in charge cluster size as a function of total charge deposit in DUT2, QGSP physics model.

purely parameterized models for all hadronic interactions for all particles, both high- and low-energy. The Geant4 webpage recommends using QGSP or QGSC for HEP tracker simulations, and ATLAS uses QGSP_BERT [1]. QGSP_BERT is a variant of the QGSP model, which uses a "Bertini cascade" to model interaction of protons, neutrons, pions and Kaons below ≈ 10 [GeV].

A superficial comparison of the output of the different models have been made, comparing values such as the incident energy on the last telescope plane (BAT3), shown in figure 3.4, and the trigger efficiency shown in table 3.1. This indicates that for the purposes of this simulation, there are no significant differences between the models, but further studies comparing for example δ-electron production should be made to say this conclusively.

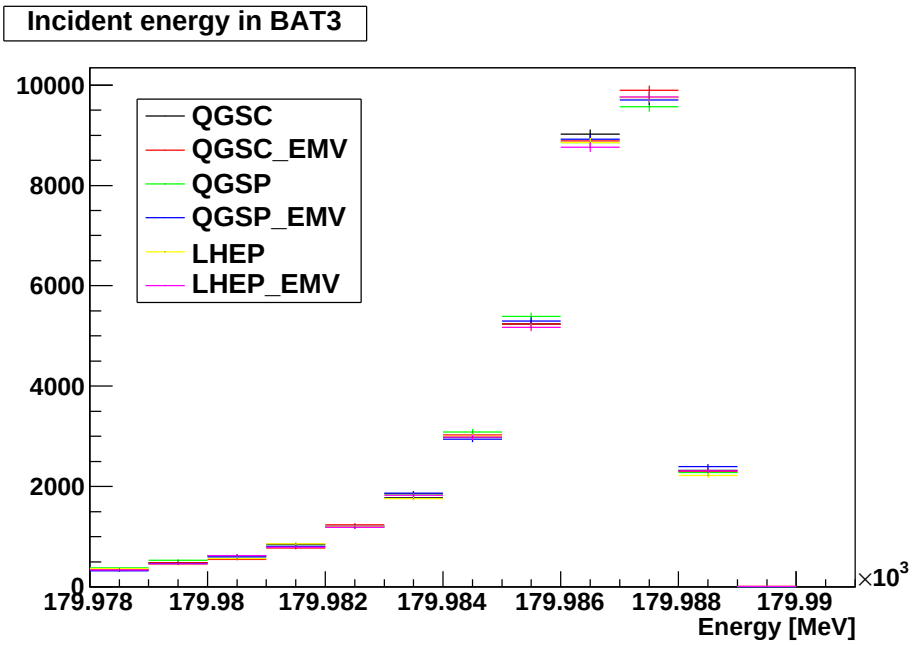


Figure 3.4: Incident energy in BAT3

A comparison of the time required to produce a single event is shown in figure 3.5. This shows that QGSP and QGSC behaves quite similarly, while LHEP models are slower. EMV variants are faster, as expected. It is surprising that LHEP is slower, as this is the opposite of what is claimed on the Geant4 webpage[3]. Note that the mean walltime per event, shown in table 3.2, is much larger than the most probable walltime indicated in figure 3.5, due to a significant fraction of events taking much more time. This benchmark was performed using `TestBeamSim` version 0.94, Geant4 version 9.3-p1 on a laptop with an Intel Core2 T7200 (2.00GHz)

[3] `http://geant4.cern.ch/support/proc_mod_catalog/physics_lists/useCases.shtml`

Model name	Trigger	Veto	Trigger and no veto
QGSC	39978	5283	34695
QGSC_EMV	39978	5456	34522
QGSP	39986	5446	34540
QGSP_EMV	39988	5420	34568
LHEP	39985	5820	34165
LHEP_EMV	39986	5897	34089

Table 3.1: Number of events where the two trigger scintillators fired, the veto scintillator fired, and where the two triggers fired but not the veto (good events), using different physics models. All from runs with 40 000 simulated primary particles.

CPU, 4GB RAM, and Fedora 13 64-bit Linux. 40'000 events where generated for each physics list.

Physics List	Mean walltime per event [ms]
QGSC	79.26
QGSC_EMW	79.31
QGSP	70.29
QGSP_EMW	64.46
LHEP	71.93
LHEP_EMW	70.80

Table 3.2: Mean walltime needed to simulate one event with different physics lists

3.2 Geometry description

The geometry used for the simulation is a description of the May 2009 testbeam at the SPS H8 beamline. A short description of this experiment, including pictures and sketches, is found in Chapter 1.2. The device positions used are based on the survey report, alignment files from tracking, and the logbook. Figure 3.6 shows a computer rendering of the setup as it would look if one could remove the magnet, and look at it from the "Jura" side[4], while figure 3.7 also shows the scintillators. The figures show the same events, including beam particles and some secondaries[5].

To simplify the set-up of the tracking program, some simplifications where made in the positioning of the devices perpendicularly to the beam axis, and they are also taken to be oriented exactly perpendicularly to the beam axis. The sensor

[4]Jura side: Side of experiment closest to Jura mountains, left-hand side when standing at the beam-axis with your back against the beam window (looking down-stream).

[5] Red tracks: Negative particles. Green: Neutrals. Blue: Positive particles.

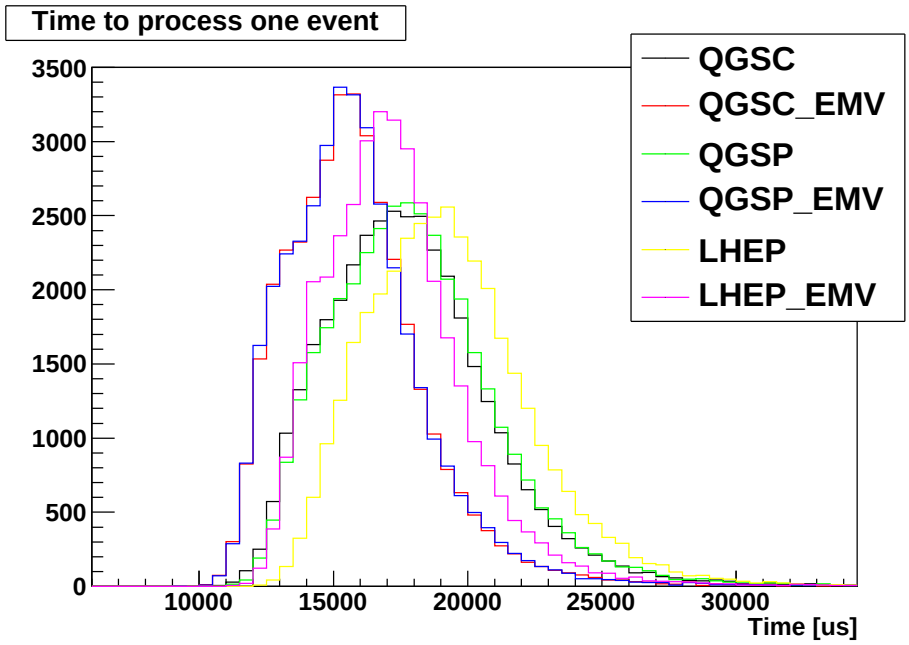


Figure 3.5: Walltime needed to simulate one event with different physics lists.

positions referred to in this section is the position of the geometric center of the sensor active material. ATLAS pixel devices are referred to as Devices Under Test (DUTs). In the horizontal direction, all DUTs are placed exactly on the beam axis. The BAT telescope planes are placed so that they are aligned with each other, 3.5 $[\mathrm{mm}]$ closer to Jura than the DUTs. In the vertical direction, all DUTs are also sitting on the beam axis, while BAT 3 and 6 is placed 6.3 $[\mathrm{mm}]$ below the beam axis. BAT 1 is placed 3.9 $[\mathrm{mm}]$ above the beam axis, making a staggered configuration with BAT 3 as shown in figure 4.1. BAT 1 is rotated $180°$ around the axis pointing towards Jura. The scintillators are placed exactly on the beam-axis. Their sizes and function is described in Chapter 3.4. The position of the devices along the beam axis is given in table 3.3, and is taken from the tracking program set-up used for experimental May 2009 data.

The orientation of BAT sensors have layer 0 (N-side) strips counting vertically and increasing upwards (for BAT 3 and 6, BAT 1 increasing downwards), while layer 1 (P-side) strips are counting horizontally, increasing towards Jura. The DUTs have 400×50 $[\mu\mathrm{m}^2]$ pixels organized in 160 rows $\times$ 18 columns. They are rotated so that the long side of the pixels are vertical, parallel to the magnetic field. This is the same orientation relative to the magnetic field as if mounted inside a large HEP tracking detector. The counting direction is then increasing horizontally towards Jura for the rows, and vertically increasing upwards for columns.

The geometry also includes a description of the mechanics of BAT planes and

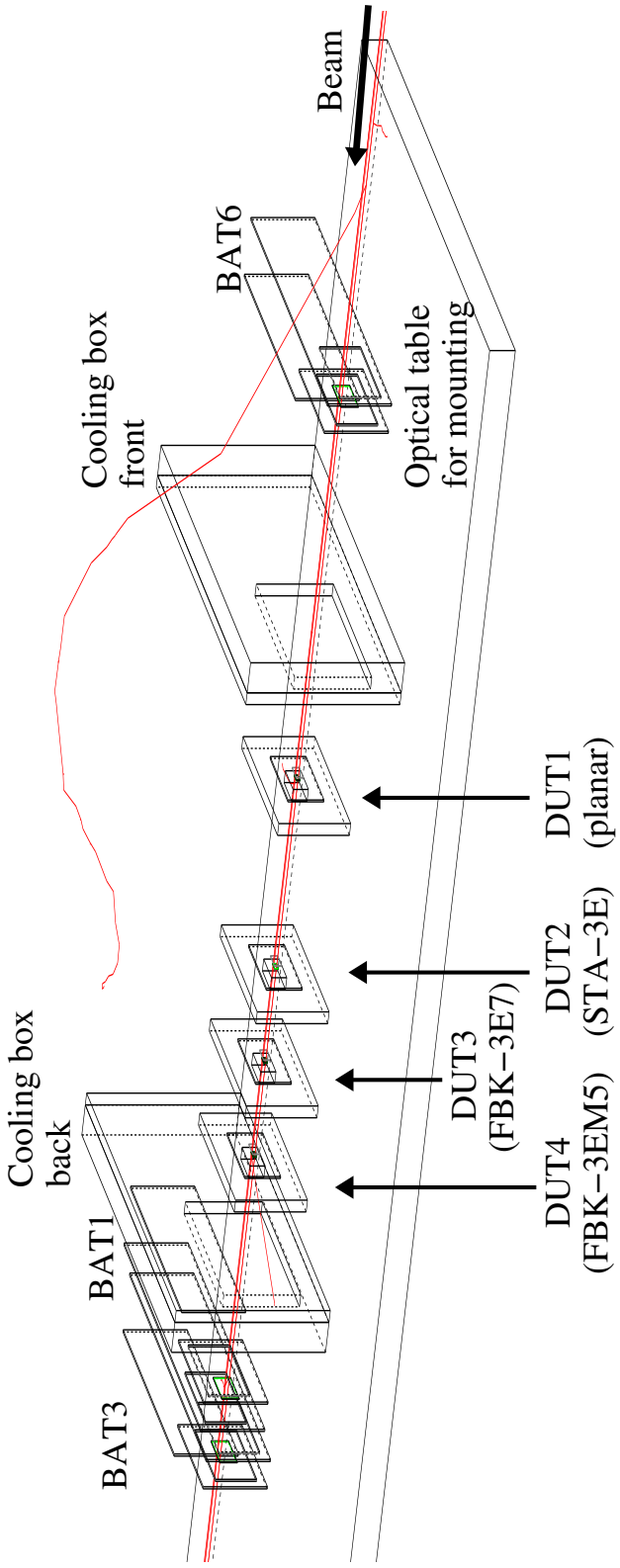


Figure 3.6: Computer rendered drawing of the simulation geometry, seen from the Jura side. Only the parts that are sitting on the optical table is shown, Morpurgo magnet suppressed, scintillators outside of view.

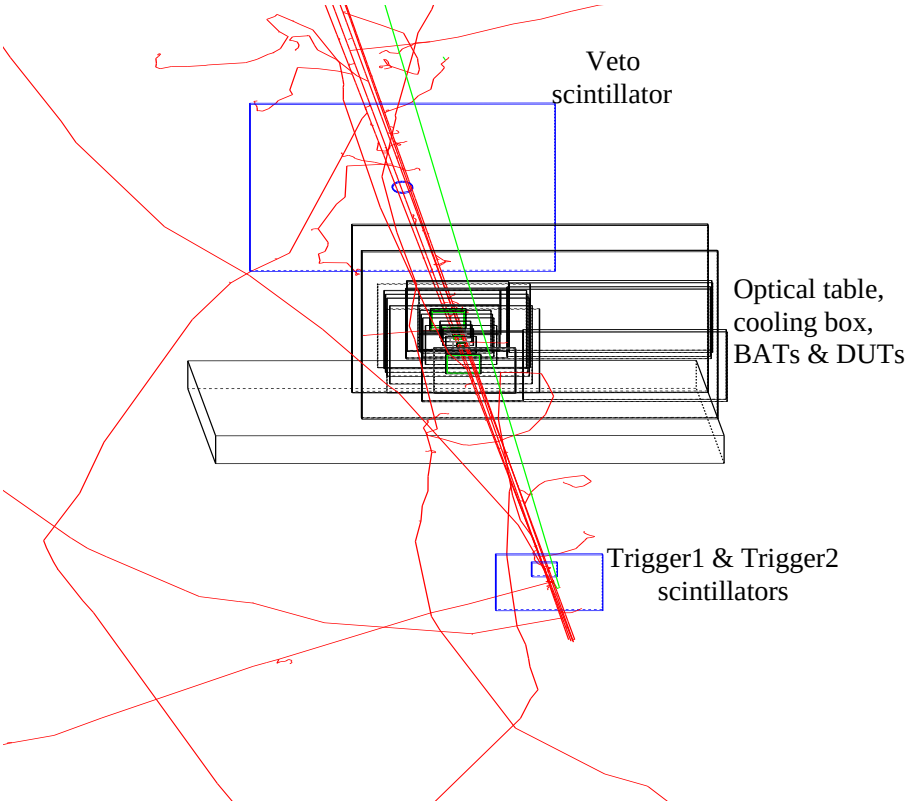


Figure 3.7: Computer rendered drawing of the simulation geometry, looking downstream and also showing scintillators. Red lines: Negative charged particles, green lines: neutrals. Both beam, scattered particles, and medium-energy secoundaries shown. Perspective compressed due to "zoom" effect in rendering program.

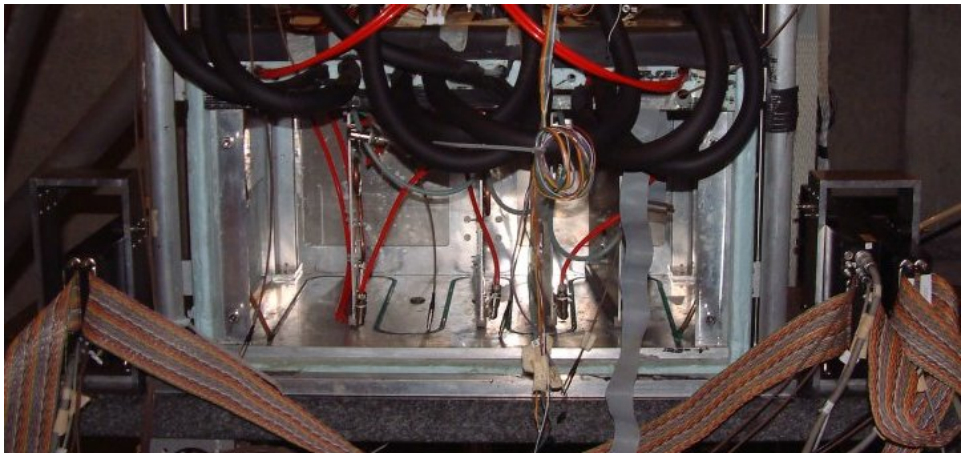


Figure 3.8: Picture of the setup on the table: BAT planes, cooling box, and DUTs.

Device name	Z-position $[\mathrm{mm}]$
Beam origin	-5000.00
Trigger1 (“Pad”)	-3770.00
Trigger2 (“Round”)	-3500.00
BAT6	868.95
DUT1 (Planar)	1195.07
DUT2 (STA-3E)	1355.09
DUT3 (FBK-3E7)	1435.10
DUT4 (FBK-3EM5)	1515.09
Veto	4500.00

Table 3.3: Position of devices along beam axis, relative to origin in the middle of the upstream opening of the Morpurgo magnet.

DUT test-boards, and is described in Chapter 4.1 and 6.1. The cooling box the DUTs are mounted in are also included in the simulation. This is built from 10 $[\mathrm{mm}]$ thick aluminum plates, and 25.5 $[\mathrm{mm}]$ thick polystyrene thermal isolation foam. The cooling box[6] geometry is seen in figure 3.6 and 3.8. There are six aluminum plates: Four sensor mounts just behind the DUT PCBs, which has a 30×30 $[\mathrm{mm}^2]$ hole for the beam, and two endplates with 150×150 $[\mathrm{mm}^2]$ holes. The sizes of these holes are guessed from pictures, as there was no exact measurements or technical drawings available. The thermal insulation foam is placed on the outside faces the endplates, but its density is in the simulation set to 10 $[\mathrm{g/m}^3]$, which is probably to low to affect anything – a more realistic value would be 10 $[\mathrm{kg/m}^3]$.

3.3 Beam

The initial state (position and momentum direction) of the primary particles in the simulation is sampled from the probability distribution functions of equations (3.6) and (3.7):

$$P(\phi) = \frac{1}{2\pi}\,, \quad P(R) = \begin{cases} \mathcal{N}(0, \sigma_R^2) & \text{for } R \leq R_c \\ 0 & \text{for } R > R_c \end{cases}\,, \quad P(z) = \delta(z - z_0) \tag{3.6}$$

$$P(\varphi) = \frac{1}{2\pi} \qquad \text{and} \qquad P(\theta) = \begin{cases} \mathcal{N}(0, \sigma_\theta^2) & \text{for } \theta \leq \theta_c \\ 0 & \text{for } \theta > \theta_c \end{cases} \tag{3.7}$$

Here the stochastic variables ϕ, R, and z describes the initial position of the particle in cylindrical coordinates. This is distributed on a disk with radius R_c perpendicular to the beam direction, with the angular coordinate distributed from a flat

[6] In the May 2009 testbeam, it was only used for mechanical mounting of the sensors, without cooling.

distribution, and the radial coordinate distributed from a truncated Gaussian distribution. For the momentum direction, this is described by the stochastic variables φ and θ. The angular coordinate φ is distributed from a flat distribution, while the azimuth angle θ, describing the angle of the emitted particle relatively to the beam axis, is distributed from a Gaussian distribution truncated at θ_c.

This seems to be a reasonable approximation, as the experimental data shows that the beam is approximately Gaussian in both profile and angle (see figure 3.9). The "stripes" in the beam profile plot is due to noisy strips in n-side of BAT 3, seen in the occupancy plot. These strips gets masked out by the reconstruction software, reducing the acceptance of particles that can be tracked.

Comparing the experimental reconstructed beam profile and angle in DUT 1 (figure 3.9) to the same from simulation (figure 3.10), we see that they are different. As expected, the simulation is symmetric between the horizontal and vertical directions. Further, as the beam shape parameters σ and cuttoff in the simulation is chosen so that $\sigma \gg$ cutoff (see table 3.4), the true beam profile and angle distributions are fairly flat, while the reconstructed distributions are not. This indicates that most of the "modulation" of the distributions seen in figure 3.10 is due to experiment acceptance, which is mostly controlled by the trigger scintillator setup discussed in Chapter 3.4.

The effect of trigger acceptance includes the structures seen in the correlation plots, showing a negative correlation corresponding to particles starting out far right needing to have a trajectory pointing leftwards in order to hit the hole in the veto scintillator and vice versa. In the simulation data this is a pure projection effect, as there are no correlations in the initial state. For the real data it is quite probable that there is a real correlation, due to the focusing of the beam. Implementing this in the simulation should lead to more realistic beamprofiles. The simulation correlation plot also shows that the position acceptance is wider (in the order of the veto hole diameter) for small-angle tracks, which is another projection effect.

The spread of beam angles seen in the simulation is an order of magnitude larger than in the real data, but are still very small, going up to approximately $0.06°$. Having to much beam spread is bad when trying to resolve electrode charge collection efficiency in 3D sensors, as discussed in Chapter 6.2.2, but this is still so low that it should not be very noticeable.

The cutoff parameters are mainly chosen to correspond to the acceptance of the trigger system; the angular cutoff is $\theta_c = \frac{\text{Radius hole in veto}}{\text{Distance beam origin - veto}} = 10^{-3}$, while the radial cutoff chosen is $R_c = \frac{\text{Radius hole in veto}}{2} = 0.5\ [\text{cm}]$. Looking at the plots in figures 3.9 and 3.10 (which was not available before the whole simulation/analysis chain was up and running), it seems reasonable that increasing the radial cutoff slightly could be a good idea. This would reduce the ratio $\frac{\text{accepted triggers}}{\text{number of simulated particles}}$ (shown in table 3.1 for current settings), which would increase the computational expense in the Geant4-part of the simulation per accepted track. The resulting beamprofile after changing the cutoffs would then indicate if σ should also be

changed.

The energy of the beam particles is fixed at 180 [GeV], and the particles are taken to be negative pions, which are charged hadrons. It is thus assumed that the beam is monochromatic; but the absolute beam energy should not matter very much as long as the beam remains relatively "stiff" with respect to scatterings, and the particles are well into the flat part of the Bethe-Block curve (see Chapter 2.1, figure 2.2).

3.4 Trigger setup

The trigger setup used in both the simulation and the real experimental setup utilizes three scintillators: Two "trigger" scintillators in front of the silicon detectors, and one "veto" downstream of the rest of the setup. In the real experiment, scintillators generate light when ionized by a passing particle, and this light is picked up by the photomultipliers attached to them, generating an electrical signal. This signal is then amplified and brought back to the electronics rack of the experiment, where it is fed into NIM modules containing discriminator/shaper circuits. These circuits sort out low-amplitude signals due to noise etc., and the output is a logic signal which is active for a setable time. The output from the discriminator is then fed through delay modules (extra cable length), and into a coincidence unit. Here the signal from the three scintillators are combined, deciding whether to send a *trigger* signal to the DAQ system to record an event. A trigger signal is sent to the DAQ if both trigger scintillators are simultaneously active, while the veto remains inactive.

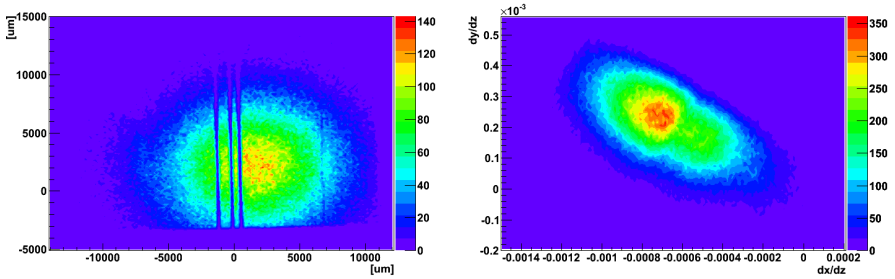
The trigger scintillators are used pairwise in order to minimize the probability of "fake" triggers from electronic noise, cosmics etc. The setup used is one large "pad" scintillator with a volume of $10.5 \times 10.0 \times 2.5$ [cm^3][7] (Trigger1), and then a smaller "finger"[8] with a volume of 2.5^3 [cm^3] (Trigger2). By adjusting how the "finger" overlaps with the "pad", the experiment acceptance can be shaped so that most of the triggers will be particles intercepting the DUTs. This is important, as recording an event takes a small amount of time in which the equipment cannot detect new particles, which means that "junk" triggers should be avoided.

The veto scintillator, placed behind the rest of the experiment, is a large ($30 \times 30 \times 2.5$ [cm^3]) scintillator with a cylindrical hole (radius 1 [cm][9]) in the middle. Beam particles with a trajectory close to parallel to the beam axis passes through this hole, while particles with a trajectory further from the beam axis miss the hole and fire the scintillator. The scintillator will also most likely be fired if the particle makes a hard interaction with the material, producing a shower or scattered at a

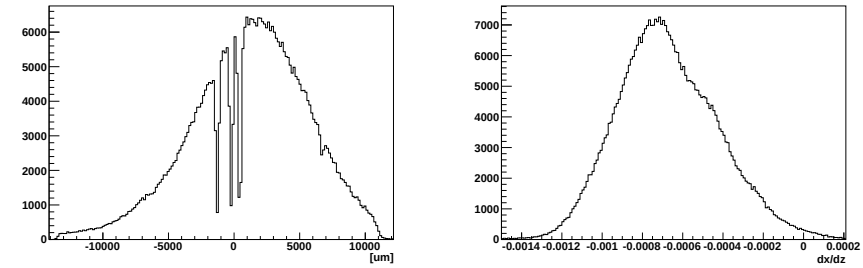
[7] Scintillator sizes from the logbook – probably not 100% accurate. 2.5 [cm] is the dimension parallel to the beam.

[8] Sometimes referred to as "round" in the logbook due to casing/light protection shape.

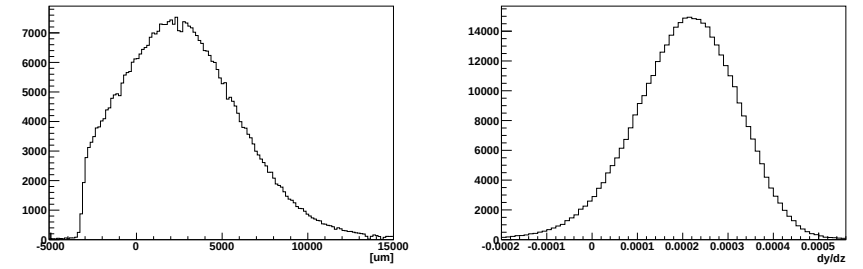
[9] The hole radius is a guess, as it was missing from the logbook. A paper currently being written states that it has a "15 [mm] hole".



(a) Beam profile (b) Beam angle

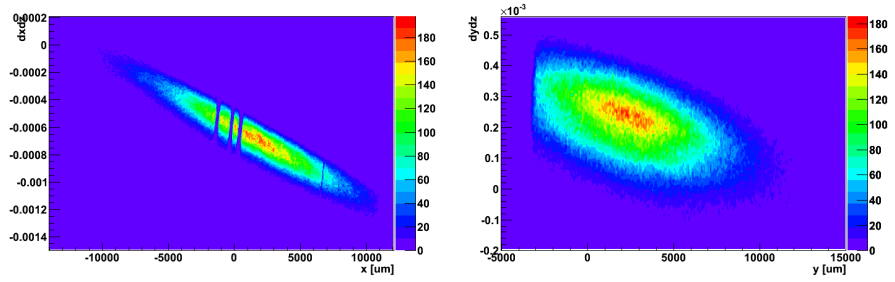


(c) Beam profile (Projection onto vertical axis) (d) Beam angle (Projection onto vertical axis)



(e) Beam profile
(Projection onto horizontal axis)

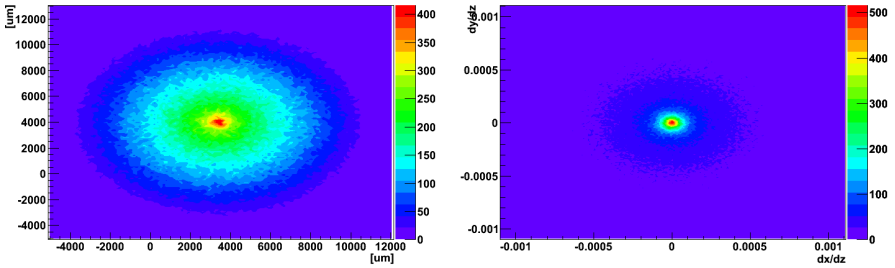
(f) Beam angle
(Projection onto horizontal axis)



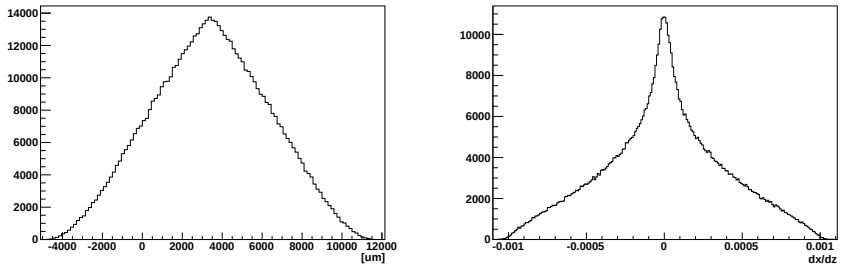
(g) Correlation beam angle versus position (vertical)

(h) Correlation beam angle versus position (horizontal)

Figure 3.9: Experimental beam profile and track angle extrapolated into the plane of DUT1. Data from 2nd run with no field, and normal incidence at May 2009 testbeam. Origin at center of pixel (0,0). “Horizontal" (Y) and “vertical" (X) is referring to direction of gravity. Tracks accepted by tracking program and with $\chi^2 < 15.0$ are included.

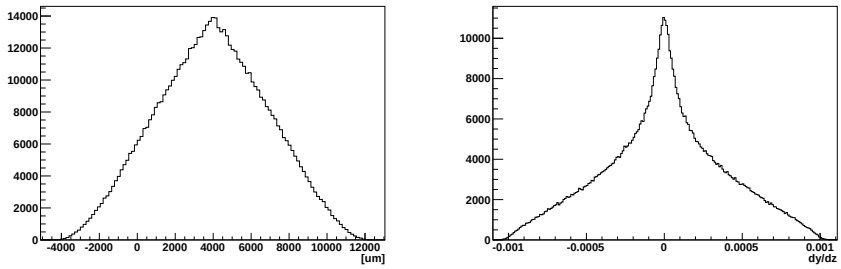


(a) Beam profile (b) Beam angle



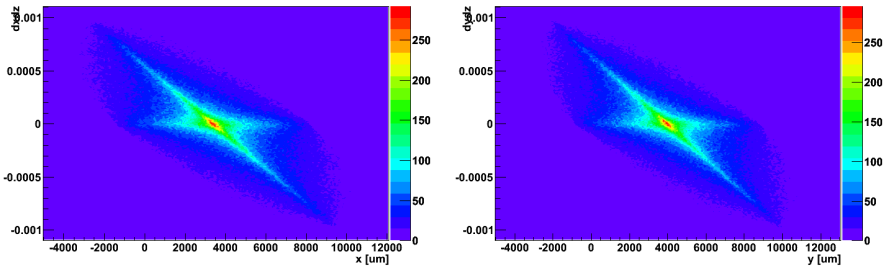
(c) Beam profile
(Projection along vertical axis)

(d) Beam angle
(Projection along vertical axis)



(e) Beam profile
(Projection along horizontal axis)

(f) Beam angle
(Projection along horizontal axis)



(g) Correlation beam angle versus position (vertical)

(h) Correlation beam angle versus position (horizontal)

Figure 3.10: Simulated beam profile and track angle extrapolated into the plane of DUT1. Normal incidence. Origin at center of pixel (0,0). “Horizontal"(Y) and “vertical"(X) is referring to direction of gravity. Tracks accepted by tracking program and with $\chi^2 < 15.0$ are included.

large angle.

The roles of different scintillators are therefore to provide trigger signal to the experiments DAQ after something interesting have happened, where interesting is defined as a "well-behaving" particle that passes through the DUTs, and have a small angle to the beam axis.

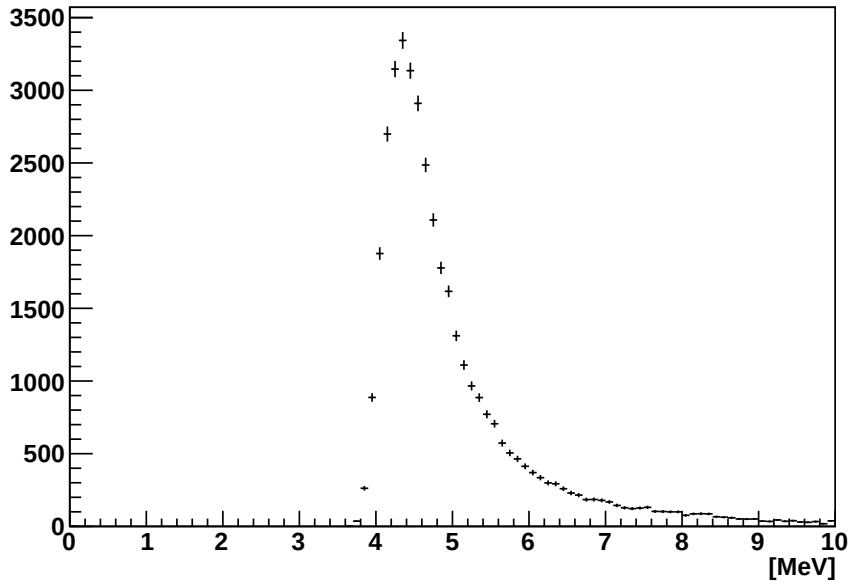


Figure 3.11: Energy deposit in Trigger1 scintillator

The simulation assumes the scintillators to be made of plastic. Further, a simplified model for scintillator response is used, only examining the energy deposition in the scintillator block. If this is above a (fairly arbitrary) threshold of 1.0 $[\mathrm{MeV}]$, the scintillator is assumed to be active. For simulating a high-energy hadron testbeam this is enough, as the particle will either hit the scintillator and produce a large energy deposit, or miss it completely (see figure 3.11). However, if making a simulation using relatively low-energy particles, such as simulating energy deposition from source-testing using β-radiation, this threshold can be important. This is shown in figure 3.12. Here selecting a too high trigger threshold results in only particles which have deposited a low amount of energy in the rest of the setup being able to reach the scintillator with enough energy to produce a trigger.

