ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Физический факультет

Кафедра вычислительной физики

БАКАЛАВРСКАЯ РАБОТА

«Пространственная реконструкция откликов частиц во время-проекционной камере эксперимента MPD на коллайдере NICA»

«Spatial reconstruction of particle responses in the time projection chamber of the MPD experiment at the NICA collider»

Выполнил студент 4 курса, 405 группы

Басалаев Артём Евгеньевич

Направление 011200 – физика

Научный руководитель, к.ф.-м.н, доцент. . . . . Немнюгин С.А.

Рецензент, к.ф.-м.н., в.н.с. . . . . Рогачевский О.В.

Работа защищена с оценкой . . . . . дата . . . . . .

Зав. кафедрой, д.ф.-м.н. . . . . Яковлев С.Л.

Санкт-Петербург - 2013 г.

Содержание

Введение

Одной из актуальных задач современной теоретической физики является исследование сверхгорячей и плотной ядерной материи. Подобное состояние вещества, называемое кварк-глюонной плазмой, можно получить в лабораторных условиях при столкновениях частиц высоких энергий.

Исследование сверхгорячей и плотной ядерной материи, в частности, является одной из приоритетных программ Объединенного института ядерных исследований. В рамках этой программы планируется создание ускорительного комплекса NICA (Nuclotron-based Ion Collider fAcility) на базе существующего ускорителя «Нуклотрон». Главным трековым детектором будущего комплекса является многоцелевой детектор MPD (Multi-Purpose Detector), а его основной камерой является время-проекционная камера TPC (Time Projection Chamber) [1]. Устройство ускорительного комплекса представлено на рис. 1.

Рис. 1. Ускорительный комплекс NICA.

В настоящий момент осуществляется математическое моделирование работы TPC с целью выяснения наилучших параметров оборудования, которое предстоит изготовить, а также с целью отработки алгоритмов компьютерной обработки данных, которые будут поступать с детектора. Моделирование осуществляется в программном комплексе MpdRoot [2].

Одним из этапов моделирования работы TPC является пространственная реконструкция откликов частиц. Откликами частиц называются сигналы, которые получает электроника детектора при попадании частиц в чувствительную область. В результате эксперимента получаются данные в виде набора сигналов со считывающих плоскостей TPC. Для того чтобы восстановить треки частиц, пролетевших сквозь время-проекционную камеру, необходимо преобразовать набор сигналов со считывающих плоскостей в последовательности точек в пространстве рабочей камеры TPC, сквозь которые пролетели частицы.

Целью данной работы было создание алгоритма пространственной реконструкции откликов частиц, удовлетворяющего установленным требованиям, а именно, имеющего малую погрешность и создающего возможно меньшие отклонения от исходных треков. Алгоритм должен удовлетврять этим требования для треков в интервале псевдобыстрот[1] $|\eta| < 1.2$ [1].

Для достижения поставленной цели решались следущие задачи:

- поиск подходящей основной идеи алгоритма среди существующих решений и реализация алгоритма реконструкции откликов частиц, учитывающего специфику время-проекционной камеры детектора MPD;
- исследование погрешностей созданного алгоритма и сравнение их с погрешностями существующих алгоритмов.

[1] Псведобыстрота – безразмерная физическая величина, определяемая как $\eta = -ln\left(tg\frac{\theta}{2}\right)$, где θ – угол между осью пучка и направлением импульса частицы.

Глава 1

Постановка задачи

1.1. Работа время-проекционной камеры

Прежде чем приступить к дальнейшему изложению, необходимо рассмотреть основные физические процессы, происходящие во время-проекционной камере MPD. Время-проекционная камера представляет собой две коаксиальные цилиндрические поверхности, между которыми находится газовая смесь из 90% аргона и 10% метана при избыточном давлении 2 мбар. Торцы TPC представляют собой 12 трапецевидных секторов с прямоугольными медными считывающими пластинами – падами [1]. Схема TPC представлена на рис. 1.1.

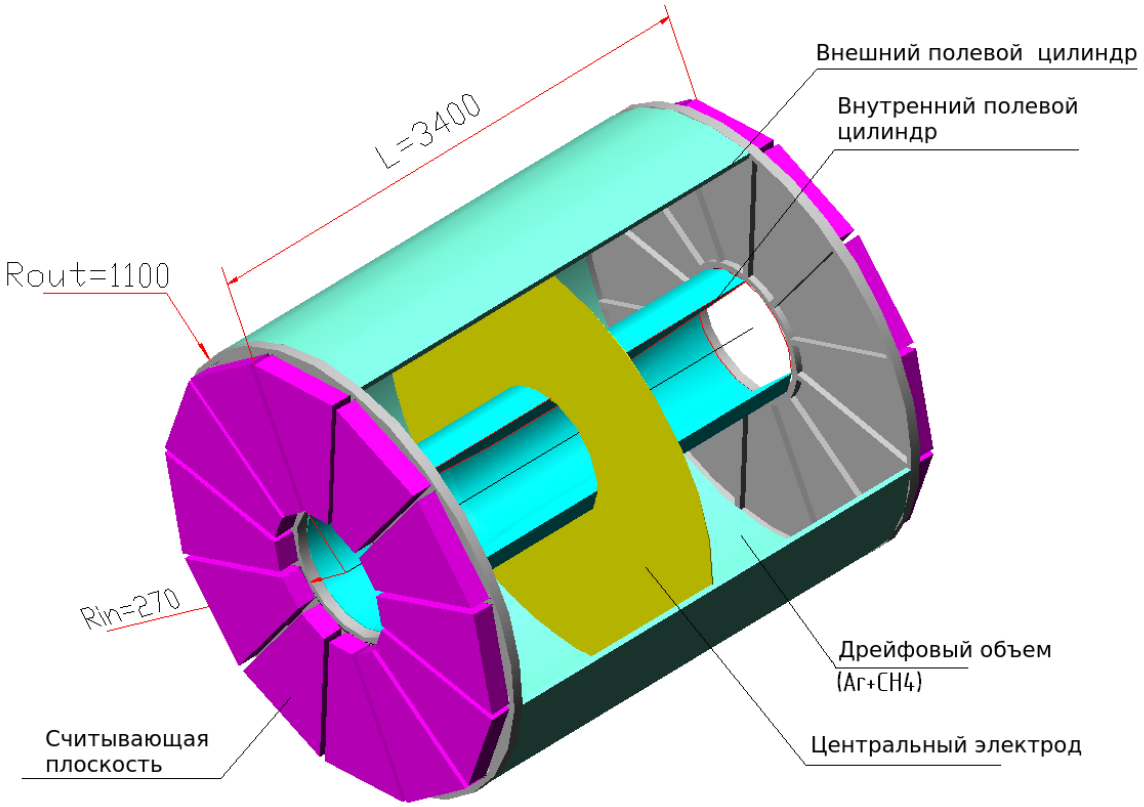

Рис. 1.1. Время-проекционная камера детектора MPD.

Основные этапы работы TPC:

- во внутренней трубке TPC происходит столкновение частиц высоких энергий; продукты столкновения, разлетаясь, попадают в рабочий объем TPC, заполненный газовой смесью;
- пролетая сквозь газ, частицы ионизируют его; под действием электрического поля положительные ионы движутся к центральным электродам, а сорванные электроны в виде сгустков (электронных кластеров) – к считывающим торцевым плоскостям;
- электронные кластеры попадают в пропорциональную камеру, где вызывают электронные лавины с зарядом, пропорциональным количеству электронов в кластере;
- лавины попадают на считывающие плоскости, где создают различные наведеные заряды на падах;

Таким образом, в результате работы TPC, получаются данные об уровне заряда на падах в зависимости от времени. Схематичное изображение наведенных зарядов на считывающей плоскости от одной электронной лавины представлено на рис. 1.2. Более темным цветом обозначены области с большим наведенным зарядом.

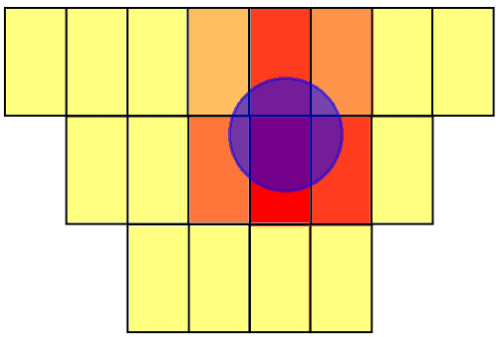

Рис. 1.2. Упрощенная схема считывающей плоскости с наведенными зарядами.

1.2. Входные данные алгоритма

Входными данными алгоритма является массив сигналов со считывающих плоскостей TPC вида TpcDigits[Sector][Row][Col][TimeBucket], где Sector – номер одного из 12 трапецевидных секторов двух считывающих плоскостей TPC, Row – номер «строки» пада, Col – номер «колонки» пада. По этим трем номерам однозначно определяются координаты X,Y центра пада. TimeBucket – номер временного отсчета, т.е. единица времени равная минимальной разрешающей способности по времени электроники детектора. Таким образом, массив содержит амплитуду сигналов, выраженную в зарядах электронов в точках с координатами X и Y, измеренными с точностью до полуширины(полувысоты) пада и временем, измеренным с точностью до временного отсчета.

В настоящее время используется массив TpcDigits, полученный в результате моделирования работы TPC, реализованного в комплексе MpdRoot. В моделировании в качестве генератора событий используется URQMD [3], транспорт частиц моделируется с помощью пакета GEANT3 [4], геометрия детектора моделируется средствами MpdRoot.

1.3. Задача алгоритма

Задачей алгоритма является восстановление исходного трехмерного распределения координат электронных кластеров по данным со считывающих плоскостей, то есть нахождение точек трека частицы.

Глава 2

Алгоритм пространственной реконструкции откликов частиц

2.1. Обзор существующих подходов

Большинство существующих подходов опирается на следующую схему:

- алгоритм исполняется для трехмерного массива вида TpcDigits[Row][Col][TimeBucket] (берется один сектор, либо секторное разбиение может отсутствовать);
- фиксируется одна координата, являющяся координатой центров пэдов одной из «строк» падов Row;
- ищутся значения в плоскости (Col,TimeBucket), содержащие сигналы выше уровня шума;
- выделяются сигналы отдельных кластеров из этих значений;
- вычисляются две оставшихся координаты кластеров усреднением или с помощью каких-либо аппроксимаций формы сигнала.

В частности, на данную схему опирается алгоритм реконструкции откликов частиц в экспериментах ALICE [5], MIPP [6], STAR [7].

