

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Programovanie modulov pre softvérový
ráamec ESAF experimentu JEM-EUSO
zameraného na detekciu častíc
s ultravysokou energiou

Diplomová práca

2015

Bc. Michal Vrábel

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

**Programovanie modulov pre softvérový
rámeč ESAF experimentu JEM-EUSO
zameraného na detekciu častíc
s ultravysokou energiou**

Diplomová práca

Študijný program: Informatika
Študijný odbor: 9.2.1 Informatika
Školiace pracovisko: Katedra počítačov a informatiky (KPI)
Školiteľ: doc. Ing. Ján Genči, PhD.
Konzultant: RNDr. Pavol Bobík, PhD.
RNDr. Blahoslav Pastirčák, CSc.

Košice 2015

Bc. Michal Vrábel

Abstrakt v SJ

Softvérový rámec ESAF projektu JEM-EUSO spracováva simulačnú a rekonštrukčnú reťaz od primárnych častíc po rekonštrukciu udalosti spŕšky z údajov získaných detektorom. Cieľom práce je implementácia nových modulov ESAF-u rozširujúc jeho možnosti. Medzi implementovanými modulmi boli moduly rekonštrukcie spŕšky pomocou algoritmu používajúceho Houghovu transformáciu, vstupný modul rekonštrukcie pre prepis knižníc generovaného UV pozadia do štruktúr ESAF-u a rozšírenie možností simulácie UV pozadia na základe modelu UV pozadia, využívajúc interpolačný algoritmus.

Kľúčové slová

JEM-EUSO experiment, softvérový rámec ESAF, algoritmus rozpoznávania vzorov, Houghova transformácia, model UV pozadia

Abstrakt v AJ

ESAF framework of JEM-EUSO project handles simulation and reconstruction chain from primary particles to shower event reconstruction. The main objective of the thesis is the implementation of new ESAF modules by extending its capabilities. Implemented modules include pattern recognition modul utilizing Hough transform algorithm, reconstruction input module for transformation of generated UV backgrounds library into ESAF structures and extending of UV background simulation capabilities by using of UV background model utilizing interpolation algorithm.

Kľúčové slová v AJ

JEM-EUSO experiment, ESAF framework, pattern recognition algorithm, Hough transformation, UV background model

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **9.2.1 Informatika**

Študijný program: **Informatika**

Názov práce:

Programovanie modulov pre softvérový rámec ESAF experimentu JEM-EUSO zameraného na detekciu častíc s ultravysokou energiou
Programming of modules for ESAF framework of JEM-EUSO experiment aimed to detect ultra high energy particles

Študent: **Bc. Michal Vrábel**
Školiteľ: **doc. Ing. Ján Genči, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce: **RNDr. Pavol Bobik, PhD.**
Pracovisko konzultanta: **Ústav experimentálnej fyziky SAV, Košice**

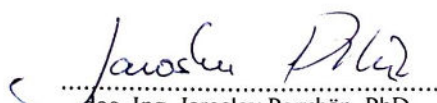
Pokyny na vypracovanie diplomovej práce:

1. Oboznámiť sa so základmi fyzikálnej problematiky detekcie častíc ultravysokých energií.
2. Preštudovať si princípy fungovania JEM-EUSO experimentu a ich implementáciu do softvérového rámca ESAF kolaborácie JEM-EUSO.
3. Oboznámenie sa so štruktúrou, fungovaním a zdrojovými kódmi softvérového rámca ESAF kolaborácie JEM-EUSO. Rozšírenie sady softvérových nástrojov na čítanie, základnú analýzu a vizualizáciu simulovaných spŕšok.
4. Implementácia kódu na rozpoznávanie spŕšok častíc ultravysokých energií (UHECR) Houghovou metódou ako samostatného modulu softvérového rámca ESAF a testovanie presnosti tejto metódy.
5. Vývoj a implementácia kódu na prepis knižníc generovaných UV pozadí do dátovej štruktúry softvérového rámca ESAF.
6. Implementácia modelu UV pozadia vyvíjaného na Ústave Experimentálnej Fyziky SAV do softvérového rámca ESAF.
7. Implementácia modelu distribúcie UV pozadia na citlivej ohniskovej ploche JEM-EUSO detektoru.

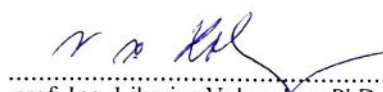
Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 30.04.2015

Dátum zadania diplomovej práce: 31.10.2014


.....
doc. Ing. Jaroslav Porubán, PhD.
vedúci garantujúceho pracoviska




.....
prof. Ing. Liberios Vokoros, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice 30. 4. 2015

.....

Vlastnoručný podpis

Podakovanie

Touto cestou sa chcem poďakovať všetkým, ktorí mi akýmkoľvek spôsobom pomohli pri vypracovaní tejto práce. Najväčšia vďaka patrí konzultantom diplomovej práce, najmä RNDr. Pavlovi Bobíkovi, PhD. za odborné vedenie, usmerňovanie a konzultácie, ktoré mi poskytoval pri vytváraní diplomovej práce. RNDr. Blahoslavi Pastirčákovi, CSc. ďakujem za úvod do problematiky a za prípravu technických prostriedkov na začiatku práce. Rovnako sa chcem poďakovať aj vedúcemu diplomovej práce doc. Ing. Jánovi Genčimu, PhD. za vypísanie tejto témy, poskytnuté rady, zoznámenie ma s RNDr. Bobíkom a umožnenie na téme pracovať.

Obsah

Úvod	1
1 Formulácia úlohy	4
2 Vysokoenergetické spršky kozmického žiarenia a experiment JEM-EUSO	6
2.1 Energetické spektrum	7
2.2 Pozorovateľné efekty	8
2.3 Detektor JEM-EUSO	9
3 Softvérový rámec ESAF	12
3.1 Softvérový rámec ROOT	14
3.2 Simulácia	16
3.2.1 Proces simulácie	16
3.2.2 Výstup simulácie	21
3.3 Rekonštrukcia	22
3.3.1 Priebeh rekonštrukcie	23
3.3.2 Modul rekonštrukcie	23
3.3.3 Trieda RecoEvent	24
3.3.4 Výstup rekonštrukcie	25
3.4 Vplyv rôznych konfigurácií na výsledok simulácie	26
4 Spúšťacie algoritmy a algoritmy rekonštrukcie vzoru v softvérovom rámci ESAF	32
4.1 Algoritmy spúšťacieho mechanizmu	32
4.1.1 Persistent Tracking Trigger	33
4.1.2 CCB-Linear Tracking Trigger	33
4.2 Algoritmy rekonštrukcie vzoru	35
4.2.1 Linear Tracking Trigger pre-clustering	35

4.2.2	Peak and WIndow SEearching	35
4.2.3	Metóda hľadania cesty	40
5	UV pozadie Zeme	41
5.1	Simulácia UV pozadia	41
5.2	Hľadanie vzorov v simulácií UV pozadia	42
6	Implementácia algoritmu Houghovej transformácie	45
6.1	Základný princíp algoritmu	45
6.2	Vstupné a výstupné údaje modulu rekonštrukcie vzoru	48
6.3	Prepis programov do jazyka C++	50
6.4	„Nultá” verzia	51
6.4.1	Algoritmus	52
6.4.2	Vylepšenie úspešnosti rekonštrukcie	53
6.5	Nové algoritmy hľadania čiar použitím Houghovej transformácie	54
6.5.1	Funkcia HoughLine	55
6.5.2	Funkcia HoughLine3D	56
6.5.3	Funkcia HoughPlane	57
6.6	Prvá verzia	57
6.7	Druhá verzia	59
6.8	Výsledky	69
6.8.1	„Nultá” verzia	69
6.8.2	Prvá verzia	71
6.8.3	Druhá verzia	72
6.8.4	Porovnanie implementácií	75
7	Prepis knižníc generovaných UV pozadí do dátovej štruktúry soft- vérového rámca ESAF	79
7.1	Prevod súradníc na identifikátor pixela	80
7.1.1	Výpočet identifikátora z pozície	81

7.1.2	Priradenie identifikátora odvodením z pozície pixela použitím mapy ohniskovej plochy	84
7.2	Informácie potrebné pre rekonštrukciu	85
7.3	Implementácia	85
7.3.1	Výpočet identifikátora z pozície	86
7.3.2	Chýbajúce údaje pre rekonštrukciu	87
7.3.3	Trieda RecoEventBuilder	88
7.3.4	Čítanie údajov z vstupného súboru	88
7.4	Výsledky	89
8	Rozšírenie možností simulácie UV pozadia	95
8.1	Model distribúcie UV pozadia na citlivej ohniskovej ploche detektora	96
8.1.1	Súbor distribúcie UV pozadia na ohniskovej ploche	96
8.1.2	Metódy interpolácie	97
8.1.3	Implementácia	98
8.1.4	Výsledky	100
8.2	Model UV pozadia	102
8.2.1	Špecifikácia modelu	103
8.2.2	Generovanie testovacích dát	104
8.2.3	Interpolácia hodnoty pozadia z modelu	105
8.2.4	Implementácia	108
8.2.5	Výsledky	110
9	Záver	112
	Zoznam použitej literatúry	114
	Zoznam príloh	117

Zoznam obrázkov

0-1 Princíp experimentu JEM-EUSO.	2
2-1 Schéma hadrónovej spršky(Blaschke, 2009).	6
2-2 Závislosť toku primárnych častíc na energii(Blaschke, 2009).	7
2-3 Schematický nákres detektora JEM-EUSO(Ebisuzaki T. et al., 2010).	10
2-4 Stavebné bloky elektroniky ohniskovej plochy detektora(Ebisuzaki T. et al., 2010).	11
3-1 Základné prúdy dát v softvérovom rámci ESAF - princíp práce s nástrojmi.	12
3-2 Diagram dátových tokov v procese simulácie.	20
3-3 Porovnanie spršok s rozdielnym zenitovým uhlom θ	27
3-4 Porovnanie spršok s rozdielnou pozíciou maxima na ohniskovej ploche detektora. Zenitový uhol simulovaných spršok θ je 60°	28
3-5 Spršky simulované s rôznou pozíciou dopadu.	28
3-6 Porovnanie spršok s rozdielnou výškou orbity.	29
3-7 Porovnanie nastavení pozície ohniskovej plochy mimo ohniska optiky (Parameter <code>PolarFocalPlane.fPos.Z</code>).	30
3-8 Porovnanie spršok s rozdielnym množstvom fotoelektrónov skutočného signálu spršky.	30
3-9 Porovnanie rozdielnych intenzít pozadia.	31
4-1 Percentuálny podiel počtu zrekonštruovaných spršok z celkového počtu použiteľných simulovaných spršok.	38
4-2 Štatistika γ_{68} rekonštrukcie zenitového uhla pri použití algoritmu PWISE pre uhly 30 až 75 stupňov, zvyšujúce sa po 5 stupňov. Pri výpočte boli použité udalosti z celej ohniskovej plochy.	39
4-3 Závislosť pozície maxima spršky a chyby γ pre algoritmus PWISE.	39
5-1 Závislosť počtu vzorov a dĺžok nájdených vzorov pre rôzne časové obdobia (Staroň, 2013).	44

6-1	Základný princíp popisu priamky v Houghovej transformácii.	46
6-2	Houghova transformácia spŕšky.	47
6-3	Diagram prúdu dát - Prvá verzia algoritmu.	57
6-4	Diagram prúdu dát - Základné bloky programu rekonštrukcie vzoru použitím Houghovej transformácie (druhá verzia).	60
6-5	Diagram prúdu dát - Druhý blok programu rekonštrukcie vzoru pou- žitím Houghovej transformácie (druhá verzia).	61
6-6	Diagram prúdu dát - Bloky <i>CorePixels</i> a <i>CleanCorePixels</i> programu rekonštrukcie vzoru použitím Houghovej transformácie (druhá verzia).	64
6-7	Vývojový diagram funkcie <i>Filter</i>	67
6-8	Porovnanie štatistík γ_{68} rôznych konfigurácií „nulteje” verzie algoritmu.	70
6-9	Závislosť pozície maxima spŕšky a chyby γ pre „nultú” verziu algoritmu.	70
6-10	Porovnanie percentuálneho podielu rekonštruovaných spŕšok rôznych konfigurácií „nulteje” verzie algoritmu.	71
6-11	Závislosť pozície maxima spŕšky a chyby γ prvej verzie algoritmu	72
6-12	Porovnanie štatistík γ_{68} prvej a druhej verzie algoritmu s algoritmom PWISE.	73
6-13	Porovnanie počtu zrekonštruovaných udalostí prvej a druhej verzie algoritmu s algoritmom PWISE.	74
6-14	Porovnanie štatistík γ_{68} prvej a druhej verzie algoritmu s algoritmom PWISE, pri použití udalostí zrekonštruovaných algoritmom PWISE.	74
6-15	Závislosť pozície maxima spŕšky a chyby γ druhej verzie algoritmu (mód <i>HoughPlane</i>).	75
6-16	Využitie pamäte virtuálnym strojom JVM pri spracovaní 10 udalostí v implementácii druhej verzie algoritmu v programovacom jazyku Java.	76
6-17	Využitie pamäte pri spracovaní 10 udalostí v implementácii druhej verzie algoritmu v programovacom jazyku C++.	77
7-1	Usporiadanie prvkov na ohniskove ploche.	82
7-2	Výsledky rekonštrukcie „falošných udalostí” pri algoritme PWISE.	92

7-3	Výsledky rekonštrukcie „falošných udalostí” pri prvej verzii algoritmu Houghovej transformácie.	93
7-4	Výsledky rekonštrukcie „falošných udalostí” pri druhej verzii algoritmu Houghovej transformácie (<i>HoughLine3D</i>).	94
8-1	Distribúcia intenzity UV pozadia zo súboru.	97
8-2	Vizualizácia interpolácie údajov zo súboru distribúcie UV pozadia na ohniskovej ploche.	100
8-3	Distribúcia intenzity UV pozadia zo súboru - porovnanie interpolácie metódy <i>najbližší sused</i> s bodmi zo súboru.	101
8-4	Vývojový diagram v zjednodušenej podobe ilustrujúci funkciu <i>FindInterpolate</i>	107
8-5	Diagram tried navrhnutých pre reprezentáciu UV modelu pozadia. . .	109
8-6	Intenzita pozadia generovaného testovacieho modelu závislá od času. .	110
8-7	Intenzita pozadia generovaného testovacieho modelu závislá od geografickej pozície.	110
8-8	Interpolácia intenzity na generovaných údajoch.	111

Zoznam tabuliek

6-1 Porovnanie priemerných časov spracovania druhej verzie algoritmu (v milisekundách)	78
6-2 Porovnanie priemerných časov spracovania prvej verzie algoritmu (v milisekundách)	78
7-1 Porovnanie výsledkov rekonštrukcie pozadia pri použití vstupného modulu <code>TxtFeeInputModule</code>	94
8-1 Časy spracovania interpolácie pre rovnakú oblasť FS pri použití rôznych veľkostí indexovaného bloku.	101

Zoznam symbolov a skratiek

CCB Cluster Control Board

EAS Extensive Atmospheric Shower - Extenzívna atmosferická sprška

EC Elementary Cell

ESA European Space Agency

EUSO Extreme Universe Space Observatory

FPGA Field-programmable gate array

FS Focal Surface

GTU Gate Time Unit

GZK Greisen Zatsepin Kuzmin

ISS International Space Station

JEM Japanese Experiment Module

JEM-EUSO Japanese Experiment Module - Extreme Universe Space Observatory

JVM Java Virtual Machine

LIDAR Light Detection and Ranging

MAPMT Multi-Anode Photomultiplier tube

PDM Photo Detector Module

PMT Photomultiplier tube

UHECR Ultra High Energy Cosmic Ray

UV Ultraviolet

Slovník termínov

Čerenkovovo žiarenie je elektromagnetické žiarenie vznikajúce pri prechode nabitej častice prostredím rýchlosťou väčšou ako je rýchlosť svetla v tomto prostredí (Blaschke, 2009).