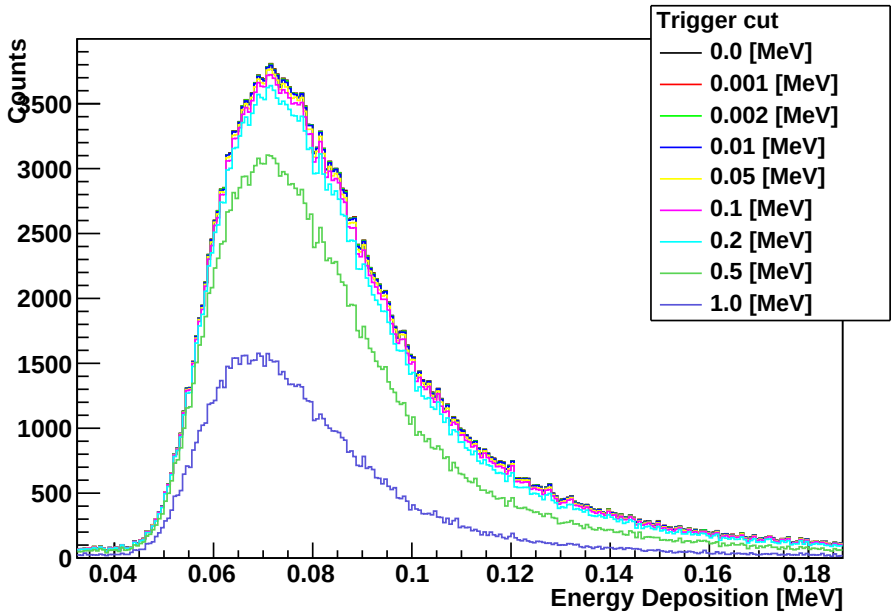


Figure 3.12: Simulation of energy deposition spectra in a 250 $[\mu\mathrm{m}]$ thick silicon sensor, exposed to β radiation from a ^{90}Sr source placed 15 $[\mathrm{mm}]$ above the sensor. Sensor "mounted" on a test board such as shown in figure 6.2, with a 2^3 $[\mathrm{cm}^3]$ plastic scintillator directly underneath the PCB, and everything in air. Different curves shows how the data looks with different trigger scintillator energy deposit thresholds applied. Produced with program `EdepSpectra`, which is also based on Geant4, and can be found at `svn+ssh://svn.cern.ch/reps/atlas3dpix/simulation/EdepSpectra`. Physics models are standard electromagnetic, QGSC, and radioactive decay, used without any special tuning.

Parameter	Value		Comment
detectorConstructionID	13		May2009 "Alligned", with detectorModules v2
particleSourceID	1		Gaussian beam
beam_energy	180	[GeV]	
beam_primaries	π^-		
beam_Rsigma	5	[cm]	
beam_Rcut	0.5	[cm]	
beam_ThetaSigma	1	[radian]	
beam_ThetaCut	$1 \cdot 10^{-3}$	[radians]	Approximate veto scintillator acceptance
physicsSetupID	13		QGSP physics list
physicsCut	5.0	[μm]	
clusterMaxCharge	400	electrons	
truthWritingCut	10	[GeV]	

Table 3.4: TestBeamSim reference parameters

3.5 Chosen simulation parameters

As discussed above, there are several parameters in the simulation that can be varied. The final values chosen for the parameters is shown in table 3.4. Some of these parameters, such as "detectorConstructionID" can be used to select between completely different simulation geometries, while others control the beam shape or physics used. Still others have no effect on the simulation itself, but can be used for deciding what particles should produce truth output when traversing the sensor ("truthWritingCut"). Many of these parameters (and more) are setable through the class `TestBeamSimMessenger`, which is described in Appendix A.4.1.

The reasons for selecting the beam parameters described in Chapter 3.3. The physics model "QGSP" was chosen as this list is one of the recommended lists for tracker simulation, is used by ATLAS and should thus have been a through validation. The physics production cut of 5.0 [μm] was chosen as a compromise between computational cost and accuracy, and it should also be noted that the normal Geant4 physics models are not accurate at very low energies[10] anyway. The choice of clusterMaxCharge, the maximum mean charge-cluster size as defined by m in equation (3.4), equal to 400 electrons is also a similar tradeoff between simulation accuracy and computational cost. The truth writing cut is set so high that it will exclude δ electrons etc., while keeping MIP(s).

[10]There are special low-energy models available.

3.6 Conclusion

In this chapter a full simulation system based on Geant4 for test beam experiments have been presented. This was applied to the May 2009 SPS H8 3D pixel testbeam, and simulation results compared to the experimental data.

A method for sampling charge clusters along the path of the simulated particles, taking the Fano factor into account is presented in Chapter 3.1.1.

The simulated beam and trigger scintillator setup is presented, and results compared to experimental data. This showed that for the fairly flat beam profile simulated, the measured profile is mostly controlled by experiment acceptance.

It is also shown that for low-energy particles, such as electrons from a β-source used in a source test, the scintillator sensitivity can significantly change the measured energy deposition spectrum.

The output of the charge cluster sampling in this simulation is used as input to the device simulation models presented in Chapter 4 and 6.

Chapter 4

BAT telescope model

As described in Chapter 1.2, the Bonn Atlas Telescope (BAT) [38, 39] is used for tracking charged particles in the testbeams that has been simulated. It is therefore necessary to provide a good model of its response to MIP radiation in order to be able to compare simulated pixel devices with real data, as the data is made using this hardware.

The BAT telescope is composed of three modules/planes, as shown in figure 4.1. Each of this planes contains a Hamamatsu S6934 double-sided strip sensor with a pitch of $50\ [\mu\mathrm{m}]$ and a sensitive area of $32\ [\mathrm{mm}] \times 32\ [\mathrm{mm}] \times 300\ [\mu\mathrm{m}]$ [15]. There are also electronics for providing power, readout of the sensor, data transfer to the DAQ system, and a data preprocessor FPGA performing zero-suppression and common-mode estimation.

Figure 4.1: Picture of two BAT planes, as mounted in the May 2009 testbeam

To make a simulation of a testbeam, a model of the beam particles interacting

with the planes (as described in Chapter 3), as well as how this creates a signal in the sensors, and how the electronics respond to this, is necessary. This Chapter describes such a model, including tuning of parameters in order to reproduce the data.

4.1 Simulation geometry

For modeling the beam interaction with the telescope planes, describing their geometry to Geant is necessary. The simulation geometry used is shown in figure 4.2, and a computer rendered drawing in figure A.1(a). The geometry is partly based on measurements directly on the planes, and for the internals, figure 3.14 from [39].

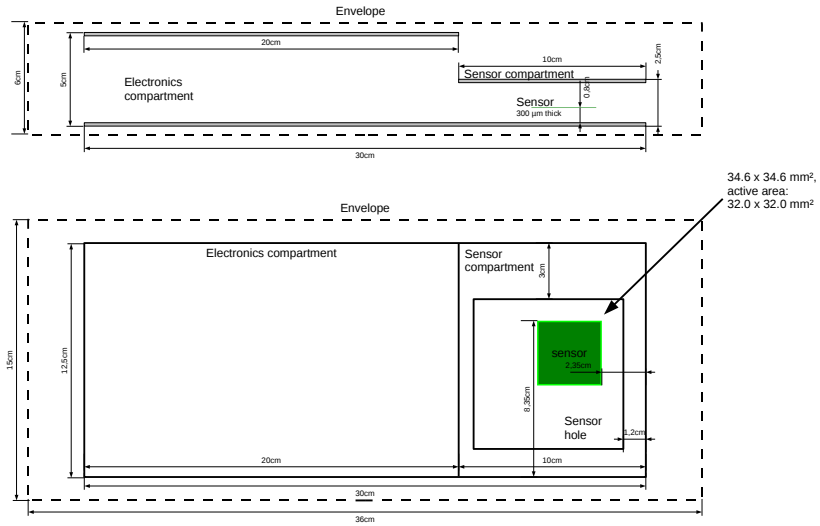


Figure 4.2: BAT simulation geometry

In addition to what is shown in figure 4.2, there is a $100\,[\mu\mathrm{m}]$ thick kapton foil covering the windows in the sensor compartments. The aluminum plates are $1.8\,[\mathrm{mm}]$ thick. As many parts of the modules are not directly in the beam, only some of it has been included in the geometry model as an illustration. This is partly done in order to simplify the model, and also because no good references on the module/hybrid mechanics was available, meaning that implementing more details of the module geometry would be dependent on a lot of guesswork. The "envelope" of air around the module is there for technical reasons – it was the original intention to run the simulation with increased resolution (shorter physics production cut-off for δ-electrons etc.) within this envelope than used elsewhere. The envelope also simplifies describing the module to Geant4, as the rest of the volumes are daughter volumes to the envelope. However it does limit how close it can be placed to other

material objects, as volumes cannot overlap[1]. This geometry is implemented as the detector module `TestBeamSimDetectorModule_BAT_v2` in `TestBeamSim`, as described in Chapter A.3.1.

In order to extract useful data for the simulation, the sensitive part of the sensor has a `TestBeamSimPixelSD` Geant4 sensitive detector attached to it, described in Chapter A.4.5. This provides a set of simulated energy depositions, and their positions locally in the sensor volume.

4.2 Simulation of sensor response

To do anything useful with the simulated energy depositions from `TestBeamSim`, it is necessary to have a *digitization* algorithm. This algorithm converts the simulation hits into detector digits, formatted the same way as the output of the real telescope system, as shown in figure 4.4.

This is done by simulating the charge transport and collection in the semiconductor, the electronic noise, and how the data preprocessing FPGA handles the digitized input. This is implemented in a series of steps shown in the data flow diagram of figure 4.3. Each of these steps is described in the subsections below, and together they form a model for a BAT sensor.

4.2.1 Charge-sharing and signal generation

The input to the digitizer from `TestBeamSim` is a set of energy-depositions, and their position in the sensor volume. These energy depositions can be assumed to create localized charge-clouds of electrons and holes, which is then transported by the electric field towards the implants. While in transport, these charge-clouds will also diffuse laterally, and become larger and less concentrated until collected at the implants (see figure 4.6). The implants are then capacitively coupled to the input of the amplifiers. The goal of this part of the digitization algorithm is to calculate how much charge is delivered to each amplifier channel.

In order to make the model manageable, certain simplifications has been made:

- Electrostatic repulsion between charges in the same cloud, and between different clouds ignored.
- Speed of charges to low for velocity saturation, as max field $0.6[\frac{\mathrm{kV}}{\mathrm{cm}}]$ with parameters as in table 4.2.
- Recombination is negligible, as drift time is on the order of $[\mathrm{ns}]$ (see chapter 4.2.1.2), while minority carrier lifetime in silicon is on the order of $[\mu\mathrm{s}]$.

The method used is similar to the one developed in the article "Charge collection in silicon strip detectors" by Belau et al. [3].

[1] Volumes cannot overlap unless they have a mother/daugther relationship. The overlap must then be complete – the mother must *contain* the daughter.

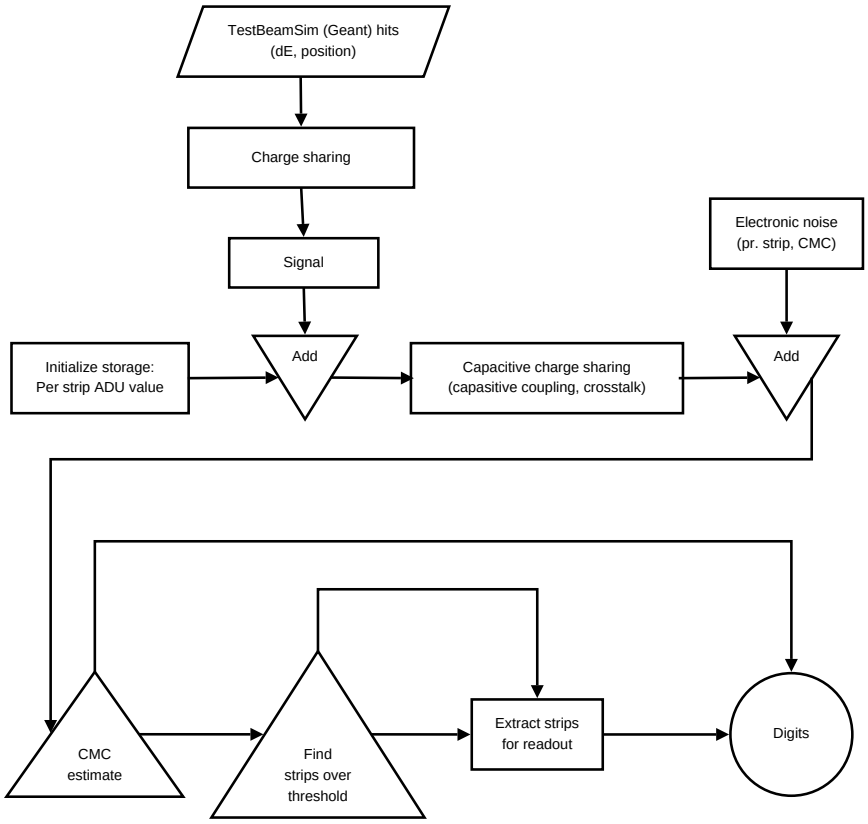


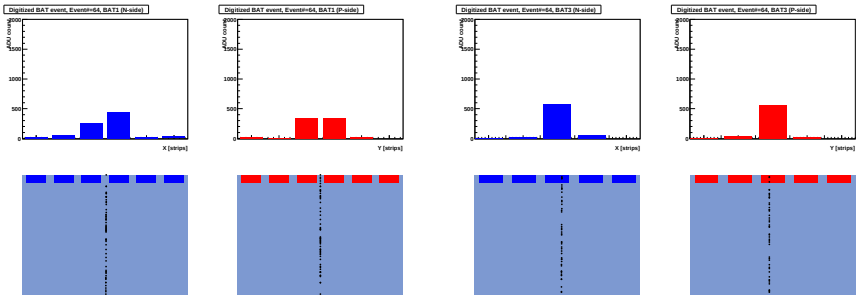
Figure 4.3: Main steps of BAT digitization algorithm. Arrows symbolize data flow, rectangles process steps, parallelograms external data input, downward-pointing triangles merging of data, upward-pointing triangles extraction of information, and circle a connection to another program.

In order to know much charge is collected on each implant, it is necessary to know the shape of the charge-cloud when it is collected. Thus the the drift and diffusion equations for electron and hole charge-clouds have to be solved, which yields the shape of the cloud as a function of time. This is dependent on the electric field, which has to be found first. The charge-clouds themselves are sampled along the path of the particle using a "Fano sampling" algorithm described in Chapter 3.1.1.

4.2.1.1 Electric field

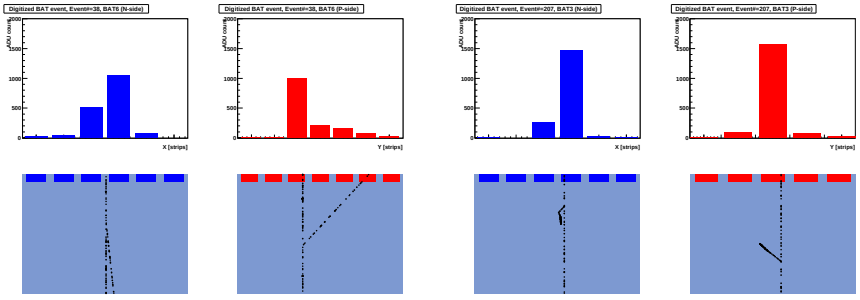
BAT utilizes Hamamatsu S6934 silicon planar strip sensors [15]. By assuming that its bulk has uniform n-type[2] doping, that it is reversed biased beyond its full

[2]N-type sensors are the most common, and as shown in chapter 4.4 the model fits the data quite well.



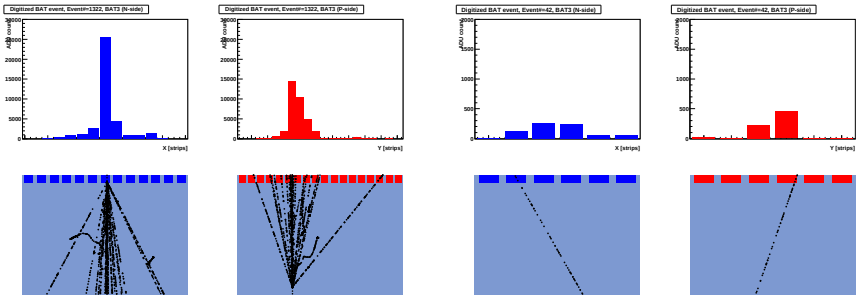
(a) Typical event – straight track between two strips, lots of sharing.

(b) Typical event – straight track near center of strip, low amount of sharing



(c) Event with δ-electron that escapes the sensor. Note that the correct hit position will be hard to reconstruct.

(d) Event with δ-electron that is stopped in the sensor. Note that the energy loss increases towards the tip of the track.



(e) Event where hadronic shower starts inside the sensor silicon. Note different scale on y-axis (max ADC range is really $2^{12} = 4096$. In the simulation, this truncation is done when writing to file). This is a quite uncommon type of event (examined 1400 events $\times$ 3 planes, found two).

(f) Event with non-perpendicular track

Figure 4.4: Gallery of plots showing energy simulated depositions and simulated detector response. Each subfigure contains four elements: Leftmost elements is showing the N-side, while the rightmost is showing the P-side. Topmost elements is showing the simulated sensor response (including noise), while elements at the bottom is showing the position of the energy deposits, projected so that strips are in the direction perpendicular to the paper. The bottom element's x-axis are scaled to show all strips above threshold, and are thus not to scale with the y-axis which always spans the sensor thickness. All data from simulation run 613, digitizer tuning as in table 4.2. Plots inspired by [34].

depletion voltage, and neglecting the areas close to the implants where the field is focused towards the implants[3], it can be assumed that the electric field is uniform and directed transversely into the sensor plane, as shown in figure 4.5.

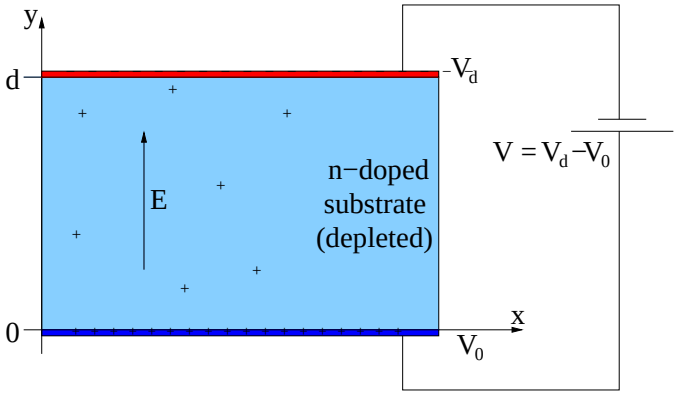


Figure 4.5: Geometry used for calculation of electric field, drift time etc. in BAT. P-type Si (top) in red, n-type in blue (bulk and bottom).

By Gauss's law, the electric field in a biased sensor with thickness d and charge-density ρ due to a concentration N_d of donor atoms, each carrying one elementary charge q, can be calculated:

$$\begin{aligned}\frac{\mathrm{d}E}{\mathrm{d}y} &= \frac{\rho}{\epsilon} = \frac{qN_d}{\epsilon} \\ \Rightarrow \int_{E_0}^{E(y)} \mathrm{d}E &= \int_0^y \frac{qN_d}{\epsilon}\,\mathrm{d}y = E(y) - E_0 = \frac{qN_d}{\epsilon}\cdot y \\ \Rightarrow E(y) &= \frac{qN_d}{\epsilon}\cdot y + E_0\end{aligned}$$

For a fully depleted sensor, the electric field $E_0 = E(y = 0)$ given a reverse bias $V \geq V_d$ can be determined by the definition of electric potential:

$$\begin{aligned}-\frac{\mathrm{d}V}{\mathrm{d}y} &= E(y) = \frac{qN_d}{\epsilon}\cdot y + E_0 \\ \Rightarrow -\int_{V_0}^{V_d} \mathrm{d}V = V &= \int_0^d \frac{qN_d}{\epsilon}\cdot y + E_0\,\mathrm{d}y = \frac{qN_d}{2\epsilon}\cdot d^2 + E_0\cdot d \\ \Rightarrow E_0 &= \frac{V}{d} - \frac{qN_d d}{2\epsilon}\end{aligned}$$

Thus the electric field can be written as:

$$E(y) = \frac{qN_d}{\epsilon}\cdot y + \frac{V}{d} - \frac{qN_d d}{2\epsilon} \tag{4.1}$$

[3]This will be treated by introducing a cutoff when transporting the charges, see section 4.2.1.2

This can be simplified by relating it to the depletion layer thickness W of a p^+n-junction and the depletion voltage V_d, and assuming full depletion [35]:

$$W = \sqrt{\frac{2\epsilon V_j}{q}\left(\frac{1}{N_d} + \frac{1}{N_a}\right)} \approx \sqrt{\frac{2\epsilon V_d}{qN_d}} \approx d \Rightarrow \frac{qN_d}{\epsilon} = \frac{2V_d}{d^2}$$

Here the junction voltage V_j is given by $V_j = V_c - V_{\text{extern}}$, where the applied voltage $V_{\text{extern}} = -V_d$, $V_d \gg V_c$, with V_c the contact potential of the junction. Inserting this into equation (4.1) yields:

$$E(y) = \frac{2V_d}{d^2} \cdot y + \frac{V - V_d}{d} \tag{4.2}$$

This is a more useful form, as the sensor thickness $d = 300\,[\mu\text{m}]$ and depletion voltage $V_d = 80\,[\text{V}]$ is listed in the sensor datasheet [15]. The bias voltage setting is however not documented anywhere, but should be somewhat larger than the depletion voltage. The bias voltage V is therefore a tunable parameter.

4.2.1.2 Drift and diffusion of charge-clouds

Knowing the electric field affecting the charge clouds, their shape at the implants where they will be collected can be calculated. Thus first the solution of the drift/diffusion equations for the charge-clouds is necessary, which can be used to calculate how much is collected on each electrode. An outline of this is shown in figures 4.6 and 4.8.

As the signal measured by the *analog-to-digital converter* (ADC) is determined by the total charge collected on a strip, only the distribution of the charge cloud laterally in the sensor plane, perpendicular to the strip that will collect the charges, is of interest. Thus the distribution perpendicular the sensor plane and parallel to the collecting strips can be neglected. The drift- and diffusion equations can then be decoupled, as the electric field has no component in the sensor plane[4]. The diffusion equation also reduces to a one-dimensional problem. A workable approximation is then to first view the charge-cloud as a point-particle[5], and calculate the time it drifts from its point of creation (y_0, x_0) to where it is collected. This is then the time available for diffusion. Putting this together then yields a description of the shape of the charge-cloud arriving at the implants as function of its point of creation.

The drift time can be found by integrating the equation of motion for an electron- and hole cloud from its point of creation $y = y_0$ and to the edge of the focused-field region $y = c_n$ (electrons) or $y = d - c_p$ (holes), where the electric field gets a lateral component which "focuses" the charges towards the closes strip, so that diffusion no longer matters. This is shown in equation (4.3) for electrons, and in equation (4.4) for holes.

[4] Neglecting electrostatic iteraction between charges

[5] The charge-cloud sizes are much smaller than the typical drift length, see figure 4.7

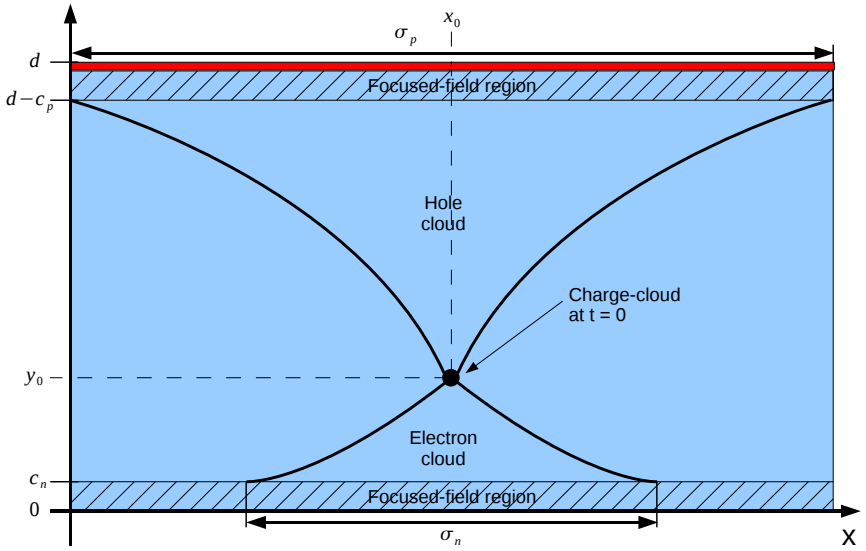


Figure 4.6: Expansion under drift of an electron/hole charge-cloud in the BAT sensor (clouds does not necessarily have the correct shape)

$$\begin{aligned} \langle v_e \rangle =& \frac{\mathrm{d}y}{\mathrm{d}t} = -\mu_n E(y) = -\mu_n \left[\frac{2V_d}{d^2} \cdot y + \frac{V - V_d}{d} \right] \\ \Rightarrow \int_0^t \frac{\mu_n}{d^2} \, \mathrm{d}t' =& - \int_{y_0}^{c_n} \frac{\mathrm{d}y}{2V_d y + d(V - V_d)} \\ \Rightarrow t_e =& \frac{d^2}{2V_d \mu_n} \log_e \left(\frac{2V_d y_0 + d(V - V_d)}{2V_d c_n + d(V - V_d)} \right) \end{aligned} \tag{4.3}$$

$$\begin{aligned} \langle v_h \rangle =& \frac{\mathrm{d}y}{\mathrm{d}t} = \mu_p E(y) = \mu_p \left[\frac{2V_d}{d^2} \cdot y + \frac{V - V_d}{d} \right] \\ \Rightarrow \int_0^{t_h} \frac{\mu_p}{d^2} \, \mathrm{d}t' =& \int_{y_0}^{d - c_p} \frac{\mathrm{d}y}{2V_d y + d(V - V_d)} \\ \Rightarrow t_h =& \frac{d^2}{2V_d \mu_p} \log_e \left(\frac{2V_d(d - c_p) + d(V - V_d)}{2V_d y_0 + d(V - V_d))} \right) \end{aligned} \tag{4.4}$$

Knowing for how long diffusion is active (t_e, t_h), the shape of the charge-cloud when arriving at the implant strips can be calculated. This is done by solving the 1D diffusion equation, as only the spread normal to the strip direction is of interest.

The diffusion equation can solved by assuming that the initial charge cloud is a delta function centered on the point (x_0, y_0), which is the point where the energy

is deposited. This yields the following demands on the charge-distribution $C(x,t)$:

$$\begin{aligned} \frac{\partial}{\partial t}C(x,t) &= D\frac{\partial^2}{\partial x^2}C(x,t) \\ \int_{-\infty}^{\infty} C(x,t)\,\mathrm{d}x &= 1 \\ C(x,0) &= \delta(x-x_0) \end{aligned} \tag{4.5}$$

Here D is the diffusion coefficient, which can be related to the carrier mobility μ, the temperature T, and Boltzman's constant k_B by the Einstein relation [35]:

$$\frac{D}{\mu} = \frac{k_B T}{q} \tag{4.6}$$

Before solving equation (4.5), a substitution of variables $x = x_0 + \xi \Rightarrow \xi = x - x_0$ is done, so that the initial condition becomes $C(\xi,0) = \delta(\xi)$. This is satisfied with equation (4.7), as a Gaussian is a Green's function of the diffusion equation.

$$C(x,t) = \frac{1}{\sqrt{4D\pi t}} e^{\frac{-\xi^2}{4Dt}} = \frac{1}{\sqrt{4D\pi t}} e^{\frac{-(x-x_0)^2}{4Dt}} \tag{4.7}$$

This Gaussian distribution describing the shape of the charge-cloud is characterized by the two parameters – the variance increase of the charge cloud during drift $\sigma^2_{\text{drift}}(t) = 2Dt$, and the mean x_0. In order to make it possible for the charge-cloud to have a finite size at $t = 0$, one can make the substitution $t \to t + t_0$. Here the relation between t_0 and the variance $\sigma_0^2 \equiv \sigma^2(t=0)$ is $\sigma_0^2 = 2Dt_0$, and thus the total variance becomes $\sigma^2(t) = 2D(t+t_0) = \sigma^2_{\text{drift}}(t) + \sigma_0^2$.

Using this, the drift time, and the Einstein relation, the variance of the Gaussian describing the electron and hole distributions as a function of charge cloud creation depth is then given by equations (4.8) and (4.9). These are also plotted in figure 4.7.

$$\sigma_e^2 = \frac{d^2 k_B T}{q V_d} \log_e\left(\frac{2V_d\ y_0 + d(V-V_d)}{2V_d\ c_n + d(V-V_d)}\right) + \sigma_0^2 \qquad \text{for} \quad y_0 > c_n \tag{4.8}$$

$$\sigma_h^2 = \frac{d^2 k_B T}{q V_d} \log_e\left(\frac{2V_d(d-c_p) + d(V+V_d)}{2V_d\ y_0 + d(V-V_d)}\right) + \sigma_0^2 \qquad \text{for} \quad y_0 < d - c_p \tag{4.9}$$

If an external magnetic field parallel to the strip direction is present, the Lorentz force would act on the clouds with a force perpendicular to the strip direction. The effect of this would be that the charge-clouds with the longest travel distance is pushed more of-course than the ones created close to the implants. A standard way of taking this into account is assuming that the charges drift through the sensor with a constant, electric field-independent Lorentz-angle θ_L with respect to the electric field [3], displacing the mean of the charge distribution $x_0 \to x_0 + \Delta y \cdot \tan\theta_L$, where Δy is the distance from the point of creation to the surface of collection. This is really only a good description of the electric field is close to constant (high *over-bias* – bias voltage much larger than depletion voltage), or if the charges quickly reach velocity saturation.

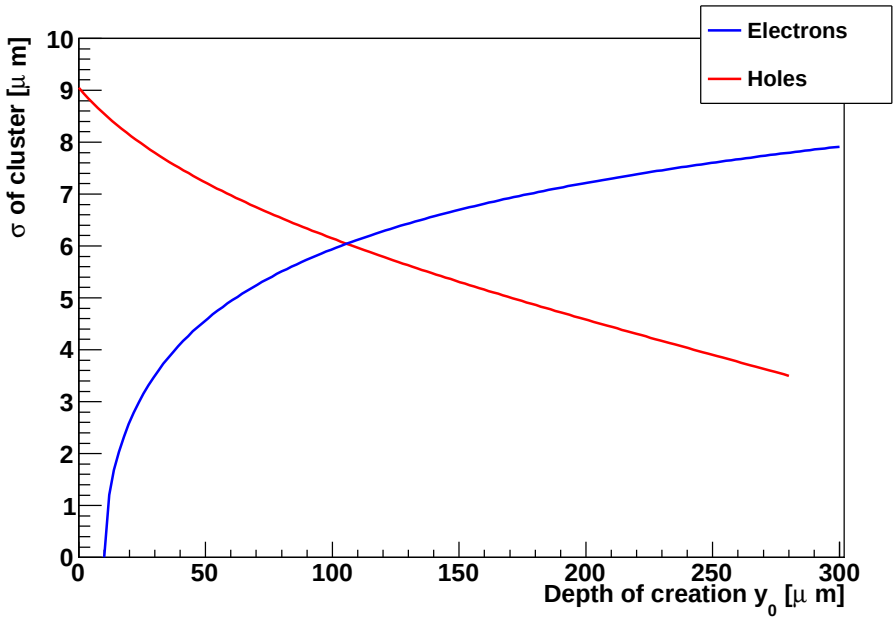


Figure 4.7: Cluster size as function of depth of creation. Plotted from equations (4.8) and (4.9). Parameters as shown in table 4.2.

Alternatively, it might be possible to drop this assumption, and instead use that the Hall-force in the strip-perpendicular direction x is given as $F_H = qv_yB_z = m^* \frac{d^2\Delta x}{d^2t}$, inserting for $v_y(t)$ and integrating for $0 < t < t_{\text{drift}}$. Here Δx is the strip-perpendicular displacement, and m^* the effective electron or hole mass. Even if this is possible, the integration probably has to be made numerically, which would make the device simulation slower and more complicated.

However, a magnetic field dependence for BAT has not been implemented in the digitization code, as the part of the simulation dealing with tracking particles and creating energy deposits (see Chapter 3) currently does not support tracking particles in the magnetic field configuration used in our experimental setup.

4.2.1.3 Charge collection on implants

Knowing the size and shape of a charge-cloud when it arrives, the amount of charge deposited on each implant can be calculated. In principle, the gaussian charge-distribution $C(x)$ extends indefinitely in both directions, but as it has a rapid fall-off the calculation is limited to the charge on the six strips closest to the deposit[6].

[6]This is about 15σ in each direction, but during parameter tuning it was found that sharing across this many strips improved the results. Number of strips to share across remains a tunable parameter in the code (restricted to positive even integers).

This is done by assuming that the charge Q_0^k deposited on strip k, centered on x_k, is given as

$$Q_0^k = \Delta \cdot K \int_{x_k-\delta/2}^{x_k+\delta/2} C(x, t_{\text{collection}})\,\mathrm{d}x \equiv \Delta \cdot K \cdot I_k \tag{4.10}$$

where Δ is the size of the charge-cloud in units of energy, K is a tunable calibration constant with units ADC unit / energy, δ is the pitch of the sensor ($50\,[\mu\mathrm{m}]$), and I_k a shorthand for the integral around x_k.

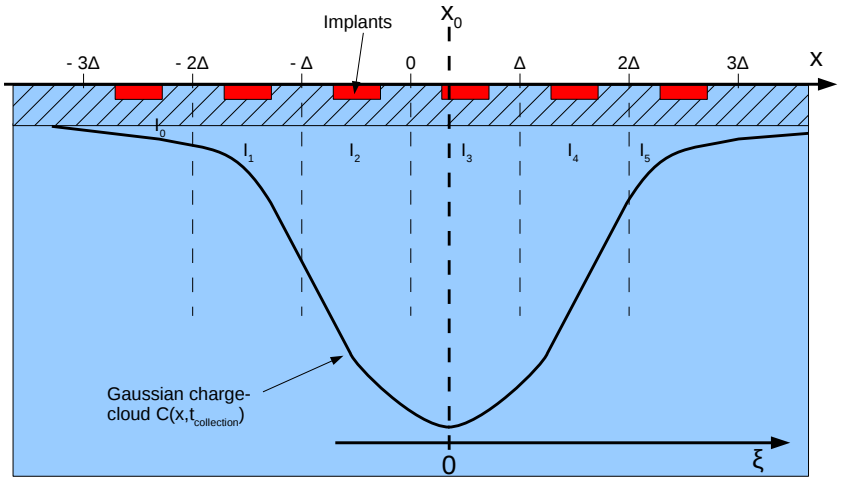


Figure 4.8: Calculated collection of a charge-cloud by 5 closest strips. Areas "under" the graph, between the limits (dashed lines) corresponding to the center-line between two implants is denoted I_k. This is the part of the cloud that is collected by the closest implant.