2.2. Реализация алгоритма

Основная идея созданного в рамках данной работы алгоритма взята из эксперимента MIPP, Fermilab [6].

Алгоритм исполняется отдельно для каждого сектора, а в секторе для каждой строки падов, то есть падов с одинаковой координатой центра Y.

Рассмотрим работу алгоритма на примере массива тестовых данных в виде сигналов от четырех электронных кластеров, причем, сигналы от двух из них перекрываются. Массив представляет собой сигналы от одной строки падов в зависимости от временного отсчета. Сигналы заданы в виде условных единиц, уровень шума (то есть минимальный уровень полезного сигнала) в алгоритме для тестовых данных установлен равным нулю. Тестовые данные изображены на рис. 2.1. Цветом показан уровень сигнала.

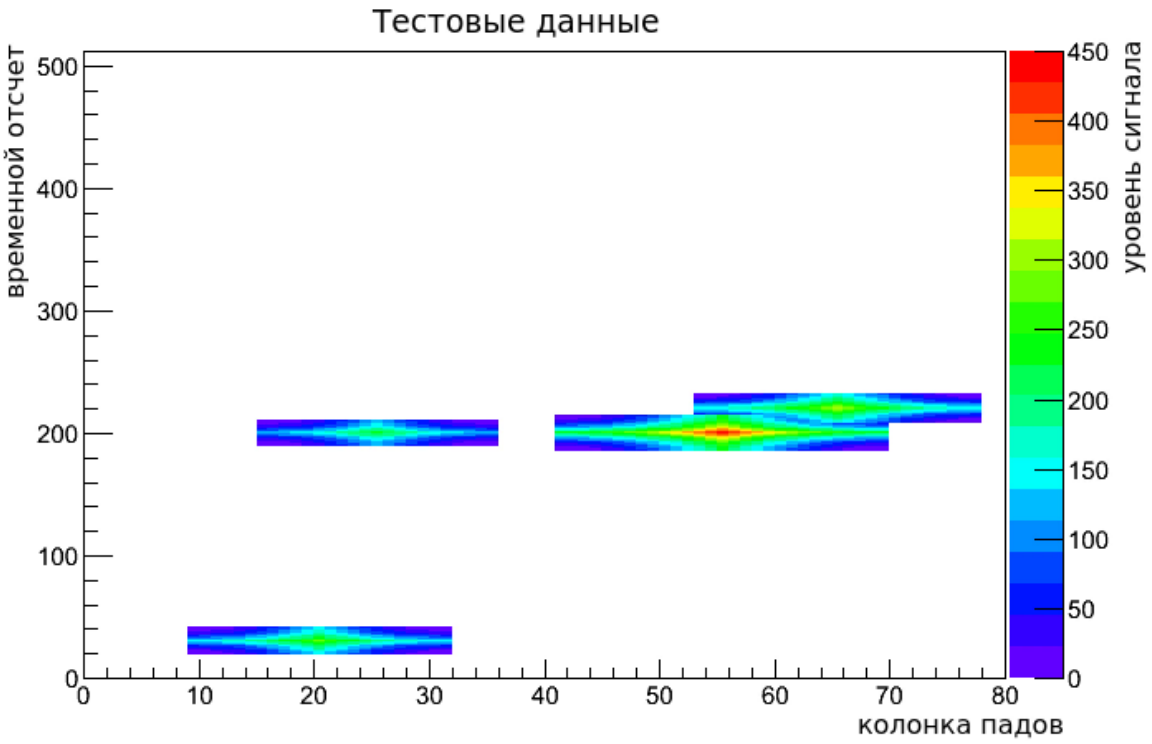

Рис. 2.1. Тестовые данные для алгоритма.

2.2.1. Поиск «протяженных кластеров»

На данном этапе:

1. осуществляется перебор массива значений;
2. если значение не отмечено, оно отмечается добавлением номера и проверяются все соседние значения;
3. если одно из соседних значений выше уровня шума и не отмечено, оно отмечается тем же номером и для него проверяются все соседние значения;
4. п.3 повторяется до тех пор пока есть неотмеченные значения;
5. если среди соседей не оказалось ни одного значения выше уровня шума, которое еще не отмечено, возвращение в п.2.

Таким образом, рекурсивно отмечаются с присвоением номера все непрерывные области в плоскости (Col, TimeBucket). Они называются «протяженными кластерами». На рис. 2.2 показано, какие протяженные кластеры найдены на тестовых данных.

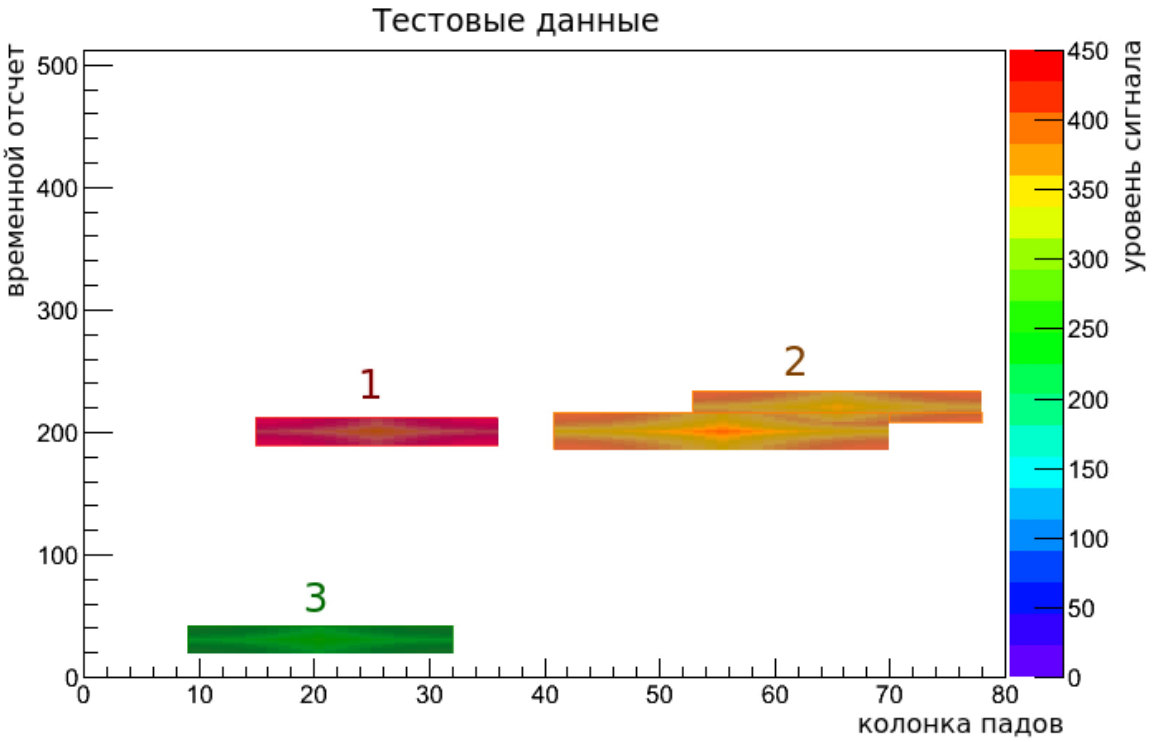

Рис. 2.2. Найденные «протяженные кластеры».

2.2.2. Пики амплитуды сигнала во времени

Для каждого пада протяженного кластера осуществляется поиск локальных пиков амплитуды сигнала во времени. Пиком считается значение сигнала, которое больше своих соседей. Для каждого из таких пиков вычисляется точное время, путем усреднения по всем соседним значениям, для которых амплитуда понижается при удалении от пика. Время пиков изображено на рис. 2.3. Видно, что у «протяженного кластера» №2 некоторые пады имеют по два пика во времени.

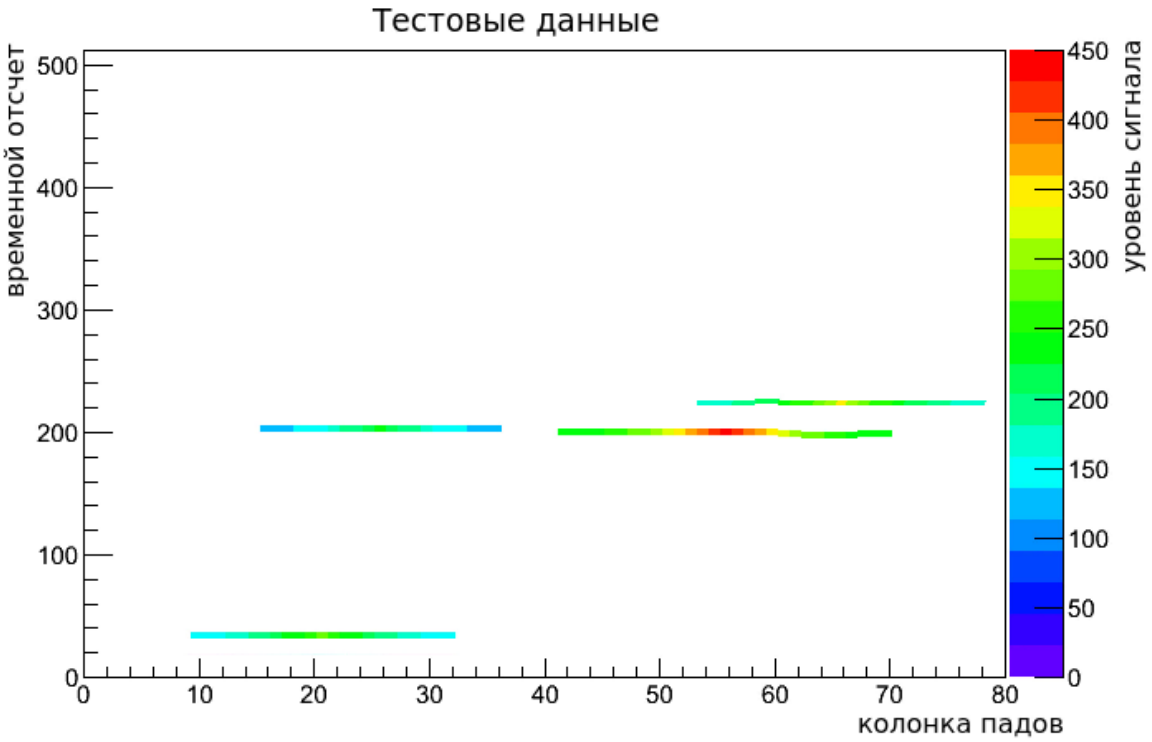

Рис. 2.3. Пики амплитуды сигнала во времени.