Extensívna atmosferická spĺška Spĺška sekundárnych častíc kozmického žiarenia dosahujúca povrch Zeme.

Fotoelektrický efekt je fyzikálny jav, pri ktorom sú elektróny vyrážané z atómov v dôsledku absorpcie elektromagnetického žiarenia (napr. röntgenového žiarenia, viditeľného svetla) látkou (kovy, polovodiče). Rozlišujeme vnútorný a vonkajší. Vnútorný - elektróny sa voľne pohybujú v látke a zvyšujú jej vodivosť. Vonkajší - elektróny sú vyžarované z látky a tieto emitované elektróny sú potom označované ako **fotoelektróny**. Ich uvoľňovanie sa označuje ako fotoelektrická emisia. (Hüfner, 2003)

Fluorescencia dusíka vzniká excitáciou atómov dusíka pozdĺž dráhy letu spĺšky sekundárnych častíc atmosférou. Pomalá deexcitácia týchto atómov vedie k slabej žiare viditeľného svetla (Blaschke, 2009).

Gate Time Unit Doba približne $2,5 \mu\text{s}$. Po uplynutí tohto intervalu je počítadlo fotoelektrónov resetované.

Greisen Zatsepin Kuzmin limit je dôsledkom interakcií častíc kozmického žiarenia s energiami nad $5 \times 10^{19} \text{eV}$ s mikrovlnným žiarením kozmického mikrovlnného pozadia. Pri zrážkach častice strácajú energiu až pokiaľ neklesne pod GZK limit.

Kaón nazývaný aj *mezón K* je pomenovanie pre elementárne častice zo skupiny štyroch mezónov, ktoré môžu niesť kvantové číslo nazvané "podivnosť" (angl. strangeness). V kvarkovom modeli obsahujú jeden podivný kvark alebo antik-

vark.

LIDAR je technika získavania informácií o objekte bez fyzického kontaktu s ním, ktorá meria vzdialenosť objektu jeho osvetľovaním laserovým lúčom a analýzou odrazeného svetla.

Mezón je hadrónová subatomárna častica zložená z kvarku a antikvarku zviazaných spolu pomocou silnej interakcie.

Pión nazývaný aj *mezón* π je spoločné pomenovanie troch subatomárnych častíc, π^0 , π^+ a π^- . Pióny sú najľahšie mezóny a zohrávajú dôležitú úlohu v objasňovaní nízkoenergetických vlastností silnej jadrovej sily.

Poissonovo rozdelenie pravdepodobnosti sa používa na vyjadrenie rozdelenia počtu výskytov nejakého javu v určitom časovom intervale. Poissonovým rozdelením sa riadi aj počet častíc v jednotke plochy alebo objemu a pod. (Daňo, Ostertagová, 2010)

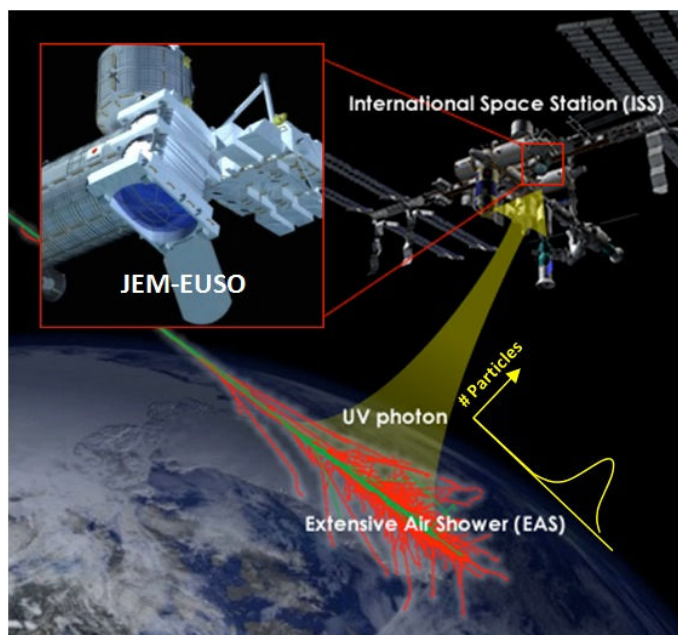
Softvérový rámec je softvér ponúkajúci generickú funkcionálnosť s možnosťou selektívnej zmeny kódom používateľa, poskytujúci aplikačne špecifický softvér.

Setter a Getter sú metódy používané pre prístup k hodnote premennej. Setter je „mutátor“ - nastavuje hodnotu. Getter je „prístupiteľ“ - číta hodnotu.

Úvod

V rozmedzí rokov 1911 až 1913 rakúsky vedec Victor Hess vykonal spolu 10 výstupov - 5 nočných, 5 denných - vodíkom plneným balónom. Tie ukázali, že zdrojmi naoko všade prítomnej radiácie nie je zem, ani slnko, a že so stúpajúcou výškou nad zemou jej intenzita vzrastá. Tento objav mu spolu s vedcom C.D. Andersonom v roku 1936 priniesol Nobelovú cenu. Neskôr, experimenty francúzskeho fyzika Pierra Augerera na vysokohorských observatóriách priniesli objav atmosferických spršok, ktoré vznikajú pri zrážkach primárnych častíc s molekulami plynov atmosféry. Zrážky spôsobujú vznik sekundárnych častíc majúcich dostatočnú energiu pre spôsobenie rozpadu ďalších častíc - vzniku ďalších sekundárnych častíc. Pri dostatočne vysokých energiách sa vzniknutá sprška lavínovite šíri až dokiaľ nedopadne na zem. Augerové merania ukázali aj to, že spršky majú rozdielne energie a so zvyšujúcou energiou sa znižuje ich počet. Od objavenia kozmického žiarenia Victorom Hessom znalosť problematiky zásadne pokročila. Okrem vedomosti jeho existencie dnes vieme, že predstavuje vysokoenergetické častice dopadajúce do atmosféry a poznáme aj jeho zloženie a spektrum jeho energií. Predpokladané zdroje kozmického žiarenia sú supernovy, aktívne jadrá galaxií, kvazáry a záblesky gama žiarenia. Cieľom projektu *Japanese Experiment Module - Extreme Universe Space Observatory*, skrátene JEM-EUSO, je realizácia astrofyzikálneho výskumu zameraného na pozorovanie častíc s extrémne vysokými energiami. Vďaka meraniu s vyššou štatistikou je cieľom upresniť charakteristiky energetického spektra kozmického žiarenia v oblasti ultravysokých energií a najmä identifikovať zdroje UHECR žiarenia.

Detekcia je vykonávaná použitím 2,5 metrového teleskopu s 30 stupňovým zorným polom do oboch strán od nadir línie, t. j. spolu 60. Teleskop má byť umiestnený na Medzinárodnej vesmírnej stanici, na vonkajšej plošine (Exposed facility) japonského modulu JEM, nazývaného *Kibó*. Obrázok 0–1 ilustruje umiestnenie detektora a princíp experimentu (Ebisuzaki T. et al., 2010).



Obr. 0–1: Princíp experimentu JEM-EUSO.

JEM-EUSO kolaborácia pracujúca na tomto projekte je aktuálne tvorená 88 výskumnými inštitúciami a univerzitami zo 16 krajín. Slovensko v JEM-EUSO kolaborácii zastupuje Ústav experimentálnej fyziky, Slovenskej akadémie vied. Slovensko je členom simulačného tímu kolaborácie, čo znamená, že ich úlohy sú zamerané na prieskum vlastností a schopností detektora, fyziky atmosférických spířšok a hornej vrstvy atmosféry. Projektu JEM-EUSO predchádzal ESA projekt Extreme Universe Space Observatory (EUSO), ktorý bol z finančných a programových dôvodov pred desiatimi rokmi zastavený.

Euso Simulation and Analysis Framework je softvérový rámec pôvodne vyvinutý pre projekt EUSO. Účelom softvérového rámca je poskytnúť projektu EUSO nástroj pre celý proces simulácie a analýzy dát. Skladá sa z dvoch základných programov *Simu* (pre simuláciu) a *Reco* (pre rekonštrukciu), okrem nich obsahuje aj viacero makier pre softvérový rámec ROOT, na ktorom je postavený, a tried určených najmä pre vizualizáciu dát.

Hlavným cieľom tejto práce je rozšíriť softvérový rámec ESAF o funkcionality a nástroje, ktoré sú potrebné pre spracovanie a analýzu dát z experimentu JEM-EUSO pracoviskom Ústavu Experimentálnej Fyziky Slovenskej akadémie vied v Košiciach. Najdôležitejším prínosom pre experiment by bola implementácia nového algoritmu rozpoznávania vzorov, ktorý by bol v nejakom ohľade lepší ako existujúce algoritmy používané v predchádzajúcich rokoch JEM-EUSO kolaboráciou. Vstup programu rekonštrukcie je v aktuálnej podobe možný len jedným spôsobom, čo nevyhovuje pre potreby testovania jej algoritmov pri použití údajov z knižníc generovaných UV pozadí. Preto cieľom práce bolo nájsť metódu integrácie týchto knižníc do ESAF-u. Posledným nedostatkom tohto softvérového rámca, riešeným v tejto práci, sú obmedzené možnosti použitia modelov UV pozadia pri simulácii. Cieľom je navrhnúť štruktúru modelu UV pozadia závislého na čase a geografickej pozícii. Doplnkovo, keďže ESAF túto možnosť nemá, je cieľ pridať schopnosť použitia externého modelu UV pozadia na ohniskovej ploche. Pri realizácii týchto úloh bude tiež potrebné vytvoriť viacero chýbajúcich nástrojov pre prácu s výstupom simulácie a rekonštrukcie.

V práci bolo potrebné vytvoriť viacero programov v jazyku C++, ktoré boli najmä rozšírenia softvérového rámca ESAF alebo nástroje pre prácu s výstupnými súborami programov ESAF-u. Skoro všetky programy používali v nejakej forme aj softvérový rámec ROOT. Práca nadväzuje na prácu Bc. Jozefa Vasilka, s ktorým som spolupracoval na vývoji algoritmu rekonštrukcie vzoru. Náročnosť práce vyplývala z potreby spájania vedomostí z viacerých oblastí, nutnosťou zoznamovania sa s ESAF-om a riešenie jeho problémov bez ucelenej, často aj neexistujúcej dokumentácie.

Práca môže tiež slúžiť aj ako zdroj informácií, keďže sa najmä z programátorského hľadiska pozerá na softvérový rámec ESAF, a takto ponúka informácie, ktoré nie sú často dostatočne alebo na jednom mieste opísané. Pre autora práce je to príležitosť sa oboznámiť s princípmi simulácie fyzikálnych udalostí a nahliadnuť do problematiky kozmickej fyziky.

1 Formulácia úlohy

Prvými úlohami tejto práce je oboznámiť sa so základmi fyzikálnej problematiky detekcie častíc ultravysokých energií a preštudovať princípy fungovania experimentu JEM-EUSO. Pri štúdiu problematiky sa bude vychádzať z odbornej literatúry a taktiež z komunikácie s odborníkom v oblasti. Podobne je potrebné preštudovať softvérový rámec ESAF, pretože pochopenie jeho funkcionality je nevyhnutné pri návrhu rozšírení. Zdrojom bude dostupná dokumentácia a jeho samotné zdrojové kódy.

Práca vyžaduje implementáciu viacerých pomocných programov pre rozšírenie možností analýzy v softvérovom rámci. Tieto programy budú vytvorené najmä ako nástroje pre riešenie podúloh vyplývajúcich z hlavných cieľov práce.

Pre kolaboráciu JEM-EUSO najdôležitejšou úlohou tejto práce je implementácia kódu na rozpoznávanie spŕšok častíc ultravysokých energií (UHECR) Houghovou metódou ako samostatného modulu softvérového rámca ESAF. Cieľom je modul implementovať a analyzovať jeho výsledky. Pre analýzu budú použité štatistické metódy štandardne používané kolaboráciou, aby výsledky mohli byť porovnané s už existujúcimi riešeniami.

Cieľom úlohy prepisu knižníc UV pozadia do dátovej štruktúry softvérového rámca ESAF, je umožniť spracovanie generovaných dát UV pozadia v rekonštrukcii podobne ako sú aktuálne spracovávané simulované dáta spŕšky. Úloha bude riešená implementáciou vstupného modulu rekonštrukcie.