This integral is evaluated by exploiting that the charge distribution is gaussian, which means that its integral is given by the error function, as shown in equation (4.11).

$$\int_{-\infty}^{x} C(x')\,\mathrm{d}x' = \frac{1}{2}\left[1 + \mathrm{Erf}\left(\frac{x - x_0}{\sqrt{2\sigma^2}}\right)\right] \tag{4.11}$$

Thus the integral I_k defined by equation (4.10) for the charge collected on strip (if N strips used) is given as:

$$I_k = \begin{cases} \int_{-\infty}^{k-N/2+1} C(x)\,\mathrm{d}x - \sum_{0\le j<k} I_j & \text{for} \quad 0 < k < N-1 \\ 1 - \sum_{j=0}^{N-2} I_j & \text{for} \quad k = N-1 \end{cases} \tag{4.12}$$

4.2.1.4 Capacitive coupling from implants to metal strips

The BAT sensor is AC-coupled, which means that the amplifier is not directly connected to the implants. In an AC coupled sensor, the implants are connected to the bias voltage source, and the amplifier is connected to metal strips separated from the implants by a thin layer of insulator (silicon dioxide [15]), as shown in figure 4.9. Thus a charge Q_0^i on implant strip i has to induce a charge on the nearby metal strips by attracting opposite charges, which has to be supplied from the input of the amplifier.

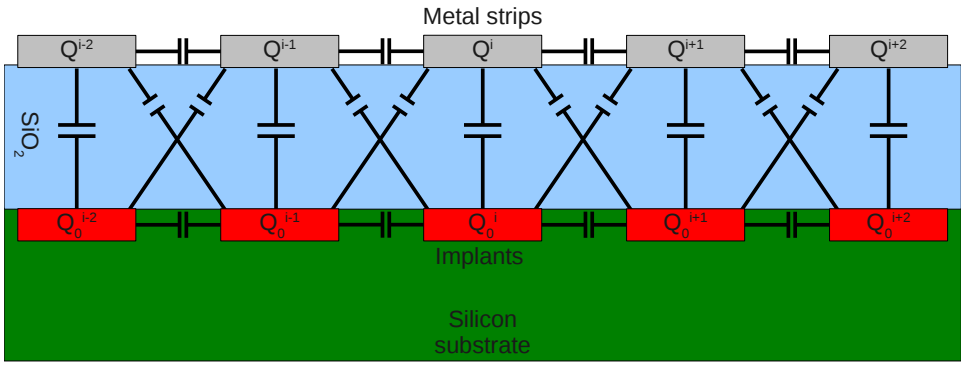


Figure 4.9: Capacitive couplings in BAT

This can be approximated by the repeating capacitor network shown in figure 4.9. If the capacitances is known, it is in principle possible to calculate the charges on the metal strips from the charges on the implants by use of Kirchoff's laws. Assuming that the charges are removed from the metal strips by the power supply much slower than the time needed by the amplifier, it can be shown that a matrix $\mathbb{C}$ transforming the the vector of implant charges $\vec{Q}_0 = (Q_0^0, Q_0^1, \ldots, Q_0^{N-1})$ into a vector of metal strip charges $\vec{Q} = (Q^0, Q^1, \ldots, Q^{N-1})$, as defined in equation (4.13), can be found.

$$\vec{Q} = \mathbb{C}\vec{Q}_0 \tag{4.13}$$

However finding this matrix for a detector with $N = 640$ strips requires finding the inverse of a 3196×3196 matrix depending on the capacitances. Inverting this matrix is certainly possible, and it only has to be done once, but as the capacitances themselves are unknown, it would add at least five more unknown parameters to the model.

A more viable way is to assume that a charge Q_0^i on implant i is only distributed across metal strips $i-2$, $i-1$, $\ldots$, $i+2$, and that the distribution is left-rigth symmetric[7]. Thus the matrix $\mathbb{C}$ can be written as shown in equation (4.14), where f_0 is the fraction of charge from implant k collected at strip k, f_1 is the fraction

[7] An asymmetry here could probably happen if the metal layer mask is shifted with respect to the implant mask. However, the data shows no evidence of significant left-right asymmetry in the signal.

collected at each strip $k+1$ and $k-1$, and f_2 the fraction collected at each strip $k+2$ and $k-2$. Conservation of charges then requires that $f_0 + 2f_1 + 2f_2 = 1 \Rightarrow f_0 = 1 - 2f_1 - 2f_2$.

$$\mathbb{C} = \begin{bmatrix} f_0+f_1+f_2 & f_1 & f_2 & 0 & & \cdots & & 0 \\ f_1+f_2 & f_0 & f_1 & f_2 & 0 & & & \vdots \\ f_2 & f_1 & f_0 & f_1 & f_2 & 0 & & \\ 0 & f_2 & f_1 & f_0 & f_1 & f_2 & 0 & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & f_2 & f_1 & f_0 & f_1 & f_2 \\ \vdots & & & 0 & f_2 & f_1 & f_0 & f_1+f_2 \\ 0 & \cdots & & & 0 & f_2 & f_1 & f_0+f_1+f_2 \end{bmatrix} \tag{4.14}$$

The input parameters to the simulation is "cShare1" and "cShare2", where $\mathrm{cShare1} = 2 \cdot f_2$ and $\mathrm{cShare2} = 2 \cdot f_2$. These are the two parameters that has to be tuned to match the real data. As the matrix has such a simple form, it is in the code implemented as a convolution rather than a matrix product.

4.2.2 Noise

In addition to the actual signal Q^i delivered to each amplifier channel, there is also noise. This both has a per-channel component, and a component that is correlated between channels ("common mode"). From the data (see figures 4.11(g) or 4.12(g) and 4.11(f) or 4.12(f)) both components are quite close to gaussian, so a usable model for the noise in channel k is

$$n_k = \mathcal{N}(0,\, \sigma_k^2) + \mathcal{N}(\mu_{\mathrm{CM}}/N,\, \sigma_{\mathrm{CM}}^2/N^2) \tag{4.15}$$

where $\mathcal{N}(\mu,\, \sigma^2)$ is a normal distribution with mean μ and variance σ^2, and N is the number of strips.

The noise n_k is then added on top of the signal Q^k, so the total signal seen by the ADC in channel k is given as:

$$s_k = Q^k + n_k \tag{4.16}$$

In the actual hardware, there is also a component of the signal that is constant between events, but varies from channel to channel, known as *pedestals*. This gives a shift in the signal to the ADC, and thus the mean of the single-channel noise is not really zero. However, the data preprocessor FPGA (discussed below) does make an attempt to subtract the pedestals from the signal. As the mean of the common mode across events is found to be different from zero, this attempt is not completely successful, but for the simulation just assuming $\mu_{\mathrm{CM}} \neq 0$ seems to work well.

Threshold setting	Threshold t_k
0	$-\infty$
1	128
2	256
3	512
4	∞

Table 4.1: Threshold and threshold setting in BAT data preprocessor

4.2.3 Data preprosessor FPGA

When a signal s_k has been created in all the channels, it is read by the ADC and a digital representation is passed on to an FPGA. This FPGA then formats the data in order to generate digits, which in presence of a trigger is written to file by the DAQ system for offline analysis. The bottom row of figure 4.3 shows a simple representation of the steps taken by the FPGA, and how this has been implemented in the model.

Many things about what this FPGA did was unknown to the current BAT user community. While making this model, documentation was rediscovered in the the PhD thesis of Johannes Treis [39], which eliminated guesswork and reverse-engineering of the FPGA output.

In order to save bandwidth on the bus connecting the BAT planes to the DAQ system, the FPGA tries to determine which strips contain a real signal from a passing particle. These strips are read out, while data from strips assumed not to contain any signal from a passing particle is discarded. The algorithm used to find which strips should be read out is a simple one – the signal (after pedestal subtraction) in all the strips is compared to a threshold t_k, which is selected with a per-channel threshold setting (see table 4.1). This means that if $s_k \geq t_k$, channel k is "marked" for readout. The strips neighbor to and next-to neighbor to a marked channel is also read out[8], in order not to loose information about the signal.

Further, a measurement of common mode noise is also made. This is done by the "Common Mode Counter" (CMC), which sums up the signal over all the channels.

$$\mathrm{CMC} \equiv \sum_{k=0}^{N-1} s_k \tag{4.17}$$

As this includes any channels that have a signal, it is a biased estimate of the common mode. This can be corrected off-line, as discussed in Chapter 4.2.3.1.

In addition to these functions, which is performed during a run, the data preprocessor also estimates the pedestal and noise of each channel at the beginning of

[8]It was previously believed that only the neighboring strips where read out, but examination of raw data from BAT showed that the number of events where number of channels read out $\in [0, 4]$ was approximately zero in the data examined. This was confirmed by looking at printouts of raw data, where a single "marked" channel was flanked by two unmarked channels on each side.

each run. The pedestal of a channel is an offset in the measured value. For most channels, the raw ADC value is in the order of a few 1000 ADC units when there are no signal, and this number (estimated for every channel) is during a run subtracted from the measured signal to get the signal s_k. The noise in each channel is also estimated as its RMS value. The results are then stored in the BAT BDT BORE (see Appendix D.3.1).

4.2.3.1 Off-line subtraction of common mode noise

It is possible to estimate the common mode noise better by only looking at the channels that are far from above-threshold-channels [39]:

$$\widehat{\mathrm{CM}} \equiv \frac{\mathrm{CMC} - \mathrm{sumADU}}{N - \text{Number of channels read out}} \tag{4.18}$$

where

$$\mathrm{sumADU} = \sum_{k \in \text{channels read out}} s_k \tag{4.19}$$

This means that it is then possible to subtract some of the noise in the detector system, by taking the corrected signal to be:

$$s'_k = s_k - \widehat{\mathrm{CM}} \tag{4.20}$$

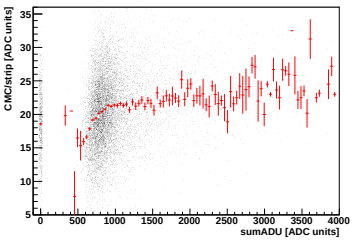
This is not done by the FPGA, but as the CMC value is stored in the data files, it is possible to do off-line. From figure 4.10 we see that this correction does seem to work, as the correlation between the signal and the noise disappears.

Figure 4.10(a) shows that the "raw" CMC is a biased estimator for the common mode, as it has a clear correlation with sumADU. By correcting the CMC by subtracting sumADU, as is done in equation 4.18, the correlation becomes weaker for events with high sumADU, as is shown in figure 4.10(b). This corrects the common mode estimate, but it still remains to correct the signal itself using equation (4.20), as is shown in figure 4.10(c).

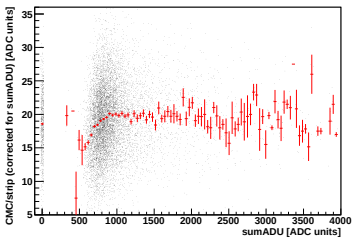
Removing the common mode noise from the signal with equation (4.20) when estimating the hit position could potentially improve the measurement resolution of the telescope planes, but has not yet been tried.

4.3 Parameter tuning

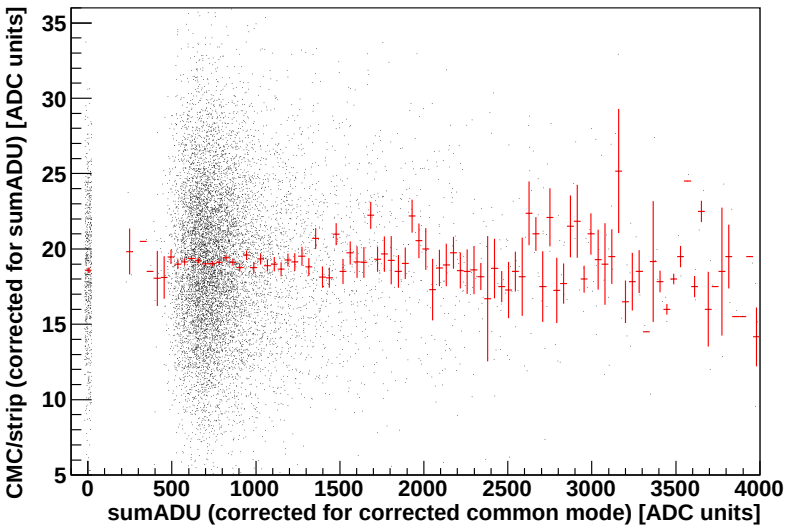
As discussed in the previous sections, the BAT model contains quite a few parameters. These are listed in table 4.2, together with the settings that was found to match the data best. Some of these parameters (such as sensor thickness and depletion voltage) are given by the sensor datasheet [15], some (such as threshold



(a) CMC/N vs. sumADU



(b) $\widehat{\mathrm{CM}}$ (eq. (4.18)) vs. sumADU



(c) $\widehat{\mathrm{CM}}$ (eq. (4.18)) vs. $\mathrm{sumADU} - \widehat{\mathrm{CM}} \times$ number of channels read out

Figure 4.10: Correlation plots between common mode estimates and signal estimates, showing how the signal can be deconvolved from the common mode noise. Data from BAT1 p-side, run705 May2009. Black: Scatterplot of common mode vs. sumADU (with different corrections applied to the estimators). Red: Profile plot of the same data.

Parameter	[Unit]	N-side	P-side
d	[μm]	300	
V_d	[Volt]	85.0	
V	[Volt]	95.0	
T	[Kelvin]	300	300
σ_0	[μm]	0.0	3.5
c	[μm]	10	10
K	[ADU/MeV]	9200	8700
cShare1	[Factor]	0.0625	0.04
cShare2	[Factor]	0.0	0.0
σ_k	[ADU]	30	18
μ_{CM}/N	[ADU]	10.0	13.5
σ_{CM}/N	[ADU]	3.908	4.6
Threshold setting		2	1
Pedestal	[ADU]	2000	2000

Table 4.2: Parameters used for BAT model

setting) is a setting from the DAQ system that is recorded in the BDT BAT BORE (see Appendix D.3.1), while others (such as per-channel noise σ_k) is estimated by the data preprocessor FPGA at the start of a run, and saved in the BDT BAT BORE. The rest has to be estimated from the data, and this is the parameter tuning process.

The estimation of parameter values, also known as *tuning*, has to be done in order to get the simulated and experimental data to match. This was done by using plots like those shown in figures 4.11 and 4.12, which compares key distributions from simulation with reference data. These plots where then produced for many sets of parameters, by the help of the program described in Appendix B.3.3. The parameters where then successively refined until a rather good match was achieved.

The reference data used for the tuning was run705 of the May 2009 testbeam, BAT1. This run was selected because it was a run with no magnetic field/no sensor tilt, and contains a decently sized data sample (10118 triggers). BAT1 was used in preference of the other planes because of its lack of noisy strips and other defects. The differences between the tune found seems to be closer to BAT1 than BAT1 is to the other planes, so even if the model does not match BAT1 perfectly, it is still representative for the response of a BAT plane.

Note that even if the histograms shown is a quite good match to the data, this does not mean that the model is correct. This was particularly revealed when plotting the no-signal CMC histogram for the P-side (see figure 4.12(g)), which showed that the mean was quite a bit off. This was not discovered during tuning, due to that this histogram was not drawn by program used for tuning.

4.3.1 Observables

As earlier mentioned, the tuning is done by examining a set of histograms. The histograms used was extracted from the real and simulated BDT data files with the help of the analysis program described in Chapter B.3.4.

Many of the histograms here refer to *clusters*. This is defined as a continuous region of three strips or more, with signal values significantly above the background, centered around a peak in strip K. The significance demand is given by that the the total s/n ratio for a 3-strip region is greater than 5.0;

$$\frac{s_{K-1} + s_K + s_{K+1}}{\sqrt{\sigma_{K-1}^2 + \sigma_K^2 + \sigma_{K+1}^2}} > 5.0$$

where σ_k is a single-strip noise estimate from the BAT BDT BORE. Further, both many analyses and the tracking software demand that there should only be only one cluster. This excludes events with multiple particle hits or high-energy δ-electrons, as these often give multiple peaks within one contiguous above-threshold region.

The histograms used are shown in figures 4.11 and 4.12, and discussed below. Note that since all the histograms are fundamentally representing probability distributions (PDFs), they have been normalized so that $\int_{-\infty}^{\infty} h(x)\mathrm{d}x = 1$, as this makes comparison between different devices, different runs, or between data and simulation possible.

4.3.1.1 sumADU

The sumADU histogram, shown in figures 4.11(a) and 4.12(a), is the spectrum of sum of the all the ADU-values read out (not suppressed), as defined by equation (4.19). When tuning, this is very useful to find the calibration constant K, which sets the scale of the energy deposits in ADC units.

4.3.1.2 ClusterMax, shoulder1 and shoulder2 spectrum

These histograms, shown in figures 4.11(d), 4.12(d) (ClusterMax), 4.11(e), 4.12(d) (Shoulder1), 4.11(f) and 4.12(f) (Shoulder2) shows the spectrum of in different strips in clusters.

If the cluster is centered around strip K (where the value of K varies from event to event), ClusterMax is the spectrum in strip K, Shoulder1 is the spectrum in strips $K + 1$ and $K - 1$ (histogram filled twice for every accepted event), and Shoulder2 the spectrum in strips $K + 2$ and $K - 2$ (also filled twice).

These histograms are particularly useful when tuning the capacitive coupling parameters cShare1 and cShare2, as the shape of the Landau tail changes significantly when altering these parameters. Shoulder2 is also of good use when tuning the noise width σ_{CM}/N, as there is very little signal here.

4.3.1.3 Mean cluster shape

This plot, shown in figures 4.11(h) and 4.12(h) is a profile plot of the single-strip spectra relative to its position within a cluster, as discussed for ClusterMax etc. above. Thus this plot yields little information that is not already in these histograms. It does show that there is no large left-right asymmetry of the clusters, however this is better shown by plotting the difference between the left- and right-shoulder1 or shoulder2 ADC count.

4.3.1.4 $\mathrm{d}N/\mathrm{d}\eta$ distribution

These histograms, shown in figures 4.11(b) and 4.12(b), shows the $\mathrm{d}N/\mathrm{d}\eta$ distribution, as defined in section 2.2.4.

The specific definition of η used is taken from the reconstruction code. This first finds a cluster, and calculates the charge-weighted mean $\hat{x}_c$ for this by using only the three center-most strips in the cluster (strips $K-1$, K, $K+1$). η is then defined as:

$$\eta = \hat{x}_c/\delta - \text{floor}\left(\hat{x}_c/\delta\right)$$

This histogram is interesting, as it describes how the charge is shared between strips. It is also the histogram used by the η-correction algorithm in the reconstruction, as described in section 2.2.4.

4.3.1.5 Consecutive number of strips above threshold

These plots, shown in figures 4.11(c) and 4.12(c), shows how often one, two, or more consecutive strips get marked as above threshold by the data preprocessor FPGA.

4.3.1.6 No-hit CMC/N

These histograms, shown in figures 4.11(g) and 4.12(g), shows the CMC value divided by number of strips for events where nothing is read out. It was not available in the reports produced by the program used for tuning. This is done to get the most unbiased (given the data available) estimate for what the CMC distribution looks like.

4.3.2 Tuning strategy

The strategy used is first to get approximate values for the most important parameters, while selecting something fairly neutral for the other parameters. Not all parameters had to be tuned, as they where given by the datasheet [15], while others are estimated by the planes and listed in BDT BAT BORE (see appendix D.3.1). This is the threshold setting selected, the pedestal, and the per-strip noise σ_k. These parameters are used as-is, and also written to the simulated BDT BAT BORE.

The parameters that must be tuned first, is the calibration constant K and the mean of the common mode μ_{CM}/N, which is mostly tuned by comparing the position of the peak and distribution shape in the sumADU histogram produced by the simulation with experimental data. This decides the amount of contribution from the signal and noise components, as seen from equation (4.16).

Another important parameter is the common mode spread σ_{CM}. This can be fairly well estimated by looking at the shoulder2 histogram, as this contains mostly noise. The CMC histogram for events which has no hits in it is also interesting, although the amount of data is fairly low. A usable strategy is to get a first estimate by fitting the no-signal CMC histogram with a Gaussian, and then assume this to be a sum of a common mode noise and per-strip noises. This yields a first estimate for the common mode noise dispersion

$$\frac{\sigma_{\mathrm{CM}}}{N} = \sqrt{\sigma^2_{\mathrm{meas.}} - \frac{1}{N}\sum_{k=0}^{N-1}\sigma_k^2} \tag{4.21}$$

where $\sigma_{\mathrm{meas.}}$ is the total width from the fit, and σ_k is the values found in the BDT BAT BORE. This also shows the problem of trying to estimate the common-mode noise and the per-strip noises, as they are convoluted in the data.

When an approximate value for K, μ_{CM} and σ_{CM} is obtained, the other parameters can be tuned in order to make the results as good as possible. This is done by varying them over physically reasonable ranges, until the the histograms from simulation output matches the data. For the "charge-cloud" parameters the most important observable is the $\mathrm{d}N/\mathrm{d}\eta$ histogram, and the clusterMax, shoulder1 and shoulder2 spectra. Somewhat easier to tune is the capacitive-sharing parameters cShare1 and cShare2, which is tuned by looking at the Landau tails of the clusterMax/shoulder1/2 spectra.

As the parameters are anything but "orthogonal", it is necessary to go back and do tuning in several passes.

While tuning the parameters, it was several times found that the sensor model used was too simple. Therefore an increasing amount of features and parameters was gradually added as the tuning progressed, which often made it necessary to redo some of the tuning. This means that making of the model and tuning it to match the data is closely related, which also makes the process hard to automate, as one always has to look out for interesting features in the histograms that may be difficult to predict or describe. It also means that tuning/model-making is a highly time-consuming activity, and one that cannot easily be fully automated.

4.3.3 Sensitivity to parameters

By scanning tuning parameters, it was seen that some parameters had a much weaker influence on the output than others. The model seemed particularly in-

sensitive to the applied voltage V in the range tested (85-100 [V])[9], showing only slight variations. The variations that did show up was mainly due to the blowup of N-side cluster sizes due to the low-field region close to the N-side implants creating low drift speeds and high drift times, happening when V was very close to V_d, as indicated by equation (4.8).

The temperature was also fairly insensitive to variations within "reasonable" parameters. Increasing the temperature did increase the amount of charge-sharing, as expected from equations (4.8) and (4.9).

The model is quite sensitive to σ_0, the cluster size at $t = 0$. Making this higher than $\approx 10[\mu\mathrm{m}]$ created strange results in the $\mathrm{d}N/\mathrm{d}\eta$ histogram, which became mostly flat, sometimes with a sharp dip in the region around $\eta = 0.5$. The values found is in the same range as the physics production range cutoff parameter in the particle-matter interaction simulation, discussed in Chapter 3.1.

Introducing the cutoff made it possible to reproduce the "pit" in the $\mathrm{d}N/\mathrm{d}\eta$ histogram around $\eta = 0.5$. The values found is of the order expected for how close it is necessary to be to the electrodes for a "pinch" in the field to be significant. Increasing it too much leads to much larger probabilities for no charge-sharing, as charges deposited inside the cutt-off region is completely collected by the nearest electrode.

4.4 Comparison of simulated data with real data

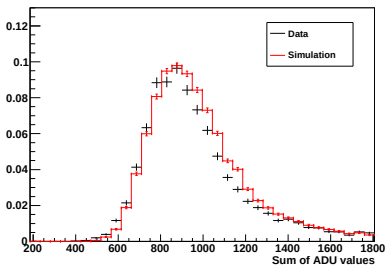
As shown in figures 4.11 and 4.12 and discussed in section 4.3, the data and the simulation generally does match quite well. The data used for comparison is taken from the May 2009 testbeam, run705, BAT1.

Even if most is in good agreement, there are some histograms that does not look as good. Notably, this includes figures 4.12(g) (P-side no-hit CMC/N distribution) and 4.11(c)) (N-side consecutive number of strips marked as above threshold). The first one is easy to explain – during parameter tuning this histogram was not monitored, and the μ_{CM}/N parameter was significantly changed in order to make other histograms look better. This is probably possible to fix by spending more time on tuning. The second histogram is is worse, and has been there since the beginning of model development. There are currently no good explanation for why this is happening.

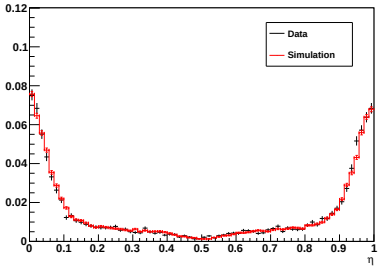
4.5 Conclusion

In this chapter, a detailed model for the response of Bonn Atlas Telescope (BAT) modules to minimum ionizing particle radiation has been presented. This model

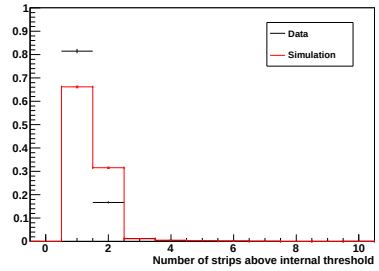
[9] This range was tested as the sensor datasheet [15] specified that $100[\mathrm{V}]$ is the sensor breakdown voltage, where the SiO_2 separating the implants from the metal strips breaks down, and the amplifier is saturated by connecting directly to the bias voltage. It was later discovered that the analog card ground potential on either side is referenced to the bias on the same side, making this not a concern.



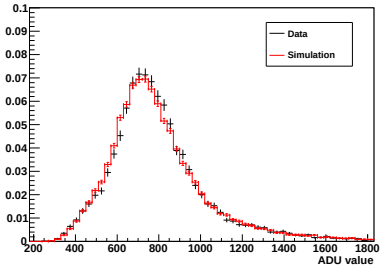
(a) sumADU (equation (4.19))



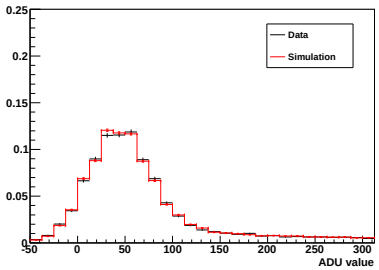
(b) 3-strip $\mathrm{d}N/\mathrm{d}\eta$



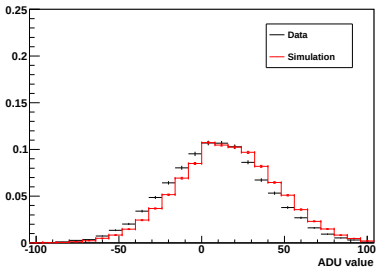
(c) Consecutive number of strips marked as above threshold



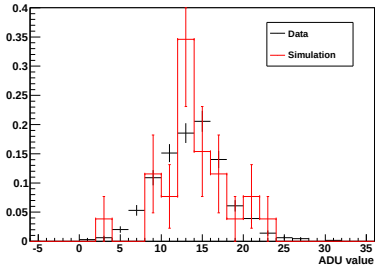
(d) Spectrum of maximum-valued/central strip of cluster



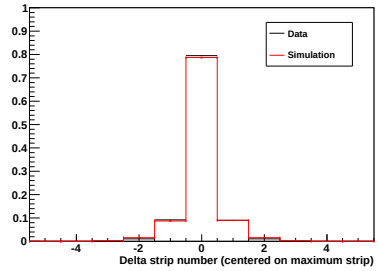
(e) Spectrum in first shoulder strips of cluster



(f) Spectrum in second shoulder strips of cluster

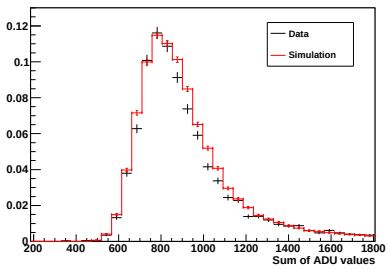


(g) CMC/N spectrum for events without hits

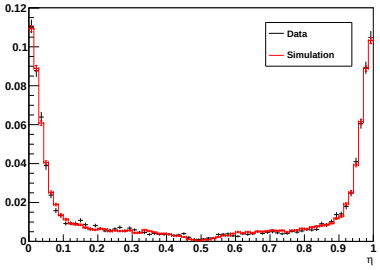


(h) Mean cluster shape profile

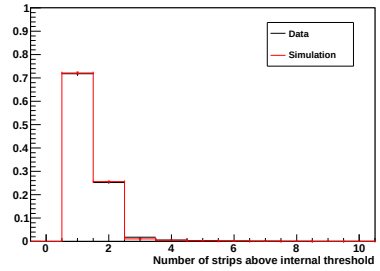
Figure 4.11: Comparison of between simulation and data, n-side



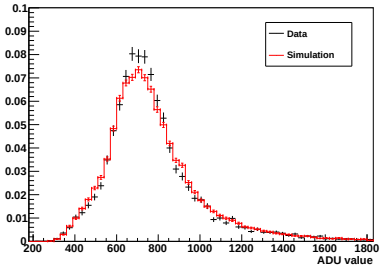
(a) sumADU (equation (4.19))



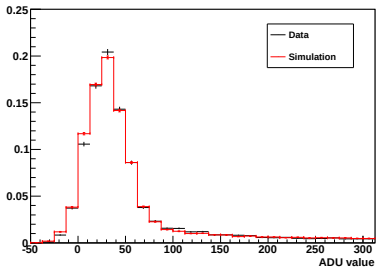
(b) 3-strip $\mathrm{d}N/\mathrm{d}\eta$



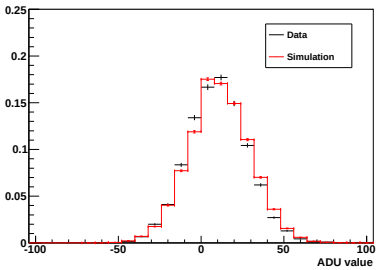
(c) Consecutive number of strips marked as above threshold



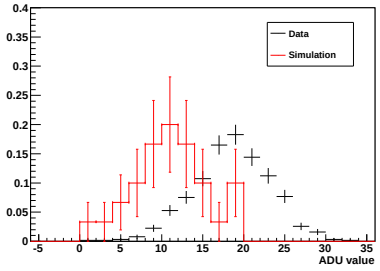
(d) Spectrum of maximum-valued/central strip of cluster



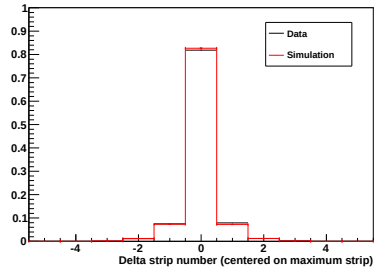
(e) Spectrum in first shoulder strips of cluster



(f) Spectrum in second shoulder strips of cluster



(g) CMC/N spectrum for events without hits



(h) Mean cluster shape profile

Figure 4.12: Comparison of between simulation and data, p-side

assumes zero magnetic field, and has been tested at single module level against experimental data using a normal incidence beam. These tests shows good agreement between the model output and data, indicating that all important processes are accounted for.

The modeling effort has also lead to a better understanding of BAT, resulting in an off-line method for subtraction of common mode noise, described in Chapter 4.2.3.1. This method will hopefully lead to a better position measurement resolution in the planes.

In order to fully qualify the model as a component of testbeam simulation, we have to be compare the output of not only one single model, but the entire experimental setup. If this is successful, we can then use the model to learn more about the properties of the setup as a whole. This is the topic of the next chapter.

Chapter 5

Performance of the tracking and alignment

The main objective for using a beam telescope like the BAT in a testbeam is to estimate where the probe particle hits the device under test (DUT). It is therefore important to know the characteristics of this hit position estimate, such as the track error distribution. The detailed modeling of the telescope described in Chapter 4 was mainly undertaken to make the hit estimate as realistic as possible.

This Chapter compares the performance of the entire telescope system as described in the simulation with what we get from the physical hardware, in order to qualify both the model developed in Chapter 4, as well as the simulation geometry etc. Further, the simulated system is used to estimate the track resolution in the DUTs, which is useful when analyzing data generated with the physical hardware.

Tracking generally means fitting a line representing the particle trajectory to measurement points. The reconstruction software used with BAT tries to fit the measurements of the particle position to a straight line using a Kalman filter, which is an optimal estimator of a linear system, equivalent to a global least squares fitter [9, 37]. This line is then extrapolated into the DUTs, providing a hit position estimate in these planes.

The measurements from the telescopes comes as numeric values representing charges collected on a set of numbered strips . Thus, in order to produce a position measurement usable for fitting, these has to be interpreted as one single measurement. This is done by the eta-correction method described in Chapter 2.2.4, which provides a hit position estimate in the local frame of the telescope sensor.

In order to fit the particle trajectory, the relative positions of the telescope sensors then has to be known, and for extrapolating the track into the DUTs, the position of these has to be known with respect to the telescope sensors. To find these relative positions with high enough accuracy to preserve the accuracy of the telescope system, an offline alignment process is used. This starts out with an estimate of the plane positions using on-site survey measurements, which is then iteratively improved by fitting tracks for many particles and looking at the resulting residual

distributions. The alignment process is included in the program doing tracking.

For simulation, alignment is unnecessary as the position of the "sensors" are completely known. This means that the simulation is expected to produce tracks with smaller error than the real experiment, even if the single devices are perfectly modeled. This becomes a problem when comparing simulated device models with experimental data, where the plane positions are not known with perfect accuracy. However, it turns out that the alignment process works well enough on the experimental data for this not to be a huge issue.

5.1 Alignment and hit resolution in BAT

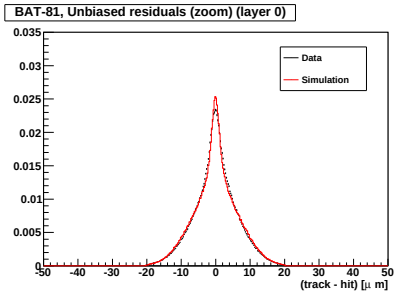
Before the simulation can be trusted to say anything about the error of the tracks extrapolated into the DUTs, the telescope model, material distribution, and particle-matter interaction model used must be validated against the real data. One possibility is to use two planes for extrapolating the path of the particle into the third plane. This yields an unbiased estimate of the particle hit position in the third plane, which can then can then be compared to a measured hit position from the plane itself in order to get an *unbiased residual*. The distribution of this error estimate can then be compared between simulation and experiment, as shown in figure 5.1.