2.2.3. Создание «хитов»

Далее происходит объединение в т.н. «хиты» пиков в соседних по Col падах, для которых разница во времени не превышает 2 временных отсчета. После чего осуществляется взвешенное усреднение по всем временным отсчетам каждого «хита». Таким образом, для каждого «хита» получаем точку со средним значением колонки Col и временного отсчета TimeBucket. На рис. 2.4 показаны пики, объединенные в «хиты» и точка со средним значением Col и TimeBucket каждого «хита».

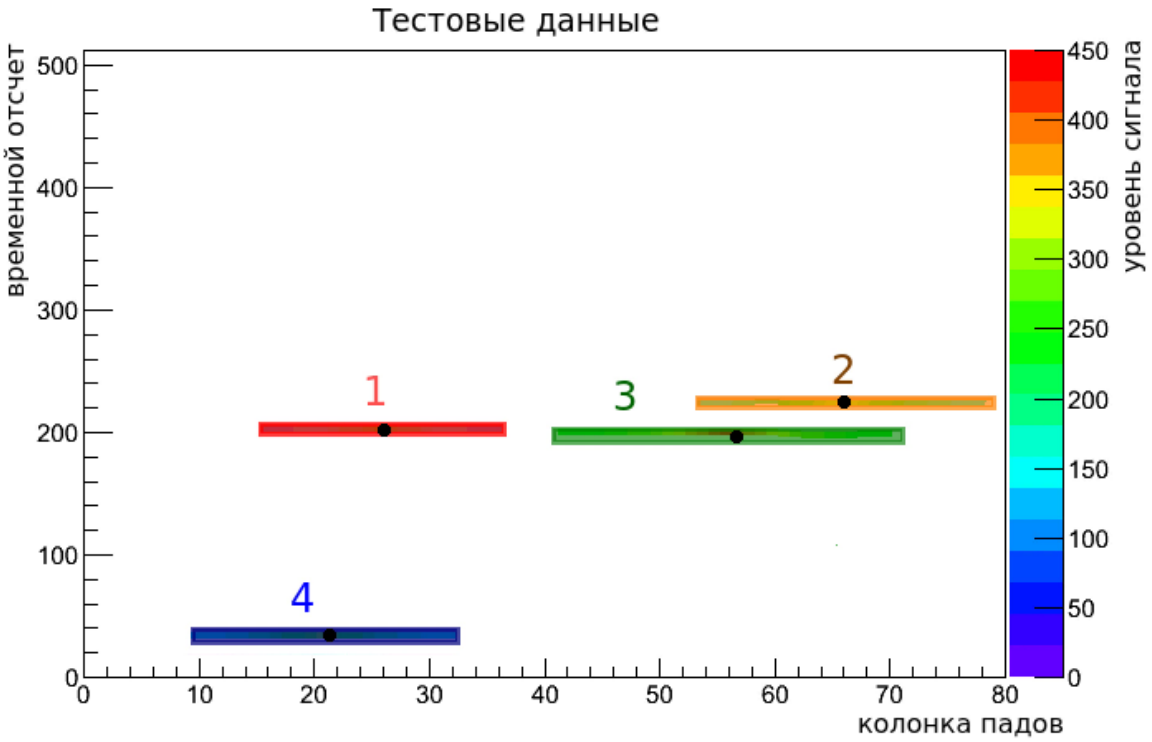

Рис. 2.4. «Хиты» и их координаты.

Таким образом, координата Y была зафиксирована, координата X однозначно вычисляется по среднему Col «хита», а координата Z вычисляется по среднему времени в TimeBucket и известной (и постоянной[1]) скорости дрейфа электронов к считывающим плоскостям.

[1] Вообще говоря, это не так. Существуют такие явления как случайное рассеяние электронов (диффузия), а также их отклонение под действием скопившегося положительного пространственного заряда в камере TPC, который создают не успевшие дойти до центрального электрода положительные ионы, образовавшиеся в результате отрыва пролетающими частицами электронов атомов газа (дисторсия). Однако в первом приближении этими явлениями можно пренебречь. В дальнейшем планируется на основе анализа распределения отклонений при наличии диффузии и дисторсии и без них (диффузию и дисторсию можно «выключать», т.к. данные получены в результате моделирования работы TPC) вычислить поправки для координат, как сделано, например, в эксперименте MIPP [6].

Глава 3

Оценка погрешностей

Оценка погрешностей алгоритма производилась как взвешенное среднеквадратичное отклонение при взвешенном усреднении по формулам:

$$rmsx^2 = \frac{\sum_{i=1}^{n} Adc_i (Col_i)^2}{\sum_{i=1}^{n} Adc_i} - \left(\frac{\sum_{i=1}^{n} Adc_i Col_i}{\sum_{i=1}^{n} Adc_i} \right)^2, \tag{3.1}$$

$$rmst^2 = \frac{\sum_{i=1}^{n} Adc_i (TimeBucket_i)^2}{\sum_{i=1}^{n} Adc_i} - \left(\frac{\sum_{i=1}^{n} Adc_i TimeBucket_i}{\sum_{i=1}^{n} Adc_i} \right)^2. \tag{3.2}$$

Здесь $Adc_i, Col_i, TimeBucket_i$ – i-ые значения амплитуды сигнала, колонки пада и временного семпла соответственно. Таким образом, полученные значения выражаются в абстрактных колонках падов и временных семплах. Чтобы перейти к сантиметрам, необходимо домножить $rmsx$ на ширину пада, а $rmst$ на «длину временного семпла», то есть на произведение скорости дрейфа электронов на время семпла (время семпла в проекте на данный момент равно 100 нс).

Были построены распределения погрешностей по данным одного события URQMD для координат X и Z отдельно для внутренних и внешних частей сектора, т.к. они содержат разные по размеру пады. В текущей геометрии проекта внутренние пады имеют размер 0.4x1.2 см, внешние – 0.5x1.8 см.

3.1. Результаты

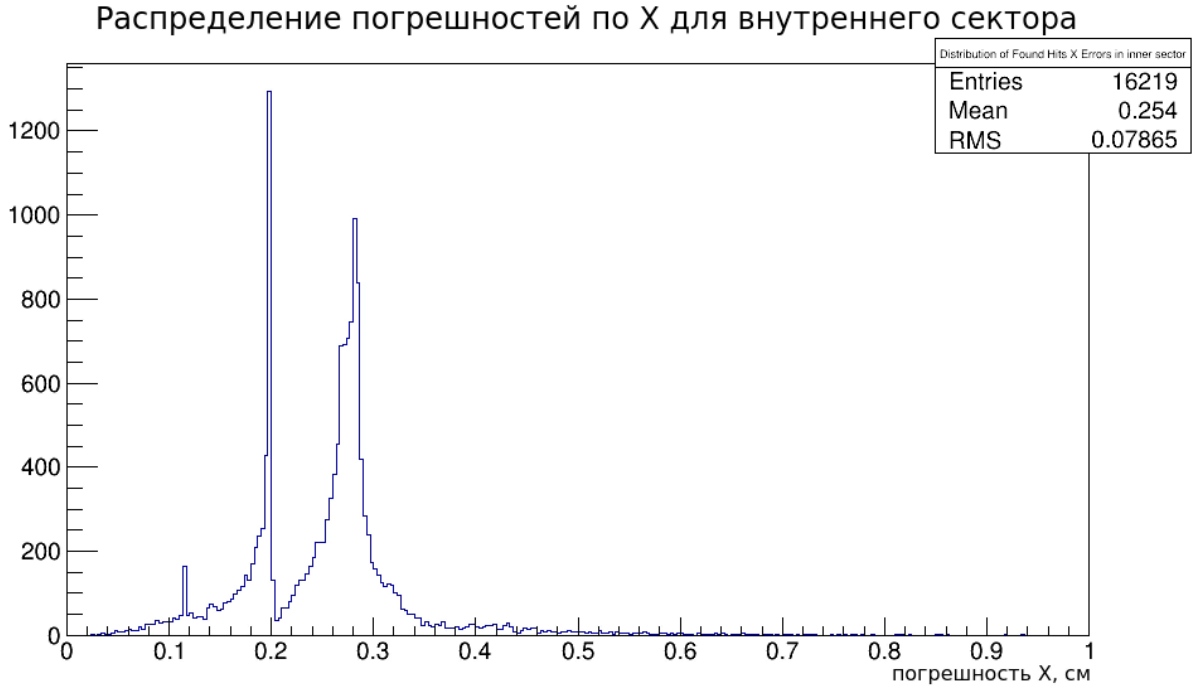

Рис. 3.1. Погрешность по X для внутренних падов.

Рис. 3.2. Погрешность по Z для внутренних падов.

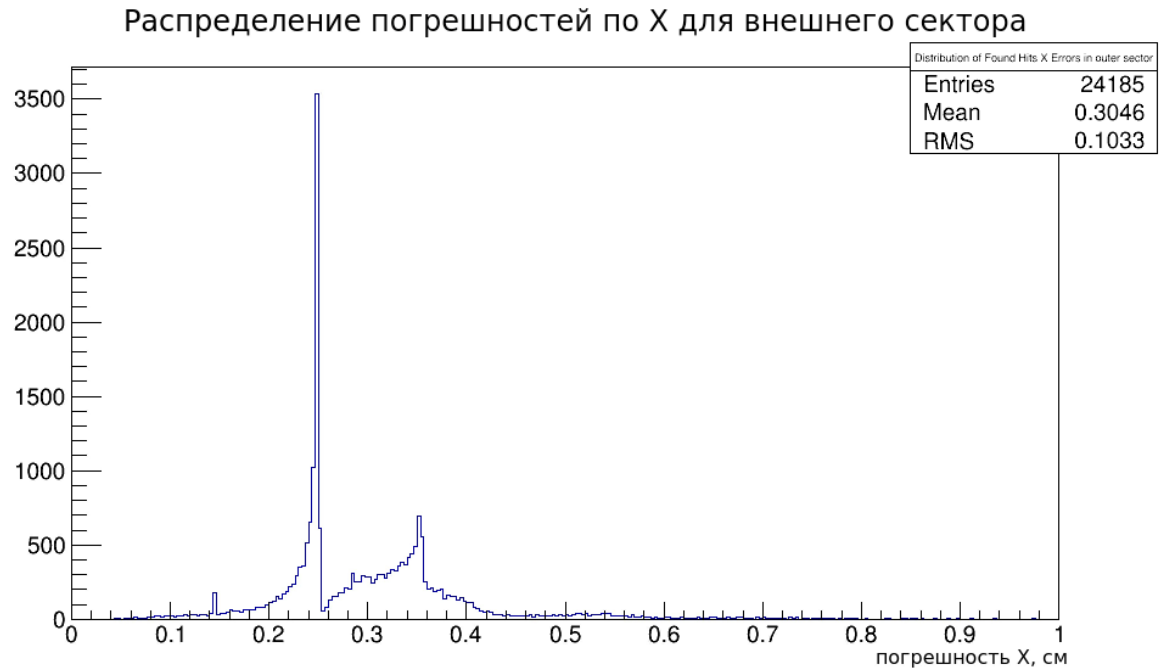

Рис. 3.3. Погрешность по X для внешних падов.