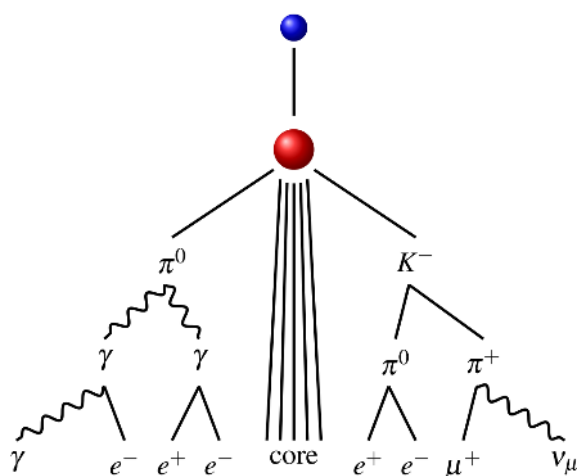
Úloha implementácie modelu UV pozadia vyvíjaného na Ústave Experimentálnej Fyziky SAV do softvérového rámca ESAF vyžaduje rozšírenie simulácie o nové možnosti a spracovávanie vstupného súboru popisujúceho tento model. Pri tomto spracovaní sa vyžaduje použitie interpolačných techník pre dopočítanie chýbajúcich dát. Táto úloha rieši pohľad detektora na Zem - závislosť pozície a času, pre ktoré sa vykonáva simulácia.

Podobný cieľ má aj úloha implementácie modelu distribúcie UV pozadia na citlivej ohniskovej ploche JEM-EUSO detektora, ale rieši sa iný aspekt problému, keďže tu sa pozerá na ohniskovú plochu detektora. Pre ňu sa simuluje rozdielna intenzita UV pozadia v závislosti od pozície na ohniskovej ploche.

2 Vysokoenergetické spršky kozmického žiarenia a experiment JEM-EUSO

Vysokoenergetické spršky kozmického žiarenia (skratka UHECR) sú tvorené primárnymi časticami - ide prevažne o protóny, α častice, ale aj jadrá ťažších prvkov, elektróny, pozitrony, fotóny a neutrína. Po dopade primárnej častice do atmosféry nastávajú zrážky s jej atómami. Tieto zrážky spôsobujú produkciu sekundárnych častíc, ich množstvo je závislé na energii primárnej častice. Produkcia sekundárnych častíc má pri dostatočne vysokej energii primárnej častice lavínový charakter. Ak sprška dosiahne až povrch Zeme, označujeme ju ako extenzívna atmosferická sprška (skratka EAS).

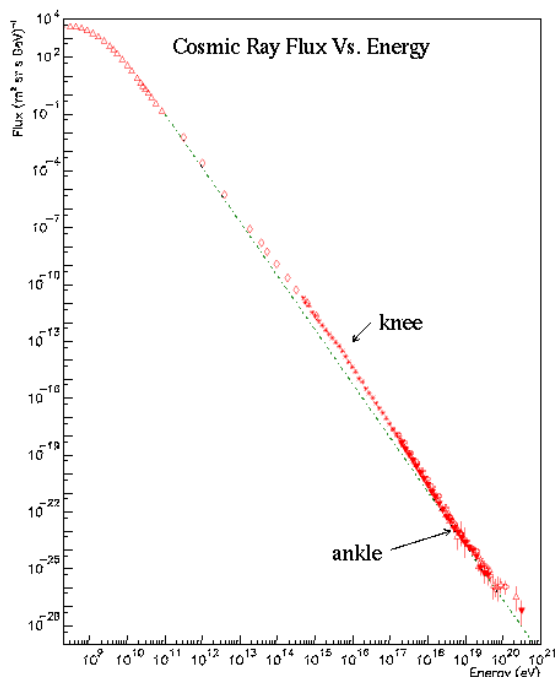
Typ primárnej častice určuje charakter lavínovej produkcie sekundárnych častíc. Napríklad ak sa jedná o neutrón, protón alebo jadro prvku, označujeme ich ako hadronové spršky (Obrázok 2–1). Najčastejšie produkty zrážok dvoch jadier sú kaóny (K^-) a pióny(π^0). Kaóny sa najčastejšie rozpadajú na pióny (21,1%) alebo mióny(μ^\pm , 63,4%). Pióny sa rozpadajú hlavne na mióny (99,9%), ktoré sa rozpadajú na elektróny (e^\pm) a neutrína (ν_μ) (Blaschke, 2009).



Obr. 2–1: Schéma hadronovej spršky(Blaschke, 2009).

2.1 Energetické spektrum

Primárne častice galaktického pôvodu sa vyskytujú v širokom spektre energií približne od $10^{10}eV$ do $10^{20}eV$. Obrázok 2–2 znázorňuje tok týchto častíc ($(m^2srsGeV)^{-1}$) v závislosti od energie primárnej častice. Takmer celé spektrum na tomto obrázku je možné popísať mocninovou závislosťou. Odchýlka začiatku spektra od tejto závislosti v blízkosti a pod energiami $10^{10}eV$ je spôsobená tým, že intenzity častíc s energiami pod uvedenou hranicou sú závislé od slnečnej aktivity - antikorelácia, nabité častice kozmického žiarenia sú deflektované medziplanetárnym magnetickým poľom. Nad touto energiou už závislosť toku (anglicky *flux*) na lokálne podmienky mizne a možno ho opísať mocninovým zákonom uvedenom v vzťahu 2.1, kde γ je spektrálny index.



Obr. 2–2: Závislosť toku primárnych častíc na energii (Blaschke, 2009).

Pre energie menej ako $10^{15}eV$ platí $\gamma \approx 2,7$, do energie $10^{18}eV$ je $\gamma \approx 3,0$, za hodnotou $10^{18}eV$ klesá tok pomalšie s $\gamma \approx 2,1$. Zlom meniaci klesanie toku medzi

$\gamma \approx 2,7$ a $\gamma \approx 3,0$ sa označuje koleno (anglicky *knee*) a zlom medzi $\gamma \approx 3,0$ a $\gamma \approx 2,1$ členok (anglicky *anckle*).

$$\frac{\delta N}{\delta E} \sim E^{-\gamma} \quad (2.1)$$

Aj napriek dvom spomenutým zlomom je závislosť toku primárnych častíc na energii bez výrazných zmien. Z fyzikálneho hľadiska je najzaujímavejší koniec spektra, teda nad energiou $10^{20} eV$. Problémom výskumu tejto časti spektra je ale nízka početnosť častíc (na jeden kilometer štvorcový dopadne jedna častica s energiou $10^{20} eV$ len približne raz za tisíc rokov), preto je pre jej výskum potrebné použiť „extrémne“ techniky. Vhodným príkladom je práve projekt JEM-EUSO. Mechanizmus akcelerácie častíc s touto energiou nie je známy. Navyše už takmer 50 rokov poznáme jav objavený K. Greinsenom, G.T. Zatsepinom a V.A. Kuzminom nazývaný GZK limit. Primárne častice s energiou nad $5 \times 10^{19} eV$ sú „brzdené“ zrážkami s mikrovlnným žiarením kozmického mikrovlnného pozadia. Z faktoru „brzdenia“ pre danú časticu je možné určiť strednú vzdialenosť nazývanú GZK limit, ktorú častica prejde bez zásadnej zmeny rýchlosti (straty energie). Ak sa nachádzame vo vzdialenosti väčšej ako GZK limit, nemali by sme UHECR častice pozorovať. Opísaný jav ešte zväčšuje „záhadu“ pôvodu týchto primárnych častíc, keďže vo vzdialenostiach od Zeme spĺňajúcich GZK limit nie sú známe žiadne možné astrofyzikálne zdroje UHECR žiarenia, napriek tomu toto žiarenie na Zemi registrujeme.

2.2 Pozorovateľné efekty

Propagáciu sekundárnych častíc atmosférou možno nepriamo sledovať využitím Čerenkovovho javu a javu fluorescencie dusíka, ktoré produkujú fotóny najmä v oblasti vlnových dĺžok 300 až 400nm pri dusíku a 250 až 600nm pri Čerenkovovom jave (Grieder, 2010).

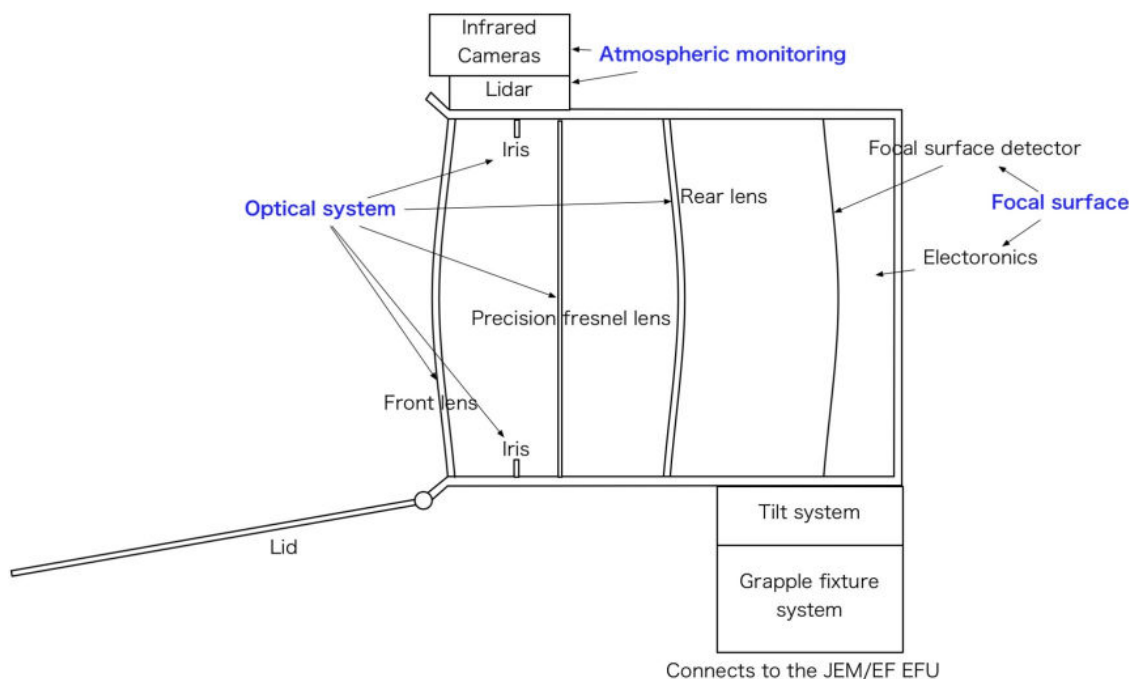
Čerenkovov jav vzniká, ak sa nabitá častica šíri rýchlejšie ako je rýchlosť svetla v médiu. Pohybujúci sa náboj vytvára oscilácie atómov, ktoré produkujú elektromagnetické vlnenie. V prípade, že rýchlosť častice je väčšia ako rýchlosť svetla v danom médiu, vlny spolu konštruktívne interferujú a vytvárajú rázovú vlnu.

Fluorescencia dusíka vzniká, keď častice sekundárnej spříšky spôsobujú excitáciu atómov dusíka v atmosfére, ich pomalá deexcitácia vedie k slabej žiare viditeľného svetla. Tento jav je pozorovaný aj pri polárnej žiare. Počet excitovaných častíc je priamo úmerný množstvu častíc spříšky. Vďaka tomuto efektu možno zistiť miesto vzniku, vývoj, tvar spříšky a pri ťažších primárnych časticách navyše aj ich hmotnosť.

2.3 Detektor JEM-EUSO

V závislosti od módu, v ktorom by bol detektor inštalovaný, je sledovaná plocha $1,3 \times 10^5 km^2$ pri pohľade detektora kolmo dole na Zem, v takzvanom nadir móde, alebo až $1,0 \times 10^6 km^2$ pri sklonenom móde pozorovania detektora. Ide o plochy rádovo väčšie, ako sú plochy existujúcich pozemných detektorov. Dostatočná výška, orientácia pohľadu zhora-nadol a orbita, na ktorej sa ISS nachádza, prinášajú ďalšie výhody.

Teleskop sa skladá z optiky tvorenej troma paralelnými fresnelovými šošovkami, predná a zadná zaostrujú svetlo, stredná vykonáva korekciu vinetácie a chromatických odchýlok. Ohnisková plocha (anglicky *focal surface*, skratka FS) je pokrytá zaoblenou mriežkou 4932 multi-anódových fotomultiplikačných trubíc (MAPMT). Sledovanie atmosféry je realizované pomocou infračervenej kamery a LIDAR-u za účelom upresnenia informácií pre rekonštrukciu spříšky. Obrázok 2–3 schematicky ilustruje časti detektora.

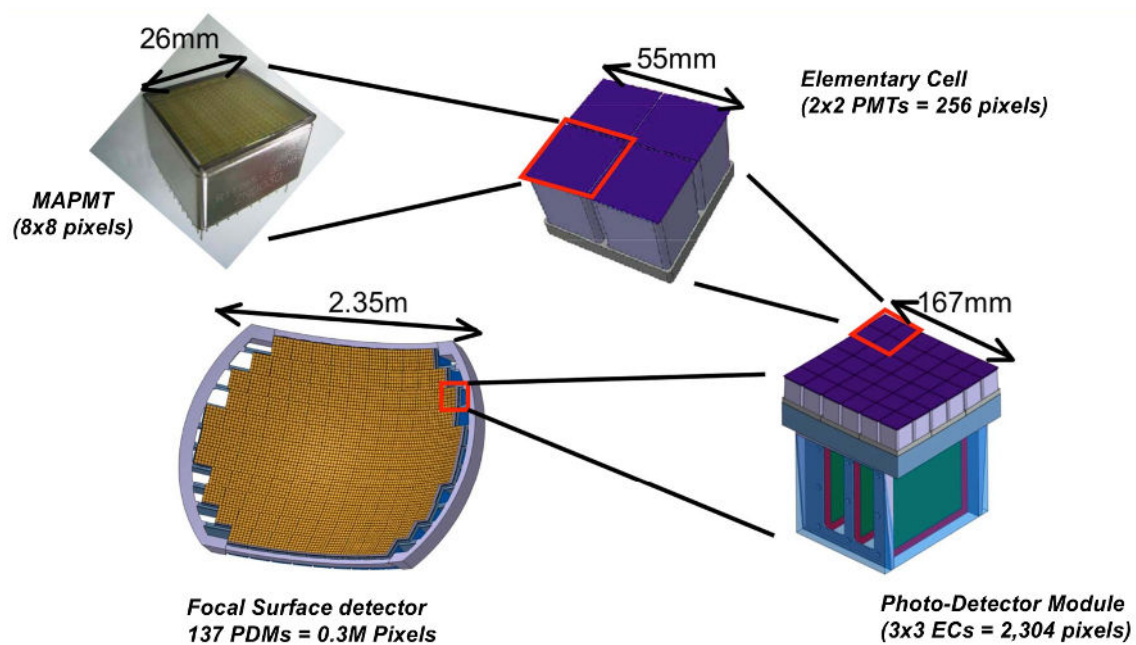


Obr. 2 – 3: Schematický náčrt detektora JEM-EUSO (Ebisuzaki T. et al., 2010).