Figure 5.1 show that see that the distribution matches in shape is fairly good, but that the simulated distributions are somewhat more peaked. A likely explanation for this is that the simulated telescope system is perfectly aligned, while the real telescope is not. It is also seen in both the simulation and experimental data that the residual distribution in plane 6 is much broader. This is to be expected from the telescope configuration (see figure 1.8), as the fit from planes 1 and 3 has to be extrapolated over a long distance and through several layers of material to reach plane 6. The angle is also poorly measured, due to the distance between plane 1 and 3 is only being 5.9 $[\mathrm{cm}]$, which further reduces the accuracy of the prediction. The simulation still reproduces the distribution, which indicates that both scattering and measurement errors in the planes are well modeled.

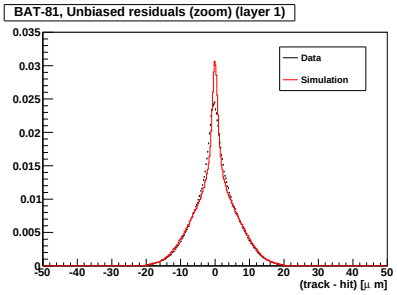
Another validation opportunity is the χ^2 distribution, shown in figure 5.2. The χ^2 variable is defined per-track by equation (5.1), which sums over both layers (X- and Y- measurement) in the three telescope planes, and the hit error estimate σ_{hit} is fixed to 5 $[\mu\mathrm{m}]$. As selected tracks are required to have exactly one hit in each plane and layer are rejected, the degrees of freedom, defined as ndof $=$ measurements $-$ fitted variables is always the same, and equal to ndof $= 6-4 = 2$.

$$\chi^2 = \sum_{\text{All planes and layers}} \frac{\left(\hat{x}_{\text{track}} - x_{\text{measured}}\right)^2}{\sigma^2_{\text{hit}}} \tag{5.1}$$

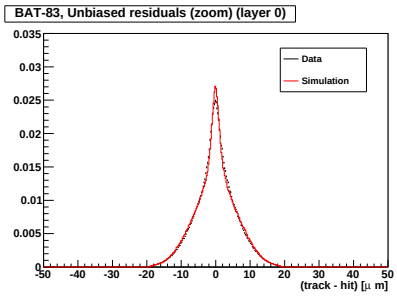
From figure 5.2, we see that the distributions match almost perfectly over two orders of magnitude. The non-matching parts are still quite close, and tracks in this region will anyway be removed by a χ^2-cut usually introduced by analysis to remove "bad" tracks. Large χ^2 values correspond to large deviations from a



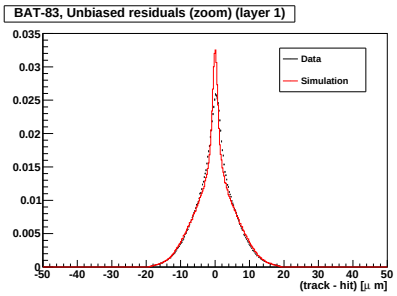
(a) BAT1, x-direction



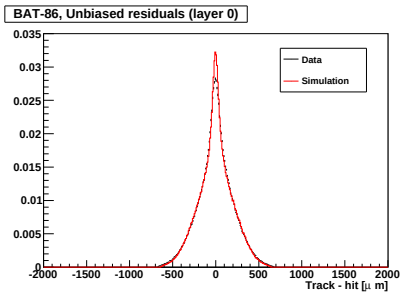
(b) BAT1, y-direction



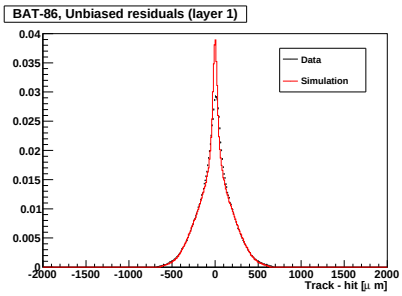
(c) BAT3, x-direction



(d) BAT3, y-direction



(e) BAT6, x-direction. Note different scale on X-axis



(f) BAT6, y-direction. Note different scale on both axis

Figure 5.1: Comparison of unbiased BAT residuals for all planes and layers. X and Y corresponds to coordinates used in tracking program and analysis software, where X points skywards, and Y to the left when looking downstream. Layer 0 is the N-side, and layer 1 is the P-side.

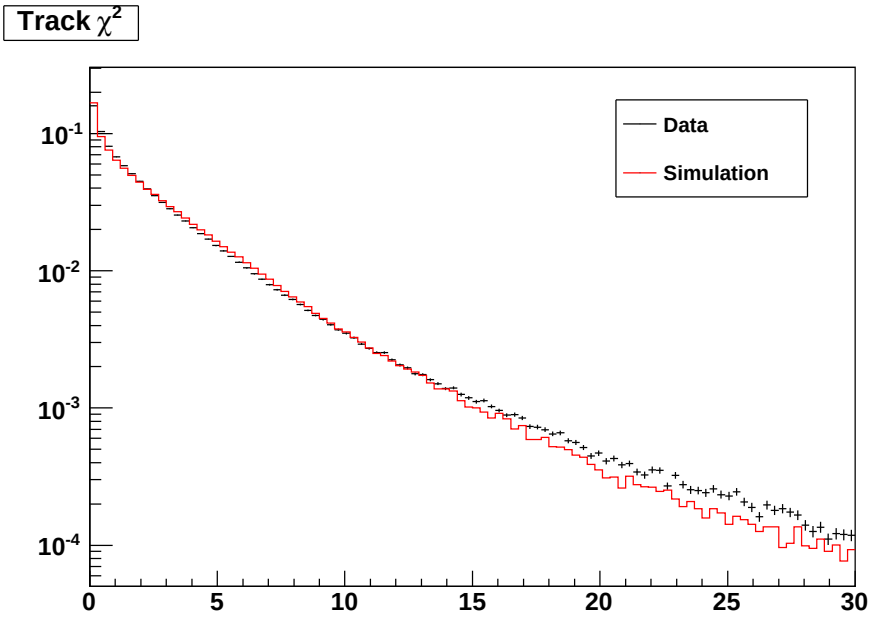


Figure 5.2: χ^2-distribution for track, from simulation and data, on a log scale

straight track, and there may be many contributions to why the experimental data contain more of these tracks. It could for example be due to incomplete modeling of particle scattering due to too little material in the simulation geometry, presence of more noise-hits in the real BAT planes than in the model, desynchronization of read out trigger numbers between BAT planes, and more.

Another interesting result is that the hit resolution (represented by the unbiased residuals) is depending on the interstrip hit position, as shown in figure 5.4. This is most probably due to a feature of the eta-correction method (described in Chapter 2.2.4) used to estimate the hit position. This method uses the charge-sharing to resolve the hit position with sub-pitch accuracy, and as there is very little "real" charge sharing when the particles passes close to the center of a strip (see the event gallery in figure 4.4), this results in higher measurement errors for those events.

The plots in figure 5.4 also show that the estimated hit position is biased for particles passing close by, but not on, the center of a strip. This can be explained by that the amount of real charge-sharing is very low for particles passing close to the center of a strip, so almost all the "charge sharing" seen when creating the $\frac{dN}{d\eta}$ distributions are really due to noise, increasing the values of η used to fill the histogram. This results in broader side-peaks on the $\frac{dN}{d\eta}$ histogram, which in turn results in less steep slopes around $\eta \approx 0.0$ or 1.0. When using this to correct a hit position estimate, the value of η for the given hit is compared to the cumulative distribution function (CDF) of the $\frac{dN}{d\eta}$ distributions. But even if the noise has al-

tered the CDF, each and every measurement is not biased, as the noise have equal probability to pull it in either direction. Thus the eta-corrected hit estimate is pulled closer to the center of the closest strip than the correct value, as sketched in figure 5.3, and this gives at least one source of bias such as seen in figure 5.4.

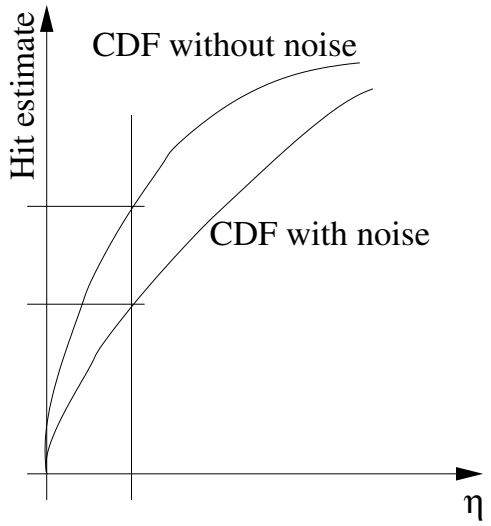


Figure 5.3: Eta-correction cumulative distribution functions (CDF)s (cartoon, see figure 2.7(b)), for cases with and without noise. Figure shows low-eta part of the CDF.)

The plots in figure 5.4 are well reproduced by the simulation, which means that unknown processes probably do not play a significant role in producing this result.

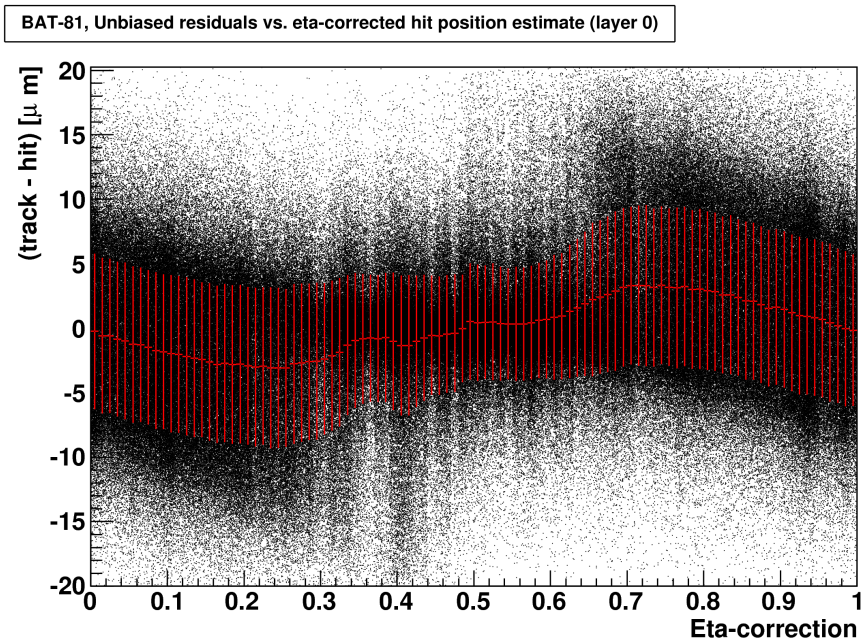
These plots probably also explains the long tails in the residual distributions shown in figure 5.1. By comparing figures 5.1(a) and 5.4(a), we see that the region in the first plot containing the tails ($\pm$ 15 – 20 $[\mu\mathrm{m}]$) is coinciding with a region in the second plot containing biased hit estimates.

5.2 Track resolution in DUTs

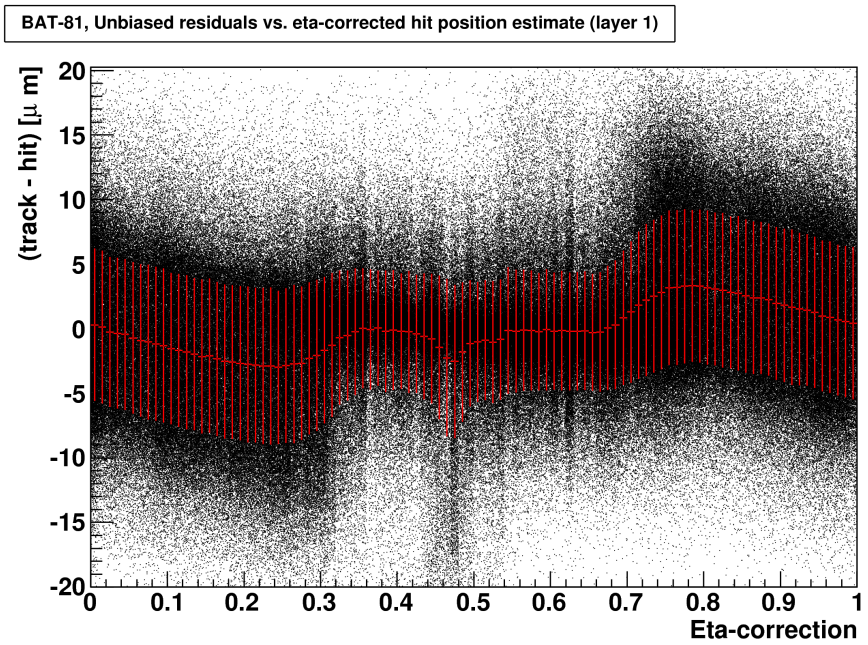
From the discussion above, the simulation seems to be a good description of the experiment. This means we can use it to compare the true hit position in the DUTs[1] with the estimated hit position from telescope tracking. This defines the hit resolution in the DUTs, and plots of this is shown in figure 5.5. As these distributions are close to Gaussian, the standard deviation of a Gaussian fitted to this is taken as the resolutions, which are listed in table 5.1.

In table 5.1, we note that the resolution get better as the device gets closer to the telescope doublet BAT1/BAT3. This is reasonable, as two planes gives a tighter

[1] Point of intersection between particle trajectory and DUT sensor, see Appendix A.3.2.

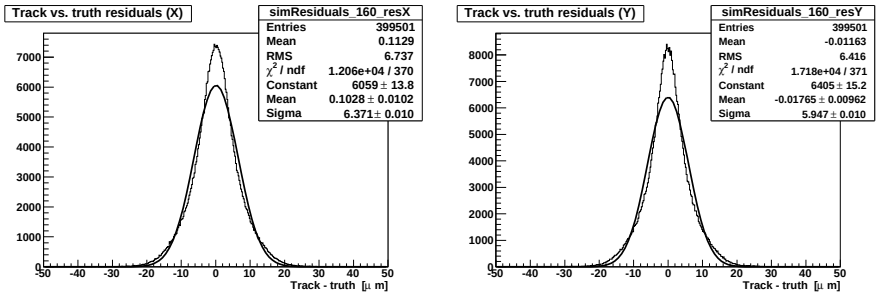


(a) X-direction



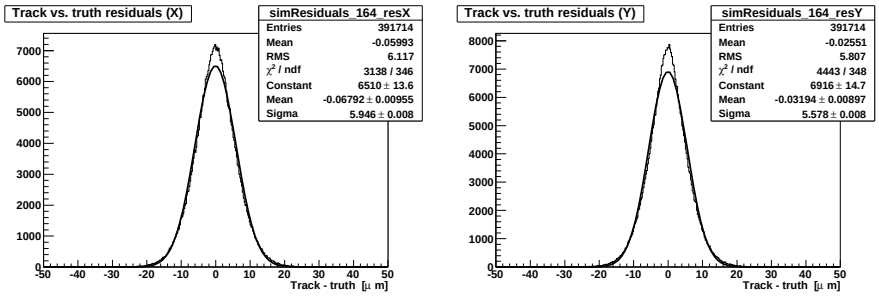
(b) Y-direction

Figure 5.4: Unbiased residuals (as shown in figure 5.1) versus estimated hit position modulo strip pitch. Scatterplot with overlayered profile histogram of the same data, error bars showing spread along the plot's y-axis. Data from real experiment.



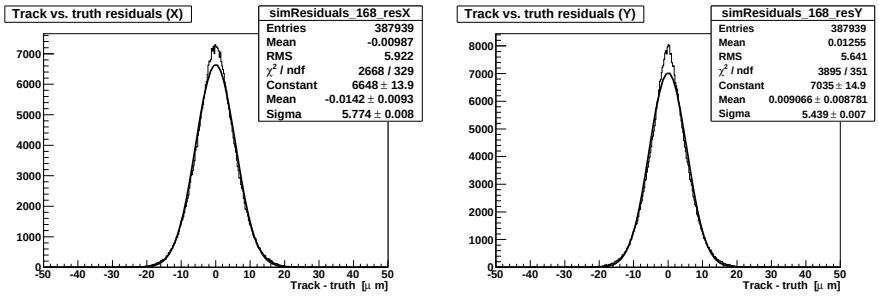
(a) DUT1 (planar), X

(b) DUT1 (planar), Y



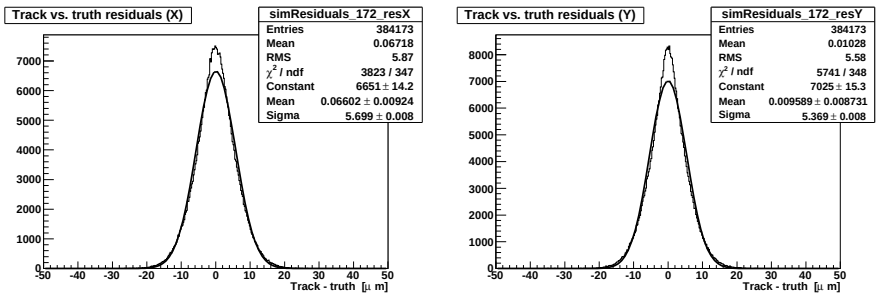
(c) DUT2 (STA-3E), X

(d) DUT2 (STA-3E), Y



(e) DUT3 (FBK-3E7), X

(f) DUT3 (FBK-3E7), Y



(g) DUT4 (FBK-3EM5), X

(h) DUT4 (FBK-3EM5), Y

Figure 5.5: Error on track in DUTs (defined as $\hat{x}_{\text{track}} - x_{\text{truth}}$), fitted to a Gaussian.

Device name	Resolution [μm]		Z-position rel. BAT6 [mm]	Z-position rel. BAT1 [mm]
	X	Y		
PLANAR	6.37	5.95	326.12	-516.98
STA-3G	5.95	5.58	486.14	-356.96
FBK-3E7	5.77	5.44	566.15	-276.95
FBK-3EM5	5.70	5.37	646.14	-196.96

Table 5.1: Estimated track resolution in DUTs from simulation, and DUT positions relative to the two closest telescope planes. Device names and geometry: See table 3.3 and figure 3.6.

constrain on the track than one, and as the sensors are also closer to the doublet than to BAT1.

As these estimates are made using simulated data only, the real values of the track error might be somewhat larger, as seen in the unbiased residual plots. This can for example be due to residual misalignment in the real system.

5.3 Conclusion

In this chapter, a validation of the simulation of particle trajectories and telescope planes has been presented. The simulation passed this test very well, showing good agreement with experimental observables.

This was then used to extract the tracking resolution in the DUTs, shown in table 5.1, showing that the resolution is approximately 6 [μm]. Knowing this resolution is important when analyzing the response of DUTs to MIP radiation, or comparing the output of DUT models to experimental data. This is the topic of the next chapter.

Chapter 6

ATLAS Pixel simulation models

The main objective of the testbeam is to test the response and performance of ATLAS 3D pixel sensors, described in Chapter 2.2.5. The testbeam simulation includes the particles trajectory and energy loss, as well as the sensor responses. Thus different models for sensor response can be tried and compared to what is measured in the experiment. Doing this is useful for testing hypothesis about the sensor response function, as well as developing response function models that can be incorporated in larger systems such as ATLAS simulation.

This chapter introduces models describing pixel sensor response, including the development of such models, and comparisons with data.

6.1 Simulation geometry

When used in a testbeam, the pixel sensors and frontend chips are mounted on PCBs providing mechanical support, as well as making it possible to connect the sensors to the DAQ and power systems. A picture of such a board is shown in figure 6.1.

As with the Bonn Atlas Telescope, the geometry of the sensor as well as the testboard must be described to Geant4. The geometry used is shown in figure 6.2, and a computer rendered drawing is shown in figure A.1(b). This includes a *sensor assembly* which is the sensor itself, the frontend (FE) chip, and solder for connecting them together and biasing the sensor. This is mounted on a a PCB, which may have a hole for letting low-energy particles through to a scintillator trigger when used in source-tests. A protective plastic cover is also mounted on the PCB, covering the sensor and wirebonds. The solder used for connecting the sensor to the FE is indium bumps, which after bump bonding are cylinders 7 $[\mu\mathrm{m}]$ tall and 18 $[\mu\mathrm{m}]$ in radius[1], one for each of the $160 \times 18 = 2880$ pixels. As modeling this as 2880 separate volumes would create an unnecessary complicated simulation geometry, this is modeled as a layer of indium covering the entire sensor/FE area, with a total mass equal to the total mass of indium in the real geometry.

[1] Giovanni Darbo, private communications

The area of the sensor and FE is taken to be $(\text{nRows} \times 400) \times (\text{nCols} \times 50) = (18 \times 400) \times (160 \times 50) = 7.2 \times 8.0\ [\text{mm}^2]$, thus ignoring ganged- or extra-long edge pixels, and bias overhang at the edges. Similarly, the FE is simply taken to be aligned and of the same size as the sensor, no overhang for wirebonding is taken into account.

The four devices under test (DUTs) have slightly different geometry, differing in the presence of a hole in the PCB behind the sensor, sensor thickness, and the presence of a aluminum bias tab. The different geometry parameters used are shown in table 6.1.

DUT ID	Name	Hole in PCB	Sensor thickness $[\mu\text{m}]$	Bias tab
1	Planar	No	250	Yes
2	STA-3E	No	210	No
3	FBK-3E7	Yes	200	Yes
4	FBK-3EM5	Yes	220	Yes

Table 6.1: Geometrical parameters of DUTs. DUT names and IDs defined in figure 3.6.

6.2 Pixel response models

Parallel to what is described in Chapter 4, models for the response of 3D pixel sensors to energy depositions has been studied. These models, described in Chapter 6.2.2, are not as closely linked to the device physics (drift-diffusion etc.) as the model developed for the BAT sensor, due to the more complex field geometry in a 3D pixel sensor. They are instead so-called effective models, describing the amount of charge collected on the hit pixel and its neighbours. An alternative, which currently has not been implemented for pixel response, is to do a complete simulation of drift-diffusion, trapping etc. for each carrier cloud.

6.2.1 Carrier cloud tracking models

As stated above, a full simulation of drift-diffusion etc. is possible, but computationally expensive and harder to implement. Thus this has unfortunately not been implemented for pixel sensors due to time constraints in the thesis work.

A viable way to make a physical model of a 3D pixel sensor may be to treat each carrier cloud as a particle, and solve its equations of motion until collection at an electrode. This is essentially what was done for the BAT sensor (here using an analytical approach), taking diffusion into account by letting the size and shape of the carrier cloud evolve with time.

For a 3D sensor, this method of including diffusion is not possible due to the field- and charge collection geometry. A possible solution is to add a random

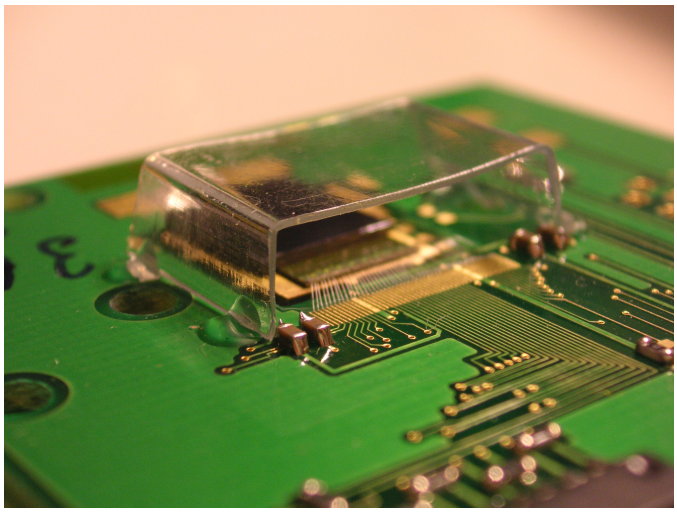


Figure 6.1: Picture of the pixel module assembly

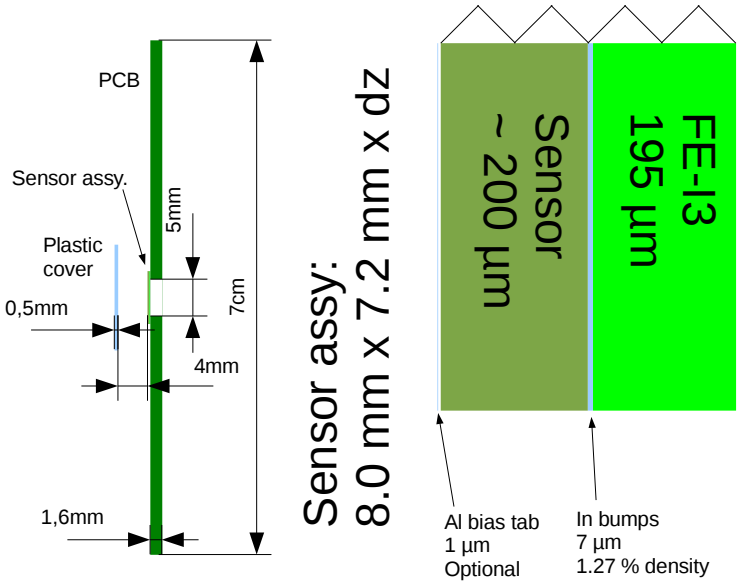


Figure 6.2: Pixel module assembly. Left panel: Overall geometry, including PCB and plastic cover. Right panel: Thickness of parts in sensor assembly.

component in the drift-diffusion equation for the charge clouds, which can be done by at each timestep sampling the new particle position from a Greens-function describing the probability distribution of new positions after the timestep[2]. Further, the Fano sampling of charge clouds (see Chapter 3.1.1) should be moved into the device simulation code, making it more flexible for tuning without re-running the whole simulation. This also enables adding a randomized transverse displacement of the clouds initial position with respect to the line between the step points, similar to the σ_0 parameter in the BAT model.

Full 3D sensors have two features which is of importance if doing such a simulation. The first one is that they have an essentially two-dimensional field geometry, the second is that the amount of charge-sharing for hits in the pixel interior are very low (see figure 2.11, 0° data).

The main advantage of having a two-dimensional field geometry is that the fieldmap can be limited to two dimensions, making it much easier and faster to generate and to access. Limiting the particle motion to two dimensions are also possible, but only removes two variables (particle position and velocity transversly to the sensor plane) from the simulation. The downsides of limiting particle motion to two dimensions is that it makes it impossible to model drift transversly to the sensor plane due to magnetic deflection. Oxide layer effects also become difficult, as they depend on transverse position, and electrostatic interaction between charge-clouds also becomes impossible to model.

The fact that 3D sensors really only see sharing at pixel edges can be used to limit the domain of full simulation to the areas close to the edges. Such a hybrid model should yield a major speedup, as the charge-clouds deposited in the central area of a pixel can immediately be "collected" in this pixel. Further, charge-clouds deposited in the edge areas then only needs to be tracked to the edge of a central area, where it can be deposited. The problem for this method is how to simulate trapping, which results in only a fraction of the total charge in a cloud being collected at the electrode, described in Chapter 2.2.2. Simulating trapping correctly is dependent on tracking the cloud all the way to collection at an electrode, since the cloud (or more realistically, parts of it) has a certain probability for getting trapped at each timestep.

6.2.2 Effective models

Effective models does not describe the detailed microscopic behavior of the electrons and holes, but instead directly describe the response as a function of position and amount of charge deposits.

The main parameter in such a model is the three-dimensional charge collection efficiency, defined in equation (6.1). The number of charges created is here the

[2]This is similar to what is done in Metropolis-Hastings Monte-Carlo (MHMC) [31]. The equation of interest must be solved is then the the Fokker-Planck equation (FPE), which describes the time evolution of the probability density describing where to find a particle affected by both drift- and diffusion.

clouds coming from Fano sampling, described in Chapter 3.1.1.

$$\epsilon_3\left(\vec{r}\right) = \frac{\text{Number of charges collected}}{\text{Number of charges created}} \tag{6.1}$$

The digital number output from each pixel above a threshold is not the number of electrons collected, but something known as *ToT* or *Time-over-Threshold*. This is the time (in units of 25 [ns]) the amplifier connected to the pixel has an output voltage above a certain *threshold* value. The response, shown in figure 6.3, is not completely linear, and is often fitted by the function given by equation (6.2)[3]. Here Q is the charge injected into the amplifier.

$$\text{ToT} = a_0 \frac{a_1 + Q}{a_2 + Q} \tag{6.2}$$

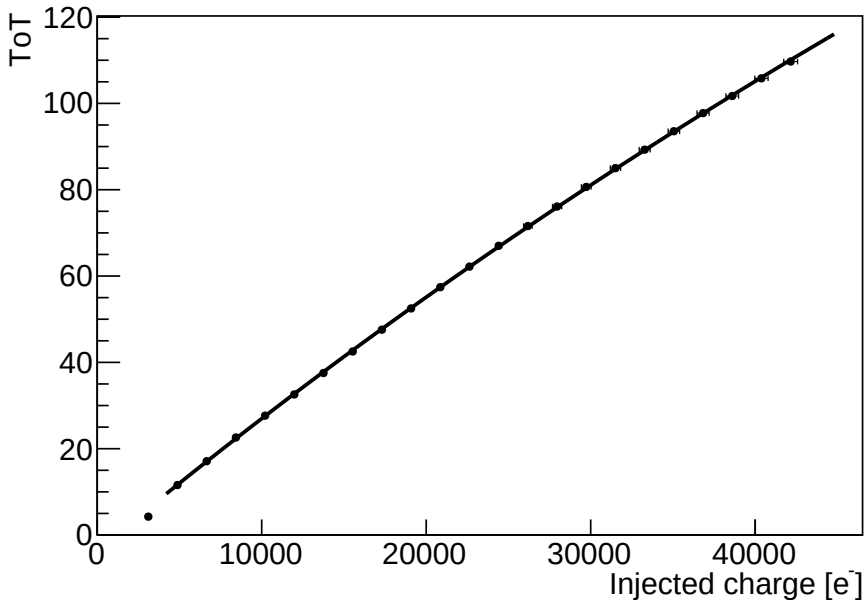


Figure 6.3: Pixel ToT as a function of charge Q injected into the amplifier by a capacitor inside the frontend chip. Data for the Stanford 3E device.

For the Stanford 3E device, the fitted parameters of equation (6.2) are given in equation (6.3). The data is fitted in the range $Q \in [4000, 45000]$.

$$a_0 = 731.525, \quad a_1 = -1134.26, \quad a_2 = 230466 \tag{6.3}$$

[3] Philippe Grenier, private communication

6.2.2.1 HolePunch model

The HolePunch (HP) model describes a sensor which has a 100% charge collection efficiency outside the electrodes, and 0% inside. This leads to the charge-collection efficiency described by equation (6.4). Here Δr_i is the distance between the center of electrode cylinder r_i and the point of deposit, and R the electrode radius. $\mathrm{H}(x)$ is the Heaviside step-function, which is 1 for $x > 0$, and 0 for $x < 0$.

$$\epsilon_3(\vec{r}) = \prod_i \mathrm{H}(\Delta r_i - R) \tag{6.4}$$

Analytical study This model can be studied analytically in the case of a beam without any divergence, leading to an analytical expression for the 2D charge collection efficiency as well as the 1D charge collection efficiency.

2D charge collection efficiency for a divergency-free beam can be defined as:

$$\epsilon_2(x_0, y_0, \theta) = \frac{\int_\lambda \epsilon_3(\vec{r})\,\mathrm{d}\vec{r}}{\int_{\lambda'} 1\,\mathrm{d}\vec{r}} \tag{6.5}$$

The integrals in this equation are path-integrals, with λ being the path of a particle parallel to z-axis (as defined in figure 6.4), passing through the point $(x_0, y_0, 0)$. θ is the tilt angle of the sensor around the y-axis, and λ' is the path of a particle passing perpendicularly through the sensor in an area which has $\epsilon_3 = 1$ along the whole path; this integral serves to normalize the 2D charge collection efficiency to unity for such a particle.

An analytical expression for ϵ_2 can be found by parameterizing the four boundary curves. This yields:

$$\begin{aligned} \vec{v}_1 &= t_1\hat{e}_1 - \tfrac{d}{2}\hat{e}_3 \quad , \quad \vec{v}_2 = t_2\hat{e}_1 + \frac{d}{2}\hat{e}_3 \\ \vec{v}_3 &= -r(y)\hat{e}_1 + t_3\hat{e}_3 \quad , \quad \vec{v}_4 = r(y)\hat{e}_1 + t_4\hat{e}_3 \end{aligned}$$

Here $r(y)$ is the "radius" of the electrode along the x-axis at a given y-position, and is given by:

$$r(y) = \sqrt{R^2 - y_0^2}$$

These vectors can then be rotated using a rotation matrix, and then solving for at which value of t they intercept the line $x = x_0$. This yields:

$$\begin{aligned} t_1 &= \tfrac{x_0 + \frac{d}{2}\sin\theta}{\cos\theta} \quad , \quad t_2 = \frac{x_0 - \frac{d}{2}\sin\theta}{\cos\theta} \\ t_3 &= \tfrac{x_0 + r(y)\cos\theta}{\sin\theta} \quad , \quad t_3 = \frac{x_0 - r(y)\cos\theta}{\sin\theta} \end{aligned}$$

Inserting this into the z-components of the boundaries yields the z-position of the interception points (z_1 through z_4). As a path integral of length L through a function of value 1 equals L, the 2D charge collection efficiency as defined by

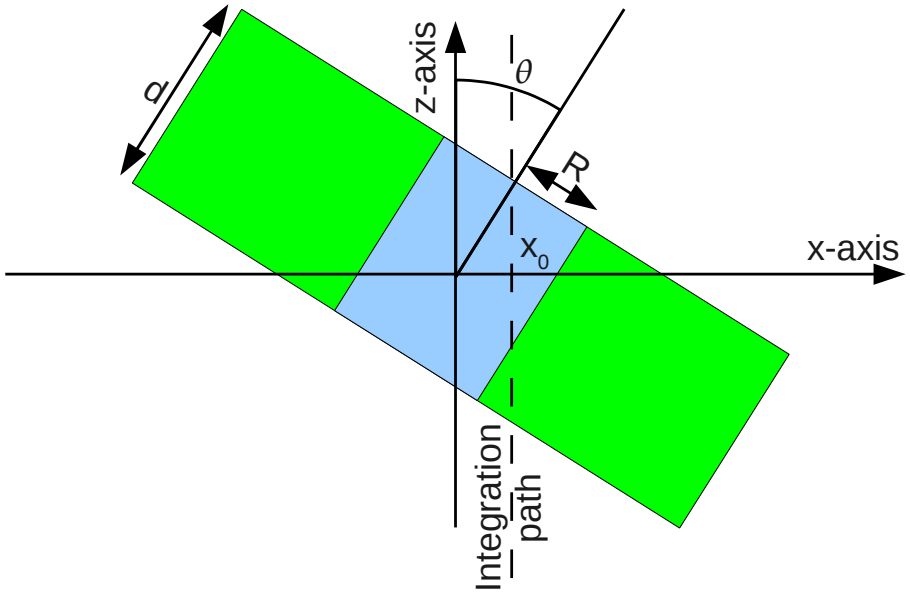


Figure 6.4: Geometry of analytical HolePunch-model

equation (6.5) can be calculated for the HP model. This leads to equation (6.6) if $0° < \theta < 90°$ and the particle misses the hole ($z_4 > z_2$ or $z_1 > z_3$ or $|y_0| > R$), equation (6.7) if $0° < \theta < 90°$ and it hits the hole, and equation (6.8) if $\theta = 0°$. This function has been plotted in figure 6.5.

$$\epsilon_2^{(\mathrm{HP})}(x_0, y_0, \theta) = \frac{z_2 - z_1}{d} \tag{6.6}$$

$$\epsilon_2^{(\mathrm{HP})}(x_0, y_0, \theta) = \frac{1}{d}\left[(z_4 - z_1)_{\text{if} > 0,\, \text{else } 0} + (z_2 - z_3)_{\text{if} > 0,\, \text{else } 0}\right] \tag{6.7}$$

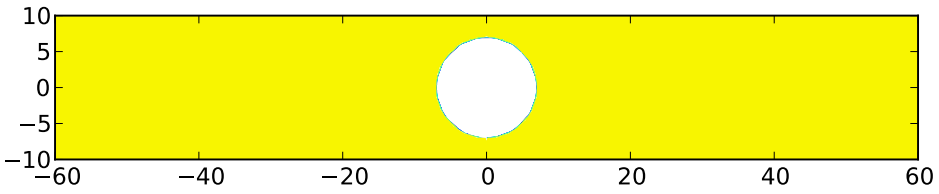
$$\epsilon_2^{(\mathrm{HP})}(x_0, y_0, \theta) = \begin{cases} 1 & \text{for} \quad x_0^2 + y_0^2 < R^2 \\ 0 & \text{else} \end{cases} \tag{6.8}$$

Further one can define the average charge collection efficiency as in equation (6.9), which is normalized to 1 for a fully efficient sensor at normal incidence[4].