Рис. 3.4. Погрешность по Z для внешних падов.

3.2. Интерпретация результатов

Распределения имеют неоднородный характер, связанный с размером пада. Наблюдаются ярко выраженные пики для одно- и двупадовых кластеров. Полученные значения погрешностей являются ожидаемыми для таких рамеров падов и соотвествуют установленным требованиям.

Глава 4

Отклонения от исходных треков

Так как исходные данные для алгоритма получены в результате моделирования, то параметры исходных треков известны, что позволяет построить распределения отклонений от исходных треков.

Исходные треки известны в виде набора точек, т.н. «Монте-карловские» точки. Они получены в результате работы генератора событий URQMD и моделирования транспортировки частиц с помощью пакета GEANT3 сквозь геометрию детектора, моделируемую средствами MpdRoot. В моделировании работы TPC на основе этого набора точек формируются электронные кластеры, распределенные между этими точками равновероятно.

Для исследования отклонений «хитов» от треков электронные кластеры создавались в «Монте-карловских» точках. Кроме того, была отключена диффузия электронов при их движении к считывающим плоскостям. То есть реализовался идеальный случай, в котором отклонения от треков могут генерироваться только алгоритмом реконструкии откликов частиц.

Отклонение по координате вычислялось как разность координаты «хита» и соответствующей координаты ближайшей (в пространстве) к «хиту» «Монте-карловской» точки.

Как и в случае с погрешностью, были построены распределения отклонений по данным одного события URQMD для координат X и Z отдельно для внутренних и внешних частей сектора, т.к. они содержат разные по размеру пады. Также была исследована зависимость отклонений от поперечного импульса и от псевдобыстроты частиц по данным 80 событий URQMD.

4.1. Результаты. Полные отклонения

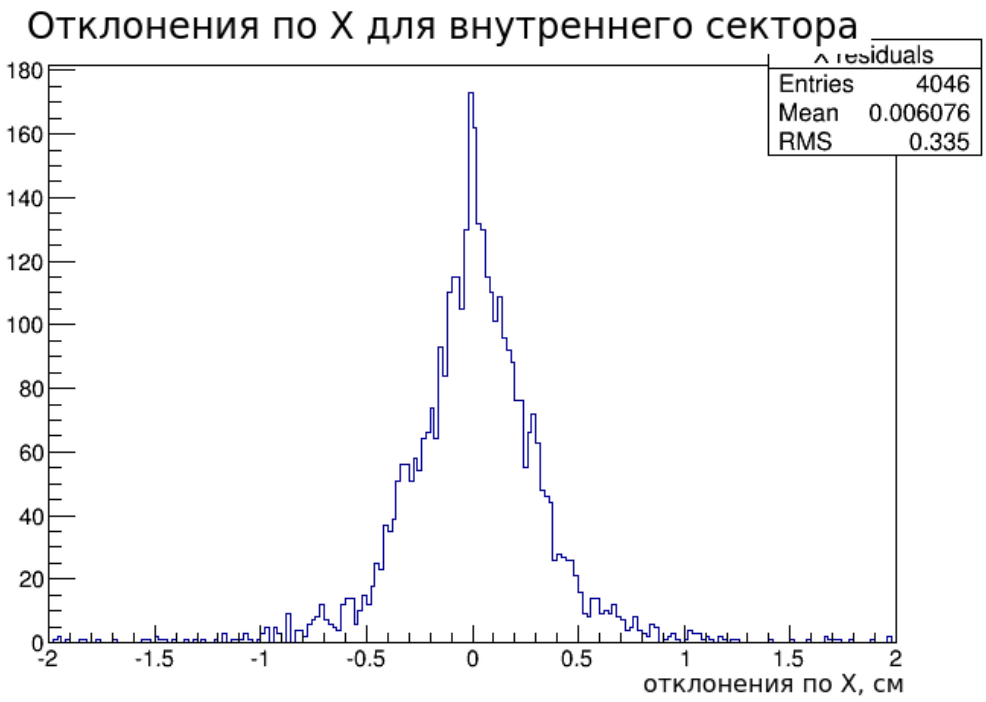

Рис. 4.1. Отклонения по X для внутренних падов.

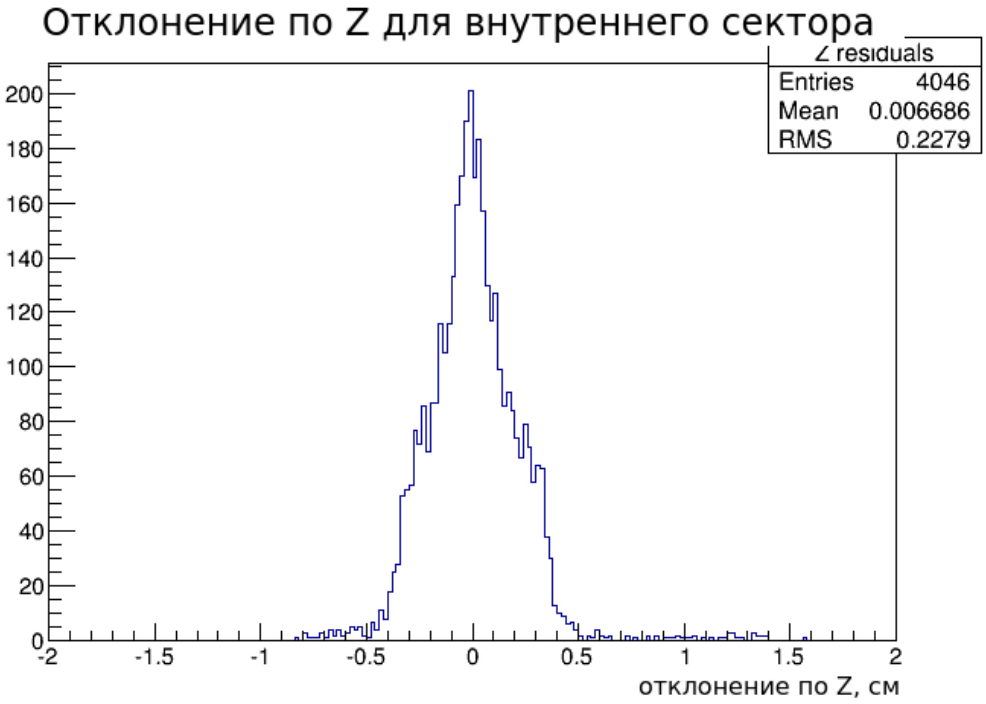

Рис. 4.2. Отклонения по Z для внутренних падов.

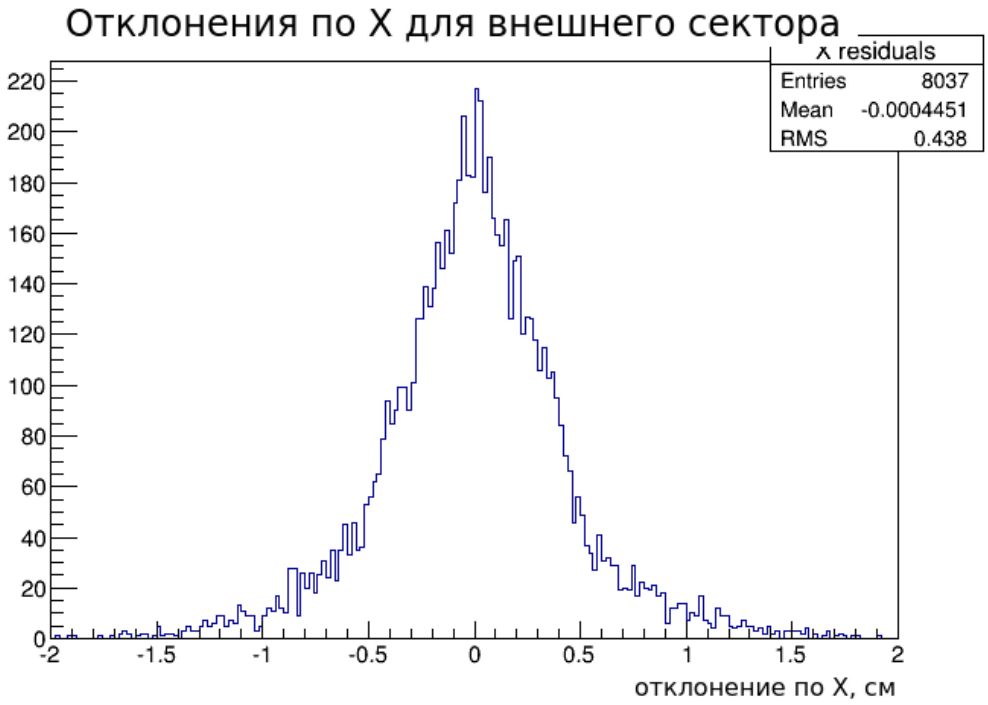

Рис. 4.3. Отклонения по X для внешних падов.