Každé MAPMT má na sebe nalepený absorpčný filter (Schott BG3) pre odfiltrovanie fotónov mimo rozsahu 300 až 400 nm. MAPMT má rozlíšenie 8x8 štvorcových pixelov. Typická veľkosť je 26x26 mm s pixelmi veľkosti 3x3 mm, čo umožňuje uhlové rozlíšenie 0,07 stupňa (uvedené hodnoty sú približné). Kvantová efektivita (pomer počtu produkovaných elektrónov k počtu zaznamenaných fotónov) je závislá na vlnovej dĺžke v rozmedzí od 30% do 42% pri vlnových dĺžkach 290 až 450 nm. MAPMT sú štrukturované v bloku 2x2 nazývanom elementárnej bunky (EC). Spolu deväť elementárnych buniek usporiadaných do skupiny 3x3 tvoria modul fotodetektorov (PDM) - spolu 2304 pixelov. Na úrovni EC sú výstupy PMT prečítané a je vykonaná digitalizácia prúdových pulzov. Integrácia počtov je vykonaná pre časovú jednotku GTU - interval 2,5 μ s. Obrázok 2–4 ilustruje spomenuté stavebné bloky. Na úrovni PDM je pomocou FPGA implementovaný spúšťač prvej úrovne ¹. Signály z ôsmich PDM sú smerované do riadiacej dosky klastra (CCB), kde je vykonávaný spúšťač druhej úrovne. Finálny logický stupeň je palubný údajový spravovací systém (SCU),

¹Princípy fungovania spúšťačov sú opísané v časti 4.1.

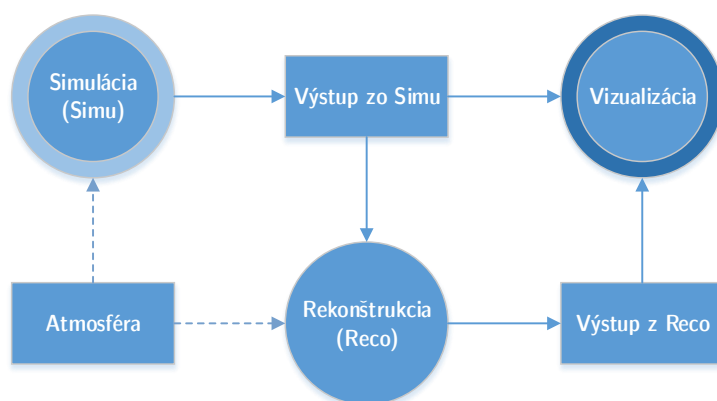
riadiaci celý inštrument (Ebisuzaki T. et al., 2010).



Obr. 2 – 4: Stavebné bloky elektroniky ohniskovej plochy detektora (Ebisuzaki T. et al., 2010).

3 Softvérový rámec ESAF

Softvérový rámec ESAF spracováva simulačnú a rekonštrukčnú reťaz od primárnych častíc v atmosfére po rekonštrukciu udalosti. Obrázok 3–1 opisuje základné prúdy dát a procesy - simuláciu, rekonštrukciu, vizualizáciu (Berat et al., 2010).



Obr. 3–1: Základné prúdy dát v softvérovom rámci ESAF - princíp práce s nástrojmi.

Zdrojové kódy systému ESAF zahŕňajú:

- simuláciu extenzívnej atmosferickej spŕšky použitím interných algoritmov, ale aj rozhraniami pre existujúce široko používané kódy,
- kompletný popis atmosféry.
- simuláciu svetla fluorescenciou aj čerenkovým efektom,
- simuláciu propagácie fotónu z bodu vzniku do detektora,
- simuláciu optiky detektora,
- simuláciu fotomultiplikátorov,
- simuláciu schémy spúšťačov (anglicky *trigger*),

- simuláciu pozadia,
- rekonštrukciu vzorov a identifikácia signálov spršky,
- rekonštrukciu smeru, energie a slant hĺbky maxima spršky².

Kód programov je napísaný hlavne v jazyku C++ s niekoľkými externými knižnicami vo Fortrane. Medzi samostatnými programami **Simu** a **Reco** je zdieľaná infraštruktúra a spoločné knižnice. Pre vytváranie simulovaných spršok častíc slúži nástroj **Simu**. Vstupné údaje tohto programu sú parametre spršky a množstvo vygenerovaných spršok. Program **Reco** vykonáva rekonštrukciu spršok (aktuálne len simulovaných) a jeho výstupom sú rekonštruované parametre spršky. Z dôvodu množstva parametrov ovplyvňujúcich vlastnosti vygenerovanej spršky, čo platí podobne aj pre metódy rekonštrukcie, sú konfigurácie oboch programov umiestnené v textových súboroch s príponou *.cfg*. Programy sú jednovláknové a v praxi ich spôsob použitia je spúšťanie viac procesov, ktoré sa líšia hodnotou analyzovaného parametra. Inštalácia a používanie programov softvérového rámca sú opísané v používateľskej príručke. Systémová príručka zas opisuje preklad zdrojových kódov ESAF-u.

Možno teoretizovať, že nedostatok príznaku na samotný vývoj softvéru, môže byť jedným činiteľom pri vzniku chýb a neoptimálnych riešení v systéme ESAF. Problémom je, že zdrojové kódy boli rozširované a prispôbované niekedy nepraktickým spôsobom bez ohľadu na konzistenciu. Preto niektoré rozšírenia vyžadujú zmeny v základných triedach.

Opisy simulácie a rekonštrukcie v nasledujúcich sekciách slúžia len pre základný popis štruktúry programu. Viac informácií je možné nájsť napríklad v práci F. Fenu (Fenu, 2013), kde autor aj detailne analyzoval výsledky jednotlivých modulov simulácie a rekonštrukcie spršok. Rozšírený opis najmä procesu simulácie sa nachádza aj v systémovej príručke. Preštudovanie tohto opisu je užitočné pre pochopenie kon-

²Vzdialenosť maxima spršky od vrchu atmosféry vyjadrená v g/cm^2 prejdeho materiálu.

ceptov opisovaných v tejto práci.

3.1 Softvérový rámec ROOT

Softvérový rámec ESAF je založený na softvérovom rámci ROOT vyvinutom organizáciou CERN pre analýzu veľkých objemov dát (Mato et al., 2014). Využívajú sa triedy umožňujúce prácu s lineárnou algebrou, náhodnými číslami, vizualizáciu dát, prácu so súborovým systémom, stromovou štruktúrou a základné prvky používateľského rozhrania.

ROOT je objektovo-orientovaný softvérový rámec vyvíjaný komunitou vývojárov - fyzikov, ktorí implementujú funkcionalitu na základe požiadaviek používateľov - fyzikov. ROOT môže byť používaný pomocou interaktívnej konzoly *CINT* (vo verzii 6 je nová interaktívna konzola *Cling*) alebo pripojením knižníc softvérového rámca a sprístupnením tried vo vlastnom programe. Štruktúra je založená na knižniciach organizovaných s cieľom minimalizácie závislostí, so základnou knižnicou *libCore.so*, funkcionalita ako napríklad grafy alebo histogramy je zahrnutá v samostatných knižniciach.

Interaktívna konzola je interpretátor zjednodušenej verzie jazyka C++ s ďalšími príkazmi pre prácu so samostatnou konzolou (ukončenie, načítanie makra, spustenie makra, atď.)³. Makrá, ktorých používanie je najtypickejšia činnosť vykonávaná pomocou tejto konzoly, môžu byť spúšťané v interpretovanom alebo skompilovanom móde, ktorý ponúka lepší výkon. Aby boli kompilované dynamické knižnice použiteľné v interaktívnej konzole, je potrebné vytvoriť slovník. Pre účel vytvorenia slovníka zo samostatných tried slúži program *rootcint*, v interaktívnej konzole je

³Pri programovaní knižníc využiteľných v interaktívnej konzole CINT používanej vo verziách nižších ako 6, je potrebné pamätať na limitácie tejto konzoly, ktoré môžu niekedy priniesť komplikácie - obmedzenie použiteľných konštrukcií jazyka C++, rozdielna interpretácia niektorých špecifických prípadov. Limitácie jazyka sú popísané v súbore <https://root.cern.ch/viewvc/branches/v5-34-00-patches/cint/doc/limitati.txt> a limitácie veľkostí sú popísané v <https://root.cern.ch/viewvc/branches/v5-34-00-patches/cint/doc/limitnum.txt>.

slovník generovaný automaticky v rámci kompilácie nástrojom *ACLiC*. To, že nie je potrebné vytvárať *makefile* a pridávať špeciálne konštrukcie, ktoré sú požadované pri použití nástroja *rootcint*, robí tvorbu jednoduchých programov používajúcich triedy softvérového rámca ROOT jednoduchšiu pomocou skompilovaných makier.

Grafické rozhranie, z pohľadu používateľa pracujúceho na typických vizualizačných úlohách (kreslenie grafov, histogramov, geometrických útvarov, atď.), je zamerané na triedu *TCanvas*. Objekty na plátne môžu byť uchytené, presúvané, deformované. Väčšina objektov je odvodená od triedy *TObject*, ktorá ponúka virtuálnu metódu *Draw()* kresliacu daný objekt na aktívne plátno.

Pre vytváranie histogramov sa využívajú triedy dediace od (abstraktnej) triedy *TH1*. Podporované sú aj viacrozmerné histogramy - dvojrozmerné *TH2*, trojrozmerné *TH3*. Ich podtriedy sú implementácie histogramov s rôznou presnosťou pre políčko teda napríklad pre políčko s presnosťou 14 čísel - typ *double* je názov triedy histogramu *TH1D*. Analogicky pre typ *integer* trieda histogramu *TH1I*, alebo pre dvojrozmerný histogram pre typ *double* je trieda *TH2D*.

Práca so súborovým systémom, lokálnym aj vzdialeným, je dostupná cez triedu *TFile*, ktorá pracuje so súbormi typu *.root*. Formát súboru je navrhnutý pre prácu s objemnými dátami, vrátane podpory kompresie. Rozhranie triedy *TFile* (a všeobecne aj celého softvérového rámca ROOT) oslobodzuje používateľa od potreby uvažovania nad serializáciou údajov alebo potrebou zaručenia medzi-platformovej kompatibility. Logická organizácia súboru je založená na kľúčoch identifikujúcich objekty uložené v tomto súbore.

Pre organizáciu záznamov údajov je určená najmä trieda *TTree*. Reprezentované môžu byť jednoduché tabuľky hodnôt, ale hlavne aj zoznamy zložitých stromových štruktúr. Využitie je nachádzané väčšinou v spojitosti s ukladaním dátových štruktúr - stromov tried asociovaných s triedou udalosti (udalosť spŕšky, zrážky častíc, atď.). *TTree* je využívaná aj pre ukládanie štruktúr tried asociovaných s triedou *EEvent*,

podobne aj `RecoRootEvent` v softvérovom rámci ESAF.

Špecifikum, ktorého použitie možno badať aj v ESAF-e, sú strojovo nezávislé typy softvérového rámca ROOT. Ich používanie nie je nevyhnutné, ale môže pomôcť predísť problémom s rozličnou veľkosťou typu na rozličných strojoch. Preto možno pozorovať, v deklaráciách používanie typov ako `Int_t`, `Double_t`, `Float_t`. Táto konvencia sa používa i v softvérovom rámci ESAF.

Pre účely tejto práce sa pracuje s verziou (vetvou) softvérového rámca ROOT 5.34, keďže v čase začiatku práce nebola verzia 6 finálna. Ďalším podstatným dôvodom je, že softvérový rámec ESAF je zatiaľ väčšinou kompilovaný na serveroch s verziou 5.34. Kompatibilitu s verziou 6 by bolo potrebné dôkladne otestovať.

3.2 Simulácia

Simulácia je rozdelená na 6 hlavných sekcií: spršky začínajúcej z očakávanej UHECR spršky, fotóny z fluorescencie a Čerenkovovho javu produkované v atmosfére, propagácia svetla, optika detektora, predná elektronika a schéma spúšťačov.

3.2.1 Proces simulácie

Vstupným bodom simulácie je trieda `SimuApplication` spracováajúca vstupné parametre programu a načítavajúca konfiguráciu použitím metód triedy `Config`. Tak tiež táto trieda inicializuje a v metóde `SimuApplication::DoEvent()` volá jednotlivé časti simulácie. Voľba tried, ktorých objekty budú vytvorené, je spravidla v celom softvérovom rámci riešená pomocou metódy továrničiek (factory metod) voliacej triedu na základe konfigurácie.

Generovanie spřšky Prvým krokom simulácie je vytvorenie primárnej častice. Pre tento účel sa používa generátor udalosti spřšky triedy `EventGenerator`. V tejto práci používaný generátor je `SlastShowerGenerator`.

Generovanie fotónov v atmosfére Dáta vytvorené generátorom spřšky sú použité pri generovaní fotónov v atmosfére, ktorý je reprezentovaný triedou `LightSource`. Aktuálne implementovaný generátor fotónov je len generátor fotónov spřšky `ShowerLightSource`.