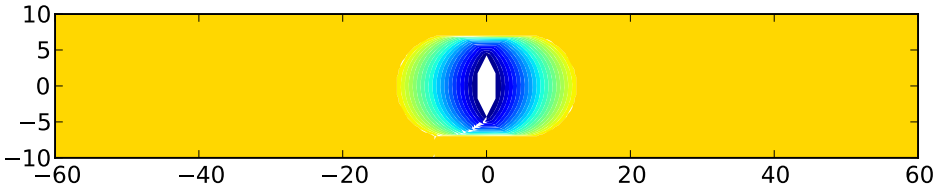
$$\epsilon_1 = \frac{1}{\text{Projected pixel area}} \iint_{\text{projected pixel area}} \epsilon_2(x_0, y_0)\, \mathrm{d}x_0\, \mathrm{d}x_0 \tag{6.9}$$

Numerically integrating the efficiency maps shown in figure 6.5 and correcting for integration area being less than the area per electrode, and the number of electrodes per pixel, this leads to the curve shown in figure 6.6.

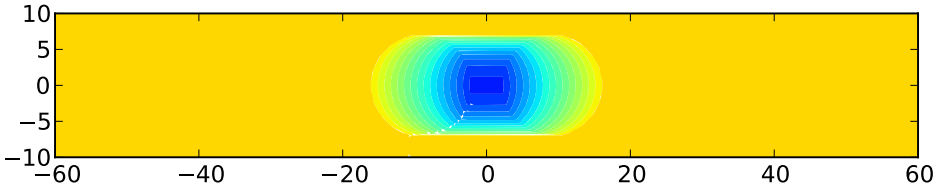
[4]Meaning that ϵ_1 as defined here can go above 1 if the beam is not hitting the sensor at normal incidence.



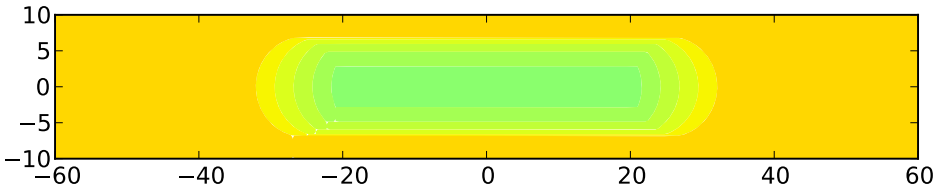
(a) 0.0°



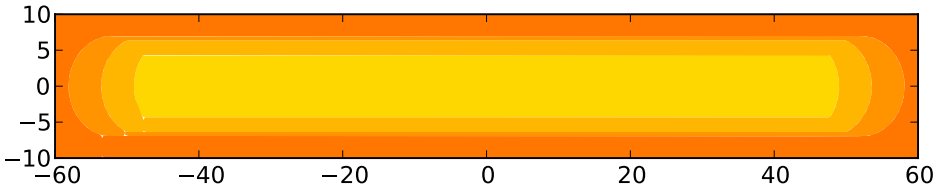
(b) 3.0°



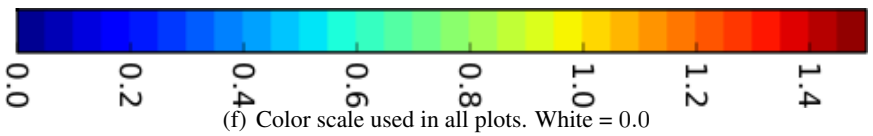
(c) 5.0°



(d) 15.0°



(e) 30.0°



(f) Color scale used in all plots. White = 0.0

Figure 6.5: 2D efficiency (as defined in equation 6.5) around a single electrode in the HolePunch model. Analytical calculation using parameters $R = 7$ [μm] and $d = 210$ [μm]. All units on axis in [μm].

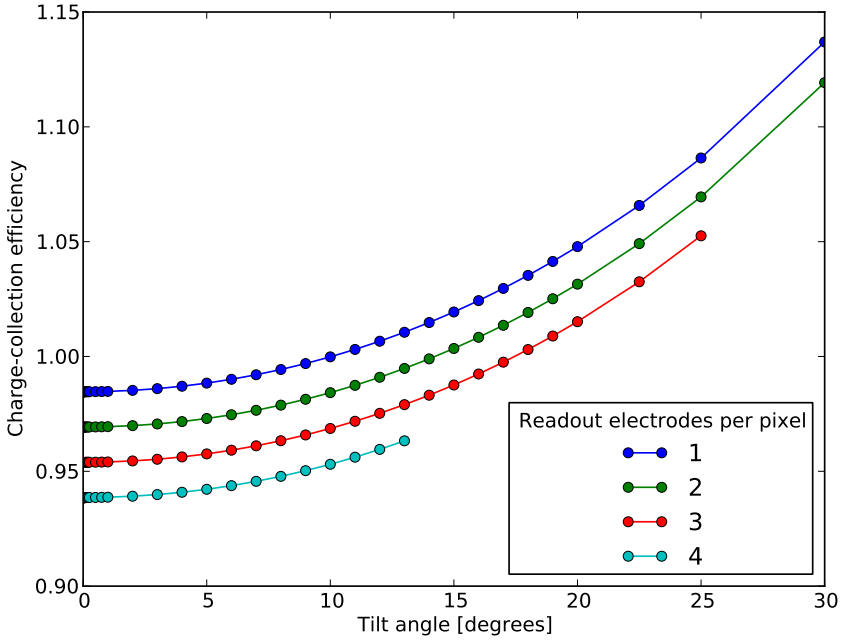


Figure 6.6: Mean collection efficiency (as defined in equation 6.9) as a function of angle and number of readout electrodes per pixel. Curves cut before the projections of electrode columns start to overlap.

This analytical model describes the charge collected from a particle depositing a fixed amount of energy per unit length. It also assumes that all particles have trajectories parallel to the z-axis, with no spread in angle. This is not true for a charged particle traversing the sensor bulk, which have an uneven energy deposit along its path. Neither is the actual beam completely parallel to the z-axis (see Chapter 3.3); but it is still an useful exercise for interpreting the data. Note that the scale of the x-axis as shown on the plots in figure 6.5 is different to the one used in the plots from real data or simulation, as the reconstruction software rotates the x-axis along with the device. Thus the plots of figure 6.5 is scaled (compressed) by a factor $\cos\theta$ along the x-axis when compared to plots from simulation or data. This only affects the axis perpendicular to the axis of rotation.

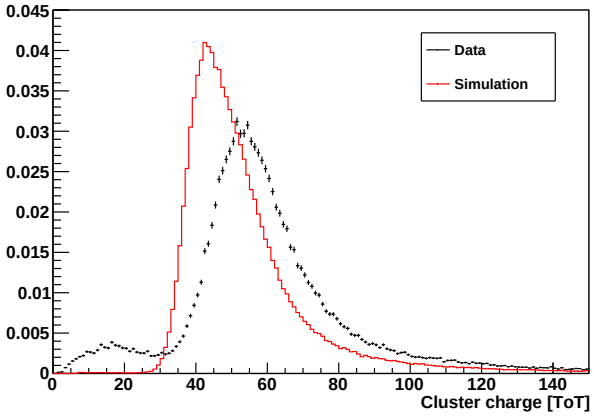
Simulation results The HolePunch-model has been implemented in the simulation as a `simDut` (see Appendix C.3), thus getting charge deposits from the Fano sampling described in Chapter 3.1.1. Electronics noise etc. is not included in the model. To make the HolePunch model, the three-dimensional charge collection efficiency of equation (6.4) has been extended to include a possibility of a effi-

ciency ϵ_3^{col} inside the electrode columns, as this seems to be necessary in order to reproduce the data. For calculating the ToT as a function of deposited charge, equation (6.2) with the parameters of equation (6.3) is used for $Q > 3200\,\text{e}^-$. This is the threshold setting of the frontend, and hits with charge deposit below this value are not registered. This results in that the simulation model has no hits where $\text{ToT}(Q) < \text{ToT}(Q = 3200) = 6$. The charge of a cluster is calculated by inverting the transformation leading to equation (3.3), so that the number of electrons in a cluster is $N_k = E'_k/w$.

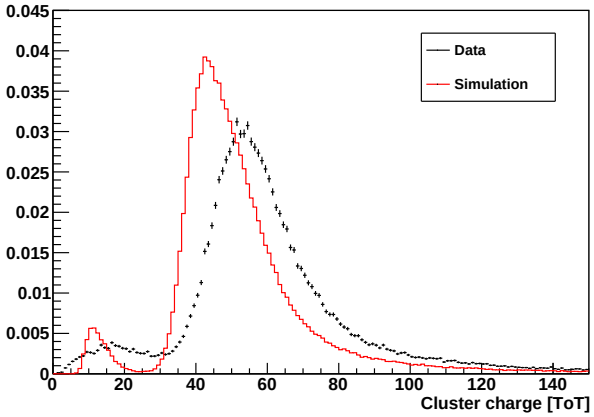
A comparison of the sum of the ToT values for pixels in clusters close to the estimated hit position is shown in figure 6.7. Here the data is compared to the HolePunch model with different values of ϵ_3^{col}. The curves do not overlap, which is to be expected as this is a simple and untuned model. However, some features, such as the extra “bump" shown in the data at low-ToT values, are reproduced when $\epsilon_3^{\text{col}} > 0.0$, but not when $\epsilon_3^{\text{col}} = 0.0$. It has been shown [13] that this feature is connected to the electrodes, and the simulation further indicates that there must be some efficiency in the undepleted columns. This claim is also further strengthened by the charge collection efficiency analysis described in Chapter 6.3. A possible explanation for this is that some of the minority carriers created when a charged particle passes through the undepleted volume can diffuse to the edge of the depletion zone, where they are swept away by the electric field and collected at another electrode of opposite polarity, creating a signal. This is in analogy to what happens in solar cells, where one normally assume that the volume of undepleted material within one diffusion length from the edge of the depletion zone is also active [35]. Some testbeam papers [40, 22] have also suggested that undepleted electrodes might have some efficiency. The last paper cited speculates that δ-electrons originating from inside the electrodes traveling into the depleted volume might be the reason for an apparent efficiency. δ-electrons are included in this simulation, but the bump is still not visible in the spectra for $\epsilon_3^{\text{col}} = 0.0$, which makes it less likely that the electrode efficiency observed is due to $delta$-electrons instead of “true” efficiency. Further simulation studies with shorter secondary radiation production cutoff (currently set to $5\ [\mu\text{m}]$) should be done to confirm or dismiss this.

It should also be noted that the $\sum$ToT-spectra from the data extends all the way down to $\sum\text{ToT} = 0$, while the simulation has a sharp cutoff at $\sum\text{ToT} = 6$. This means that there is a response below this threshold, which must be included in an accurate model. The peak of the spectra is also displaced towards a lower ToT-value, which might be due to a problem with the calibration used to get the parameters in equation (6.3), charge multiplication in the real device leading to $\epsilon_3 > 0$, or the amount of deposited energy in the simulation being too low.

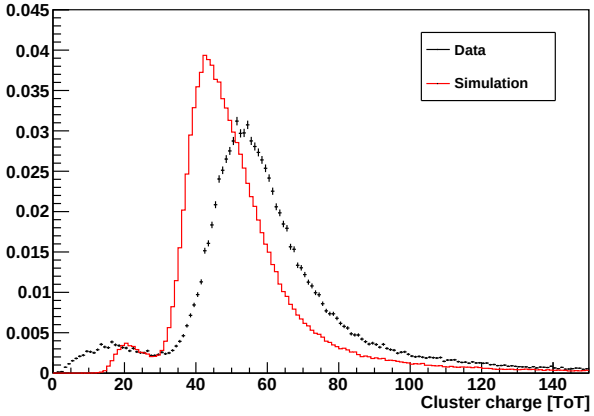
Figures 6.8 and 6.9 shows the hit efficiency for both the real data and the HolePunch model, as a function of hit position. The hit efficiency ϵ^h is defined as the probability of getting one or more pixels close to the hit position above threshold, and is related to the charge collection efficiency by equation (6.10). Here $P(Q)$ is the probability distribution describing the charge deposited in an event,



(a) $\epsilon_3^{\text{col}} = 0.0$

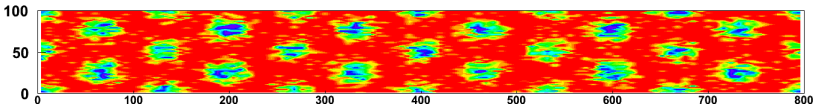


(b) $\epsilon_3^{\text{col}} = 0.3$

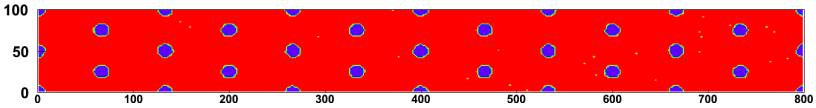


(c) $\epsilon_3^{\text{col}} = 0.5$

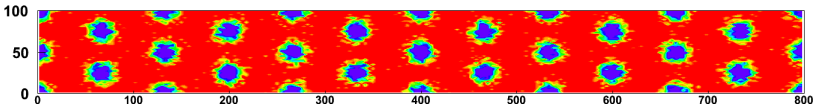
Figure 6.7: Comparison of cluster $\sum$ ToT spectra for clusters matched with tracks, using data for different ϵ_3^{col} in the HolePunch model, electrode $R = 7.0$ [μm]. All curves normalized to unit area.



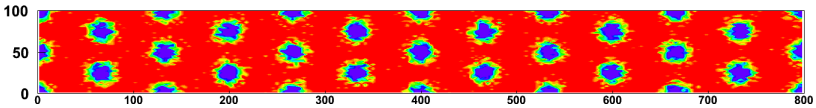
(a) Sta-3E sensor (no field), real experiment.



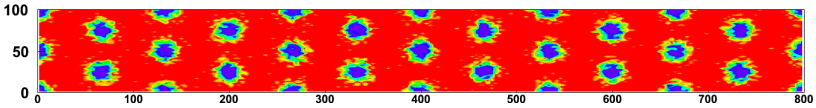
(b) HolePunch model ($\epsilon_3^{\text{col}} = 0.0$), true hit position.



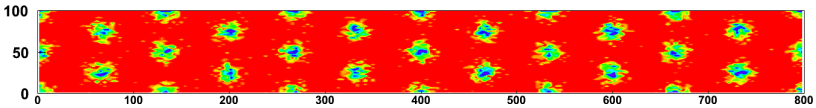
(c) HolePunch model ($\epsilon_3^{\text{col}} = 0.0$), simulated BAT for tracking.



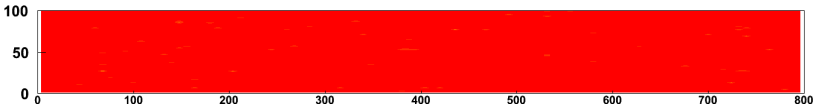
(d) HolePunch model ($\epsilon_3^{\text{col}} = 0.1$), simulated BAT for tracking.



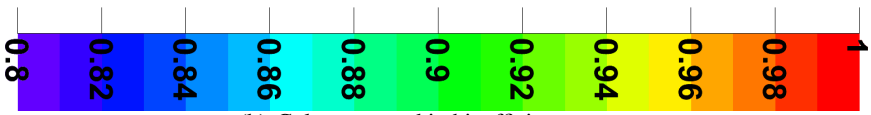
(e) HolePunch model ($\epsilon_3^{\text{col}} = 0.15$), simulated BAT for tracking.



(f) HolePunch model ($\epsilon_3^{\text{col}} = 0.2$), simulated BAT for tracking.

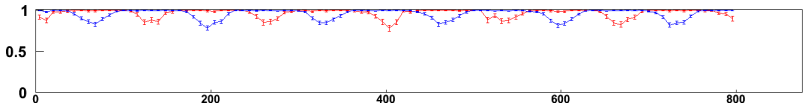


(g) HolePunch model ($\epsilon_3^{\text{col}} = 0.3$), simulated BAT for tracking.

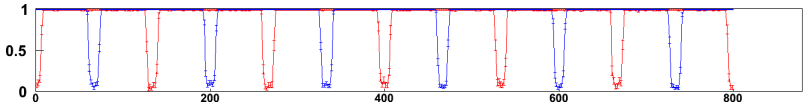


(h) Colormap used in hit efficiency maps

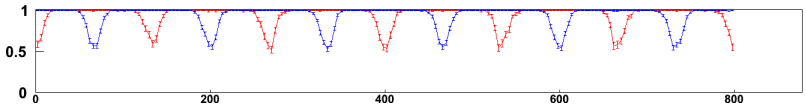
Figure 6.8: 2D maps of hit efficiency for real data and HolePunch model with electrode radius $R = 7$ [μm]. Both real data and simulation uses $\approx 0^\circ$ beam incidence. All units on axis in μm.



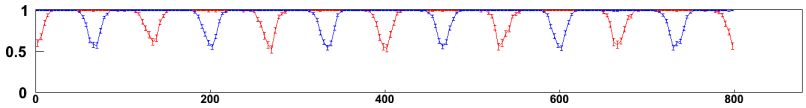
(a) Sta-3E sensor (no field), real experiment.



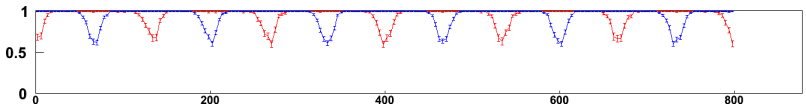
(b) HolePunch model ($\epsilon_3^{\text{col}} = 0.0$), true hit position



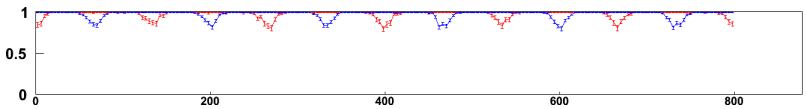
(c) HolePunch model ($\epsilon_3^{\text{col}} = 0.0$), simulated BAT for tracking



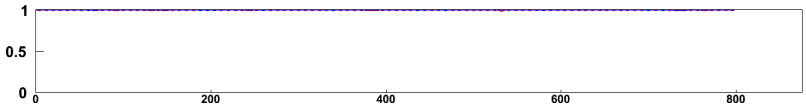
(d) HolePunch model ($\epsilon_3^{\text{col}} = 0.1$), simulated BAT for tracking



(e) HolePunch model ($\epsilon_3^{\text{col}} = 0.15$), simulated BAT for tracking



(f) HolePunch model ($\epsilon_3^{\text{col}} = 0.2$), simulated BAT for tracking



(g) HolePunch model ($\epsilon_3^{\text{col}} = 0.3$), simulated BAT for tracking

Figure 6.9: Same data as the plots in figure 6.8, projection onto x-axis of 10 $[\mu\text{m}]$ slices around the electrodes. Red = slice around readout electrodes, blue = slice around bias electrodes.

Model	Hit efficiency ϵ^{hit} [%]
Data (Sta-3E, no magnetic field) [13]	96.7
HolePunch, $\epsilon_3^{\text{col}} = 0.0$	95.7
HolePunch, $\epsilon_3^{\text{col}} = 0.1$	95.8
HolePunch, $\epsilon_3^{\text{col}} = 0.15$	96.5
HolePunch, $\epsilon_3^{\text{col}} = 0.0$	98.5
HolePunch, $\epsilon_3^{\text{col}} = 0.0$	100.0

Table 6.2: Comparison of overall hit efficiencies for full 3D sensor and simulation models. $\approx 0^\circ$ angle of incidence used everywhere.

which is something close to a Landau distribution (see Chapter 2.1). Note that equation (6.10) is valid both for the overall hit efficiency, and the hit efficiency as a function of hit position. However one should change the normalization of the 2D charge collection efficiency used in equation (6.5) so that it is normalized to unity[5], as the Landau distribution is depending on pathlength in both mean and shape (see Chapter 2.1). The overall hit efficiency for the different models is listed in table 6.2.

$$\epsilon^h = \int_{\text{threshold}}^{\infty} \epsilon \cdot P(Q)\, \mathrm{d}Q \tag{6.10}$$

Comparing the results shown in figures 6.8 and 6.9 to data is complicated by the poorly understood low-ToT response of the frontend. However it seems likely that the hole radius is approximately right, and also that the hole efficiency is smaller than $\epsilon_3^{\text{col}} \approx 0.2$.

Comparing figures 6.8(b) versus 6.8(c) and 6.9(b) versus 6.9(c) clearly shows the effect of uncertainty in hit position estimation by the telescope. Here the effect of δ-electrons is also visible, as the hit efficiency does not go completely to 0, even when $\epsilon_3^{\text{col}} = 0.0$ and in the middle of the electrodes, as seen in figure 6.9(b).

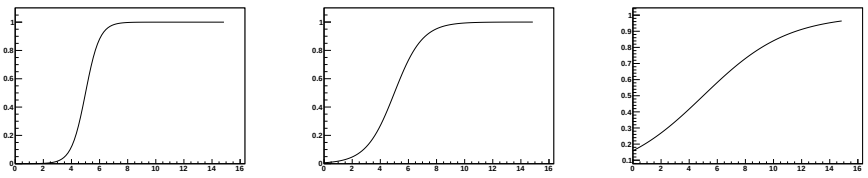
A source of error in the interpretation of this data, especially the real experimental data, is the angle distribution of the beam (discussed in Chapter 3.3). If the sensors are mounted perfectly perpendicular to the beam, as they are in the simulation, the angle distribution will cause some of the particles to travel some of the distance inside the electrodes, and some inside the depletion zone. This leads to an additional smearing of efficiency maps beyond what is caused by the tracking resolution, as seen in figure 6.9(b) (although this is probably also affected by δ-electrons). The real beam does not only have an angular spread, but likely also a non-zero mean angle. This leads to an apparent elongation of the projection of the electrodes, as seen in the analytic results in Chapter 6.2.2.1, and maybe also in figure 6.12(a).

[5] Change the path λ' to being the actual path taken by the particle.

6.2.2.2 S-curve model

Another model of the efficiency which has been less studied, is the S-curve model proposed by Vadim Kostyukhin. Here the efficiency is given by by equation (6.11), which contain two parameters: The electrode radius R and the sharpness σ. The effect of changing the new parameter σ is illustrated in figure 6.10. Here we see that in the limit of small σ, the model tends towards the $\epsilon_3^{\text{col}} = 0.0$ HolePunch model, while for larger σ the charge collection efficiency in the center of the electrode is not zero.

$$\text{eff}(\vec{r}) = \prod_i \frac{1}{1 + \exp\left(\frac{\Delta r_i - R}{\sigma}\right)} \tag{6.11}$$



(a) $\sigma = 0.5$ [μm] (b) $\sigma = 1.0$ [μm] (c) $\sigma = 3.0$ [μm]

Figure 6.10: S-curve model, charge-collection efficiency as a function of distance from electrode center in [μm] for different σ

This model shows a less peaked response in the low-ToT area, as seen in figure 6.11. However, the "hump" seen in the data is still not reproduced, as the peak is smeared out. This is also seen in the charge collection efficiency analysis, figure 6.14.

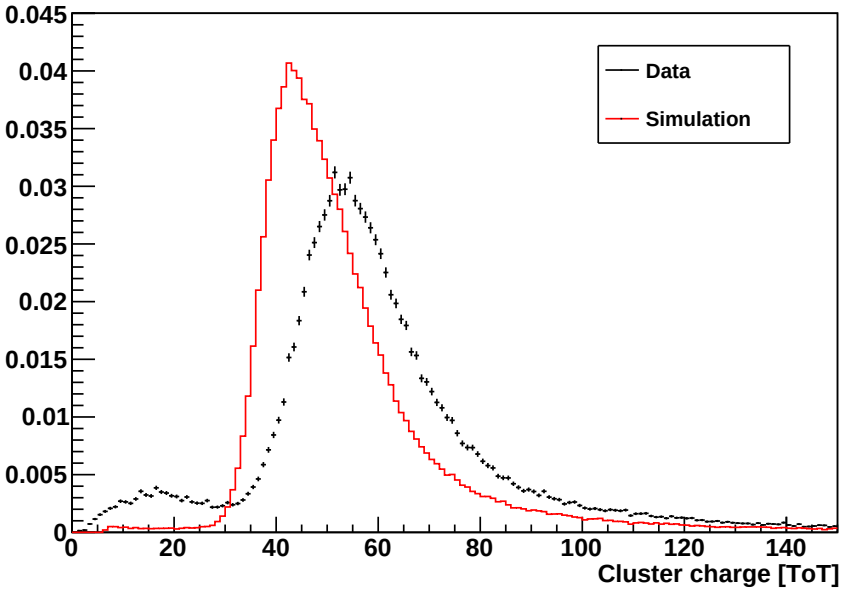
6.3 Pixel charge collection efficiency analysis

When tuning pixel simulation models, the information provided by the hit efficiency analysis (such as shown in figures 6.8 and 6.9 and table 6.2) is not sufficient. This is because the main tunable parameter in an effective model is the charge collection efficiency, of which the hit efficiency is a function (equation (6.10)).

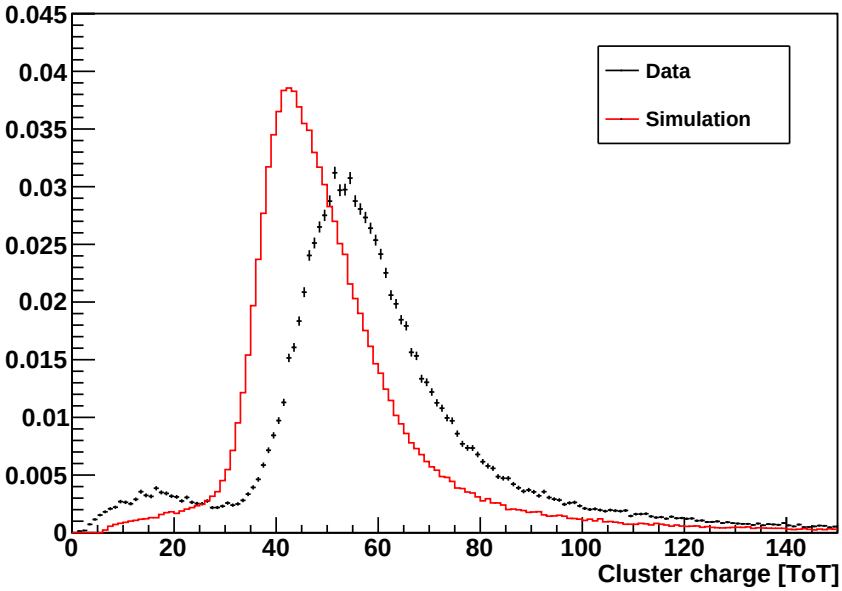
An analysis addressing this has been implemented, histogramming the $\sum$ ToT and $\sum Q$ value[6] for a hit cluster as a function of hit position within a pixel. Results from this analysis is shown in figures 6.13, 6.14 and 6.12.

Figure 6.13 clearly show that the low bump of the $\sum$ ToT spectra (figure 6.7) is created in the area around the electrodes, and the shape of the spectra in this area is also possible to recognize. This quite different in the HP model and the real data, with the HP showing a more "peaked" low-ToT bump, a feature also seen in figure

[6] If the function ToT(Q) is known for the sensor



(a) $\sigma = 0.5$ [μm]



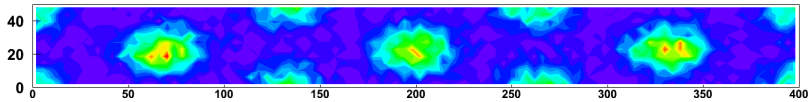
(b) $\sigma = 3.0$ [μm]

Figure 6.11: Comparison of cluster $\sum$ToT spectra for clusters matched with tracks, using data for different σ in the HolePunch model, electrode $R = 7.0$ [μm]. All curves normalized to unit area.

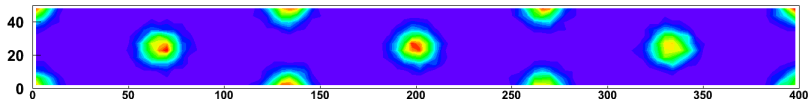
6.7. For the S-curve model (figure 6.14), this peak has been smeared out so much that the bump is no longer visible.

Another interesting feature is how the spectra changes as a function of position around the transition between column and bulk. Here the data shows a "lifting" of the bottom edge of the distribution when approaching the edge of the columns. This may indicate a smooth transition between low- and normal efficiency regions of the sensor. This is not reproduced by the HP model, while the S-curve model does reproduce it to some extent.

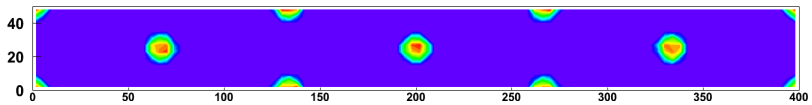
The $\epsilon_3^{\text{col}} = 0.0$ plots in figure 6.13 is clearly different from the rest of the plots, both data and simulation with higher column efficiency. As there are no efficiency in the columns, this results in that only particles passing through the electrodes at an angle so that it traverses sensitive material, or emits secondary radiation, are being registered. On the other hand, particles passing straight through the columns are not registered at all.



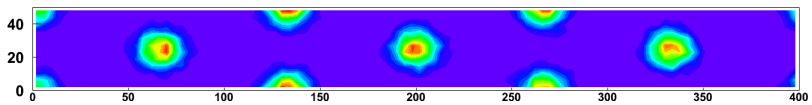
(a) Experimental data, Stanford 3E full 3D device, BAT telescope



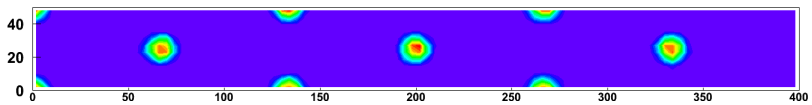
(b) HolePunch model, $\epsilon_3^{\text{col}} = 0.3$, simulated BAT



(c) HolePunch model, $\epsilon_3^{\text{col}} = 0.3$, true hit position



(d) S-curve model, $\sigma = 3.0$, simulated BAT



(e) S-curve model, $\sigma = 3.0$, true hit position

Figure 6.12: Amount of hits with $\sum \text{ToT} < 25$ and a hit cluster matching track position. All units on axis in $[\mu\text{m}]$.

Using this data, it may also be possible to make out the shape of the electrodes.

Plotting the estimated hit position within a pixel cell for tracks with deposit below a set threshold, we can find an approximate shape of the electrodes. This is shown in figure 6.12, and the electrode columns are clearly visible for both simulation and experimental data. Looking at hits below a set threshold produces better results than looking at hits above the same threshold, as the Landau distribution have a rapid rise and a long high-energy tail, meaning that high-energy hits in low-efficiency areas can fluctuate above the threshold, while the other way around is much less probable.

Comparing the electrode shape seen in the simulation data using hit position estimation from simulated BAT tracking versus true hit position, it is clear that most of the smearing seen in the experimental plot (figure 6.12(a)) is not due to the telescope resolution alone. It is not possible to conclusively say what causes the "extra" smearing from these plots alone, but from the $\sum$ ToT-spectra it is clear that the response function is more complicated than what is used in the simulation. Another interesting feature in the experimental plot is that the electrodes are not reconstructed as round, but has an oval shape. Whether this is an effect of projection due to tracks intercepting the sensor at an angle, electrodes not being identical and identically placed in different pixels, poor angular offline alignment of the sensor tilt-angle around the axis which lies in the sensor plane and normal to the long pixel direction, or something else, is up to further analysis to find out.

6.4 Conclusion

In this chapter, two models for charge collection efficiency have been presented and compared to experimental data. These models have primarily been selected for their simplicity, but are nevertheless a starting point for further study. This should include a better described $\mathrm{ToT}(Q)$ function, electronic noise, charge sharing, and crosstalk between pixels.

Further, the results strongly indicates that there are some real charge collection efficiency in the undepleted electrode columns. However, some further study is necessary to say this conclusively, especially to understand any effect of low-energy δ-electrons.