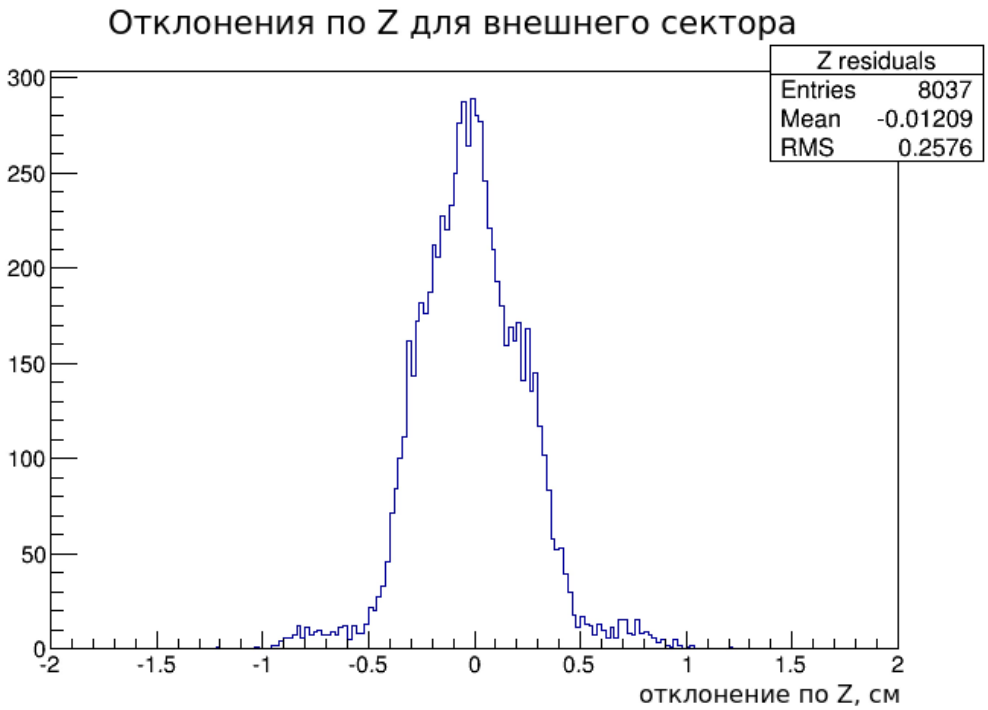

Рис. 4.4. Отклонения по Z для внешних падов.

4.2. Результаты. Отклонения в зависимости от поперечного импульса и псевдобыстроты

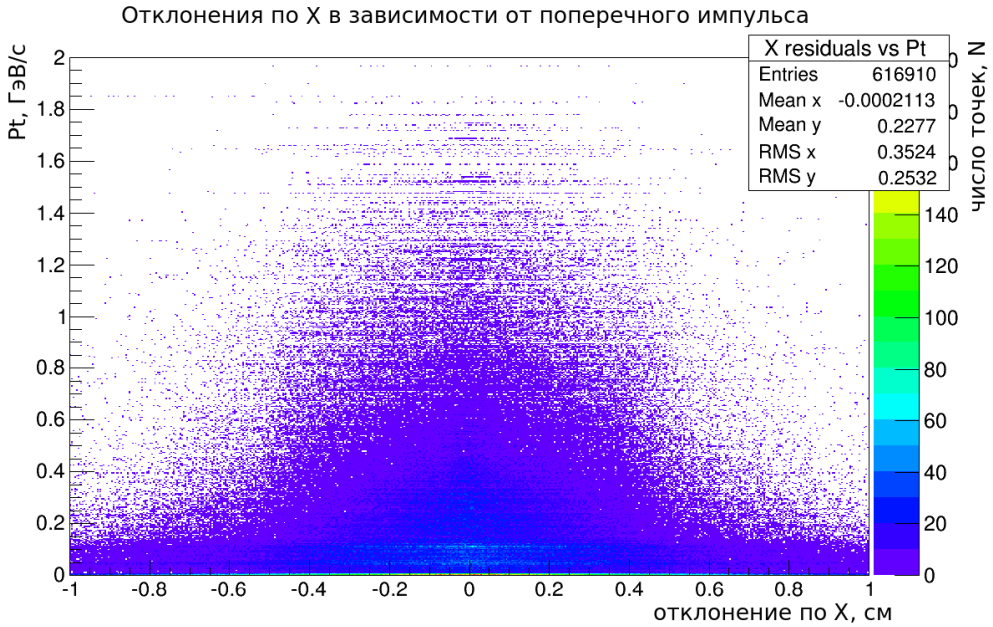

Рис. 4.5. Отклонения по X в зависимости от поперечного импульса.

Рис. 4.6. Отклонения по Z в зависимости от поперечного импульса.

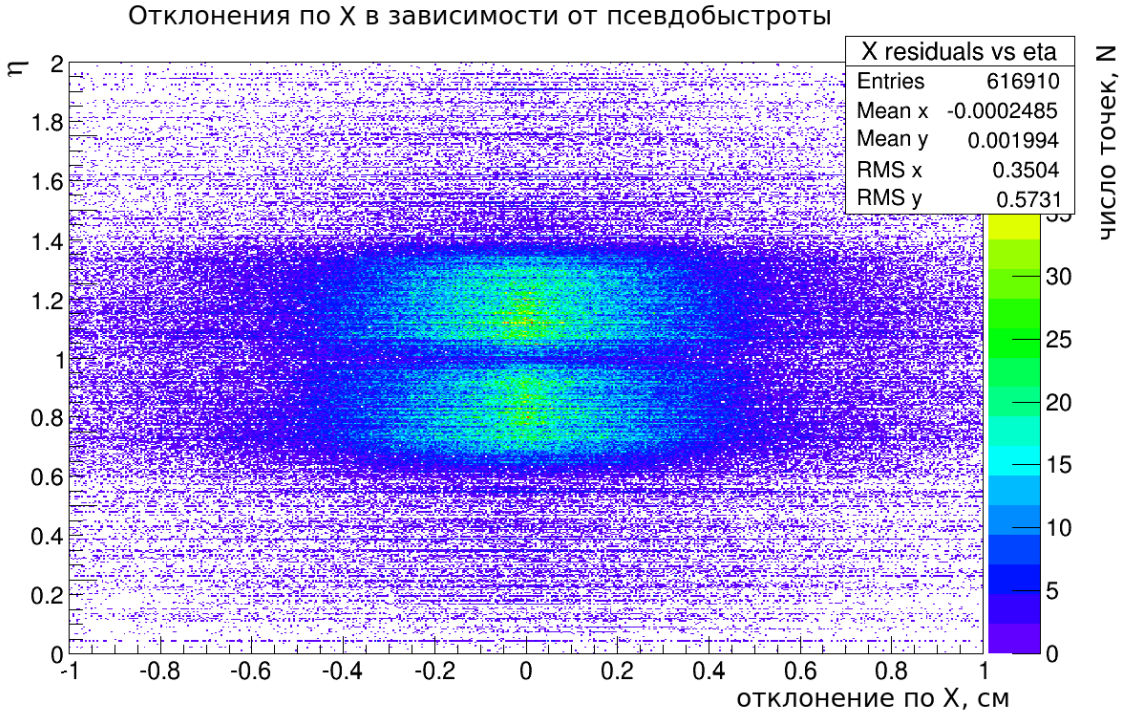

Рис. 4.7. Отклонения по X в зависимости от псевдобыстроты.

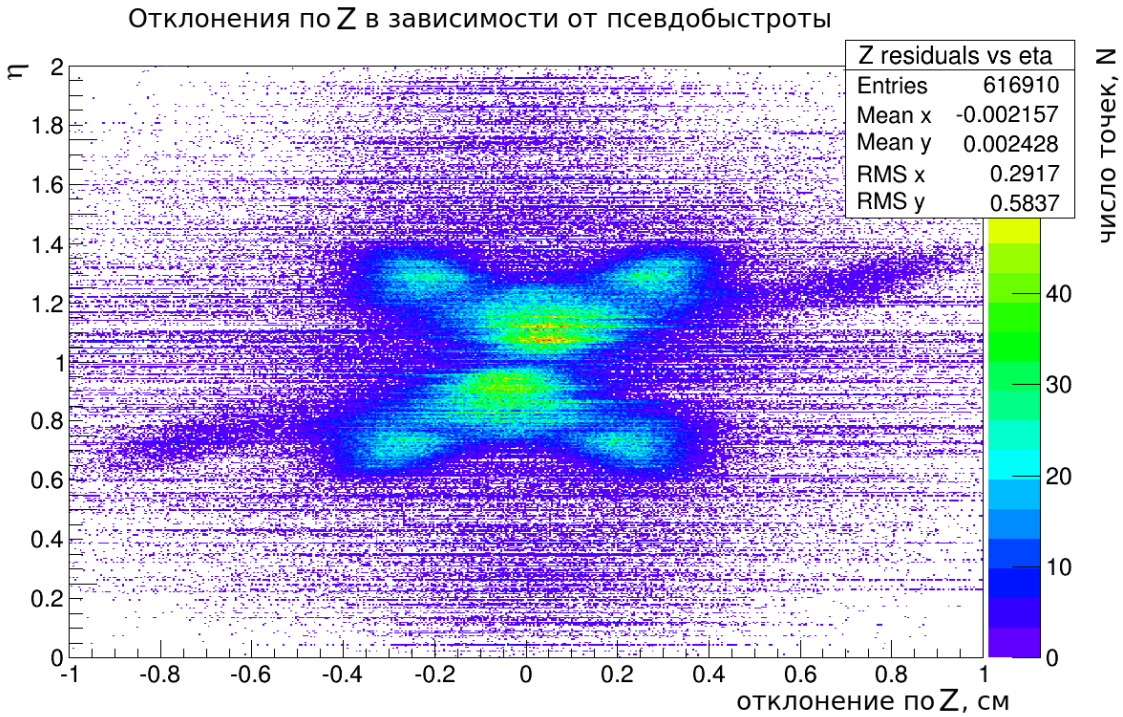

Рис. 4.8. Отклонения по Z в зависимости от псевдобыстроты.

4.3. Интерпретация результатов

Как и ожидалось, отклонения зависят от размера падов, причем по Z зависят слабее, т.к. длительность временного отсчета одинакова для всех падов. Числовые значения результатов для полных отклонений (рис.4.1-4.4) получаются приемлемыми, в сравнении, например, с экспериментом STAR [8].

Характер зависимости отклонений от величины поперечного импульса получился ожидаемым. Поперечный импульс характеризует «закрученность» трека. Чем меньше поперечный импульс, тем сильнее закручен трек. В связи с этим увеличивается неопределенность при восстановлении исходных координат кластеров. Это характерно как для координаты X(рис. 4.5), так и для Z(рис. 4.6). На рисунках видно, что при низких значениях поперечного импульса увеличивается дисперсия точек.

Характер зависимости отклонений от величины псевдобыстроты получился менее ожидаемым. Для отклонений по координате X по полученному изображению (рис. 4.7) нельзя сказать что наблюдается ярко выраженная зависимость, увеличение яркости цвета к центру вызвано тем, что точек для таких псевдобыстрот больше по количеству, при этом однозначно сделать вывод об увеличении дисперсии точек при росте псевдобыстроты по данному рисунку нельзя. Для отклонений по координате Z(рис. 4.8) при увеличении псевдобыстроты явно наблюдаются корреляции, плотность вероятности имеет два или даже три максимума. Это говорит о возможной систематической ошибке в алгоритме при определении координаты Z и требует тщательного изучения.