Propagácia fotónov cez atmosféru Ďalším krokom je propagácia fotónov cez atmosféru až na šošovku detektora. Pre tento účel slúžia triedy dediace od triedy `RadiativeTransfer`. Predvolenou metódou je `BunchRadiativeTransfer`. V triede `BunchRadiativeTransfer` sa tiež používa trieda `EAtmosphereBunchAdder` pre pridanie skupín fotónov do výstupného súboru. V prípade štandardne používaného generátora svetla spřšky `StandardLightToEuso` je pre simuláciu atmosféry použitá trieda dediacia od triedy `Atmosphere`. Trieda atmosféry ponúka tabulkové údaje a prepočty charakterizujúce zvolený model. Aktuálne používaná implementácia je `LowtranAtmosphere`, ktorá používa programy balíka `lowtran 7` napísanom v jazyku Fortran. Účelom programov je simulácia propagácie fotónu založená na modele „*Standard US Atmosphere*”.

Propagácia fotónov cez optiku detektora Simulácia detektora je zaobalená v triede `Detector`, respektíve v potomkoch tejto triedy - aktuálne používaná trieda `EusoDetector`. V prvom kroku fotóny na šošovke sú odovzdané metóde objektu triedy dediacej od triedy `VirtualDetectorTransportManager`, aktuálne používaná trieda `DetectorTransportManager`, ktorá pre každý fotón vykonáva transport fotónov optikou detektora. Na základe parametrov optického systému a pozície fotónu, sú do zmeny pozície zapojené steny detektora a tienidlo okolo optiky detektora.

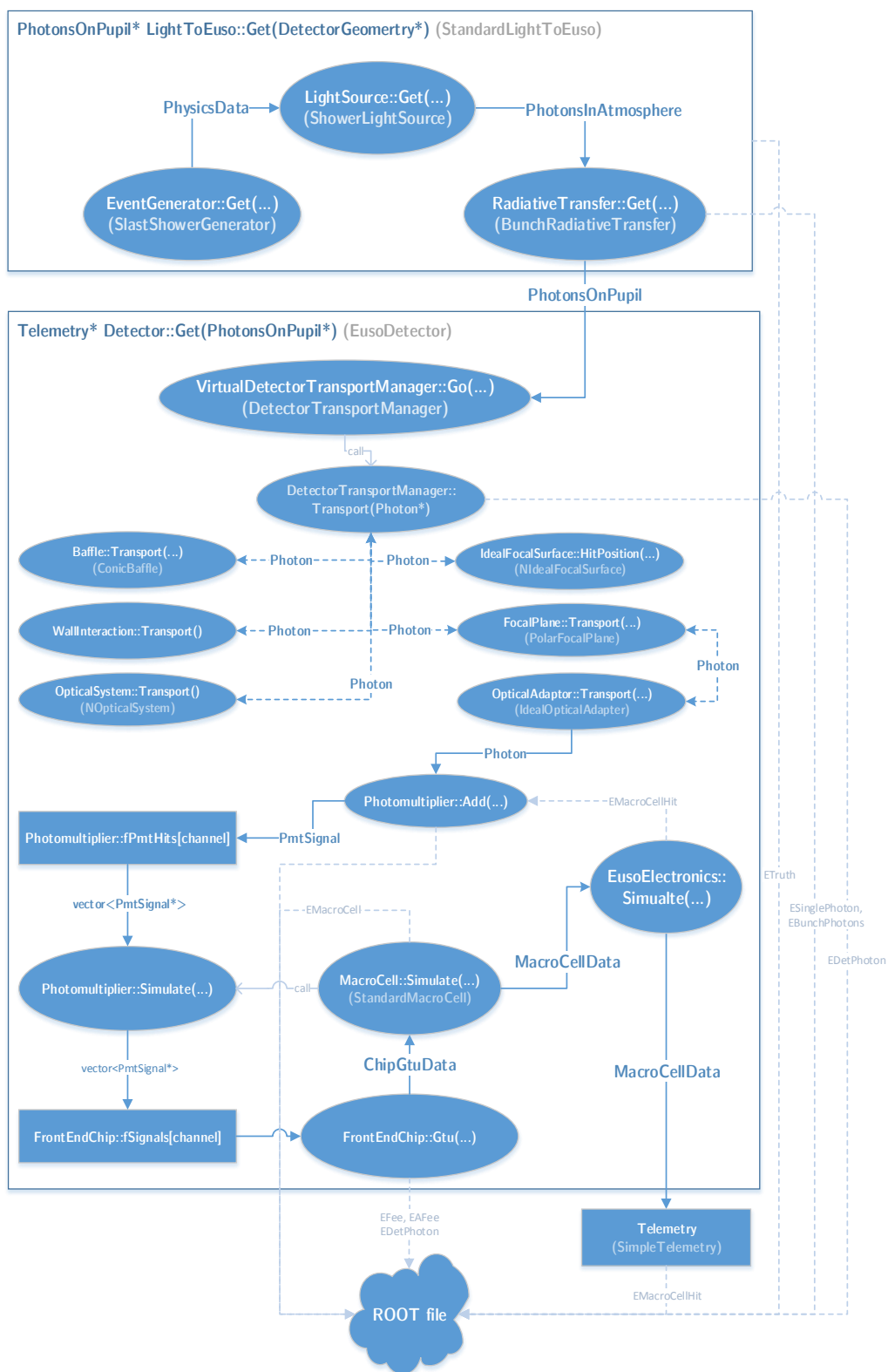
Je vypočítaný bod interakcie fotónu s optikou a fotón je cez ňu transportovaný. V prípade, že fotón trafil fotomultiplikátor (trieda `Photomultiplier`), je na tomto fotomultiplikátore zaznamenaný.

Elektronika detektora Simulácia elektroniky je reprezentovaná triedou `EusoElectronics`. Vykonáva sa tu simulácia reakcie PDM na zaznamenané fotoelektróny. Simuluje sa tu záznam zachytených fotoelektrónov prednou elektronikou ako signálov, ktorých čas je zaznamenaný. Po simulácii fotomultiplikátorov je určený posledný čas úderu, pomocou ktorého je vypočítaný počet GTU. Následne pre všetky GTU je pre všetky EC (trieda `ElementaryCell`) získaný čip prednej elektroniky (trieda `FrontEndChip`). Pre tento čip je simulovaný interval GTU volaním metódy `FrontEndChip::Gtu (...)`. V tejto metóde sa počíta čas aktivácie jednotlivých kanálov EC - pixelov, aplikuje sa vplyv nočnej žiary, počíta sa počet zaznamenaných fotónov. Počty fotónov a počty zaznamenané na pixeli (prednej elektronike) sú pridané do výstupného súboru pomocou objektov tried `EDetectorPhotonDataAdder` a `EEventFrontEndDataAdder`.

Schéma spúšťačov Posledným krokom je simulácia schémy spúšťačov reprezentovanej triedou `TriggerEngine`. Je dobré poznamenať, že to či bolo PDM aktivované neovplyvní jeho uloženie do výstupného súboru. Aktuálne používané spúšťače sú opísané v časti 4.1.

Diagram prúdu dát Obrázok 3–2 ilustruje opísaný proces simulácie využitím diagramu prúdu dát. V časti ilustrujúcej simuláciu fotónov použitím metódy `LightToEuso::Get(DetectorGeometry*)` sú údaje prechádzajúce medzi metódami ich návratové hodnoty. V časti detektora je štruktúra mierne komplikovanejšia. Metóda `VirtualDetectorTransportManager::Go(PhotonsOnPupil*)` volá pre každý fotón metódu `DetectorTransportManager::Transport(Photon*)`, tento fotón sa

v metódach `Transport` upravuje. Ak fotón spĺňa kritéria, je ako `PmtSignal` pridaný do signálov fotomultiplikátora. Ďalším krokom simulácie detektora je volanie metódy `EusoElectronics::Simulate()`, ktorá volá pre každú PDM `MacroCell::Simulate(Int_t)`, čo zase zabezpečuje volanie pre každý jeho fotomultiplikátor `Photomultiplier::Simulate()` vykonávajúce transformáciu signálov na čipy prednej elektroniky. Tieto údaje sú potom využité neskôr, keď `MacroCell::Simulate(Int_t)` volá pre každý čip prednej elektroniky volá `FrontEndChip::Gtu(...)`. Doplnkovo, prerušovanou čiarou sú vyznačené triedy použité vo výstupnom súbore, ktorých objekty sú napĺňané údajmi. Metódy, z ktorých v diagrame vychádza táto prerušovaná čiara údaje zapisujú použitím objektov s triedami s názvami končiacimi príponou `Adder`. Napríklad pre záznam údajov prednej elektroniky je to trieda `EEventFrontEndDataAdder`.



Obr. 3–2: Diagram dátových tokov v procese simulácie.

3.2.2 Výstup simulácie

Údaje simulácie sa ukladajú do „stromu“ triedy `EEvent`, ktorá vytvára koreň stromu tried reprezentujúcich mnohé údaje simulovanej spŕšky, jej objekt je uložený do `.root` súboru. Tento súbor obsahuje stromovú štruktúru triedy `TTree` s vetvami mappovanými na triedu `EEvent`. Kôli spôsobu označovania spŕšky v ESAF-e ako „*event*“ sa bude v tomto texte používať pre spŕšku aj označenie „*udalosť*“ (Pavllavicini, Thea, 2004).

Čítanie z výstupného súboru simulácie Výpis 1 uvádza zjednodušený príklad (nezohľadňuje spracovanie chýb) čítania údajov z výstupného súboru simulácie typu `.root`. V príklade sú použité premenné `file_name` a `entry_index`. Premenná `file_name` predstavuje reťazec (typ `const char*`) cesty k čítanému súboru a premenná `entry_index` určuje index (typ `Long64_t`) čítanej udalosti.

Výpis 1: Princíp čítania objektu triedy `EEvent` z výstupného súboru simulácie.

```
TFile *f = f = new TFile(file_name);
TTree * etree = (TTree*)f->Get("etree");
EEvent * ev = new EEvent();
ev->SetBranches(etree);
etree->GetEntry(entry_index);
```

Strom triedy `EEvent` Táto trieda zaobahuje údaje udalosti simulovanej spŕšky. Spolu s triedami asociovanými s touto triedou sa vytvára „strom“, z ktorého sú prístupné aspekty tejto udalosti. Niektoré najpodstatnejšie „getter“ metódy triedy `EEvent`:

- `EHeader *GetHeader()` - Základné informácie o udalosti. Napríklad číslo udalosti, číslo behu, zemepisná šírka a výška.

- `ETruth *GetTruth()` - Všeobecné údaje pre ktoré bola udalosť simulovaná. Napríklad energia, zenitový uhol, hĺbka interakcie.
- `EShower *GetShower()` - Všeobecné informácie o spríške - informácie o krokoch spríšky, energia spríšky, zenitový uhol.
- `EAtmosphere *GetAtmosphere()` - Fotóny v atmosfére.
- `EDetector *GetDetector()` - Informácie o reakcii detektora na zaznamenané fotóny. Podstatnou metódou je triedy je `EFee* GetFee(Int_t index)`, jej výsledok je záznam pre jeden pixel prednej elektroniky obsahujúci počet fotoelektrónov, GTU, pozíciu.
- `ERunParameters *GetRunPars()` - Parametre detektora počas simulácie. Napríklad rozmer strany PMT, priestor medzi PMT, nočná žiara atmosféry, mapa pixelov detektora, pozície bodov, indexovanie bodov, atď.
- Údaje spúšťačov.

3.3 Rekonštrukcia

Cieľom rekonštrukcie je rekonštruovať vlastnosti primárnej častice, ktorá vytvorila spríšku sekundárneho kozmického žiarenia. Rekonštrukcia využíva údaje získané z detektora za pomoci rozličných metód implementovaných ako moduly zodpovedné za špecifickú úlohu rekonštrukcie. Rekonštruuje sa smer, energia, slant hĺbka maxima spríšky. Tieto moduly môžu zapisovať údaje do výstupného súboru rekonštrukcie a vytvárať (sprístupňovať) údaje pre ďalšie moduly. Vstup rekonštrukcie môže byť zatiaľ len výstupný súbor simulácie, no práve cieľom tejto práce je pridať ďalšiu možnosť.

3.3.1 Priebeh rekonštrukcie

Rekonštrukcia začína načítaním vstupného súboru, využíva sa tu vstupný modul - podtrieda triedy `InputModule`, typicky `RootInputModule`. Ak má udalosť (`RecoEvent`) nejaké záznamy pixelov detektora, nasleduje postupné volanie modulov, ktoré sú reprezentované objektom triedy dediacej od `RecoModule`. Volané sú metódy `RecoModule::PreProcess()`, `RecoModule::Process(RecoEvent* anEvent)`, `RecoModule::PostProcess()` a `RecoModule::SaveRootData(RecoRootEvent *fRecoRootEvent)`. Ak akákoľvek z týchto metód zlyhá - vráti `FALSE`, rekonštrukcia udalosti sa ukončí. Po spracovaní údajov modulu sa smerník na jeho verejné údaje (trieda `RecoModuleData`) uloží do stromu objektu spracováanej udalosti. Postupnosť volania modulov je určená konfiguráciou (parameter konfigurácie `RecoFramework.ModuleFile` - cesta k súboru s postupnosťou názvov použitých modulov).

V aktuálne používaných scenároch, prvý modul spracovávajúci udalosť je spracovanie vzoru. Typicky sa používa algoritmus PWISE (modul `PWISEModule`). Fungovanie algoritmov implementovaných v použiteľných moduloch rekonštrukcie je opísané v časti 4.2.

Ďalším postupom je rekonštrukcia smeru spršky inklináčného (zenitového) uhla θ a azimutového uhla ϕ . Pre tento účel sa aktuálne využíva modul `TrackDirection2Module`. Nasleduje určenie rozmerov a pozície spršky v atmosfére, z čoho možno nakoniec určiť energiu spršky. Tu je aktuálne používaný modul `PmtToShowerReco`.

3.3.2 Modul rekonštrukcie

Trieda `RecoModule` definuje modul rekonštrukcie. Triedy dediace od tejto triedy musia implementovať tieto abstraktné metódy:

- `virtual Bool_t Init()` - typické úlohy sú inicializácia modulu a načítanie

konfigurácie, volané pred spracovaním udalostí zo vstupného súboru,

- `virtual Bool_t PreProcess()` - metóda volaná pred spracovaním procesu,
- `virtual Bool_t Process(RecoEvent*)` - spracovanie udalosti,
- `virtual Bool_t PostProcess()` - metóda volaná po spracovaní procesu,
- `virtual Bool_t SaveRootData(RecoRootEvent*)` - uloženie informácií výstupného objektu rekonštrukcie, metóda volaná po metóde `PostProcess()`,
- `virtual Bool_t Done()` - metóda volaná na konci rekonštrukcie (pred ukončením programu).