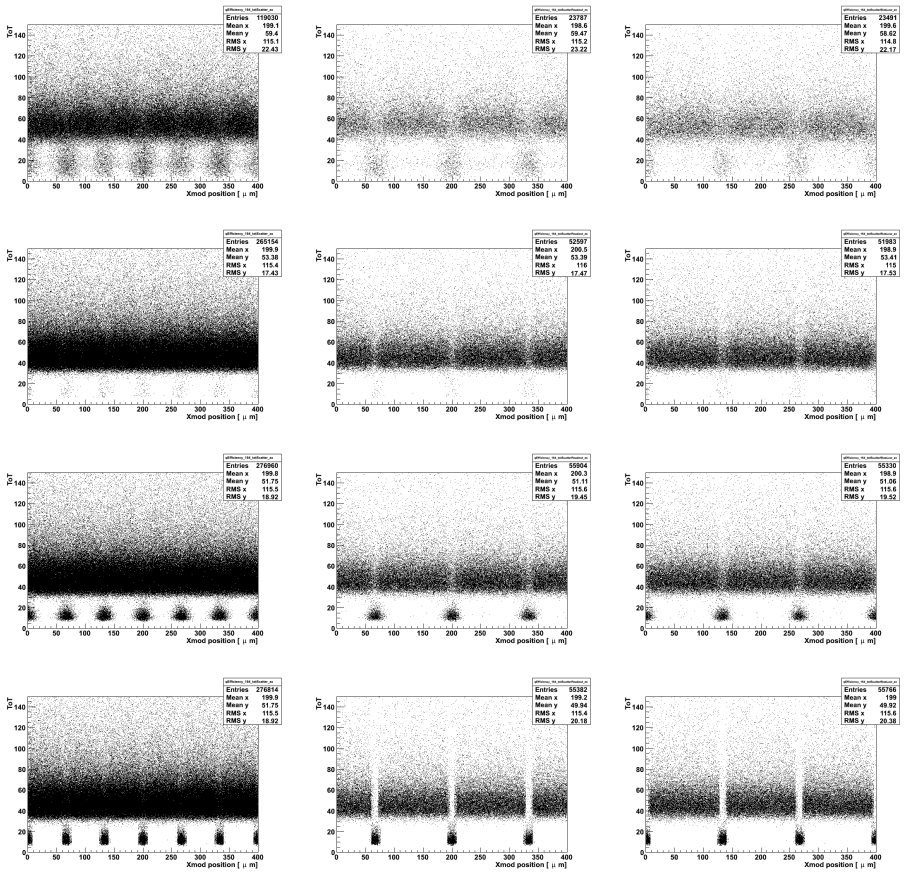


Figure 6.13: Output from pixel charge collection efficiency analysis, histograms of $\sum$ ToT as function of hit position projected onto axis along pixel long direction, for different models and data.
Left column: All data; Middle: Only hits in middle of pixel (short direction) ± 5 [μm] (Readout electrodes); Right: Only hits along long edge of pixel ± 5 [μm] (Bias electrodes).
Top row: Experimental data, Stanford 3E full 3D device, BAT telescope; 2nd row: HolePunch model, $\epsilon_3^{\text{col}} = 0.0$, simulated BAT; 3rd row: HolePunch model, $\epsilon_3^{\text{col}} = 0.3$, simulated BAT; Bottom row: HolePunch model, $\epsilon_3^{\text{col}} = 0.3$, true hit position.

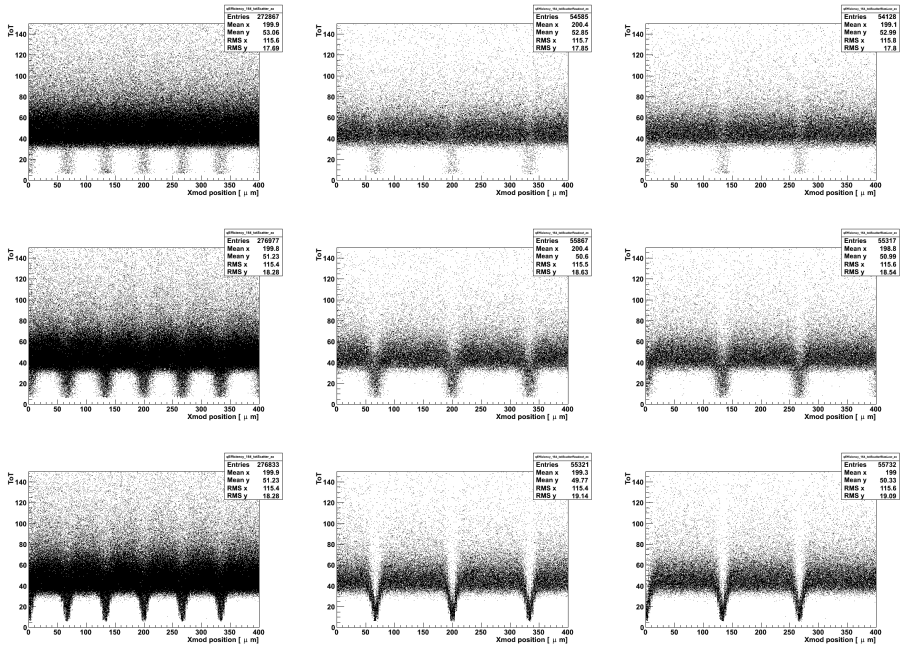


Figure 6.14: Output from pixel charge collection efficiency analysis, histograms of $\sum$ ToT as function of hit position projected onto axis along pixel long direction, for S-curve model with different parameters.
Left column: All data; Middle: Only hits in middle of pixel (short direction) ± 5 [μm] (Readout electrodes); Right: Only hits along long edge of pixel ± 5 [μm] (Bias electrodes).
Top row: S-curve model, $\sigma = 0.5$, simulated BAT; 2nd row: S-curve model, $\sigma = 3.0$, simulated BAT; Bottom row: S-curve model, $\sigma = 3.0$, true hit position.

Chapter 7

Conclusions and outlook

In the work described in this thesis, a Geant4-based full simulation system for testbeams has been developed. This system is split in two parts, respectively handling simulation of the passage of beam particles through matter, and sensor response from the energy deposited by the particles. The system has been used to model and characterize the experimental setup, particularly the Bonn Atlas Telescope (BAT), and has also been used to test models of Device Under Test (DUT) response against experimental data. This enables better understanding of the DUTs, and the modeling process also helps pointing out where our understanding of the sensors are lacking.

In order to find a model that describes the telescope sensor response well, the real hardware had to be understood. This interplay between modeling and analysis resulted not only in a detailed model describing the telescope sensor and readout system (see Chapter 4), but also a method for off-line common-mode subtraction on the telescope data (see Chapter 4.2.3.1). This method can potentially reduce the telescope hit position measurement uncertainty in future data takings with BAT, and as it is an off-line method, also makes it possible to go back and rerun the reconstruction of earlier data with better results than previously possible.

The simulation system, including the BAT response model, was then used to produce full simulation data for the testbeam setup. The simulated data was reconstructed with the normal offline alignment and reconstruction software also used for experimental data.

Having both reconstructed simulated and real experimental data presented a validation opportunity for the simulation, comparing the residual distribution between the measured hit position and the unbiased track position in the telescope plane. The simulation passed this test very well, reproducing experimental data (figure 5.1) with high accuracy. Having the simulation thus enable many opportunities for testing the assumptions and robustness of the offline tracking and alignment software in a completely controlled environment.

Further, comparisons of the estimated hit position in the DUTs with simulation truth enabled studying of the hit position error distribution in the DUTs, shown in

figure 5.5. This provided estimates for the uncertainty of the hit position estimate in DUTs, shown in table 5.1.

The simulation system itself is flexible, and can very easily be adapted to other testbeam configurations using BAT (see Appendix A and B). Currently, the ATLAS pixel R&D groups are also starting to use the EUDET telescope [27]. Thus implementing the EUDET geometry, telescope sensor response, and readout system is important for the future usefulness of the software. Modeling the EUDET is interesting for better understanding the performance of this telescope, just as it helped understand BAT. This has not yet been undertaken.

Another potential for improving the simulation system is to replace the ASCII file format currently used with a binary format. This will improve performance of detector simulation greatly, and also avoid roundoff errors that may reduce accuracy. When redoing the energy deposition file output routines, moving the Fano sampling routine (described in Chapter 3.1.1) out of the particle-matter interaction simulation and into detector simulation is also a useful improvement that will increase flexibility in detector response modeling.

Implementing propagation of the simulated particles and telescope response in a magnetic field is also interesting. This will enable studies of DUT response models under these conditions, as well as checking the performance of the off-line tracking and alignment software for particles bent in a magnetic field.

Further, the energy deposition processes needs to be understood with finer spatial resolution, both in the direction transverse to the particle's path due to low-energy δ-electrons, and also the modulation of the energy deposit density along the particle's path, which is currently described only taking into account variations around the mean deposit density along each step originating from the Fano factor. This is important for simulation-supported studies of electrode efficiency, especially of the boundary region between the electrodes and the depleted bulk, but also when studying at Double-side Double Type Column 3D sensors, which have a truly 3-dimensional geometry.

For the pixel sensor models, the results so far indicates that neither the "Hole Punch" or the "S-curve" models described in Chapter 6.2.2 satisfactory describes the data. Still, much have been learned from them, including what does not work, and how the output from the given models look from the perspective of data analysis.

An important reason for the current mismatch between the models and the data seems to be a lack of understanding of the detector charge response function $\mathrm{ToT}(Q)$, especially for low values of deposited charge Q. A good starting point for attacking this problem would be to study models currently in use for planar sensors in ATLAS simulation, as they use the same frontend. Further, none of these models currently describe charge sharing due to carrier drift-diffusion, noise, or crosstalk. This also have to be addressed in order to get a realistic model of the sensor response to high-energy charged particles.

It is worth noting that the early results from the modeling and analysis described in Chapter 6 strongly indicates that some electrode charge collection effi-

ciency is necessary when comparing to experimental data.

There are many possibilities to be explored when making new models, including the "carrier cloud tracking models" conceptually described in Chapter 6.2.1. The main problem with this type of model is that while it has the possibility of being very accurate, they are not as easy to understand and analyze as the effective models. However, it is an interesting approach that should be explored.

To sum up, the simulation system provides an opportunity for studying the output generated from different hypotethical response models, and directly compare their output to the experimental data. This enable us to better utilize the data collected in the test beam campaigns for learning how the sensors work. To do this efficiently, a more systematic and less intuition-dependent approach than what has been used in the modeling effort so far would be beneficial. This requires picking some key observable numbers, for example fit parameters to model/experimental distributions, and study how different model parameters in different models affect these observables. This can then be used to find the optimal parameters and models. Having good models are useful both for understanding how the sensors work, and is also important when simulating larger experimental setups such as ATLAS, where the objective is to gain insight in detector physics performance.

Most of the challenges described above have been taken on by participants of the ATLAS 3D Pixel R&D collaboration. It is therefore highly probable that this modeling and simulation effort will continue with greater strength, answering questions not yet solved, and uncovering new unknowns in the process.

Appendix A

TestBeamSim technical documentation

TestBeamSim is the part of the simulation system dealing with particles and their interaction with the surrounding environment – sensors and other material. It is built with C++ using the Geant4 framework [2], which provides both physics models, a skeleton for the program, and many useful utilities.

TestBeamSim is meant to be quite configurable and flexible – allowing the user to choose between multiple geometry setups at run-time, and to create new geometries using ready-made "modules", representing pixel devices, telescope planes, scintillators etc. These modules are also responsible for recording data to file, which is later used in `TbAna` (see Appendix B) and by the `TbMon` simulation extensions (see Appendix C). Another description of TestBeamSim, focusing more on physics results, is found in Chapter 3.

A.1 Installing TestBeamSim

As TestBeamSim is built using the Geant4 framework, this needs to be installed first. Installing Geant4 can be somewhat complicated (as it has its own dependencies), but the process is documented in the installation guide [11]. For development versions 9.2-patch1 and 9.3-patch1 of Geant4 was used, compiled from source on RHEL5 and Fedora 13 Linux.

When Geant4 has been installed, installing TestBeamSim is the next step. The TestBeamSim sources can be downloaded from the web (`http://folk.uio.no/kyrrens/master`), or from SVN (`svn+ssh://<CERN-user>@svn.cern.ch/reps/atlas3dpix/simulation/TestBeamSim`). You should download it into your Geant4 workdir, which is usually found at `$HOME/geant4/`. Once downloaded, compiling it should be as easy as typing `gmake` in the TestBeamSim folder. This will place the TestBeamSim executable in a place executable from your `$PATH`, ready to be run.

A.2 Configuring and running

Once Geant4 and TestBeamSim has been successfully installed and compiled, the executable is ready to be run. It's command usage is as follows:

```
$ TestBeamSim {i | <numEvents>} <preInitScript> (<postInitScript>)
```

Here the first argument decides whether to run in interactive- or batch-mode. If this argument is an "i", interactive mode is selected. Here TestBeamSim initializes itself, and then presents an interactive Geant4 prompt, where the user may enter commands starting runs, changing parameters etc. If the second argument is an integer number batch mode is selected, where TestBeamSim initializes, then generates the set number of events, and finally exits.

The second and third arguments are names of Geant4 macros; text-files containing commands as interpreted by the Geant4 prompt. The `preInitScript` is executed before Geant4 is fully initialized, enabling selection of geometry, beam, physics processes, which folder to store output files in, and more. An example `preInitScript`, mostly the same as `preInitScript.mac` (provided with the sources), is shown below:

```
#G4 setup
/run/verbose 0
/event/verbose 0
/tracking/verbose 0

#TestBeamSim params:
/TBsim/outputFolder out
/TBsim/truthWritingCut 10 GeV

/TBsim/detectorConstructionID 3
/TBsim/particleSourceID 1
/TBsim/dutAngle 0 deg

/TBsim/surfChk true

/TBsim/physicsSetupID 13
/TBsim/physicsCut 5 um
/TBsim/physicsCut_modules 1 um #Not used

#Set to negative to disable:
/TBsim/stepLimitSensor -1 um
#In units of number of e-h pairs:
/TBsim/clusterMaxCharge 400
```

This exemplifies the syntax used by the Geant4 prompt, based on "folders" (such as `/TBsim/`) containing commands (such as `/TBsim/detectorConstruct-`

ionID) which takes an argument of some form, sometimes with a unit. For a description of commands specific to TestBeamSim, see Appendix A.4.1.

The postInitScript, which is an optional argument to TestBeamSim and may be omitted, specifies commands that should be ran after initialization but before starting a simulation run or presenting a prompt. This usually contains commands to setup a visualization system, and two examples are provided with the source:

postInitScript_OGLIX.mac Creates a OpenGL ImmediateX visualization (nice for getting visual confirmation on setup, per-event display of particle tracks)

postInitScript_DAWN.mac Useful for creating high-quality illustrations.

A.3 Output

TestBeamSim produces output in the form of several ASCII files stored in a folder specified in the preInitScript. These files contain the size and position of energy deposits in sensors, truth data for particles hitting sensors, performance data, and run metadata.

ASCII was chosen over a binary format because of ease of writing and parsing[1], and because compiling an application that is linked with both Geant4 and ROOT proved difficult. It also had the advantage of being easy to inspect with a text editor etc. The main disadvantages of ASCII compared to binary storage is that it is large, that parsing is slow, and there is also a loss of numerical precision.

Software to decode the machine-readable output files (everything but metadata) is provided with TbAna (see Appendix B) and the TbMon simulation extensions (see Appendix C).

A common element in many of the output files (both modules and truth) are real 3-vectors, which are stored as follows:

```
(<real_number>,<real_number>,<real_number>)
```

Note that there are no spaces in this representation. The decimal separator in use for real numbers is everywhere "." (period).

A.3.1 Detector modules

As said in the introduction to TestBeamSim, the experiment geometry is created using "modules". These modules generally represent one detector element, and are responsible for managing their own file output. Each module writes one file, which is stored in the output folder.

All of these modules have a unique name, which is reflected in the name of the output file. There is three different kinds of modules: "BAT", "DUT", and

[1] Most of the text-parsing complexity is hidden in libraries, and Python (which TbAna is written in) provides excellent text processing abilities.

"Scint". Thus the files are named `TBsim_moduleBAT_<name>.dat`, `TBsim_moduleDUT_<name>.dat`, or `TBsim_moduleScint_<name>.dat`, where `<name>` is the unique name of the module. The format of these files is described below.

A.3.1.1 BAT and DUT: PixelSD

As both BAT and DUT modules use the same same sensitive detector, PixelSD (which is described in Appendix A.4.5), their output file format is identical. Their output is a simulation of charge-clouds created in silicon detectors, including the Fano-factor etc. This means that for each charged particle traversing the sensitive volume of the detector, it samples many charge-clouds along each step, so that in mean[2] the sum of deposited charge (in units of energy) written to file equals the ionizing energy deposit by the particle. For uncharged particles, all the ionizing energy deposit in the step is taken to be at the step endpoint.

The structure of the files is:

```
<event number> <number of records in this event>
    <record1>
    <record2>
    ...
<next event number> <number of records>
    <record1>
    <record2>
    ...
...
```

Here each record has the following format:

```
<edep> <global pos.> <local pos.>
```

The fields used are:

event number Number of simulation event; integer starting at 0 and increasing by one for each beam particle simulated.

number of records Number of lines with records (energy deposits) in this event.

edep Size of energy deposit in MeV (real number)

global pos. Position of energy deposit in the global frame, units mm (real 3-vector)

local pos. Position of energy deposit in the local frame of the sensitive volume (origo in the middle of the volume), units mm (real 3-vector)

[2] See Chapter 3.1.1

A.3.1.2 Scintillators

Scintillator modules write a much simpler data format, using the following structure:

```
<event number> <edep>
<next event number>   <edep>
...
```

The event number field is the same as for pixelSD-devices, while the edep field is the total energy deposit in the scintillator during the current event.

A.3.2 Truth

In addition to energy deposit data used for further detector simulation ("digitization") in `TbAna`, TestBeamSim also writes truth data. The file is named `TBsim_truth.dat`, and contains data about the position, momentum direction, kinetic energy, type etc. of particles transported in the setup. Data is recorded when a particle with kinetic energy above a some threshold is transported either into or out of a volume declared as a sensitive detector. The threshold is set through the TBsim messenger (see Appendix A.4.1) in the preInitScript (see Appendix A.2), command `/TBsim/truthWritingCut`.

The overall file structure is the same as written by pixelSD, but the fields in the records are different:

```
<pos> <posLocal> <momDir> <momDirLocal> <kinE> <planeID>
    <firstStep> <particleID> <particleType> <stepNum>
```

A linebreak is added to the text above in order to fit the page.

The fields used are:

pos Position of particle (global frame), units mm (real 3-vector).

posLocal Position of particle (local frame of detector), units mm (real 3-vector).

momDir Direction of particle momentum (global frame), (3-vector of unit length).

momDirLocal Direction of particle momentum (local frame of detector), (3-vector of unit length).

kinE Kinetic energy of particle, in units MeV.

planeID Module unique name (string), as discussed in beginning of section A.3.1.

firstStep Boolean encoded as "1" (true) or "0" (false), indicating if this is first step in volume (entering) or last step (exiting).

particleID Unique ID (within the current event) for the transported particle. This is an integer number from Geant4's physics engine. Note that the particleID may change in some interactions (even if there is only one particle of the given kind in the initial and final-state of the interaction), something which will also reset `stepNum`.

particleType PDG ID number of particle [14, "Monte Carlo particle numbering scheme"].

stepNum Number indicating how many steps the particle has been transported by the Geant4 physics engine.

A.3.3 Performance data

In addition to the physics data described above, some simple performance data is also written. This is usable when benchmarking the code, comparing different physics models, settings etc., and for verifying that nothing surprising happens when making changes in the code.

The file is named `TBsim_performance.dat`, and has a simple structure closely resembling that of scintillator modules:

```
<event number>      <walltime>
<next event number> <walltime>
...
```

Here the event number is as explained above, while the "walltime" is the wall clock time used for processing the event in microseconds.

A.3.4 Metadata

In addition to the files mentioned above, which are meant to be parsed by machines, a metadata file named `TBsim_metadata.dat` is written. This file contains the settings used for the current run (along with text-descriptions for "numeric switches"; see Appendix A.4.1), geometry information, the random number seed used for the run, some rudimentary performance data, and version number of TestBeamSim.

A.4 Internals

As stated in the introduction, TestBeamSim is meant to be easily extendible by anyone who knows C++ [36] and Geant4 [10]. This section assumes that C++/Geant4 basics are known, and describes the internal structure of TestBeamSim and how to extend it.

The basic structure of all Geant4 programs is based around interface classes provided by Geant4, which plugs into the Geant4 `RunManager` – the "kernel"

of Geant4. `G4RunManager` is responsible for connecting the different classes representing different parts of the simulation, calling defined functions in them at the right time, and for controlling simulation runs, events, and steps.

As in any C++ program, there exists a `main()` function, which is located in the file `TestBeamSim.cc` in the root directory of the sources. This parses command line arguments, uses the messenger to parse the `preInitScript`, and then starts setting up the program. This includes setting up the correct geometry, physics list, and primary generator action (particle source), initializing the random number generator[3], and `UserActions` for writing truth and performance data. When finished initializing these classes, `main()` kicks off a `postInitScript` if one is specified by command line argument, and then either goes into interactive mode or runs the number of events specified by command line argument. When this is done (end of run or `exit` command / `control-D` if interactive), it writes the metadata file, cleans up, and exits.

The rest of TestBeamSim is a set of classes, most of them specific implementations of Geant4 interfaces. Of these there is three main groups:

UserAction These classes are called at specific points of the simulation, such as the start and end of a run, start and run of every event, at each simulation step, generation of primary particle(s) etc.

UserInitialization These classes are used when initializing the simulation, and provides descriptions of the geometry, defines which physics processes are in use, and more.

Other classes In addition to the classes mentioned above, there are other things such as messengers, sensitive detectors, hits, physics processes, and much more.

The implementation of these used in TestBeamSim is described in the subsections below. Note that there are some dependencies restricting the order of initialization; especially `TestBeamSimEventAction` has to be initialized early as it serves as an "information hub" for other classes. Also the messenger has to be initialized before running the `preInitScript`, which has to be done before initializing anything else (as this contains options etc. used by these classes, and in some cases even decides which version of the classes should be used).

A.4.1 Messenger

The messenger class `TestBeamSimMessenger` is implemented in the files `TestBeamSimMessenger.hh` and `TestBeamSimMessenger.cc`. It is a single-

[3]TestBeamSim uses the `RanLuxEngine` from `CLHEP`. It is initialized at the beginning of the run by use of the number of seconds since the UNIX Epoch (00:00 UTC, 1/1/1970).

ton[4] class, and is responsible for setting up the `G4UIcommands` that are used to control the simulation from the `preInitScript`[5]. This includes the `G4UI-directory` `/TBsim/`, and all the contained commands.

The commands defined are:

outputFolder Allows selection of where to store the output files (path name, relative or absolute).

detectorConstructionID ID number (integer) of wanted detector geometry.

physicsSetupID ID number (integer) of wanted physics list.

physicsCut Default production cut (expected stopping length).

physicsCut_modules Production cut in detector modules (expected stopping length) (set to negative if just using default cut values everywhere) (Parameter currently ignored by the simulation).

stepLimitSensor Stepping limit within position sentitive volumes (set to negative if not in use).

clusterMaxCharge Max number of electrons in a charged-particle energy deposit.

particleSourceID ID number (integer) for particle source wanted.

truthWritingCut Minimum energy of particle for writing truth info.

surfChk Run surface check during initialization? (produces lots of output) (true/-false).

dutAngle DUT rotation angle about y-axis.

For the allowable values of the commands, and description of the identifiers, please see the source code.

In addition to the commands themselves, two getters are provided for most commands: One for getting the set value itself. with units etc. calculated in. The other getter returns a `G4String`[6] containing a nicely formatted version of the set value including units etc. The string getter is useful when printing or logging the current settings, and is defined for all commands operating on non-trivial datatypes (number with a unit, vectors, etc.). For numeric IDs, the string also contain a short description text.

If implementing a new command, its value should be printed from `Test-BeamSimMessenger::Print()`, and be written to the metafile (end of `main()`).

[4] That a class is singleton means that there exists only one instance of the class. To get class instance, use the class method `TestBeamSimMessenger::Instance()`, which will return a reference to the existing instance, or create a new one if one does not exists. Do *not* use the constructor directly!

[5] See Appendix A.2 for an example `preInitScript`.

[6] `G4String` is derived from the Rogue Wave implementation of `RWCString`.

A.4.2 Physics

As Geant4 is intended to be as general as possible, the set of physics processes used is not preset; but code for simulating many processes are included with Geant4. This leaves it up to the application developer to pick and match processes and particles, and to define his/her own processes or particles.

To get this right is a tedious and error-prone task, but luckily there are also quite a few ready-made physics-lists included, which reduces the problem down to selecting a best possible physics list. In TestBeamSim, this is mainly done through `TestBeamSimPhysicsListFactory`, implemented in the files `TestBeamSimModularPhysicsList.hh` and `TestBeamSimModularPhysicsList.cc`.

The factory class `TestBeamSimPhysicsListFactory` sets up and returns one of the applicable predefined Geant4 physics lists. Which one is used is depending on the numeric identifier `physicsSetupID` from the messenger. In addition, an extra physics process constructor `StepLimiterBuilder` is added to the lists. This enables limiting the step length for charged particles in the silicon detector volumes through the use of `G4StepLimiter`. Step limiting was originally done in order to get more samples for simulating detector response, but this way of sampling has later been obsoleted by the Fano sampling routine in `TestBeamSimPixelSD`.

A.4.3 DetectorConstruction

Probably the most interesting class for anyone adapting TestBeamSim to a new testbeam geometry is the class `TestBeamSimDetectorConstruction`, defined and implemented in the files `TestBeamSimDetectorConstruction.hh` and `TestBeamSimDetectorConstruction.cc`. This class defines the geometry the simulated experiment. Currently, this is done monolithically: There is one large class with which defines all geometries. Several geometries are implemented, and which is used is decided by the numeric identifier `detectorConstructionID` from the messenger. Currently, all geometries are implemented in `TestBeamSimDetectorConstruction`'s constructor, which is unfortunately structured as one gigantic if-else.

The detector geometries are constructed with the usual Geant4 constructs (solids, logical- and physical- volumes) for parts such as tables, mountings etc., and with `TestBeamSimDetectorModules` (described below) for the active detectors. This enables rapid and flexible description of new experimental setups.

A.4.4 Detector modules

Detector parts are often common between different test-beam setups, and also the part of the setup with the most detailed description. It is therefore a good idea to share this code between different setups, and this is accomplished by the use of "detector modules".

A detector module is a class inheriting the interface `TestBeamSimDetectorModule`, defined and implemented in the files `TestBeamSimDetectorModule.hh` and `TestBeamSimDetectorModule.cc`. This interface defines two important virtual functions: `TestBeamSimDetectorModule (G4LogicalVolume* LV_parent, G4Transform3D placement, G4String name)` which builds the module, and `virtual void process_event (const G4Event* event)` which is a purely virtual method started at the end of each event, responsible for writing any data to file. In addition, a virtual constructor should be implemented for closing files and cleaning up memory.

The arguments taken by the constructor are things needed for all modules. `placement` is the wanted position and orientation of the center of the active sensor volume. `LV_parent` is the logical volume which the detector module should be placed in. Note that the position specified by `placement` is relative to the logical volume. `name` is the unique identifier discussed in Appendix A.3.1.

When implementing a new detector module, remember to call the interface constructor from your constructor. This is best done when implementing your constructor, using the syntax shown below:

```
1 Inheriting_module::Inheriting_module(G4LogicalVolume* LV_parent,
2                                      G4Transform3D placement,
3                                      G4String name,
4                                      G4type myArg1,
5                                      G4type myArg2) :
6       TestBeamSimDetectorModule(LV_parent, placement, name) {
7   // Actual implementation
8 }
```

The actual constructor implementation contains the code for setting up materials, creating the geometry by placing volumes at positions determined by `placement`, inside the volume `LV_parent`, opening output file, setting up the sensitivity, and installing a step limiter if this is asked for. These classes also print some info at the end of initialization.

The interface class is used for tasks common to all modules, such as registering them with the `EventAction`, which calls `process_event()` on all modules at the end of each event. It also keeps some common data fields, which is useful to be able to access in a unified way across different modules:

LV_parent Parent logical volume, as described above.

placement Position and orientation for center of sensitive part of module.

name Unique name of module.

fileName Use this field to store the name of the output file in use, including folder name (full or relative path).

ofile File stream for output file.

eventAction Pointer to `TestBeamSimEventAction` class in use.

PV_sensitive Physical volume with sensitive detector

PV_envelope Physical volume of an "envelope", if this is present. This is by default set to `NULL`.

Some of these common fields has to be set explicitly by the implementing class constructor.

Three main detector modules are provided with the TestBeamSim sources, all named `TestBeamSim_<modName>`, where all possible `<modname>` are listed below. The scintillator is provided in the same files as the interface class, while version 2 of the BAT- and DUT- modules (see also figure A.1) is defined and implemented in the files `TestBeamSimDetectorModules_v2.hh` and `TestBeamSimDetectorModules_v2.cc`.

scintillator This defines a (plastic) scintillator, which records the energy deposited in this event. The size of the scintillator, and whether it has a hole in the middle, is controlled by arguments to the constructor.

BAT_v2 This defines a single BAT plane

DUT_v2 This defines a single DUT (pixel device under test). The thickness of the sensitive part, whether the PCB has a hole, and whether there is an aluminum bias tab is controlled by arguments to the constructor.

For more information on these devices models see Appendix A.3.1, and Chapters 3.4, 4, and 6. In addition to the "version 2" DUT- and BAT- models mentioned, there exists version 1 variants as well. These use a slightly different geometry[7] and a simpler sampler (`TestBeamSim_BAT_SD`) that only write energy deposition samples at `preStepPoints`.

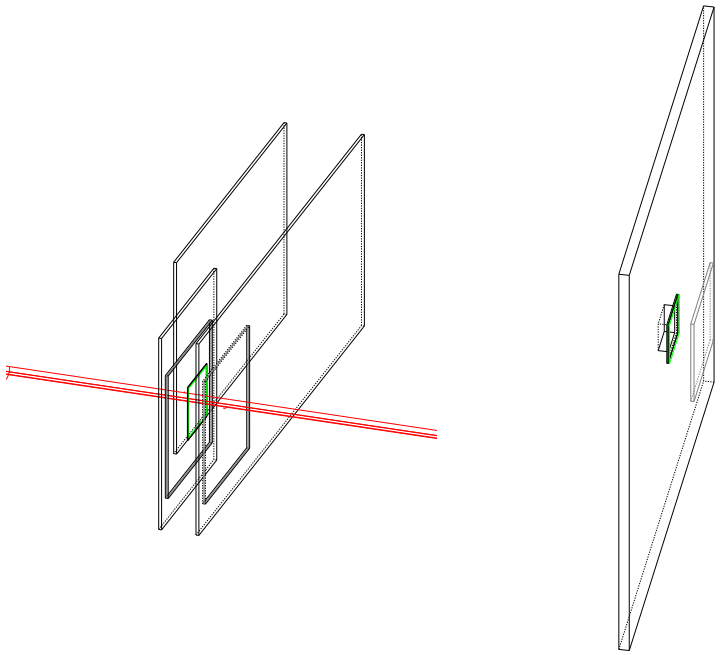
A.4.5 PixelSD sensitive detector

This class is used for handling "hits" in the position-sensitive silicon detectors `BAT_v2` and `DUT_v2`. It is derived from the interface `G4VSensitiveDetector`, and is defined and implemented in the files `TestBeamSimPixelSD.hh` and `TestBeamSimPixelSD.cc`.

As stated in Appendix A.3.1.1, the output of this sensitive detector is a set of energy depositions and their positions in both the global- and local-[8] coordinate systems. For charged particles, these energy deposits are created by Fano-sampling along each step taken within the sensitive volume, while for uncharged particles the energy is taken to be deposited at the step endpoint.

[7] BAT version 1 has a more detailed geometry, but it is not entirely correct about sensor placement relative to the rest of the chassis. DUT version 1 are slightly off with sensor size, and does not feature a plastic cover. Neither features the air envelope originally intended to be used to define an area of shorter physics cutoff. There are other differences as well.

[8] Sensitive-volume-local coordinates: Coordinate system specified by `placement` argument to detector module. Centered in middle of the sensitive volume, rotated along with it.



(a) Bonn Atlas Telescope (v.2) (b) Device Under Test (v.2)

Figure A.1: Detector modules used in simulation

Fano sampling is discussed more thoroughly in Chapter 3.1.1. In short, when a charged particle makes a step through the sensitive volume, it deposits energy because of interactions with the material, determined by the active physics processes.

For charged particles this deposit is divided up into N samples, which is spread uniformly along the step. The number of samples N is calculated from equation A.1.

$$N = \left\lceil \frac{\Delta}{m} \right\rceil \tag{A.1}$$

Here Δ is the ionizing energy deposit from Geant4, and m the maximum allowed mean cluster charged (converted to units of energy).

The size of each single energy E'_k deposit is then sampled from a normal distribution $\mathcal{N}(\mu, \sigma^2)$, with parameters as defined in equation A.2.

$$\mu = \frac{\Delta}{N} \quad , \quad \sigma^2 = F\mu \cdot w \tag{A.2}$$

Here F is the Fano factor, which has a value $F = 0.1$ for silicon [8]. w is the mean energy deposit required to produce an ion pair, which according to [8] has a value $w = 3.62\,[\mathrm{eV}]$ in silicon. These values are stored as the class variables `fanoFactor` and `ePair`.

For uncharged particles there is only one deposit, which is taken to be at the end of the step. This is also smeared by the Fano factor, so the effective energy deposit

is sampled from a normal distribution with parameters as defined in equation (A.3).

$$\mu = \Delta \quad , \quad \sigma^2 = F \cdot \Delta \cdot w \tag{A.3}$$

Due to a bug in the original implementation, which is discussed at depth in Chapter 3.1.1.1, a correction had to be made without invalidating the BAT device model tunings. Thus a boolean flag `sampleBug` was added (with default value `false`) to the constructor of `TestBeamSimPixelSD`. Setting this to `true` selects the old and incorrect sampler, and this is done in the constructor of `TestBeamSimDetectorModuleBAT_v2`

A.4.6 UserActions

UserActions are classes that are called by the Geant4 RunManager at specific points of the simulation. They may perform tasks that are critical to the simulation (such as creating primary particles), or they may do other user-defined tasks such as writing output data to files. The UserActions used in TestBeamSim is described below.

A.4.6.1 PrimaryGeneratorAction

Three PrimaryGeneratorAction classes are available in TestBeamSim, and are selectable by the preInitScript messenger command `/TBsim/particleSourceID <number>`. Both `GausGun` and `PointGun` are defined and implemented in the files `TestBeamSimPrimaryGeneratorAction.hh` and `TestBeamSimPrimaryGeneratorAction.cc`.

PointGun (ID 2) This particle source produces negative pions with kinetic energy $E_k = 180[\mathrm{GeV}]$, momentum direction parallel to the z-axis, and initial position at $(-5, 0, 0)[\mathrm{m}]$. This is mostly useful as a help for checking alignment and geometry.