Заключение

В данной работе был реализован алгоритм реконструкции откликов частиц во время-проекционной камере детектора MPD на коллайдере NICA. Для этого были исследованы имеющиеся решения. На основе идей этих решений реализован уникальный алгоритм, учитывающий специфику TPC MPD. При оценке погрешности алгоритма были получены ожидаемые приемлемые результаты. Также были оценены отклонения от исходных треков. По этому параметру были получены приемлемые результаты, в целом соответствующие другим подобным экспериментам.

В целом, удалось создать алгоритм, удовлетворяющий заявленным требованиям. В дальнейшем планируется более подробное исследование погрешностей и отклонений, выяснение зависимости отклонений от различных параметров исходных треков и их минимизации, а также вычисление поправок на диффузию и дисторсию. Кроме этого, планируется улучшение быстродействия алгоритма за счет различных оптимизаций.

В заключение хочется выразить благодарности научному руководителю, Немнюгину С.А. за корректуру данной работы, ценные советы и замечания; Рогачевскому О.В. за информацию об известных способах решения задачи, оценку результатов и предложенные способы их улучшения; Мерцу С.П. за проверку и оптимизацию исходного кода, выявленные ошибки и недочеты.

Литература

1. MPD K. Многоцелевой детектор MPD для изучения столкновений тяжелых ионов на ускорителе NICA(Концептуальный дизайн-проект). Дубна: ОИЯИ, 2010. 224 с.

2. MpdRoot. Simulation and Analysis Framework for NICA/MPD Detectors. http://mpd.jinr.ru.

3. The Ultrarelativistic Quantum Molecular Dynamics model. http://urqmd.org/.

4. GEANT - Detector Description and Simulation Tool. http://wwwasd.web.cern.ch/wwwasd/geant/.

5. Fabjan C. W., Jirden L. C., Lindestruth V. K. et al. ALICE trigger data-acquisition high-level trigger and control system : Technical Design Report: Tech. Rep. 062: CERN LHC, 2003.

6. Yang X. Centrality Dependence of Strangeness Production in Proton-Nucleus Collisions at AGS Energies: Ph. D. thesis / COLUMBIA UNIVERSITY. 2000.

7. Lisa M. The STAR-TPC Clusterfinder/Hitfinder - STAR Note 238. Lawrence Berkley National Laboratory, Berkley, CA 94720 USA, 1996. — February.

8. The STAR time projection chamber: a unique tool for studying high multiplicity events at RHIC // Nuclear Instruments and Methods in Physics Research. 2003.