3.3.3 Trieda `RecoEvent`

Objekt triedy `RecoEvent` sprístupňuje údaje spracovávané rekonštrukciou, a taktiež údaje vytvorené modulmi rekonštrukcie. Pri tvorbe modulov rekonštrukcie je práca s touto triedou nevyhnutná. Metódy triedy `RecoEvent` podstatné z pohľadu tejto práce:

- `RecoEventHeader &GetHeader()` - generálne informácie o udalosti. Napríklad počet všetkých pixelov zaznamenaných detektorom, skutočný inklináčny uhol, atď.
- `RecoModuleData *GetModuleData(const string&)` - Kontajner rôznych výstupných údajov modulov.
- `vector<RecoPixelData*> &GetRecoPixels()` - Informácie o pixeloch zaznamenaných detektorom. Napríklad počet všetkých zaznamenaných fotoelektrónov.
- `RecoGlobalData* FindGlobalData(const string& name)` - Kontajner

globálnych dát, do ktorých môžu moduly rekonštrukcie zapísať údaje. Tieto dáta nie sú viazané na názov modulu ako je to pri `RecoModuleData`.

3.3.4 Výstup rekonštrukcie

Objekt triedy `RecoRootEvent` je výstup rekonštrukcie uložený do výstupného súboru softvérového rámca ROOT reprezentovaného triedou `TFile`. Jednotlivé záznamy sú organizované ako záznamy stromov využitím objektu triedy `TTree`. Principiálne sa dá porovnať s triedou `EEvent`, no strom tried s koreňom `RecoRootEvent` je oveľa jednoduchší a zahŕňa menej dát. Ilustruje to aj veľkosť výstupného súboru rekonštrukcie, ktorého veľkosť je v stovkách kilobajtov, veľkosť výstupného súboru simulácie má veľkosť v stovkách megabajtov (najmä v závislosti od uhlov a energií simulovaných spŕšok). Fakt, že táto štruktúra bola navrhnutá len pre uchovávanie finálnych výsledkov, pri analýze (najmä vlastností algoritmov rekonštrukcie) vyžaduje, buď čítanie údajov z výstupného súboru simulácie (inak povedané dostupnosť týchto súborov), alebo obídenie štandardnej metódy a zápisu údajov (v metóde `SaveRootData(RecoRootEvent*)`) a zápisom priamo do otvoreného súboru softvérového rámca ROOT (reprezentovaného triedou `TFile`) v samotných moduloch rekonštrukcie. Výpis 2 zjednodušene ilustruje princíp ako získať `RecoRootEvent` z výstupného súboru rekonštrukcie.

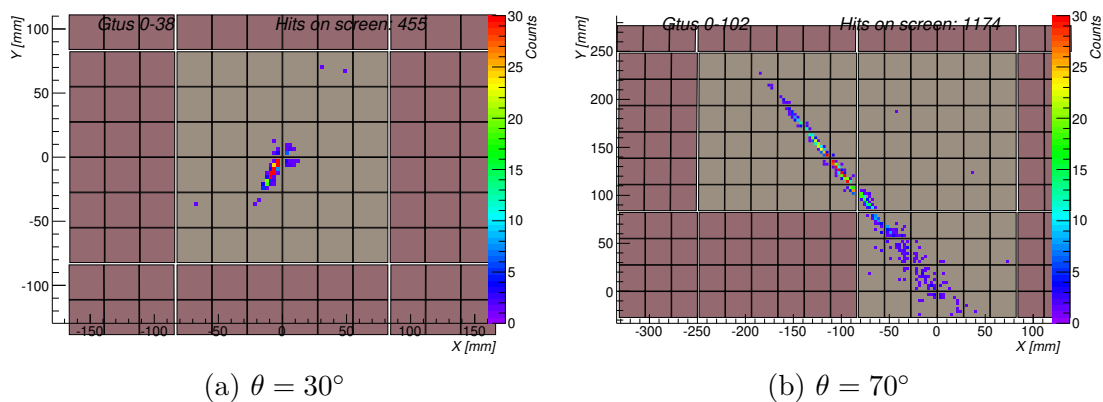
Výpis 2: Princíp čítania objektu triedy `RecoRootEvent` z výstupného súboru simulácie.

```
TFile *f = f = new TFile(file_name);  
TTree * recotree = (TTree*)f->Get("recotree");  
RecoRootEvent * evsreco = new RecoRootEvent();  
recotree->SetBranchAddresses("events",&evsreco);  
recotree->GetEntry(entry_index);
```

3.4 Vplyv rôznych konfigurácií na výsledok simulácie

Problémom pri generovaní vhodných spŕšiek pre nasledovné testovanie algoritmov rekonštrukcie bolo použitie optimálnej konfigurácie - viaceré testované dávali odlišné výsledky. Pre účely tejto práce bola sledovaná závislosť konfiguračných parametrov a pixelov zaznamenaných na detektore. Sledoval sa počet zaznamenaných fotoelektrónov na pixeloch, počet zaznamenaných pixelov a pozícia zaznamenaných pixelov. Vizualne sa zaznamenaná spŕška dá porovnať pohľadom na tie pixely, ktoré zaznamenali fotoelektróny vytvorené z fotónov skutočnej spŕšky.

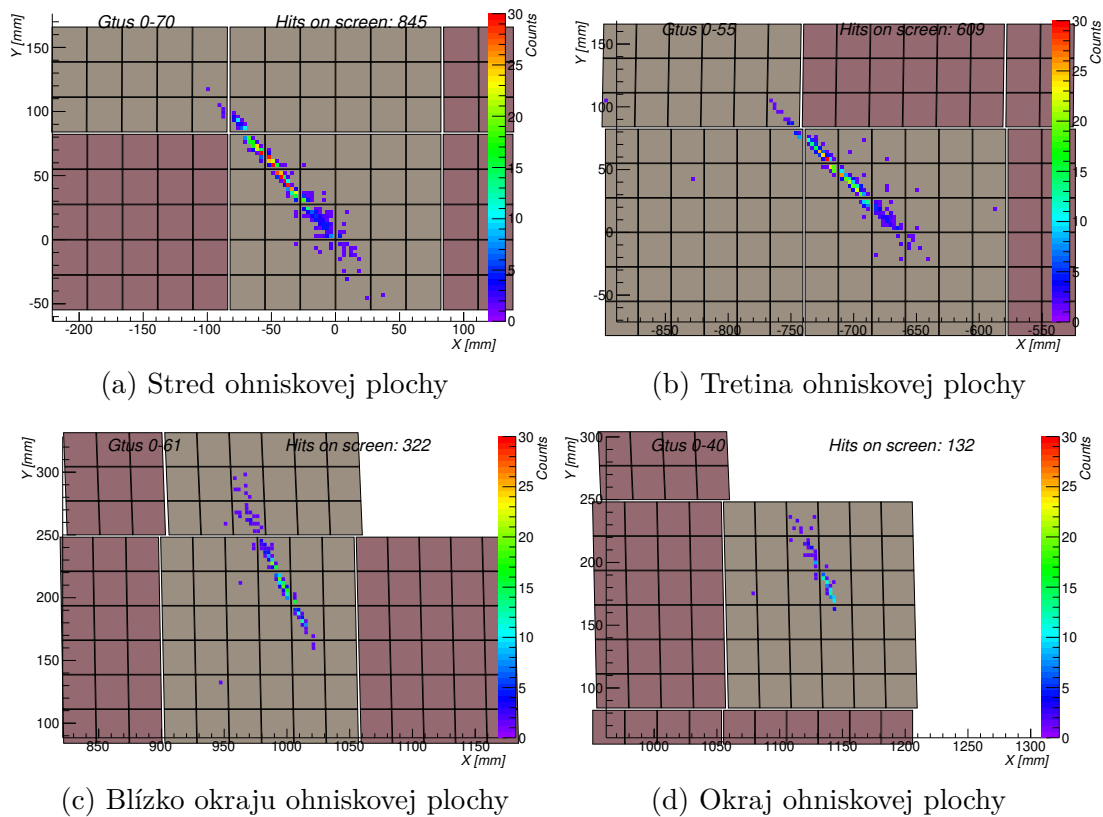
Zenitový uhol spŕšky Prvým dobre badateľným faktorom ovplyvňujúcim zaznamenanú spŕšku je zenitový uhol θ , pod ktorým bola spŕška simulovaná. Pri použití generátora spŕšok reprezentovaného pomocou triedy `SlastShowerGenerator`, používaného vo všetkých simuláciách v tejto práci, sa tento parameter nastavuje parametrami `GeneratorLightToEuso.ThetaRangeMin` a `GeneratorLightToEuso.ThetaRangeMax`. Zenitový uhol spŕšky je uhol medzi kolmicou na povrch Zeme a smerom, z ktorého primárna častica prichádza. To znamená, že z pohľadu detektora, čím nižší uhol θ , tým je zaznamenaná spŕška kratšia. Krátke spŕšky sú ťažšie rekonštruovateľné, keďže z malého zhluku pixelov je smer menej jasný. Tento rozdiel v dĺžkach spŕšok ilustruje obrázok 3–3. Na tomto obrázku sú vizualizované spŕšky, ktorých jediný rozdiel v konfiguračných parametroch bol rozsah zenitových uhlov. Počet fotoelektrónov pixelov na obrázku je vypočítaný ako suma skutočných signálov zo spŕšky pre všetky GTU tejto spŕšky.



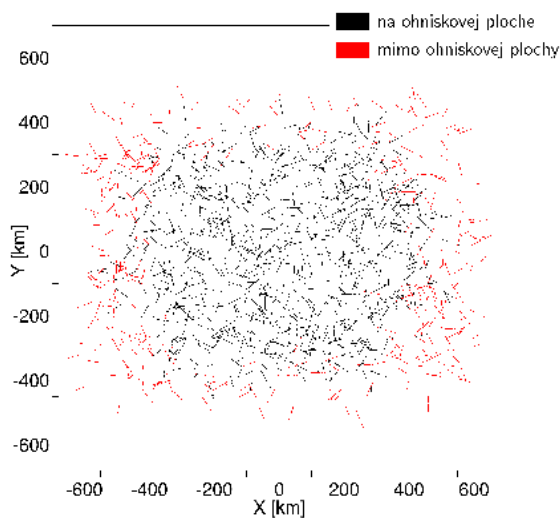
Obr. 3–3: Porovnanie spřšky s rozdielnym zenitovým uhlom θ .

Pozícia spřšky na ohniskovej ploche Ďalším faktorom vizuálne ovplyvňujúcim simulovanú spřšku je pozícia, na ktorej je spřška na detektore zaznamenaná. Efektivita optiky detektora je nižšia na okraji, a teda na tieto časti ohniskovej plochy dopadá menej fotónov spřšky v porovnaní so stredom. Taktiež v závislosti od konfigurácie simulovanej nočnej žiary úroveň pozadia môže byť nižšia. Parametre konfigurácie ovplyvňujúce túto veličinu sú rozsahy pozície dopadu spřšky v kilometroch `GeneratorLightToEuso.ImpactXmin`, `GeneratorLightToEuso.ImpactXmax`, `GeneratorLightToEuso.ImpactYmin`, `GeneratorLightToEuso.ImpactYmax`. Obrázok 3–4 porovnáva spřšky s maximom na rozdielnej pozícii ohniskovej plochy. Pre tieto obrázky platí rovnaké ako v predošlom prípade. Z obrázkov možno pozorovať, že efekt je významný až na okraji ohniskovej plochy. Obrázok 3–5 ilustruje simulované spřšky s rozdielnou pozíciou dopadu, červené neboli na detektore zaznamenané⁴.

⁴Pre vygenerovanie tejto vizualizácie bolo použité makro `MvEventsStat.C` opísané v používateľskej príručke.

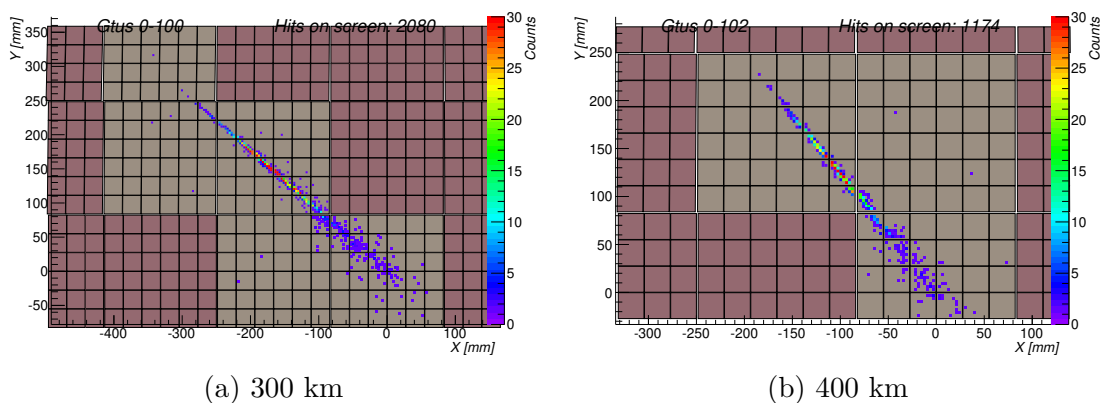


Obr. 3–4: Porovnanie spŕšok s rozdielnou pozíciou maxima na ohniskovej ploche detektora. Zenitový uhol simulovaných spŕšok θ je 60° .



Obr. 3–5: Spŕšky simulované s rôznou pozíciou dopadu.