GausGun (ID 1) This particle source is similar to the PointGun, but the particles have a randomized initial position and momentum direction. The PDFs used for the initial position is given in polar form as equation (A.4), while the PDFs for the initial momentum direction is given by equation (A.5).

$$P(\phi) = \frac{1}{2\pi}\,, \quad P(R) = \begin{cases} \mathcal{N}(0, \sigma_R^2) & \text{for } R \leq R_c \\ 0 & \text{for } R > R_c \end{cases}, \quad P(z) = \delta(z - z_0) \tag{A.4}$$

$$P(\varphi) = \frac{1}{2\pi} \qquad \text{and} \qquad P(\theta) = \begin{cases} \mathcal{N}(0, \sigma_\theta^2) & \text{for} \quad \theta \leq \theta_c \\ 0 & \text{for} \quad \theta > \theta_c \end{cases} \tag{A.5}$$

Here the angular coordinate $\phi = \arctan(y/x)$ of the initial position (x, y, z) follows a flat distribution, while the radius $R = \sqrt{x^2 + y^2}$ is distributed according to a truncated normal distribution. For the initial momentum direction, the we see

that the angular coordinate follows a flat distribution, while the azimut angle is distributed according to a truncated normal distribution.

The parameters for the truncated normal distributions (σ and cutoff) are defined in the user `GausGun` class constructor. The truncated normal distribution itself is generated with a Gaussian RNG and a Von Neuman accept-reject Monte Carlo method [19].

GeneralParticleSource (ID 3) This particle source, which is included with Geant4[25], enables the user to setup all the parameters from the `preInitScript`. It is included for doing radioactive source test simulations within the TestBeamSim framework. To do this, one must also include `G4RadioactiveDecayPhysics` in the physics list.

To setup a ^{90}Sr radioactive source at the global origo, one could for example use the following commands in the `preInitScript`:

```
#General Particle source config (90Sr)
/gps/position 0 0 0
/gps/energy 0 keV
/gps/particle ion
/gps/ion 38 90 0 0
```

Note that this does not take into account source material effects.

(^{90}Sr is a β-emitter). Further, ^{90}Sr will decay to the β-unstable isotope ^{90}Y within the same Geant4 event. As the half-life of ^{90}Y is of the order of days, these decays will not end up within the same trigger window, and a mechanism for separating energy deposits according to its "mother ion" is therefore necessary for accurate simulation of radioactive source tests, but is currently not provided with TestBeamSim[9].

A.4.6.2 RunAction

Methods in this class, defined and implemented in the files `TestBeamSimRunAction.hh` and `TestBeamSimRunAction.cc`, are called at the beginning and end of every run. In TestBeamSim this is only used for calculating the mean wall-time per event, which is reported in the metadata output file.

A.4.6.3 EventAction

Methods in this class, defined and implemented in the files `TestBeamSimEventAction.hh` and `TestBeamSimEventAction.cc`, are called at the beginning and end of every event. In TestBeamSim this is used for collecting performance data, notifying other classes about the end of event so that they can write

[9]However, it is implemented in another Geant4-based program called `EdepSpectra`, which is available at `svn+ssh://svn.cern.ch/reps/atlas3dpix/simulation/EdepSpectra`

their data to file, and for searching for detector module given a pointer to its sensitive volume.

This role as a information hub and notifier makes it necessary that the EventAction is initialized before any detector modules and before the SteppingAction.

A.4.6.4 SteppingAction

Methods in this class, defined and implemented in the files `TestBeamSimSteppingAction.hh` and `TestBeamSimSteppingAction.cc`, are called at the end of each step. This is used for writing the truth file described in Appendix A.3.2.

The SteppingAction is depending on EventAction for notifications about end of event, and for mapping sensitive volumes into detector module unique names.

A.5 Benchmarking

As a rule of thumb one can expect a mean walltime of $0.1[\mathrm{s}]$[10] pr. event when using May2009 alligned geometry, QGSP physics, and Fano sampling. For a detailed benchmark comparing different physics models, see Chapter 3.1.2.

[10] On `stau.uio.no`, a $3.00[\mathrm{GHz}]$ dual-core Pentium D with 4 GB of RAM, running Geant 4.9.2-p1 on RHEL5.5

Appendix B

TbAna technical documentation

TbAna is a collection of Python scripts and libraries, which forms a package capable of many things:

- Handling and analysis of `TestBeamSim` output
- Trigger- and detector-response simulation
- Tuning of detector response simulation parameters
- Encoding/decoding of raw detector data (BDT files)
- Automated production runs of `TestBeamSim`
- Visualization and pretty-plotting of results

Most of the general logic, such as I/O of different file-formats, are handled by library functions. This means that it is relatively trivial to try out ideas, make new plots etc. without duplicating common/core functionality.

B.1 Installation

In order to install TbAna, there are some dependencies which needs to be fulfilled first. These include the following programs

- Python (version 2.5 on RHEL5 used for development)
- ROOT with Python support (version 5.24 used for development)
- numpy and scipy for some special math functions and array datatypes

When these are successfully installed, TbAna may be downloaded from the web (`http://folk.uio.no/kyrrens/master`), or from SVN (`svn+ssh://<CERN-user>@svn.cern.ch/reps/atlas3dpix/simulation/TestBeamSim`). Install the folder `TestBeamSim-analysis` somewhere on your computer, and it should be ready to use.

B.2 Initial configuration and overview

Inside the folder `TestBeamSim-analysis`, there are three subfolders (`bin`, `lib`, and `plotters`), containing a lot of `.py`-files. In addition to this, there is a file `pathsetup.sh`. This file setups the current shell for running the TbAna software, and contains the path of the installation directory. It typically looks similar to what is shown below:

```
1 #!/bin/bash
2 export PATH=~/master/TestBeamSim-analysis/bin:$PATH
3 export PATH=~/master/TestBeamSim-analysis/plotters:$PATH
4 export PYTHONPATH=~/master/TestBeamSim-analysis/lib:$PYTHONPATH
```

For it to work on your setup, change the installation path `~/master/TestBeamSim-analysis` to the one used in your setup. The setup is then simply done by sourcing this script, as in:

```
$ source ~/master/TestBeamSim-analysis/pathsetup.sh
```

When this is done, TbAna is ready to run, which can be verified by checking that the TbAna executables show up:

```
$ Tb <hit TAB key twice>
TbAna_bdtHistos.py                          TbAnaPlot_bdtHistos_compareHistos.py
TbAna_bdtPrint.py                           TbAnaPlot_bdtHistos_etaPlot.py
TbAna_clusterBarchart.py                    TbAnaPlot_driftTimePlot.py
TbAna_digitizer.py                          TbAnaPlot_physicsEngine.py
TbAna_digiTune_BAT.py                       TbAna_test1.py
TbAna_globalFit.py                          TbMonPlot_chi2compare.py
TbAna_hitposEstimators.py                   TbMonPlot_histoCompare.py
TbAna_physicsEngine.py                      TbMonPlot_landauEdep.py
TbAnaPlot_bdtHistos_CMCcorrPlot.py          TbSim_runMaker.py
```

As we see, TbAna is composed of several executable Python files. These can then be called from everywhere in the filesystem, and it therefore simple (and advisable) to keep the data separate from the TbAna installation directory.

The three subfolders `bin`, `lib`, and `plotters` contain python files with different functions:

bin This directory contains Python files meant to be executed as command line programs, controlled by command line arguments. Common for these programs is that they are mostly geared towards data processing and analysis.

plotters These Python-files are also executables. They differ from those in `bin` by purpose – these use histogram .ROOT files generated by other analysis to make pretty plots for presentation, printing etc. Files named `TbAnaPlot_*` are used for pretty-plotting output from TbAna, while `TbMonPlot_*` are used for pretty-plotting output from `TbMon`.

lib These Python-files contain functions common to the programs in `bin`, such as file I/O and detector simulation models.

B.3 Using TbAna: Important executables

As stated above, TbAna contains many executable programs, performing various tasks related to test beam simulation and analysis. Only the most important ones are listed below, but they are all documented with usage-prompts and comments in the sourcecode.

Note that many of these programs operate on runs from `TestBeamSim`. These programs then generally takes the path to a "workdir" folder containing the data (see Appendix A.3 for details), and often assumes that the filenames and device positions are the same as in the `TestBeamSim` geometryID #3 (May2009 with aligned devices).

B.3.1 TbAna_digitizer

This program uses simulated energy deposits from `TestBeamSim` to simulate detector response and trigger decisions. The output is then written to the file `bdtOut.bdt`, which is saved in the same directory as the input data. This file contains simulated detector digits from BAT and DUTs, formatted identically to the BAT DAQ system. It also writes out a file `digiSyncMap.dat`, which is used by `TbMon` for synchronizing reading events by trigger number (as in the BDT) with primary-particle-counter event number from the underlying Geant4 simulation.

It's command usage is as follows:

```
$ TbAna_digitizer.py <workdir> { <maxEvents> }
```

Here `<workdir>` is the data directory containing the .dat-files, while the voluntary argument `<maxEvents>` is used to limit the number of simulated triggered events.

B.3.1.1 Operation as a library

TbAna_digitizer is special in that it is also usable as a Python module (library), because all the "working" code is encapsulated into one function, defined as:

```
1  def MakeDigits(workdir, maxEvents = 0, BATconfig=None, TPLLconfig=None,\
2                 dummyID_BAT = -1, dummyID_TPLL = -1, writeSyncMap=False):
3      """
4      Function that actually makes the digits,
5      either called from this modules main(),
6      or called from another module in batch processing.
7
8      If dummyID == -1, don't use dummy digitizers
9      If dummyID != -1, use dummy digitizer on all OTHER modules
10                       than the one with headerName=dummyID
11     if dummyID == -2, use dummy digitizer on ALL modules of this class
12
13     If writeSyncMap=True, output a file digiSyncMap.dat in the working dir,
14     containing a mapping of BDT event numbers -> TestBeamSim eventNumbers.
```

```
15     This is then read by the simBaseBuilder in tbmon.
16     """
17     ...
```

This means that it is simple to automate digitization of a run, and this is done in `TbSim_runMaker.py` (see Appendix B.3.2) and `TbAna_digiTune.py` (Appendix B.3.3).

B.3.1.2 digiSyncMap.dat file output format

The purpose of this file is to enable access of "raw" simulation data (truth and energy deposits) from `TbMon`. To do this, one must remember that the `.bdt` files output from `TbAna_digitizer` does not contain all events from `TestBeamSim`. This is due to that simulation events are corresponding to primary particles, but not all primary particles produces a trigger. When this happens, we get a desyncronization between event numbers in the `.bdt` file and the `TestBeamSim` output. It is therefore necessary to have a mapping between those event numbers, so that when we in `TbMon` can re-syncronize the two data streams.

The simple ASCII file format is as shown below, with event numbers counting from 0 in integer steps, and BDT event number always increasing by 1 each line:

```
<BDT event number> <TestBeamSim event number>
<next BDT event number> <next TestBeamSim event number>
...
```

B.3.2 TbSim_runMaker

The purpose of this program is to run the Geant4-simulation (`TestBeamSim`) several times; either with the same `TestBeamSim` configuration for generating large amounts of statistics using the same configuration, or one run per physics list for comparing them.

For generating statistics, its command line usage is:

```
TbSim_runMaker.py <runNumberStart> <numRuns> <numEvents>
   {physicsSetupID=13} {clusterMaxCharge=400}
   {dutAngle=0.0} {physicsCut=5.0}
```

For doing a physics list scan (lists 11-16), the command line usage is:

```
TbSim_runMaker.py PhysScan <runNumberStart> <numEvents>
   {clusterMaxCharge=400} {dutAngle=0.0} {physicsCut=5.0}
```

Here `<runNumberStart>` is the ID-number of the first run generated, `<numRuns>` the number of runs to generate, and `<numEvents>` the number of primary particles to simulate in each run. The arguments shown as `{optionName=defaultValue}` are optional, and may be omitted. They correspond to

configurations in the `preInitScript` used for `TestBeamSim` (see Appendix A.2).

In addition to running `TestBeamSim`, it will run `TbAna_ditigizer.py` and move all the resulting `.bdt`-files into a common folder named `bdt` when in "statistics mode", while it will run `TbAna_physicsEngine.py` when in "PhysScan" mode.

B.3.3 TbAna_digiTune_BAT

This program is used for finding the optimum BAT configuration, as described in Chapter 4. This is done by running the digitizer (using the "library mode" described in Appendix B.3.1) several times over the same data, using several configurations/-tunes. The resulting `.bdt`-files are then analyzed using `TbAna_bdtHistos-.py` (described in Appendix B.3.4), and key distributions from the simulation and from a reference real-data BDT is plotted on top of each other. The plots are then presented as a web page, sorted by the digitizer configuration/tune used to make them, as shown in figure B.1.

The output is a folder containing a report in html format named `report-.html`, the plots as `.svg` vector files, the reference `.bdt.root` file, a `.root` file with all the canvases used (for zooming or other processing), all created `.bdt` and `.bdt.root` files, and a logfile that can be useful for debugging.

The command line usage of the program is (typed as one single line):

```
$ TbAna_digiTune_BAT.py -r <RefFile.[bdt | root]>
    -d <RefDevice> -o <OutputName> {-n <NumEvents>}
    {-s <N|P|B>} {-c "comment"}
    -p:param1 <value1> -p:param2 <value2> ...
```

Here one needs to specify a reference file in BDT or ROOT format[1], a reference device (BAT plane BDT name – 1,3, or 6 in May2009 data), and the name of a folder to store the output in.

Further, one usually want to change one or more of the parameters to the BAT digitization. This is done by using the `-p:param value` interface, where `value` can be a single value or describe a linear or logarithmic range[2]. Zero or more parameters can be set, and unset parameters will assume the default value from `TbDigitizerConfig.py::TbDigitizerConfigBAT_v3`.

When ranges are specified, parameter tuning points for each combination of the parameters are used, labeled $[p_1, p_2, \ldots, p_n]$. Here p_i is an index into the range of parameter i, and n the total number of specifiable parameters.

In addition to the mentioned arguments, `TbAna_digiTune_BAT.py` can also use arguments `-c` for providing a comment[3] to the current tuning run (for

[1] If BDT data is specified for reference data, it will be analyzed by `TbAna_bdtHistos.py`, which will create a ROOT file with the relevant histograms

[2] For information about the syntax, see the program usage string

[3] To provide a multi-word string as a comment, enclose the string in quotation marks.

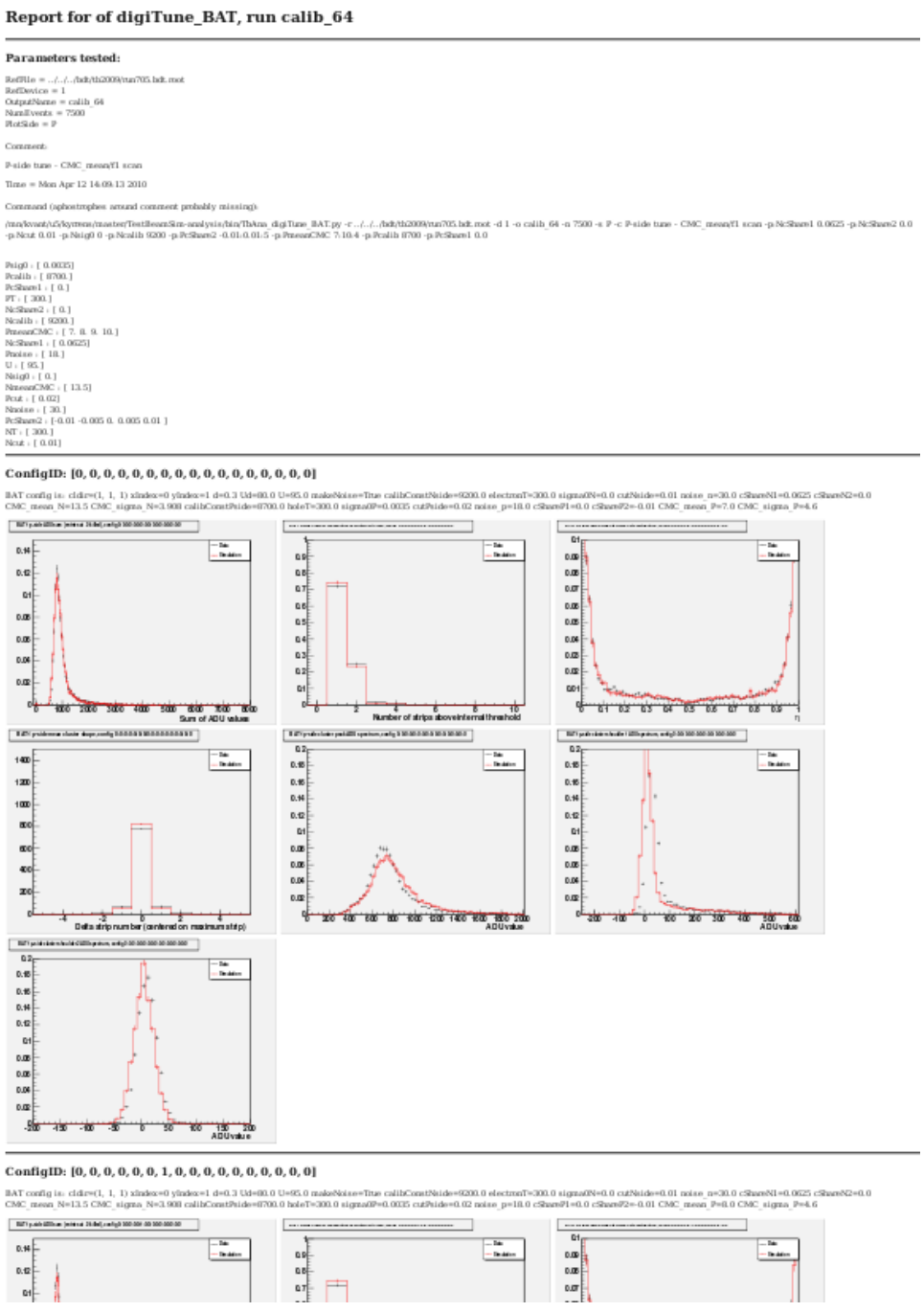


Figure B.1: Screenshot of example webpage created by `TbAna_digiTune_BAT.py`

remembering the purpose of the tuning run), and `-s` for only showing plots for N- or P-side in the HTML report.

B.3.4 TbAna_bdtHistos

This program directly analyzes BDT files, and outputs a .root file containing histograms. The command line usage is:

```
$ TbAna_bdtHistos.py <BDTfile> {--batch} { <maxEvents>=0 }
```

If `--batch` is not used, a `TBrowser` is shown at the end of processing, allowing the user to click through the plots.

The output of `TbAna_bdtHistos` is a .root file with the same name as the input .bdt file, with .root appended. This file contains histograms from BORE and DATA parts of the BDT file, both BAT and PLL.

B.3.5 TbAna_physicsEngine

This program is used for comparing different physics lists, as described in Appendix A.4.2. As input it uses a folder of output files from `TestBeamSim`, and as output it produces a root file with histograms with information about what happens at each module (energy deposit, number of hits per event for `PixelSD`s and kinetic energy of incident particle), and performance data (walltime per event, running average graph of walltime per event). These root files are useful for comparing different physics models, and `TbMonPlot_physicsEngine.py` can be used for plotting the distributions for sets of physics models on top of each other. The command line usage is:

```
$ TbAna_physicsEngine.py <workdir> {<maxEvents>} {--batch}
```

Here the optional argument `--batch` disables using a `TBrowser` to click through plots at the end of processing.

B.3.6 TbAna_globalFit

This program works as a simple tracking program. It makes "hits" in the planes by clustering raw energy deposits based on proximity, and the energy-deposit weighted mean of each cluster is taken to be the x-y hit position of the cluster[4]. One may optionally enable smearing of the hits (convolution of the hit position with a gaussian).

After making the hits in the local plane of the modules, it uses knowledge of the position of the telescope planes (hard-coded into the program) to make a straight-line global fit of the track. This track is then extrapolated into the DUT planes and compared to the estimated hit position there.

[4]This is done by the "digitizer" `TbDigitizerCluster`, found in `lib/TbDigitizers.py`

The program also simulates trigger the same way as `TbAna_digitizer.py`, requiring that the energy deposit in the trigger scintillators is above a certain threshold, and that the deposit in the veto scintillator is below a similar threshold.

The output of the program is a `.root` file containing histograms of the track parameters, the residuals in all planes (x-, y- direction and $\sqrt{x^2 + y^2}$), and squared sum of pulls in the same directions. This output was used to estimate the telescope resolution in the DUTs for a Spåtind 2010 talk [32].

Command line usage:

```
$ TbAna_globalFit.py <workdir> {<maxEvents>}
```

B.4 Libraries

In order to avoid code duplication, the TbAna package includes Python libraries for common functions like data I/O and digit generation. This is the files in the `lib` subdirectory. There are a few different "sublibraries", which are mostly independent of each other, and described in the sections below.

B.4.1 Parsing of raw simulation data

An overview of the classes related to simulation data parsing and datastructure is shown in B.2.

The classes inheriting from interface[5] `TbModule` are parsing output files from `TestBeamSim`, with different implementations reflecting different kinds of `TestBeamSim` output files (described in Appendix A.3). These are used by giving the name of the datafile to read as an argument to the classes constructor. One can then call `ReadNextEvent()` to read one event further from file, until this method returns `False` at the end of file. A special implementation `TbModuleList` can be used for easily getting data from multiple files synchronously.

When a module has parsed an event from file, it will store the data as a `TbEvent` in the field `currentEvent`. Again `TbEvent` is an interface, with implementations for the different kinds of output files. Further, more complicated kinds of output files may contain several hits per event, and these hits are stored in a subclass of `TbHit`.

B.4.2 Simulating detector response

The most important function of TbAna is to test models for detector response simulation, and to simulate realistic hits for BAT telescope from the `TestBeamSim` energy deposits.

[5] An interface is a class which defines a set of functions etc., but not how these functions act. Thus you cannot have an instance of an interface class. However interface classes does not really exist in Python, but the general idea has been implemented by having a set of classes defining names of data fields and functions, but leaving the "virtual" functions "empty" by use of the `pass` Python keyword

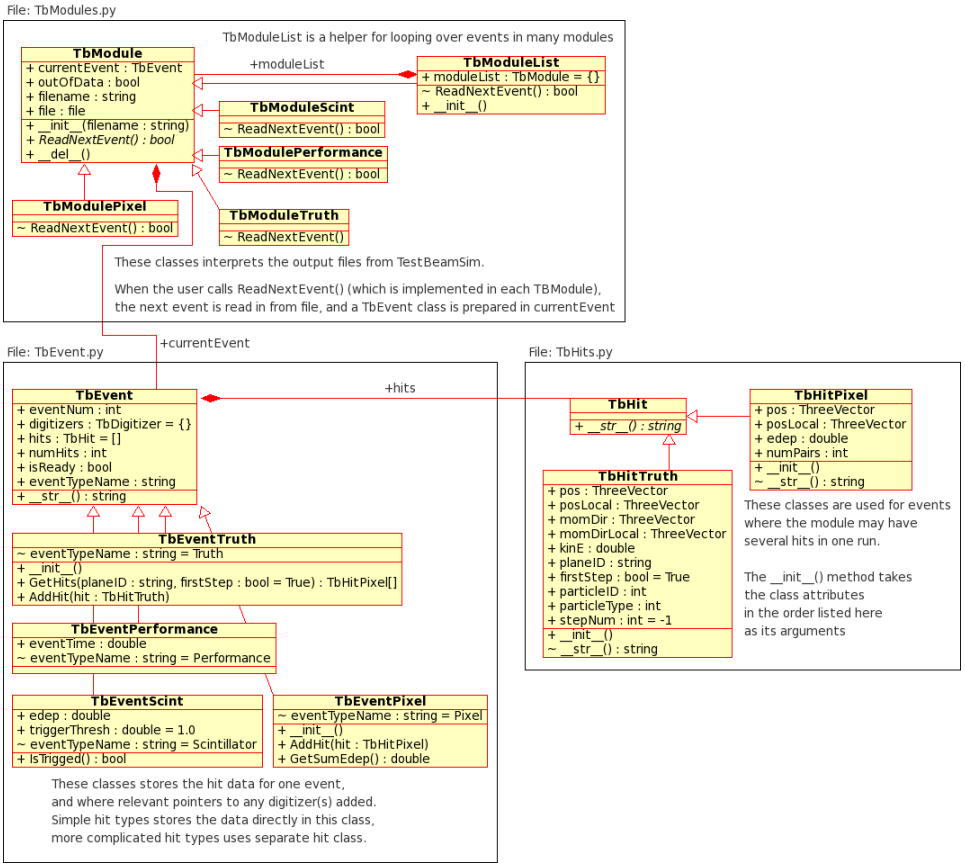


Figure B.2: UML diagram of the analysis/digitization library, only the part dealing with data input and storage.

This is done by use of the classes shown in figure B.3. The most important class here is `TbDigitizer`, which is an "interface" to different digitization algorithms. The classes implementing this interface can then be pushed onto the map `TbEvent::digitizers`, and ran by calling the digitizers method `DoDigitize()`. The digitizers then produce a set of `TbDigits`, which represents individual digits. As the digitizers all have configurations and tunings, the interface `TbDigitizerConfig` defines a mechanism for storing and accessing this.

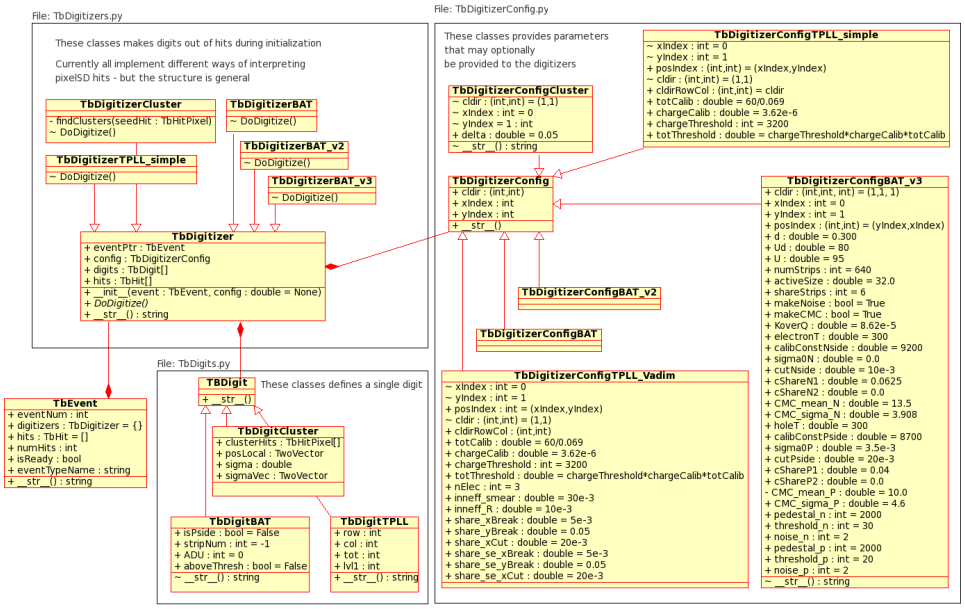


Figure B.3: UML diagram of the analysis/digitization library, only the part dealing with digitization simulation. Fields and methods in `TbDigitizerConfig_BAT` and `TbDigitizerConfig_BAT_v2` not written out.

B.4.3 Reading and writing raw experiment data (BDT)

The Bonn Atlas Telescope DAQ system uses a format known as `BDT` to store its data. This format is described in Appendix D. As TbAna needs to both write (`TbAna_digitizer.py`) and read (`TbAna_bdtHistos.py`) this format, there are Python libraries to do it. An overview of the classes are shown in figure B.4.

This library is fully contained within the file `TbBDTlib.py`, which is depending on `TbDigits.py` and `TbDigitizerConfig.py`. The main class is `BDTfile`, which is representing a single BDT file. The name of this file is passed as the argument to the constructor, along with a flag indicating whether to open the file for reading or creating a new file for writing. For detailed documentation about how to use this library, please see its source code.

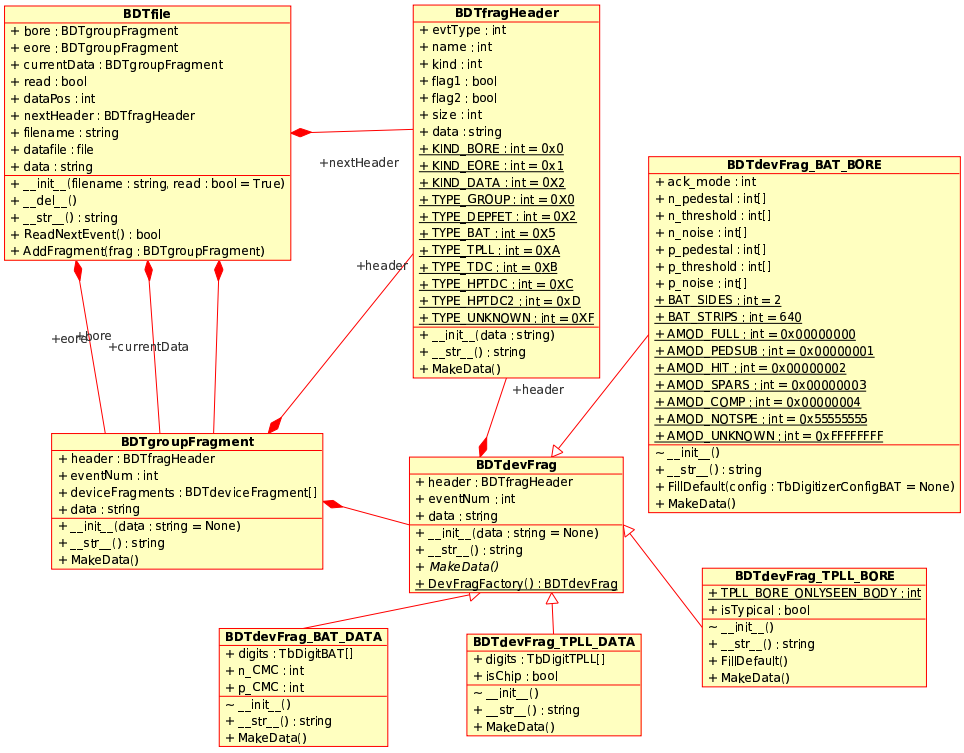


Figure B.4: UML of the BDT reading/writing library

Appendix C

TbMon simulation extensions

TbMon is the offline analysis framework used by the 3D pixel and PPS groups. Its main purpose is to make data, cuts etc. easily available to analysis, thus avoiding code duplication.

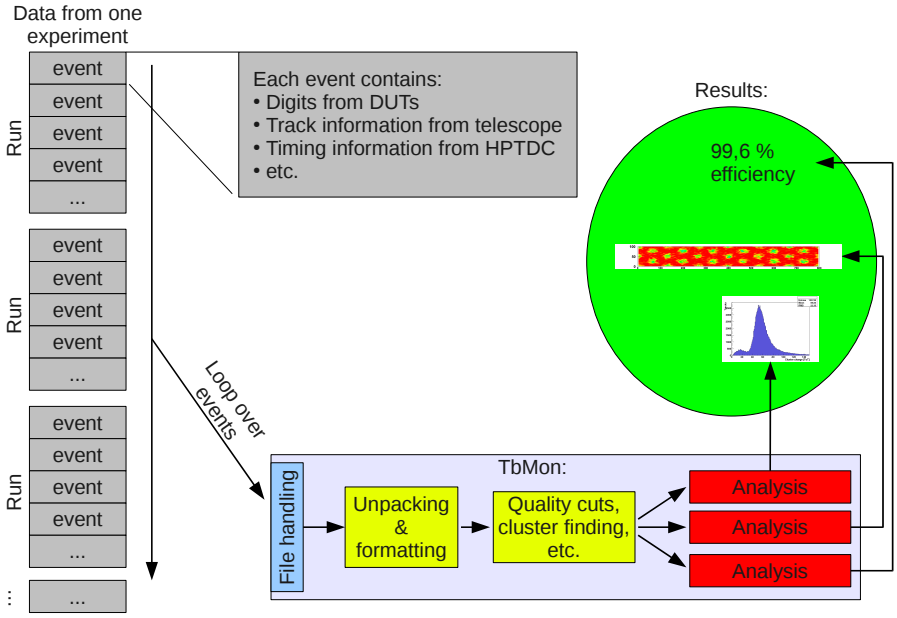


Figure C.1: Overview of TbMon: Input, output, and schematical internal structure

TbMon assumes that the data is structured in `configurations`, which is generally corresponding to one experiment or testbeam period. One single configuration is built up of many runs, which are `.root` ntuples containing detector digits and tracking information. In addition to the ntuple files, a configuration is

also consisting of `event builders` that handle data I/O for these ntuples, and sets cut flags on the events. The output of the `event builders` are an `event` object, which contains all the relevant information for the current event and DUT. These events are then passed on to one or more analysis, which makes plots, calculations etc. based on data accumulated for a single DUT.

For more information on TbMon in general, please see the official documentation `doc/tbmon.html` inside the TbMon source folder. The presentation "TbMon offline analysis framework: Getting started, newly added features", found at `http://folk.uio.no/kyrrens/master/talks/EudetTutorial_tbmon/tbmon_moreStuff.pdf` may also be useful. This presentation also contains installation and configuration instructions. For the rest of this appendix, non-simulation specific parts of TbMon is assumed known.

To effectively handle simulation data as well as real data, some extensions have been added to TbMon. These enable reading the raw simulation data created by `TestBeamSim` (described in Appendix A.3) and making it available, and an infrastructure for rapidly developing and testing models for DUT response. All of this is described in the sections below.

C.1 Configuring simulation extensions

For simulation extensions to work, they need access to the raw simulation data from `TestBeamSim` in addition to the reconstructed ntuples. To do this, one must specify the path where the `workdirs` are located in the part of `siteconfig.h` for the `configuration`, and also set the flag `config.isSimulation`. Further, one needs to include at least the `event builder` called `simBaseBuilder`, which synchronizes the event numbers for reconstructed and raw simulation data (see Appendix B.3.1 for more information). To do something useful with the simulation data, the event builders `simPixelEdepBuilder` (gives access to the raw charge clusters as output from `PixelSD`, see Appendix A.3.1.1) and `simTruthBuilder` (gives access to truth information, see Appendix A.3.2) should also be loaded. For testing pixel detector models (`simDuts`), `simDutRunner` is required.

For an example of how to set this up, please see the `configuration simBat2010` from `driver.cc` and `siteconfig.h.example`.