Приложение А

Исходный код алгоритма

```
void TpcClusterFinderTask::Exec(Option_t* opt) {

    cout << "TpcClusterFinder::Exec started" << endl;
    // Reset output Array
    if (!fHitsArray) Fatal("TpcClusterFinder::Exec)", "No FoundClustersArray");
    fHitsArray->Delete();

    vector<TpcFoundHit*> tpcHitList; // output vector of TpcHits

    for (UInt_t iSec = 0; iSec < nSect; ++iSec) {
        for (UInt_t iRows = 0; iRows < nRows; ++iRows) {
            Float_t** f2dArray = new Float_t* [fNumOfPadsInRow[iRows] * 2];
            for (UInt_t iPad = 0; iPad < fNumOfPadsInRow[iRows] * 2; ++iPad) {
                f2dArray[iPad] = new Float_t[nTimeBins];
                for (UInt_t timeBin = 0; timeBin < nTimeBins; ++timeBin) {
                    f2dArray[iPad][timeBin] = fStore->fBuffer[iSec][iRows][iPad][timeBin];
                }
            }
            vector<Tpc2dCluster*> extClusters; //vector of extended clusters
            Find2DClusters(&extClusters, f2dArray, iRows, iSec);
            vector<TpcPeak*> peakList; //vector of all peaks in cluster
            for (UInt_t iClust = 0; iClust < extClusters.size(); ++iClust) {
                FindPeaksInCluster(extClusters.at(iClust), &peakList);
                vector<TpcPeak*> collectedPeakList;
                while (!peakList.empty()) {
                    CollectPeaks(peakList, extClusters.at(iClust), &collectedPeakList);
                    CreateHit(collectedPeakList, extClusters.at(iClust), &tpcHitList);
                    for (UInt_t i = 0; i < collectedPeakList.size(); ++i) {
                        vector<TpcPeak*>::iterator pkItr = find(peakList.begin(), peakList.end(),
                            collectedPeakList[i]);
                        peakList.erase(pkItr);
                    }
                }
            }
            for (UInt_t iPad = 0; iPad < fNumOfPadsInRow[iRows] * 2; ++iPad)
                delete [] f2dArray[iPad];
        }
    }


    for (UInt_t iHit = 0; iHit < tpcHitList.size(); ++iHit) {
        TpcFoundHit* hit = tpcHitList.at(iHit);

        const Int_t Sect = hit->GetSect();
        const Float_t sectPhi = Sect * TwoPi() / 12;

        const Float_t lX = hit->X(); // local X coordinate
        const Float_t lY = hit->Y(); // local Y coordinate
        const Float_t lZ = hit->Z(); // local Z coordinate

        const Float_t gX = (lY + r_min) * Cos(sectPhi) - lX * Sin(sectPhi); // global X coordinate
        const Float_t gY = (lY + r_min) * Sin(sectPhi) + lX * Cos(sectPhi); // global Y coordinate
        const Float_t gZ = (Sect < nSect / 2) ? lZ : lZ * (-1); // global Z coordinate

        const Float_t Q = hit->QADC(); //charge of hit
        Int_t outSize = fHitsArray->GetEntriesFast();
        new((*fHitsArray)[outSize]) MpdTpcHit(0/*detector ID*/, TVector3(gX, gY, gZ), TVector3(hit->errX(),
            hit->errY(), hit->errZ()), 0/*index of MC*/); //TODO correct index and detID
        MpdTpcHit* outHit = (MpdTpcHit*) (fHitsArray->At(outSize));
```

```
            outHit->SetQ(Q);
            outHit->SetLocalXYZ(lX, lY, lZ);
            outHit->SetLayer(hit->Cluster()->Row());

            if (fMakeQA) {
                fHisto->_hXY->Fill(lX, lY, Q);
                fHisto->_hX->Fill(lX, Q);
                fHisto->_hY->Fill(lY, Q);
                fHisto->_hZ->Fill(lZ, Q);
                fHisto->_hX_global->Fill(gX, Q);
                fHisto->_hY_global->Fill(gY, Q);
                fHisto->_hZ_global->Fill(gZ, Q);
                fHisto->_hXY_global->Fill(gX, gY, Q);
                fHisto->_hPeak->Fill(Q);
                if (hit->PadRow() < nInRows) {
                    fHisto->_hErrX_inner->Fill(hit->errX());
                    fHisto->_hErrZ_inner->Fill(hit->errZ());
                } else {
                    fHisto->_hErrX_outer->Fill(hit->errX());
                    fHisto->_hErrZ_outer->Fill(hit->errZ());
                }
                fHisto->_hErrY->Fill(hit->errY());
                fHisto->_h3D->Fill(gX, gY, gZ, Q);
                if (hit->GetSect() == 3 && lY <= 1.0) {
                    fHisto->_hXT_clust_row1->Fill(lX, lZ, Q);
                }

                fHisto->_hNumOfDigitsInCluster->Fill(hit->Cluster()->GetNumDigits());
                fHisto->_hSect->Fill(hit->Cluster()->GetSect(), hit->Cluster()->GetADC());
                fHisto->_hNumOfPadsInCluster->Fill(hit->Cluster()->GetNumPads());
                fHisto->_hNumOfTimeBinsInCluster->Fill(hit->Cluster()->GetNumTimeBins());
            }
        }
        if (fPrintDebugInfo) cout << "Number of Found Clusters = " << fHitsArray->GetEntriesFast() << endl;
        cout << "TpcClusterFinder::Exec finished" << endl;
    }

    Bool_t TpcClusterFinderTask::Find2DClusters(vector<Tpc2dCluster*> *extClusters, Float_t **f2dArray, UInt_t
        row, UInt_t sec) {

        Float_t curADC;
        UInt_t curDigit[4];
        Bool_t result;
        UInt_t cluscount = 0;

        Bool_t** fADCMarks = new Bool_t* [fNumOfPadsInRow[row] * 2];
        for (UInt_t i = 0; i < fNumOfPadsInRow[row] * 2; ++i) {
            fADCMarks[i] = new Bool_t[nTimeBins];
            for (UInt_t j = 0; j < nTimeBins; ++j)
                fADCMarks[i][j] = kFALSE;
        }

        for (UInt_t pad = 0; pad < fNumOfPadsInRow[row] * 2; ++pad) {
            for (UInt_t tBin = 0; tBin < nTimeBins; ++tBin) {
                curADC = f2dArray[pad][tBin];
                if (curADC < fNoiseThreshold) continue;

                curDigit[0] = row;
                curDigit[1] = pad;
                curDigit[2] = tBin;
                curDigit[3] = curADC;

                if (!fADCMarks[pad][tBin]) { //if current ADC has no mark yet

                    Tpc2dCluster* cluster = new Tpc2dCluster(row, sec);
                    fADCMarks[pad][tBin] = kTRUE;
```

```
                cluster->SetID(cluscount++);

                if (!cluster->Insert(row, pad, tBin, curADC)) {
                    cout << "Failed to insert digit into cluster..." << endl;
                    return kFALSE;
                }

                // initialize the max,min buckets and columns.
                cluster->SetMinBkt(tBin);
                cluster->SetMaxBkt(tBin);
                cluster->SetMinCol(pad);
                cluster->SetMaxCol(pad);

                // now that we have the first digit in the 2d cluster, the following
                //  will find the rest of the connected digits
                result = GetNextDigit(curDigit, cluster, row, f2dArray, fADCMarks);

                if (!result) return kFALSE;
                if (fPrintDebugInfo && cluster->GetADC() < fNoiseThreshold) {
                    cout << "Warning: empty cluster!" << endl;
                }
                extClusters->push_back(cluster);
            }
        }
    }

    for (UInt_t iPad = 0; iPad < fNumOfPadsInRow[row] * 2; ++iPad)
        delete [] fADCMarks[iPad];
    return kTRUE;
}

Bool_t TpcClusterFinderTask::GetNextDigit(UInt_t* currdig, Tpc2dCluster* Clus2d, UInt_t row, Float_t **
    fDigitsArray, Bool_t **fADCMarks) {

    UInt_t thisCol, thisRow, thisBkt;
    UInt_t nextCol, nextBkt, nextADC;

    thisCol = currdig[1];
    thisRow = currdig[0];
    thisBkt = currdig[2];
    for (int i = -1; i < 2; ++i) {

        // check left-right bounds...
        if (thisCol == 0 && i < 0) continue;
        if (thisCol == fNumOfPadsInRow[row] * 2 - 1 && i >= 0) continue;
        nextCol = thisCol + i; // look right, left, check if we're at the edge

        for (int j = -1; j < 2; ++j) {

            if (i == 0 && j == 0) continue;

            // check up-down bounds...
            if (thisBkt == 0 && j < 0) continue;
            if (thisBkt == nTimeBins - 1 && j >= 0) continue;
            nextBkt = thisBkt + j; // look down, up, check if we're at the edge

            nextADC = fDigitsArray[nextCol][nextBkt];
            if (nextADC < fNoiseThreshold) continue;
            UInt_t nextdig[4] = {row, nextCol, nextBkt, nextADC};

            if (!fADCMarks[nextCol][nextBkt]) {
                if (!Clus2d->Insert(row, nextCol, nextBkt, nextADC)) {
                    cout << "Failed to insert digit into cluster..." << endl;
                    return kFALSE;
                }
```

```
                fADCMarks[nextCol][nextBkt] = kTRUE;

                if (Clus2d->MaxBkt() < thisBkt) Clus2d->SetMaxBkt(thisBkt);
                if (Clus2d->MinBkt() > thisBkt) Clus2d->SetMinBkt(thisBkt);
                if (Clus2d->MaxCol() < thisCol) Clus2d->SetMaxCol(thisCol);
                if (Clus2d->MinCol() > thisCol) Clus2d->SetMinCol(thisCol);

                GetNextDigit(nextdig, Clus2d, row, fDigitsArray, fADCMarks);
            }
        }
    }
    return kTRUE;
}

void TpcClusterFinderTask::FindPeaksInCluster(Tpc2dCluster* clust, vector<TpcPeak*> * PeakList) {

    const UInt_t fPeakValleyRatio = 2; // maximum of peak / minimum of peak
    const Int_t mincol = clust->MinCol();
    const Int_t minbkt = clust->MinBkt();
    const Int_t nbkt = clust->GetNumTimeBins();
    const Int_t ncol = clust->GetNumPads();
    const Int_t ndig = clust->GetNumDigits();

    Float_t xyADC[ncol][nbkt];

    for (Int_t i = 0; i < ncol; ++i)
        for (Int_t j = 0; j < nbkt; ++j)
            xyADC[i][j] = 0.0;

    for (Int_t i = 0; i < ndig; ++i) // data extraction block.
        xyADC[clust->Col(i) - mincol][clust->Bkt(i) - minbkt] = clust->Adc(i);

    // now loop through each pad and extrapolate individual peaks

    /// Set proper values for our TPC
    ///minthresh is a minimum ADC required for peak to be created
    Float_t adcmin = 1000.0;
    Float_t minthresh = adcmin + 2 * sqrt(adcmin);

    for (Int_t i = 0; i < ncol; ++i) {

        Float_t thresh = 0.;
        Float_t max = 0.;
        Float_t min = 1.e9;
        Int_t imax = -1;
        Int_t imin = -1;
        Int_t npeak = 0;

        TpcPeak* peak = new TpcPeak();
        peak->SetCol(i + mincol);
        peak->AttachCluster(clust);
        Bool_t FoundPeak = kFALSE;

        for (Int_t j = 0; j < nbkt; ++j) {

            Float_t adc = xyADC[i][j];

            if (adc > minthresh) {
                if (peak->NSamples() == 0) {
                    peak->SetBktOff(minbkt + j);
                }
                peak->Insert(adc);
                if (adc > max) {
                    max = adc;
                    imax = i;
                    thresh = max - 2 * sqrt(max);
```

```
            }
        }

        if (FoundPeak && adc < min) {
            min = adc;
            imin = i;
        }

        if (adc < thresh) {
            FoundPeak = kTRUE;
        }

        if ((FoundPeak && adc < minthresh) ||
                // ^-- here we're below our min. threshold, so we save
                // the peak info. and create a new peak
                (FoundPeak && adc > min)) {
            //   ^-- here we've found a peak but the slope is rising... could be
            // entering a new peak, so save the peak info. and create a new one

            if (adc > min) // remove current ADC hit from peak
                peak->Remove(peak->NSamples() - 1);

            FoundPeak = kFALSE;

            if (peak->NSamples() > 1 && (peak->Max() / peak->Min() > fPeakValleyRatio)) {
                PeakList->push_back(peak);
                ++npeak;
                peak = new TpcPeak();
            } else {
                peak->Clear();
            }

            peak->SetCol(i + mincol);
            peak->SetBktOff(minbkt + j);
            peak->AttachCluster(clust);
            if (adc > minthresh) peak->Insert(adc);

            thresh = 0.;
            max = 0.;
            min = 1.e9;
            imax = -1;
            imin = -1;

        } // end check for 2nd peak (or clear end of 1st peak)
    } // end loop through buckets in this column

    // need to add check that _last_ peak was not already added to list!

    if (FoundPeak && peak->NSamples() > 1 && (peak->Max() / peak->Min() > fPeakValleyRatio)) {
        // check that we haven't already added this peak, just to be safe...
        vector<TpcPeak*>::iterator pkItr = find(PeakList->begin(), PeakList->end(), peak);
        if (pkItr == PeakList->end()) {
            PeakList->push_back(peak);
            ++npeak;
        }
    }

    if (npeak == 0) { // no peaks were found, so we'll store the ADC info
        // as a single "peak" anyway...

        peak->Clear();
        peak->SetCol(i + mincol);
        peak->AttachCluster(clust);