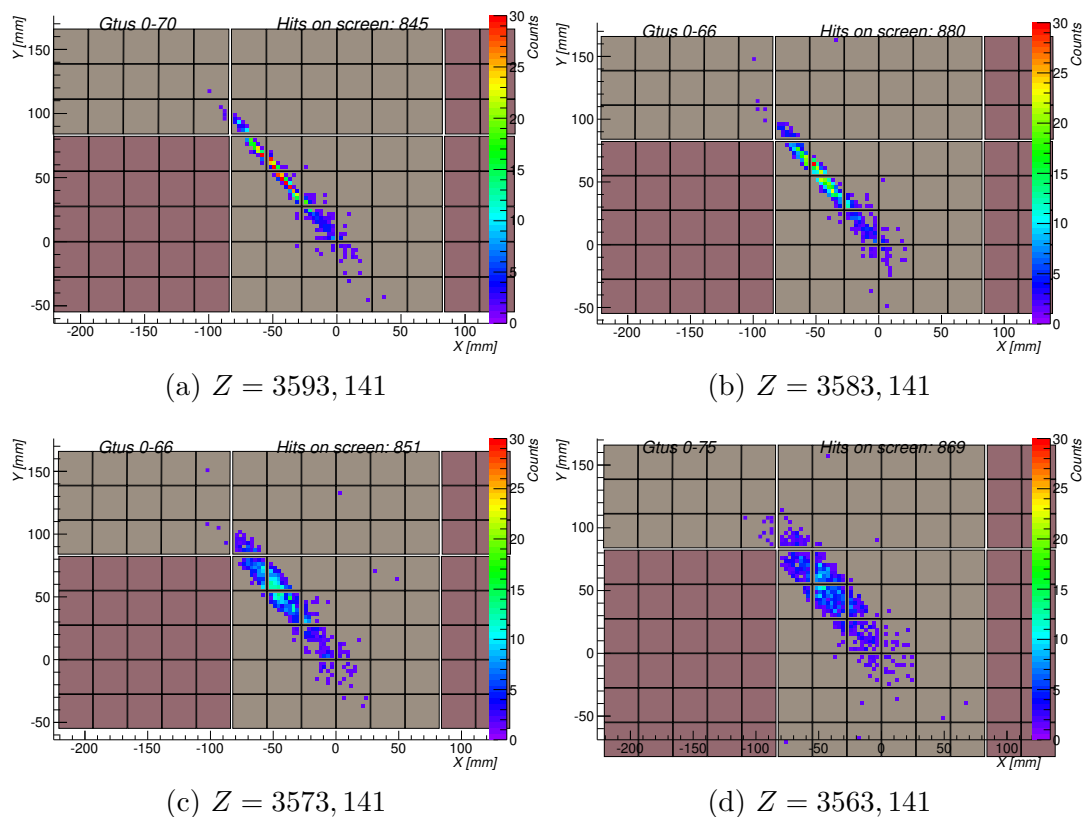
Výška ISS Malá zmena výšky orbity ISS nespôsobí žiadne zásadne pozorovateľné zmeny vo výsledkoch a v každom prípade je tento údaj pri rekonštrukcii známy. Obrázok 3–6 porovnáva spŕšky simulované v dvoch rôzne vysokých orbitách (respektíve pohľad z rozdielnych výšok pri eliptickom orbite).



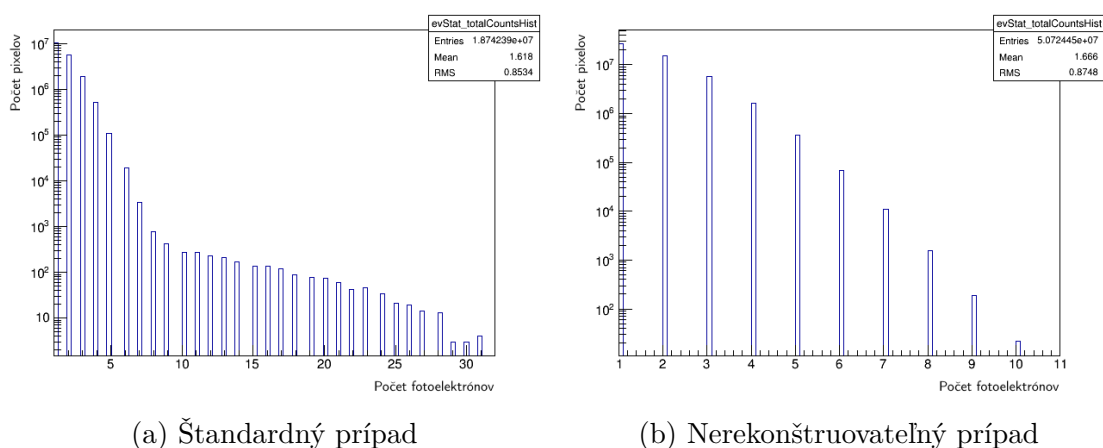
Obr. 3 – 6: Porovnanie spŕšok s rozdielnou výškou orbity.

Nastavenie optiky detektora Správne nastavenie pozície prednej elektroniky detektora je podstatné, pretože so znižujúcou ostrosťou zaznamenanaj spŕšky sa stráca možnosť jej odlíšenia od pozadia. Obrázok 3–7 porovnáva nastavenie pozície ohniskovej plochy parametrom `PolarFocalPlane.fPos.Z`, ktorá je postupne „rozostrovaná“ zo správnej pozície. Počet zaznamenaných fotoelektrónov pre rozostrenú spŕšku je nízky, v extrémnych prípadoch sa fotoelektróny produkované z fotónov spŕšky úplne stratia v šume, čo ilustrujú histogramy na obrázku 3–8⁵.

⁵Pre vygenerovanie týchto histogramov bolo použité makro `MvEventsStat.C` opísané v v používateľskej príručke.

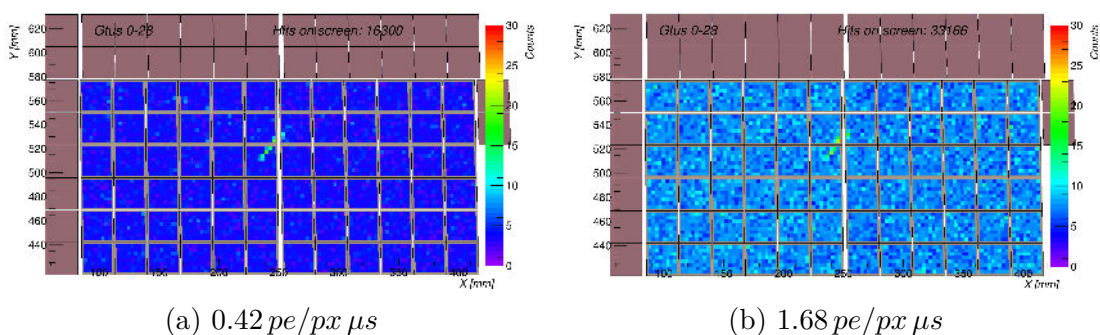


Obr. 3–7: Porovnanie nastavení pozície ohniskovej plochy mimo ohniska optiky (Parameter PolarFocalPlane.fPos.Z).



Obr. 3–8: Porovnanie spršok s rozdielnym množstvom fotoelektrónov skutočného signálu spršky.

Intenzita UV pozadia Podstatným parametrom simulácie je intenzita UV pozadia, ktorá je v softvérovom rámci závislá od použitého modelu. Všeobecne sa v tejto práci používalo nastavenie priradzujúce rovnakú úroveň intenzity pre každú elementárnu bunku na ohniskovej ploche. Obrázok 3–9 porovnáva štandardne používanú úroveň pozadia $0.42 \text{ pe/px } \mu\text{s}$ (obrázok 3–9a) s vyššou intenzitou (obrázok 3–9b). Na obrázkoch sa pre pixel zobrazuje jeho najvyšší počet zaznamenaných fotoelektrónov. Obrázky vizuálne ukazujú, že záznamy spŕšok s vyššou intenzitou UV pozadia sú ťažšie rozpoznateľné.



Obr. 3–9: Porovnanie rozdielnych intenzít pozadia.

Nastavenia spúšťačov Nastavenia spúšťačov neovplyvňujú generovanú spŕšku, keďže informácia o tom, ktoré PDM boli aktivované, je doplnková. Mimo simulácie by zmena nastavenia spúšťačov ovplyvnila to, aké spŕšky by boli zaznamenané. Aktivované spúšťače, ale môžu ovplyvniť rekonštrukciu, keďže napríklad modul `PmtToShowerReco` podmieňuje začatie výpočtov tým aké spúšťače boli aktivované.

Vlnová dĺžka generovaných fotónov Rozmedzie 300 až 400 nm je najpodstatnejšie, keďže najviac produkovaných fotónov (fluorescence yield) sa nachádza v tomto rozmedzí - rozmedzie blízkeho UV. Nastavenie vlnových dĺžok zásadne mimo tohto rozsahu môže spôsobiť problémy simulácie.

4 Spúšťacie algoritmy a algoritmy rekonštrukcie vzoru v softvérovom rámci ESAF

Spracovanie údajov zaznamenaných prednou elektronikou detektora prebieha vo viacerých fázach. Prvá fáza sa deje ešte s pomocou elektroniky detektora. Toto spracovanie sa označuje ako *spúšťací mechanizmus*. Detektor je navrhnutý s dvoma takýmito úrovňami PTT spúšťačom a CCB_LTT spúšťačom (Ebisuzaki T. et al., 2010). Druhá fáza je rekonštrukcia, ktorú vykonáva program Reco. Pri rekonštrukcii sú aktuálne dostupné algoritmy *PWISE* a *Robust* (označovaný aj ako Track Finding Method - Metóda hľadania dráhy), ktoré môžu byť voliteľne doplnené algoritmom *LTT pre-clustering*.

4.1 Algoritmy spúšťacieho mechanizmu

Hlavným dôvodom existencie spúšťacieho mechanizmu je obmedziť množstvo zaznamenaných dát z meraní detektora. Bez tohto mechanizmu by bolo nutné zaznamenávať 125 GB/s, čo je nereálne berúc do úvahy, že množstvo dát vymedzené pre JEM-EUSO je 3 GB/deň - celý výpočet aj s ďalšími podrobnosťami opisuje Fenu (Fenu, 2013). Prvý spúšťací (aktivovací) algoritmus PTT je vykonávaný použitím FPGA v každej PDM a verifikuje výskyt perzistencie signálu na každej z jeho deviatich EC. Zaznamenaný výskyt spúšťacieho signálu v PDM aktivuje ďalší spúšťací algoritmus CCB_LTT na úrovni CCB, ktorá fyzicky spája skupiny 8 až 9 PDM. Aktivácia spúšťačov vychádza z analýzy pixelov patriacich rovnakému PDM, ale ostáva preskúmaná možnosť analýzy vo väčšom rozsahu.

4.1.1 Persistent Tracking Trigger

PTT algoritmus zamietá väčšinu fluktuácií pozadia požadovaním lokálne perzistentného signálu nad úrovňou pozadia trvajúc viacero GTU. Autor algoritmu je O.Catalano, optimalizácie na algoritme urobil M. Bertiana (Adams et al., 2012). PPT nastavuje prah na každom pixel-GTU, čo definuje prebytok hodnoty na pixeli. Pixely sú zlúčené v 3×3 skupinách. Všetky prebytky vo vnútri skupín sú integrované a priradené k centrálnemu pixelu. Zároveň prítomnosť minimálne jedného pixela nad prahom v EC je podmienkou pre inkrementáciu perzistentného počítadla. Ak takýto pixel nie je prítomný, perzistentné počítadlo je nastavené na 0. Perzistentné počítadlo je teda zvyšované pokiaľ nenastane stav, že sa v EC nenachádza pixel nad prahom. Ak bola podmienka perzistencie verifikovaná, mapa prebytkov hodnôt na pixeli je uložená do kruhového zásobníka. Ak podmienka perzistencie nenastala, kruhový zásobník je resetovaný. Prah perzistencie je nastavený na určitý počet GTU - v aktuálnych konfiguráciách 5. Ak je prah perzistencie prekročený prebytky vo všetkých skupinách sú integrované pre všetky predchádzajúce GTU v zásobníku, resp. do „hlbky” prahu perzistencie. Finálne, ak táto integrácia prekročí prah integrácie je nastavený signál spúšťača.

4.1.2 CCB-Linear Tracking Trigger

Základná myšlienka algoritmu CCB_LTT je integrácia počtov sústredených v oblasti (skupine, štvorci) posúvanej pozdĺž preddefinovanými smermi. Autorom tohto algoritmu je M. Bertiana (Adams et al., 2012). Algoritmus je náročnejší ako PPT a vykonáva sa pomocou elektroniky CCB - doska riadenia klastra. Kombinácie pixelov a GTU (skrátene pixel-GTU), na ktorých bol aktivovaný PPT spúšťač slúžia ako začiatkový stav pre aplikáciu celej CCB procedúry spúšťačov. Následne je okolo tohto stavu definovaná množina pixel-GTU, v aktuálnej konfigurácii s rozmermi 3×3 v rozmedzí ± 7 GTU. Všetky pixel-GTU v tomto rozsahu (135 pixel-GTU) slúžia

ako začiatkové podmienky pre sériu testovacích integrácií. V každej z nich je definované 324 integračných smerov. Každý smer je definovaný ako projekcia ideálnej spŕšky v zornom poli detektora na ohniskovej ploche. Zoznam smerov musí brať do úvahy možné smery príchodu EAS. Nízke zenitové uhly (θ) spŕšok budú projektované ako stojaca integrácia, zatiaľ čo horizontálne budú pretínať väčšie frakcie ohniskovej plochy. Integrácia bude vykonaná v rozsahu ± 7 GTU od štartovacieho stavu. V každom GTU sa oblasť pohybuje na základe rovníc $\Delta_x = \Delta_t * K_x$, $\Delta_y = \Delta_t * K_y$, kde Δ_x a Δ_y sú posunutia osí vyjadrené v počte pixelov vo vzťahu k štartovacej pozícii integrácií a Δ_t je posunutie v GTU vo vzťahu k začiatkovému času integrácie. K_x a K_y sú definované v rovniciach 4.1, 4.2.

$$K_x = \sqrt{\frac{\tan^2 \frac{\hat{\theta}}{2}}{\left(\frac{\Delta_{Pix}}{GTU_{Light}}\right)^2 * (1 + \tan^2 \hat{\phi})}} \quad (4.1)$$

$$K_y = K_x * \tan \hat{\phi} \quad (4.2)$$

V rovniciach 4.1, 4.2 $\hat{\theta}$ a $\hat{\phi}$ reprezentujú polárny a azimutový uhol v polárnom súradnicovom systéme, ktorých polárna os je čiara od spŕšky k detektoru. Oboje tieto parametre sú uvedené v preddefinovanej 324 riadkovej tabuľke. Parametre Δ_{Pix} a GTU_{Light} reprezentujú veľkosť na Zemi v pixeloch a vzdialenosť prejdenú svetlom za jeden GTU. Týmto spôsobom je možné vypočítať pre každé z ± 7 GTU pozíciu 3×3 pixelov integračnej oblasti. Všetky počty v tejto oblasti a prekračujúce prah pixela sú potom integrované. Ak zhromaždený signál prekračuje aktuálny integračný prah, tak je vyprodukovaný spúšťač signál.