C.2 Analysis: Access to raw simulation data

The `event builders` all store their data in `event` objects, and this is also true for simulation event builders. The simulation data interesting for analysis is stored inside the `simDataKeeper` object pointed to by the field `Event::simData`.

Before accessing this object, the analysis should check that the flag `config.isSimulation` is set to `True`. If this is not the case, `Event::simData` will be a `NULL` pointer.

The `simDataKeeper` object itself contains vectors containing the data from `simEdepBuilder` and `simTruthBuilder`. The fields of the elements contained in this vector are mostly the same as in the output of `TestBeamSim` – however note that `TestBeamSim` and TbMon use different coordinate systems. This is solved by converting the local coordinates of the `edep`s and `truthHit`s inside the `eventBuilder`s, before they are passed on to analysis.

C.3 SimDuts and pixel detector model testing

In order to test models for detector response given energy deposits, the TbMon simulation extensions includes support for "simDuts". These are classes that for each event converts raw `TestBeamSim edep`s into pixel digits. These pixel digits are then inserted into the event object, overwriting the digits from the ntuple. This enables rapid development and testing of new device models, which can then be analyzed with the same tools as used with real data.

To implement a new SimDut model, inherit the interface `Simdut`, and implement the model inside the purely virtual function `digitize(...)`. For an example on how to do this, see the already included SimDuts in sourcecode.

In order to run the SimDut model, it has to be listed inside `simDutRunner::init(...)`. It can then be ran by using the command line argument `-P:B_simDutRunner_model <modelName>`, and analyzed with the normal analysis classes. The output of the models can thus be compared to real data.

Appendix D

Description of the BDT file format

This appendix describes the BDT file format, used by the Bonn Atlas Telescope (BAT) data acquisition system. It has mostly been reverse-engineered from the `SiTBeAn` sourcecode, and python code parsing and writing the format has been written, documented in Appendix B.4.3. This code seems to decode and encode the format (version VER_2005) properly, as data produced with it is correctly read back by both the SiTBeAn code, as well as the `tbtrack` software used for tracking and alignment of our testbeam campaigns. It has been tested on data from the May 2009 BAT testbeam in the SPS H8 beamline.

D.1 Main elements and structure

The main element of a BDT file is a *word*, which is a 32-bit little-endian field. These are in turn organized into *fragments*, which may have sub-fragments.

One fragment, as shown in figure D.1, is composed of first one frame header word, followed by an optional padding, and then as many data words as necessary (limited upwards to $2^{20} - 1$ words by the *size* field in the header).

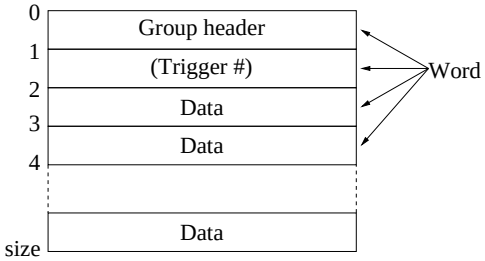


Figure D.1: One BDT fragment

The file is built up from a set of events, which are written consecutively to

the file. Each event is represented by one *group fragment*, containing a number of device fragments as its data words. The first event (called the *BORE* – Begin Of Run Event) is special, and contains information about the setup used in the current file. There should be one and only one BORE in each file. After the BORE, several *DATA* fragments follows. There is also a possibility for a special End Of Run Event (*EORE*), however this has not been seen in actual data.

An illustration of a group fragment containing several device fragments is shown in figure D.2. When looking at several group fragments, all the devices

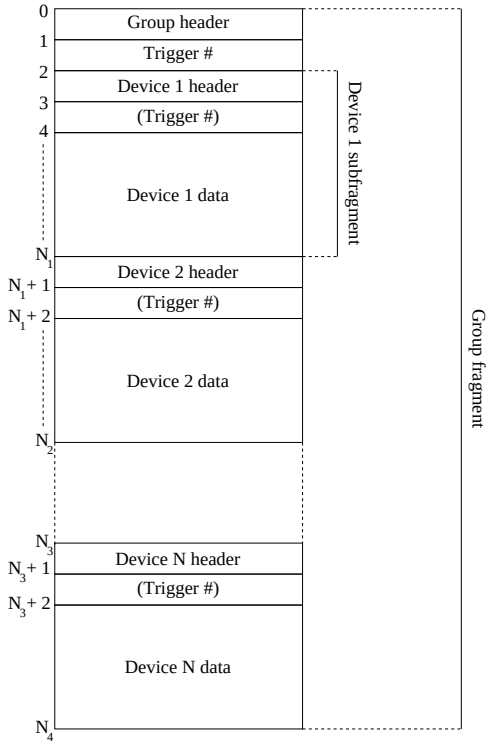


Figure D.2: A BDT group fragment containing several device sub-fragments

always writes a sub-fragment, and the sub-fragments always appear in the same order across different group fragments. However, when looking at different files, the order seems random.

All group fragments contains a «trigger number» word right after the header. Also, each single DATA and EORE device fragment contains a trigger number. This is a unsigned little-endian 32-bit integer, which is increased by one for each event, starting at 0. It seems to be written by the DAQ software, as the «trigger numbers» stays in sync even if the data itself is desynchronized. Where present in BORE, it is set to `0x55555555`.

D.2 Fragment header

Each header is composed of several fields (see figure D.3), describing the type and contents of the fragment.

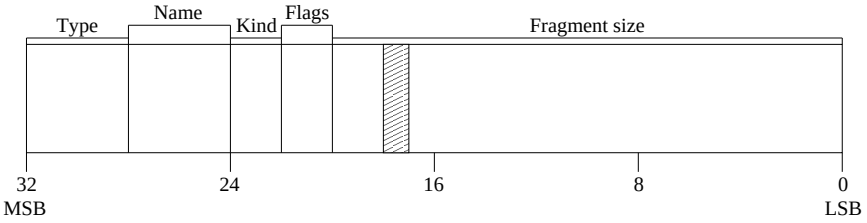


Figure D.3: BDT fragment header

The first 20 bits[1] contains the size of the fragment. This is the number of words the fragment contains, inclusive the header, padding etc.

Bit 21 and 22 is known as *flag1* and *flag2*. This is only used for Atlas pixel device fragments. Here flag1=1, flag2=0 if and only if we have a single chip assembly, and flag1=0, flag2=1 if and only if it is a pixel module. If it is not an Atlas pixel device fragments, these fields are ignored and set to 0.

Bit 22 and 23 is the *kind* field. It tells whether this fragment belongs to a BORE, EORE, or DATA event. It is set in both group- and sub-fragment headers. The magic bits are listed in table D.1.

Name	Bits
BORE	`0x0`
EORE	`0x1`
DATA	`0x2`

Table D.1: Kind field magic bits

Bit 24-28 is the *name* field. This is some 4-bit identifier, used for separating different devices of the same type. It is ignored for group fragments (set to 0).

Bit 28-32 is the *type* field. Together with the kind field, it tells how to decode the data contained in the fragment. The magic bits are listed in table D.2.

Note that the only type allowed at toplevel is GROUP. UNKNOWN means that the device is unknown to the encoding program. The second last entry is probably some kind of (HP)TDC, encountered in the May 2009 data.

D.3 BAT data

There are two kinds of BAT data packages written – one kind fore BORE events, and another for DATA events (real, triggered events).

[1] The SiTBeAn package ignores bit 18. I have not found that this bit is used for anything else, and therefore assumed that this is a bug. This assumption is also done in the tbtrack software.

Name	Bits
GROUP	0x0
DEPFET	0x2
BAT	0x5
TPLL	0xA
TDC	0xB
HPTDC	0xC
???	0xD
UNKNOWN	0xF

Table D.2: Type field magic bits

D.3.1 BORE

A BAT BORE data package contains information about the status of the BAT telescope – the data format currently in use, and pedestals, thresholds, and noise measurements for each strip. The general layout of the format is shown in figure D.4.

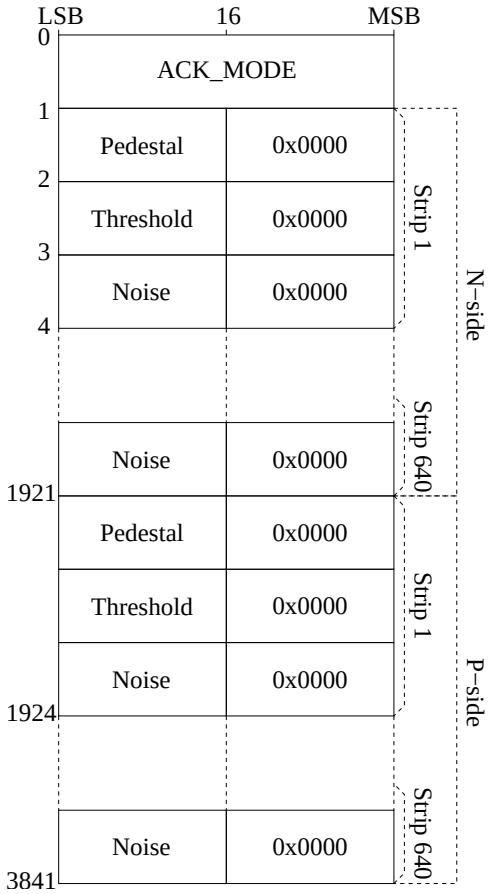


Figure D.4: BAT BORE data

The first word is the acquisition mode in use for the current run, and is one of the magic words listed in table D.3. If AMOD_NOTSPE is specified, AMOD_COMP is assumed. This is what is seen in the actual data files, and what will be documented here.

Name	Bits
AMOD_FULL	0x00000000
AMOD_PEDSUB	0x00000001
AMOD_HIT	0x00000002
AMOD_SPARS	0x00000003
AMOD_COMP	0x00000004
AMOD_NOTSPE	0x55555555
AMOD_UNKNOWN	0xFFFFFFFF

Table D.3: BAT BORE acquisition mode magic word

The first word is followed by 3840 words of pedestals etc., all encoded as signed 16 bit little-endian integers, with two bytes of zeroes appended. The data for each strip is written in three words, as shown in figure D.4. The strips are written consecutively, first the n-side, and later the p-side. There is no trigger number in BAT BORE device fragments.

D.3.2 DATA

The BAT DATA data packages are a bit more complicated, and involves five different words. It is written in 16-bit half-words (HWs), if necessary with a *fill* HW appended so that the number of half words fits into a whole number of words. A rough sketch of such a package is shown in figure D.5. In addition to what is shown here, there is a trigger number right after the header (before the start of the data).

The two first words are CMC values for the n- and p-side. These are encoded as a little-endian 32-bit signed integers. After these the 16 bit HWs follow. These are built up as shown in figure D.6. There are five kinds, possible to separate from their header:

- Fill: These HWs have header $= (1001)_2$. If neccessary, they are used at the end of a DATA data package to match the word boundary.

- N-strip value: These HWs have header $= (S0T0)_2$. They contain the ADU (ADC Units) value of a read-out strip, with the pedestal subtracted. It looks like it is initially encoded as 16-bit signed integer, but bit number 12-15 is overwritten by the header, effectively reducing it to a 2^{12+1} bit signed integer. The sign convention used is two's compliments, so if the sign bit is set, the possible conversion methods are either direct promotion to a 16 bit signed int:

$$\text{value} = (\text{HW} \;\&\; \text{0x0FFF}) \mid \text{0xF000}$$

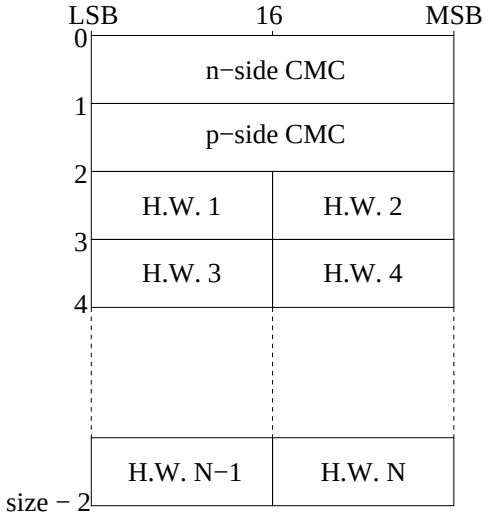


Figure D.5: BAT DATA data

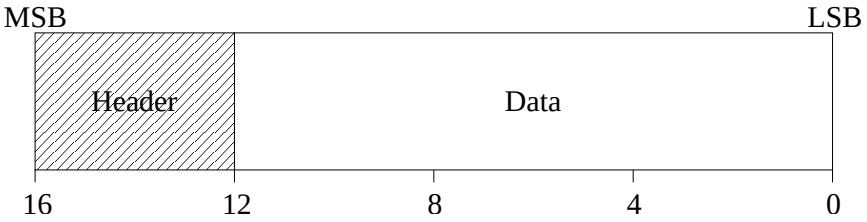


Figure D.6: BAT DATA data half-word

, or the more mathematical approach

$$\text{value} = -(2^{12} - \text{HW})$$

, where HW is the 16-bit half-word.

The ADU values of the strips are written consecutively, so the reading program needs to have a counter which is first set (or re-set) by a n-strip number HW, and then incremented by one for each n-strip value encountered.

The bit marked "T" seems to indicate if this strip was above the threshold for the zero-suppression or not. It is therefore set to 1 if this strip was above threshold, and 0 if it was read out because a neighbour or next-to-neighbour strip was above threshold. This bit seems to be ignored by the SiTBeAn code.

- P-strip value: These HWs have header $= (S1T0)_2$. Other than this they are encoded the same way as the n-strip values. Note that the n- and p-strip counters are separate.

- N-strip number: These HWs have header $= (0001)_2$. They set which n-strip value will be read next from the file. It is decoded as

$$\text{n\_strip} = ((\text{HW} \;\&\; \text{0x0FFF}) - 21)\,/2$$

- P-strip number: These HWs have header $= (0101)_2$. These set the strip number counter for the p-side, similar to the n-strip number. They are decoded as

$$\text{p\_strip} = ((\text{HW} \;\&\; \text{0x0FFF}) - 22)\,/2$$

D.4 TPLL data

There are two modes of writing TPLL (pixel) data – *single chip* and *module*. With single chip mode, a separate TPLL (Turbo Pixel Low Level VME card) and TPCC (Turbo Pixel Control Card) is used for each pixel sensor, while module mode is used when each TPLL/TPCC controls several pixel sensors. In the BDT file, single chip mode is indicated by setting flag1=1, flag2=0 in all device fragment headers belonging to the device, while module mode is indicated by setting flag1=0, flag2=1. The "module" mode is used in the May 2009 data, and is documented below.

D.4.1 BORE

I have found (in May2009 data) these fragments to always be set like shown in figure D.7.

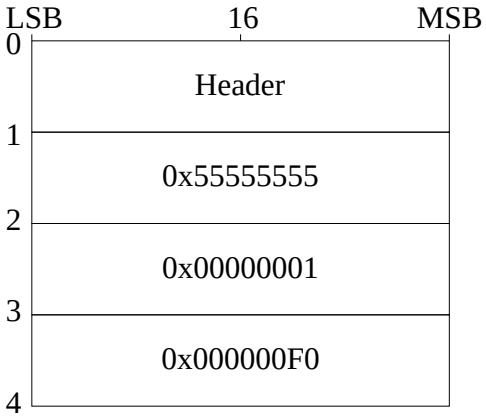


Figure D.7: TPLL BORE fragment

D.4.2 DATA

TPLL single chip data has a rather simple format, where each data word represent a pixel that has fired. This is encoded into the format shown in figure D.8. In addition

to simple «pixel hit» data, it is also possible to store “EOE” (End Of Event), and ChipError. When decoding these data, remember that a single pixel device has 160 rows and 18 columns. In the data, they are counted from 0.

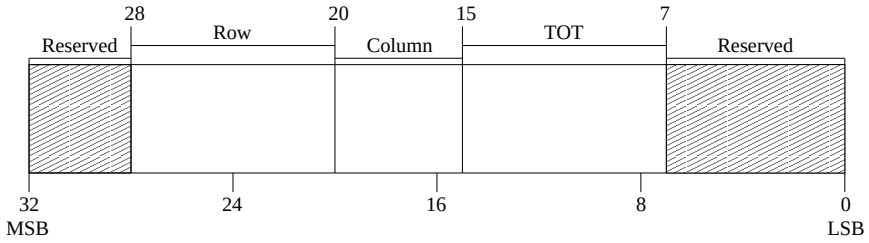


Figure D.8: TPLL DATA word

Here EOE is encoded by setting row=225 or row=240, while chipError is encoded by setting row>159 (but not 225 or 240).

If several events with different LVL1 timestamps are read out, a EOE is emitted when LVL1 is increased.

Appendix E

Dictionary of often used words

Word	Description
ADC	Analog Digital Converter, circuitry that converts an analog signal (voltage, current, charge) into a digital representation.
ADC unit	Arbitrary unit used by an ADC – one ADC unit is the increase in input necessary to increment the output by one.
ATLAS	A Toroidal LHC ApparatuS, Particle physics experiment at the LHC
BAT	Bonn Atlas Telescope; particle tracking system. See Chapters 1.2 and 4.
BDT	Binary data format used by BAT. Documented in Appendix D.
BORE	Begin Of Run Event; special type of "event" at start of BDT file. See Appendix D.
CDF	Cumulative Distristribution Fuction; For a probability distribution function (PDF) $P(X)$, the CDF CDF(x) is defined such that CDF(x) equals the probability that a random number X drawn from P is greater than or equal to x.
CLIC	Compact LInear Collider. A possible design of a new high-energy electron-positron collider.
CM, Common Mode	Noise in an detector common for all channels in an event.
CMC	Common Mode Counter, function of the BAT FPGA. Used to estimate the common mode. See Chapter 4.2.3.
DAQ	Data Acquisition; System collecting data from sensors and storing them for analysis.
Digit	Element of raw data from detector.

Digitization	Process of converting simulation hits (energy deposits) into detector digits.
DRIE etching	Deep Reactive Ion Etch. Technology for etching steep-sided holes and threnches. The most used DRIE technique, the Bosch process, works by switching between two modes: Deposition of a protecting substance everywhere, and a isotropic plasma etch containing some ions attacking the surface perpendicularly, removing the protecting layer at the bottom of the holes but leaving it at the side-walls.
EHP	Electron-Hole-Pair. Created by exiting an electron from the valence band into the conduction band, leaving behind a hole.
Fano factor	Physical effect in semiconductors “smearing" the amount of electrons produced for a certain energy deposit. See Chapter 2.2.3.
FPGA	Field-Programmable Gate Array; Integrated circuit (IC) that can change function by uploading new “software”.
Frontend	First stage of electronics in a detector, connected directly to the sensor. In HEP silicon detectors this is usually a chip containing all the analog parts of the readout system (amplifiers etc.), and also the ADC.
HEP	High Energy Physics, the study of particle interactions at high energy.
Hit	A raw energy deposit in a sensor, or the estimated position of where a particle intersected the sensor.
IBL	Insertable B-Layer; a proposed fourth layer of pixel sensors, which is to be placed inside the current B-layer. See Chapter 1.1.2.
IC	Integrated Circuit; small package with electronic circuitry. Also known as (electronic) chip.
LHC	Large Hadron Collider, Proton accelerator at CERN in Geneva, Switzerland. See Chapter 1.
MIP	Minimum Ionizing Particle; a relativistic particle, having the “optimum” momentum for an as low as possible $\langle dE/dx \rangle$. See Chapter 2.1 and figure 2.2.

NIM	Nuclear Instrumentation Module; Electrical and mechanical specification of type of electronics modules often used in experimental physics. In testbeams, trigger systems are often built using such modules.
On-line	A step in data processing that happens in real-time while the data is recorded, or as a part of the data recording process.
Off-line	A step in data processing that happens independently of the data taking, and may thus be repeated.
PCB	Printed Circuit Board; Flat plastic/glass-fiber substrate with conducting traces, providing electrical interconnection and mechanical support for electronic components.
PDF	Probability Distribution Function; A real-valued function $P(x)$ that describes the probabilities of drawing random numbers from a distribution. If X is distributed according to $P(x)$, then $P(x_a \leq X \leq x_b) = \int_{x_a}^{x_b} P(x)dx$.
Pedestal	In BAT: The signal value from a strip that has not been hit by a particle. This is subtracted from the signal when reading out runs, and its value is reported in the BORE of the BDT file.
Pile-up	The presence of more than one underlying event inside one detector event.
SLHC	Super LHC. A proposed upgrade of the LHC proton collider increasing its luminocity.
Spectrum	Histogram representing the rate of occurrence of different amounts of energy.
SVG	Scalable Vector Graphics; XML-based vector graphics format that with modern browsers can be embedded in HTML pages.
TbAna_*	Collection of Python programs handling digitization and analysis not depending on tracking. See Appendix B.
TestBeamSim	Geant4-based simulation between the beam and matter. See Chapter 3 and Appendix A.
ToT	Time over Threshold; A measurement of signal size, the amount of time an amplifier shows an output voltage above some threshold value. See Chapter 6.2.2.

TPCC	Turbo Pixel Control Card; PCB that sits between the TPLL and one or several pixel test boards (frontend and sensor).
TPLL	Turbo Pixel Low Level VME card; VME card that connects the TPCC to the VME bus (which is connected to the DAQ PC). Also handles triggering for the pixel devices.
Underlying event	One single collision in a particle physics experiments. See Chapter 1.
VME	Computer bus standard, used for *crates* providing power and communications between *cards* that may perform various tasks.
δ-electron, δ-radiation	When a charged particle traverses a material, it may sometimes transfer a non-trivial amount of energy to a single electron. This electron, which now has a non-neglible kinetic energy, can then travel through the material and produce more ionization along its path. See section 2.1.

List of Figures

List of Tables

Bibliography

[1] Atlas simulation validation – performance benchmarking - platform slc4-32bit. http://atlas-computing.web.cern.ch/atlas-computing/packages/simulation/geant4/validation/ComparisonsStable/ComparisonsSLC432.html, March 2010.

[2] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, F. Behner, L. Bellagamba, J. Boudreau, L. Broglia, A. Brunengo, H. Burkhardt, S. Chauvie, J. Chuma, R. Chytracek, G. Cooperman, G. Cosmo, P. Degtyarenko, A. Dell'Acqua, G. Depaola, D. Dietrich, R. Enami, A. Feliciello, C. Ferguson, H. Fesefeldt, G. Folger, F. Foppiano, A. Forti, S. Garelli, S. Giani, R. Giannitrapani, D. Gibin, J. J. Gómez Cadenas, I. González, G. Gracia Abril, G. Greeniaus, W. Greiner, V. Grichine, A. Grossheim, S. Guatelli, P. Gumplinger, R. Hamatsu, K. Hashimoto, H. Hasui, A. Heikkinen, A. Howard, V. Ivanchenko, A. Johnson, F. W. Jones, J. Kallenbach, N. Kanaya, M. Kawabata, Y. Kawabata, M. Kawaguti, S. Kelner, P. Kent, A. Kimura, T. Kodama, R. Kokoulin, M. Kossov, H. Kurashige, E. Lamanna, T. Lampén, V. Lara, V. Lefebure, F. Lei, M. Liendl, W. Lockman, F. Longo, S. Magni, M. Maire, E. Medernach, K. Minamimoto, P. Mora de Freitas, Y. Morita, K. Murakami, M. Nagamatu, R. Nartallo, P. Nieminen, T. Nishimura, K. Ohtsubo, M. Okamura, S. O'Neale, Y. Oohata, K. Paech, J. Perl, A. Pfeiffer, M. G. Pia, F. Ranjard, A. Rybin, S. Sadilov, E. Di Salvo, G. Santin, T. Sasaki, N. Savvas, Y. Sawada, S. Scherer, S. Sei, V. Sirotenko, D. Smith, N. Starkov, H. Stoecker, J. Sulkimo, M. Takahata, S. Tanaka, E. Tcherniaev, E. Safai Tehrani, M. Tropeano, P. Truscott, H. Uno, L. Urban, P. Urban, M. Verderi, A. Walkden, W. Wander, H. Weber, J. P. Wellisch, T. Wenaus, D. C. Williams, D. Wright, T. Yamada, H. Yoshida, and D. Zschiesche. G4–a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250 – 303, 2003.

[3] E. Belau, R. Klanner, G. Lutz, E. Neugebauer, H.J. Seebrunner, A. Wylie, T. Böhringer, L. Hubbeling, P. Weilhammer, J. Kemmer, U. Kötz, and

M. Riebesell. Charge collection in silicon strip detectors. *Nuclear Instruments and Methods in Physics Research*, 214(2-3):253 – 260, 1983.

[4] M Benoit, A Lounis, and N Dinu. Simulation of guard ring influence on the performance of atlas pixel detectors for inner layer replacement. *Journal of Instrumentation*, 4(03):P03025, 2009.

[5] E. Bolle, M. Borri, M. Boscardin, G.-F. Dalla Betta, G. Darbo, C. Da Vià, O. Dorholt, S. Fazio, C. Gemme, H. Gjersdal, P. Grenier, S. Grinstein, P. Hansson10, J. Hasi, F. Huegging, P. Jackson, C. Kenney, M. Kocian, A. La Rosa, A. Mastroberardino, P. Nordahl, F. Rivero, O. Røhne, H. Sandaker, K. Sjøbæk, T. Slaviec, D. Su, J. Tsung, D. Tsybychev, N. Wermes, and C. Young. 3d pixels - recent results. In *Proceedings of Science, Vertex 2009*, 2009.

[6] ATLAS Collaboration. Atlas insertable b-layer technical design report (in preparation). Technical report, CERN, 2010.

[7] Geant4 Collaboration. Reference physics lists. `http://geant4.cern.ch/support/proc_mod_catalog/physics_lists/referencePL.shtml`.

[8] International commison on radiation units (ICRU). Icru report 31: Average energy required to produce an ion pair, May 1971.

[9] R. Fruhwirth. Application of kalman filtering to track and vertex fitting. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 262(2-3):444–450, December 1987.

[10] Geant4 Collaboration. *Geant 4 User's Guide for Application Developers*.

[11] Geant4 Collaboration. *Installation Guide: For setting up Geant4 in your computing environment*.

[12] Geant4 Collaboration. *Physics Reference Manual*.

[13] H. Gjersdal, E. Bolle, M. Borri, C. Da Via, O. Dorholt, S. Fazio, P. Grenier, S. Grinstein, P. Hansson, J. Hasi, F. Huegging, P. Jackson, C. Kenney, M. Kocian, A. La Rosa, A. Mastroberardino, P. Nordahl, F. Rivero, O. Røhne, H. Sandaker, K. Sjøbæk, T. Slaviec, D. Su, J. Tsung, D. Tsybychev, N. Wermes, and C. Young. Tracking effciency and charge sharing of 3d silicon sensors at different angles in a 1.4 tesla magnetic field. In *Nuclear Instrumentation Methods A (proceedings)*, 2009.

[14] Particle Data Group. *Review of particle physics*. Elsevier, 2008.

[15] HAMAMATSU PHOTONICS K.K., Solid State Division. *Datasheet for Hamamatsu Si Strip detector S6934 (AC-coupled double-sided Si strip detector for particle tracking)*.

[16] P. Hansson, J. Balbuena, C. Barrera, E. Bolle, M. Borri, M. Boscardin, M. Chmeissan, G.-F. Dalla Betta, G. Darbo, C. Da Via, E. Devetak, B. DeWilde, D. Su, O. Dorholt, S. Fazio, C. Fleta, C. Gemme, M. Giordani, H. Gjersdal, P. Grenier, S. Grinstein, J. Hasi, K. Helle, F. Huegging, P. Jackson, C. Kenney, M. Kocian, I. Korolkov, A. La Rosa, A. Mastroberardino, A. Micelli, C. Nellist, P. Nordahl, F. Rivero, O. Rohne, H. Sandaker, D. Silverstein, K. Sjoebaek, T. Slaviec, J. Stupak, I. Troyano, J. Tsung, D. Tsybychev, N. Wermes, and C. Young. 3d silicon pixel sensors: Recent test beam results. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, In Press, Corrected Proof:–, 2010.

[17] Zhong He. Review of the shockley-ramo theorem and its application in semiconductor gamma-ray detectors. *Nuclear Instruments and Methods in Physics Research A*, 463:250–267, 2001.

[18] K Helle, E Bolle, B Buttler, C Da Via, O Dorholt, S Fazio, G Gjersdal, J Hasi, C Kenney, A La Rosa, D Miller, V Linhart, H Pernegger, O Rohne, K Sjobak, H Sandaker, T Slavicec, B Stugu, M Tomasec, S Watts, and C Young. Testbeam studies of 3d silicon sensors. Technical Report ATL-COM-UPGRADE-2010-007, CERN, Geneva, Jun 2010. The spirit of the note is first of all to make sure that this study is communicated to the tracker upgrade community. If fit for 'PUB' release , we would of course be happy to get it reclassified.

[19] M. Hjorth-Jensen. *Lecture Notes in Computational Physics*. (unpublished), 2010.

[20] C. Kenney, S. Parker, J. Segal, and C. Storment. Silicon detectors with 3-d electrode arrays: fabrication and initial test results. *Nuclear Science, IEEE Transactions on*, 46(4):1224 –1236, aug 1999.

[21] W.R. Leo. *Techniques for Nuclear and Particle Physics Experiments (2nd edition)*. Springer, 1994.

[22] M. Mathes, M. Cristinziani, C. Da Via, M. Garcia-Sciveres, K. Einsweiler, J. Hasi, C. Kenney, S.I. Parker, L. Reuen, M. Ruspa, J. Velthuis, S. Watts, and N. Wermes. Test beam characterization of 3-d silicon pixel detectors. *Nuclear Science, IEEE Transactions on*, 55(6):3731 –3735, dec. 2008.

[23] M. Morpurgo. A large superconducting dipole cooled by forced circulation of two phase helium. *Cryogenics*, July:411–417, 1979.

[24] S. I. Parker, C. J. Kenney, and J. Segal. 3d – a proposed new architecture for solid-state radiation detectors. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 395(3):328 – 343, 1997. Proceedings of the Third International Workshop on Semiconductor Pixel Detectors for Particles and X-rays.

[25] QinetiQ. Geant4 general particle source. Electronic. `http://reat.space.qinetiq.com/gps/`.

[26] Simon Ramo. Currents induced by electron motion. *Proceedings of the I.R.E*, 27:584–585, 1939.

[27] Philipp Roloff. The eudet high resolution pixel telescope. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 604(1-2):265 – 268, 2009. PSD8 - Proceedings of the 8th International Conference on Position Sensitive Detectors.

[28] A. La Rosa, M. Boscardin, G.-F. Dalla Betta, G. Darbo, C. Gemme, H. Pernegger, C. Piemonte, M. Povoli, S. Ronchin, A. Zoboli, N. Zorzi, E. Bolle, M. Borri, C. Da Via, S. Dong, S. Fazio, P. Grenier, S. Grinstein, H. Gjersdal, P. Hansson, F. Huegging, P. Jackson, M. Kocian, F. Rivero, O. Rohne, H. Sandaker, K. Sjobak, T. Slavicek, W. Tsung, D. Tsybychev, N. Wermes, and C. Young. Preliminary results of 3D-DDTC pixel detectors for the ATLAS upgrade. *PoS*, RD09:032, 2009.

[29] William Shockley. Currents to conductors induced by a moving point charge. *Journal of Applied Physics*, 9:635–636, 1938.

[30] Kyrre Ness Sjøbæk. 3d silicon: The future of radiation-hard silicon sensors? `http://folk.uio.no/kyrrens/diverse/roros_talk/3d%20silicon.pdf`, August 2009. Annual meeting of the Norwegian Physical Society (Fysikermøtet) 2009.

[31] Kyrre Ness Sjøbæk. A system for variational monte carlo calculations. Electronic, April 2009. `http://folk.uio.no/kyrrens/kode/vmc-classes/project1.pdf`.

[32] Kyrre Ness Sjøbæk. Test of 3d silicon pixel detectors: Simulations and measurements. `http://folk.uio.no/kyrrens/diverse/spaatind_talk/kyrre_spaatind.pdf`, January 2010. Spåtind 2010 Nordic conference on particle physics.

[33] Helmuth Spieler. *Semiconductor Detector Systems*. Oxford Science Publications, 2005.

[34] S Straulino, O Adriani, L Bonechi, M Bongi, S Bottai, G Castellini, D Fedele, M Grandi, P Papini, S B Ricciarini, P Spillantini, F Taccetti, E Taddei, and E Vannuccini. Spatial resolution of double-sided silicon microstrip detectors for the pamela apparatus. *Nucl. Instrum. Methods Phys. Res., A*, 556(hep-ex/0510012):100–114. 28 p, Oct 2005.

[35] Ben G. Streetman and Sanjay Kumar Banerjee. *Solid state electronic devices (6th edition)*. Pearson Prentice Hall, 2006.

[36] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1997.

[37] The Analytic Sciences Corporation Technical Stapp. *Applied Optimal Estimation*. MIT press, 1974.

[38] J. Treis, P. Fischer, H. Krüger, L. Klingbeil, T. Lari, and N. Wermes. A modular pc based silicon microstrip beam telescope with high speed data acquisition. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 490(1-2):112 – 123, 2002.

[39] Johannes Treis. *Development and operation of a novel PC-based high speed beam telescope for particle tracking using double sided silicon microstrip detectors*. PhD thesis, Universität Bonn, Physikalisches Institut, 2002.

[40] C. Da Via, E. Bolle, K. Einsweiler, M. Garcia-Sciveres, J. Hasi, C. Kenney, V. Linhart, Sherwood Parker, S. Pospisil, O. Rohne, T. Slavicek, S. Watts, and N. Wermes. 3d active edge silicon sensors with different electrode configurations: Radiation hardness and noise performance. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 604(3):505 – 511, 2009.

[41] C. Da Viá, J. Hasi, C. Kenney, V. Linhart, Sherwood Parker, T. Slavicek, S.J. Watts, P. Bem, T. Horazdovsky, and S. Pospisil. Radiation hardness properties of full-3d active edge silicon sensors. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 587(2-3):243 – 249, 2008.

[42] Eric W Weisstein. Normal sum distribution. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/NormalSumDistribution.html`.

[43] Norbert Wermes and G Hallewel. *ATLAS pixel detector: Technical Design Report*. Technical Design Report ATLAS. CERN, Geneva, 1998.

[44] R. Wunstorf, M. Benkert, N. Claussen, N. Croitoru, E. Fretwurst, G. Lindström, and T. Schulz. Results on radiation hardness of silicon detectors up to neutron fluences of 1015 n/cm2. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 315(1-3):149 – 155, 1992.