        for (Int_t j = 0; j < nbkt; ++j) {
            if (xyADC[i][j] > minthresh) {
```

```
                    if (peak->NSamples() == 0) peak->SetBktOff(minbkt + j);
                    peak->Insert(xyADC[i][j]);
                }
            }

            if (peak->NSamples() > 1)
                PeakList->push_back(peak);
        }

        // finally, check that peak has not been added to the peak list...
        // if not, then delete it!

        vector<TpcPeak*>::iterator pkItr = find(PeakList->begin(), PeakList->end(), peak);
        if (pkItr == PeakList->end()) delete peak;

    } // end loop thru pads in this column

}

void TpcClusterFinderTask::CollectPeaks(vector<TpcPeak*> peakList, Tpc2dCluster* clust, vector<TpcPeak*>*
    collectedPeakList) {

    Int_t mincol = clust->MinCol();
    Int_t ncol = clust->GetNumPads();
    Int_t npeaks[ncol];

    for (Int_t i = 0; i < ncol; ++i) npeaks[i] = 0;

    if (fitGamma)
        FitPeaks(peakList, npeaks, mincol);
    else {
        for (UInt_t i = 0; i < peakList.size(); ++i) {
            TpcPeak* peak = peakList[i];
            peak->SetChi2(0.0); // TMP
        }
    }

    collectedPeakList->clear();
    collectedPeakList->push_back(peakList[0]);
    // collect all peaks that line-up with each other
    Float_t tprev, tnext, tdiff; //time of current and previous peaks and time shift between two peaks
    Int_t cprev, cnext, cdiff; //pad of current and previous peaks and pad shift between two peaks

    if (peakList[0]->Chi2() > 0.)
        tprev = peakList[0]->PeakTime();
    else {
        Float_t x1 = peakList[0]->NSamples();
        //Float_t dt1 = par[0]*(1 - exp(-(par[1] - x1)*(par[1] - x1) / par[2]));
        tprev = peakList[0]->Mean() + peakList[0]->BktOff(); // - dt1;
    }

    cprev = peakList[0]->Col();

    for (UInt_t i = 1; i < peakList.size(); ++i) {
        if (peakList[i]->Chi2() > 0.)
            tnext = peakList[i]->PeakTime();
        else {
            Float_t x1 = peakList[i]->NSamples();
            //Float_t dt1 = par[0]*(1 - exp(-(par[1] - x1)*(par[1] - x1) / par[2]));
            tnext = peakList[i]->Mean() + peakList[i]->BktOff(); // - dt1;
        }
        tdiff = tnext - tprev;
        cnext = peakList[i]->Col();
        cdiff = cnext - cprev;

        if (fabs(tdiff) < 2. && abs(cdiff) == 1) {
```

```
            collectedPeakList->push_back(peakList[i]);
            cprev = cnext;
        }
    }
}

void TpcClusterFinderTask::CreateHit(vector<TpcPeak*> collectedPeakList, Tpc2dCluster* clust, vector<
    TpcFoundHit*> *hitList) {

    Bool_t oneHitCluster = kFALSE;
    if (clust->GetNumPads() == 1 || clust->GetNumTimeBins() == 1) oneHitCluster = kTRUE;

    if (!oneHitCluster && !collectedPeakList.empty()) {

        TpcFoundHit* hit = new TpcFoundHit(clust);

        // now calculate <t>, sig<t>, <x> and sig<x>:
        Float_t avgt = 0.0, avgx = 0.0;
        Float_t sumq = 0.0;
        Float_t sigt = 0.0, sigx = 0.0;
        Float_t w = 0.0, dw = 0.;
        Bool_t GoodHit = kFALSE;

        if (collectedPeakList.size() == 1) { // handle the case where a single _clear_
            // peak is found, but no neighboring peaks ...
            TpcPeak* peak = collectedPeakList[0];
            Float_t pTime = peak->PeakTime();
            Float_t pInt = peak->Integral();
            Int_t pCol = peak->Col();
            if (peak->Chi2() > 0. && peak->Chi2() < 30. && peak->Max() > 100.) {

                hit->SetQFit(pInt);
                hit->SetSigQFit(peak->IntegSig());
                hit->SetType(TpcFoundHit::kFitPeak);

                // now estimate sigx and sigt...
                // draw a 3x3 box around peak
                Float_t xyADC[3][5];
                Float_t xADC[3];
                memset(xyADC, 0, sizeof (xyADC));
                memset(xADC, 0, sizeof (xADC));
                int ncol = 1;

                for (int ih = 0; ih < clust->NDigits(); ++ih) {
                    int ic = (Int_t) (clust->Col(ih) - pCol);
                    int ib = (Int_t) (clust->Bkt(ih) - pTime);
                    if (abs(ic) <= 1 && abs(ib) <= 2) {
                        xyADC[ic + 1][ib + 2] = clust->Adc(ih);
                        xADC[ic + 1] += xyADC[ic + 1][ib + 2];
                    }
                }

                if (xADC[0] > 0) ++ncol;
                if (xADC[2] > 0) ++ncol;

                if (ncol > 1) {

                    // first deal with x-position
                    avgx = 0.33 * xADC[0] * (pCol - 1) + 0.33 * xADC[2] * (pCol + 1) + pInt * pCol;
                    avgx /= (0.33 * (xADC[0] + xADC[2]) + pInt);
                    sigx = 0.33 * xADC[0] * (pCol - 1 - avgx) * (pCol - 1 - avgx) + 0.33 * xADC[2] * (pCol +
                        1 - avgx) * (pCol + 1 - avgx) + pInt * (pCol - avgx) * (pCol - avgx);
                    sigx /= (0.33 * xADC[0] + pInt + 0.33 * xADC[2]) * (ncol - 1);

                    if (sigx < 0.0001) // this is completely unrealistic, so we assume something really bad
                        happened here...
```

```
                sigx = kOneOverSqrt12;
            else
                sigx = sqrt(sigx);

            // now deal with y-position
            Float_t y1 = 0;
            for (int ib = -2; ib <= 2; ++ib)
                y1 += (Int_t) (pTime + ib) * xyADC[0][ib + 2];
            if (xADC[0] > 0)
                y1 /= xADC[0];

            Float_t y2 = 0;
            for (int ib = -2; ib <= 2; ++ib)
                y2 += (Int_t) (pTime + ib) * xyADC[2][ib + 2];
            if (xADC[2] > 0)
                y2 /= xADC[2];

            avgt = 0.33 * xADC[0] * y1 + 0.33 * xADC[2] * y2 + pTime * pInt;
            avgt /= (0.33 * (xADC[0] + xADC[2]) + pInt);

            if (fabs(pTime - avgt) < 1.) {
                // looks reasonable, so continue

                sigt = 0.33 * xADC[0] * (y1 - avgt) * (y1 - avgt) + 0.33 * xADC[2] * (y2 - avgt) * (
                    y2 - avgt) + pInt * (pTime - avgt) * (pTime - avgt);
                sigt /= (0.33 * xADC[0] + pInt + 0.33 * xADC[2]) * (ncol - 1);

                if (sigt < 0.001)
                    sigt = peak->SigMean() / peak->SumADC();
                else
                    sigt = sqrt(sigt);

                /// Correction for TPC MPD may be added here
                //          if (fUseTCorr)
                //            avgt -= fTCorrTable->TCorr(peak->Row(),peak->Col());

                GoodHit = kTRUE;
            }
        }
    }
} else { // more than one peak found in cluster...
    GoodHit = kTRUE;
    for (UInt_t i = 0; i < collectedPeakList.size(); ++i) {
        TpcPeak* peak = collectedPeakList[i];
        if (peak->Chi2() > 0.) { // this means the peak was successfully fit.
            w = peak->Integral();
            dw = peak->IntegSig();
            hit->SetQFit(hit->QFit() + w);
            hit->SetSigQFit(hit->QFitSig() + dw * dw);
            hit->SetType(hit->Type() | TpcFoundHit::kFitPeak);
            Float_t myt = peak->PeakTime();
            /// Correction for TPC MPD may be added here
            // if (fUseTCorr) myt -= fTCorrTable->TCorr(peak->Row(),peak->Col());
            avgt += myt*w;
            sigt += myt * myt*w;
        } else { // fit was not successfully fit
            w = peak->SumADC();
            hit->SetQADC(hit->QADC() + peak->SumADC());
            hit->SetType(hit->Type() | TpcFoundHit::kWMPeak);
            Float_t x1 = peak->NSamples();
            //Float_t dt1 = par[0]*(1 - exp(-(par[1] - x1)*(par[1] - x1) / par[2]));
            Float_t myt = peak->Mean() + peak->BktOff(); // - dt1;
            /// Correction for TPC MPD may be added here
            //    if (fUseTCorr) myt -= fTCorrTable->TCorr(peak->Row(),peak->Col());
            avgt += myt * w;
            sigt += myt * myt * w;
```

```
            }
            avgx += peak->Col() * w;
            sigx += peak->Col() * peak->Col() * w;
            sumq += w;
        }
        hit->SetSigQFit(Sqrt(hit->QFitSig()));
        avgt /= sumq;
        avgx /= sumq;
        sigt = (sigt / sumq - avgt * avgt);
        sigx = (sigx / sumq - avgx * avgx);

        if (sigt <= 0.0) sigt = kOneOverSqrt12; //(hitPeaks.size()) * kOneOverSqrt12;
        else sigt = Sqrt(sigt);

        if (sigx <= 0.0) sigx = kOneOverSqrt12; //sigx = (hitPeaks.size()) * kOneOverSqrt12;
        else sigx = Sqrt(sigx);
    }

    Float_t pos[3], dpos[3];

    if (GoodHit) {

        Float_t padW, padH;
        Int_t Row = clust->Row();

        if (Row < nInRows) {
            padW = pwIn;
            padH = phIn;
            pos[0] = padW * ((Float_t) avgx - (Float_t) fNumOfPadsInRow[Row] + 0.5); // x-coordinate of
                pad center
            pos[1] = padH * ((Float_t) Row + 0.5); // y-coordinate of pad center
        } else {
            padW = pwOut;
            padH = phOut;
            pos[0] = padW * ((Float_t) avgx - (Float_t) fNumOfPadsInRow[Row] + 0.5); // x-coordinate of
                pad center
            pos[1] = fSectInHight + ((Float_t) (Row - nInRows) + 0.5) * padH; // y-coordinate of pad
                center
        }

        pos[2] = ((Float_t) avgt + 0.5) * (zDrift / nTimeBins);

        hit->SetPadCol(avgx);
        hit->SetTimeBkt(avgt);
        //adding errors
        Float_t timeBktSize = zDrift / nTimeBins;
        dpos[0] = sigx * padW; //dx
        dpos[1] = padH * kOneOverSqrt12; //dy
        dpos[2] = sigt * timeBktSize; //dz

        hit->SetPos(pos, dpos);

        hitList->push_back(hit);
    } else
        delete hit;

} else { //oneHitCluster is true  or collectedPeakList is empty

    Int_t iPad, iRow, iTime;
    Float_t y;

    // Initialize variables to 0
    Float_t avgx = 0.0, avgz = 0.0;
    Float_t rmsx = 0.0, rmsz = 0.0;
    Float_t adc = 0.0, sumadc = 0.0, invsum = 0.0;
```

```
Int_t ndig = clust->GetNumDigits();
for (UInt_t i = 0; i < ndig; ++i) {

    iPad = clust->Col(i);
    iTime = clust->Bkt(i);
    iRow = clust->Row(i);
    adc = clust->Adc(i);

    // y-coordinate of pad center
    y = (iRow < nInRows) ? (phIn * ((Float_t) iRow + 0.5)) : (fSectInHight + ((Float_t) (iRow -
        nInRows) + 0.5) * phOut);

    // Compute weighted sums of x, t, x^2, and t^2
    avgx += adc * iPad;
    avgz += adc * iTime;
    rmsx += adc * iPad * iPad;
    rmsz += adc * iTime * iTime;
    sumadc += adc;
}
// Make sure that we had at least one digit with charge
if (sumadc > 0.0) {
    invsum = 1.0 / sumadc;
    avgx *= invsum;
    avgz *= invsum;
    rmsx = rmsx * invsum - avgx * avgx;
    rmsz = rmsz * invsum - avgz * avgz;
}

if (clust->GetNumPads() == 1 || rmsx <= 0.0) {
    Float_t xErrOnePad = (y < nInRows * phIn) ? (pwIn * kOneOverSqrt12) : (pwOut * kOneOverSqrt12);
    clust->SetErrX(xErrOnePad);
    cntr++;
} else {
    Float_t xErr = (y < nInRows * phIn) ? (pwIn * Sqrt(rmsx)) : (pwOut * Sqrt(rmsx));
    clust->SetErrX(xErr);
}

if (rmsz <= 0.0) clust->SetErrZ(zDrift / nTimeBins * kOneOverSqrt12);
else clust->SetErrZ(Sqrt(rmsz) * zDrift / nTimeBins);

if (clust->GetSect() > nSect / 2) avgz *= -1;

clust->SetX(avgx);
clust->SetY(y);
clust->SetZ(avgz);

Float_t yErrOneRow = (y < nInRows * phIn) ? (phIn * kOneOverSqrt12) : (phOut * kOneOverSqrt12);
clust->SetErrY(yErrOneRow);
clust->SetADC(sumadc);

Float_t pos[3];
Float_t hitErr[3];

if (clust->Row() < nInRows) {
    pos[0] = pwIn * ((Float_t) clust->GetX() - (Float_t) fNumOfPadsInRow[clust->Row()] + 0.5); // x-
        coordinate of pad center
    pos[1] = phIn * ((Float_t) clust->Row() + 0.5); // y-coordinate of pad center
} else {
    pos[0] = pwOut * ((Float_t) clust->GetX() - (Float_t) fNumOfPadsInRow[clust->Row()] + 0.5); // x-
        coordinate of pad center
    pos[1] = fSectInHight + ((Float_t) (clust->Row() - nInRows) + 0.5) * phOut; // y-coordinate of
        pad center
}

pos[2] = ((Float_t) clust->GetZ() + 0.5) * (zDrift / nTimeBins);
```

```
            //adding errors
            hitErr[0] = clust->GetErrX(); //dx
            hitErr[1] = clust->GetErrY(); //dy
            hitErr[2] = clust->GetErrZ(); //dz

            TpcFoundHit* tmphit = new TpcFoundHit(pos, hitErr);
            tmphit->AttachCluster(clust);
            tmphit->SetTimeBkt(clust->GetZ());
            tmphit->SetPadCol(clust->GetX());
            tmphit->SetNumHits(1);
            tmphit->SetType(TpcFoundHit::kWMPeak);
            tmphit->SetQADC(clust->GetADC());
            hitList->push_back(tmphit);
        }
    }
```