4.2 Algoritmy rekonštrukcie vzoru

Algoritmy rekonštrukcie vzoru sú implementované v rekonštrukčnej časti *Reco* systému ESAF. Moduly rekonštrukcie je možné používateľsky aktivovať resp. deaktivovať a je možný scenár, že sú na vstupné dáta použité viaceré moduly rekonštrukcie vzoru (možné to je v závislosti od implementácie daných modulov). Časová a pamäťová náročnosť môže byť pri týchto algoritmoch zásadne vyššia uprednostňujúcej kvalitu rozpoznania vzoru v porovnaní so spúšťacími algoritmi.

4.2.1 Linear Tracking Trigger pre-clustering

Algoritmus LTT pre-clustering a jeho implementácia v systéme ESAF `LTTPatternRecognition` nie je určený priamo ako samostatný modul rekonštrukcie, ale ako modul použitý pred samotným rozpoznávaním vzoru. Algoritmus je analógiou k spúšťaciemu algoritmu `CCB_LTT` a cieľom jeho použitia je zo všetkých údajov daných PDM, ktoré sú vstupom rekonštrukcie, vybrať tie pixel-GTU, ktoré popisujú spíškku. Teda podobne ako `CCB_LTT` pixely najväčšími hodnotami sú vybrané a posúvaním integračnej oblasti preddefinovanými smermi vyhľadáva cestu pretínajúcu tento bod. Pri výslednej zmenšenej analyzovanej množine sú teda odstránené akumulácie pixelov, ktoré by potenciálne mohli byť omylom rozpoznané ako signál.

4.2.2 Peak and Window SEarching

PWISE algoritmus a jemu prislúchajúci modul rekonštrukcie `PWISEModule`, je v systéme ESAF v čase písania práce hlavný používaný algoritmus pre rekonštrukciu vzoru spíškky z dát detektora. Autorom tohto algoritmu aj jeho implementácie je A. Guzman (Guzman et al., 2013).

PWISE pozerá na každý počet zaznamenaný pixelom ako na funkciu času. Použí-

vajúc tieto informácie vyhľadáva počty na pixele, ktoré odpovedajú očakávanému správaniu hýbajúceho sa bodu EAS, ktorý žiari na daný pixel. Tento proces je opakovaný pre všetky aktivované pixely. Výstup modulu je zoznam zvolených pixelov, kde každý pixel má pridelené časové okno, ktoré odpovedá očakávanému času spršky od toho, čo začiatok spršky zažiaril na tento pixel. Okrem výberu počtov na pixeloch, ktoré pochádzajú z EAS, PWISE tiež filtruje viaceré rozptýlené fotóny, čo má výsledok približný („fuzzy”) obraz dráhy. Tento efekt sa objavuje ako následok ich posunutého času príchodu z dôvodu viacerých rozptylov. Cieľ je poskytnúť dostatočne počty signálov pre algoritmus rekonštrukcie uhla a optimalizovať ich výkon aj za cenu straty svetla spršky filtrovacím procesom.

Algoritmus PWISE je vo viacerých zdrojoch (Biktemerova et. al, 2013), (Guzman et al., 2013), (Fenu, 2013) prezentovaný v použití spolu s algoritmom Linear Tracking Trigger pre-clustering. Testy ale ukázali, že použitie tohto algoritmu pred algoritmom PWISE neprináša zásadné vylepšenie výsledkov (Mernik et al., 2012).

Priebeh algoritmu PWISE

Krok 1 Sú vybrané pixely, ktorých najvyššia hodnota je nad určeným prahom.

Krok 2 Je vyhladané časové okno s najvyšším pomerom signálu ku šumu definovanom (SNR) v rovnici 4.3.

$$SNR = \left(\frac{1}{\Delta\tau * RMS} \right) \sum_{\Delta\tau} pc(t) \quad (4.3)$$

V rovnici 4.3 $pc(t)$ označuje hodnotu pixela ako funkciu času, $\Delta\tau$ je dĺžka časového okna centrovaná na najvyššiu hodnotu pixela a RMS je stredná kvadratická odchýlka všetkých hodnôt pixela.

Krok 3 Kontrola, či maximálne SNR je nad prahom SNR. Ak áno, v danom okne sú vybrané hodnoty ktoré maximalizujú SNR.

Štandardná konfigurácia Pre všetky analýzy v tejto práci sa pre algoritmus PWISE používa konfigurácia uvedená vo výpise 3. Podstatnými parametrami, ktoré je potrebné si všímať, sú `PWISEModule.fThreshold` a `PWISEModule.fSNRejection`.

Výpis 3: Štandardná konfigurácia algoritmu PWISE.

```
PWISEModule.fAbsThreshold = 9
PWISEModule.fLowThreshold = 1
PWISEModule.fSNRejection = 5
PWISEModule.fThreshold = 8
PWISEModule.fPWISER = no
```

Vlastná analýza algoritmu PWISE Pre účely porovnania bol algoritmus PWISE analyzovaný na rovnakých dátach, s ktorými sa robila analýza implementácie algoritmu Houghovej transformácie. Pri analýze sa vychádzalo zo simulovaných 1000 (a viac udalostí pre každý uhol)⁶, z ktorých časť nedopadla na ohniskovú plochu detektora a časť nebola použiteľne rozpoznaná algoritmom PWISE. Simulácia s použitou konfiguráciou vyprodukovala 66,2% až 73,0% použiteľných udalostí⁷.

Obrázok 4–1 zobrazuje percentuálny podiel počtu zrekonštruovaných spŕšok z celkového počtu použiteľných spŕšok. Tento údaj je podstatný, pretože ukazuje aký podiel spŕšok prejdených rekonštrukciou vzoru je použiteľných v module rekonštrukcie uhla `TrackDirection2`. Dôvod nepoužitelnosti rozpoznávaných vzorov je zvyčajne málo bodov - v tejto analýze to bolo 10 bodov⁸. Z týchto udalostí mohla byť vypočítaná štatistika hodnôt metriky γ_{68} . γ vyjadruje rozdiel medzi skutočným a zrekonštruovaným uhlom a γ_{68} je hodnota, v ktorej kumulatívna distribúcia γ dosahuje 0.68.

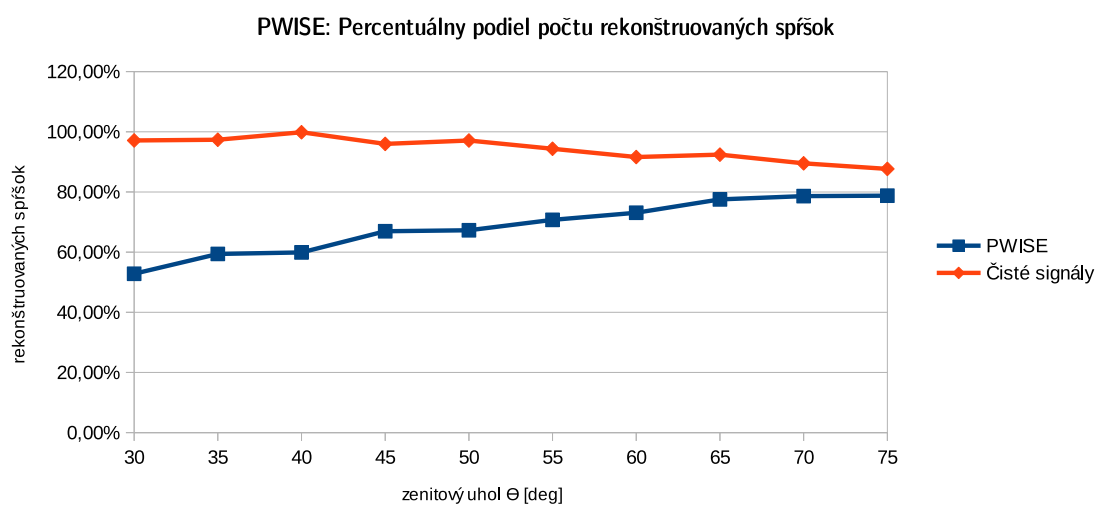
Namerané výsledky sú na obrázku 4–2 pre zenitové uhly θ vo veľkostiach 30, 35, 40,

⁶Semienko generátora náhodných hodnôt - parameter konfigurácie `EsafRandom.fSeed` bol pre všetky simulácie nastavený na hodnotu 11274.

⁷Nepoužiteľné udalosti vznikli simulovaním spŕšky ktorej svetlo nedopadlo v dostatočnej miere na ohniskovú plochu detektora, keďže realizovaná simulácia pokrývala celé možné pole pohľadu detektora.

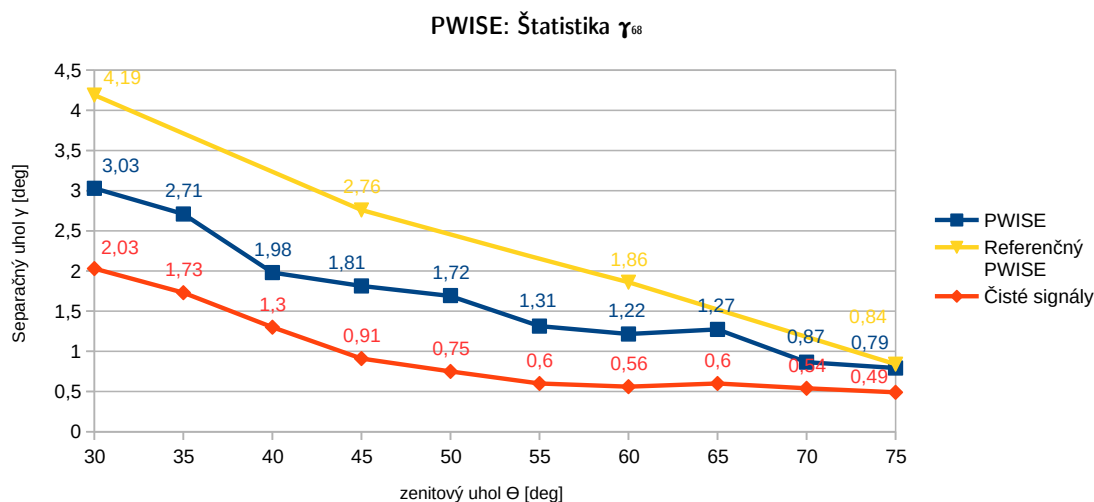
⁸Minimálny akceptovaný počet bodov modulom `TrackDirection2` je nastaviteľný parametrom `TrackDirection2Module.fNumPointsMin`.

45, 50, 55, 60, 65, 70, 75 stupňov⁹ - teda zvyšujúci sa po 5 stupňoch. Graf označený „PWISÉ reference” sú hodnoty prezentované v článku (Biktemerova et. al, 2013), graf označený „PWISÉ analysis” zase označuje namerané hodnoty. Možno pozorovať, že namerané hodnoty sú lepšie ako referenčné. Táto analýza bola vykonaná na rôznych sadách simulovaných spršok s postupne zväčšujúcou sa štatistikou. Prezentované sú výsledky namerané z najväčšej použitej množiny simulovaných udalostí. Dobrý pohľad na efektívnosť algoritmu rekonštrukcie vzoru ponúka aj trojrozmerný histogram používajúci pre osi X a Y pozíciu maxima spršky a ako os Z hodnotu γ , teda rozdiel medzi rekonštruovanou a skutočnou veľkosťou uhla zenitového θ , ktorý je uvedený na obrázku 4–3. Obrázky porovnávajú metódu PWISÉ s výsledkami rekonštrukcie, kde vybrané pixely modulom rekonštrukcie vzoru boli tie, ktoré majú počet skutočných signálov viac ako 1.



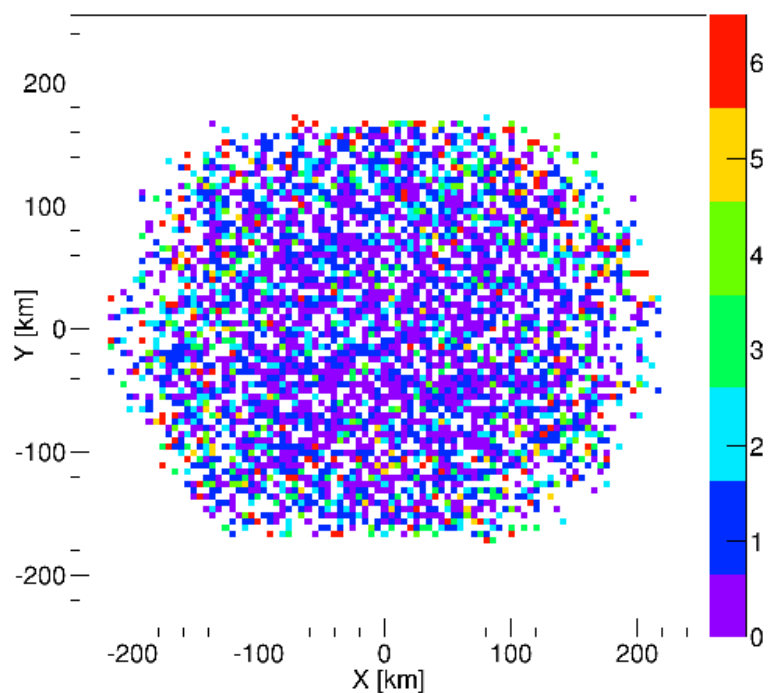
Obr. 4–1: Percentuálny podiel počtu zrekonštruovaných spršok z celkového počtu použiteľných simulovaných spršok.

⁹Uhly sú približné, keďže použitý generátor spršok `SLastShowerGenerator` nedovoľuje rovnakú maximálnu aj minimálnu hodnotu zenitového uhla θ .



Obr. 4–2: Štatistika γ_{68} rekonštrukcie zenitového uhla pri použití algoritmu PWISE pre uhly 30 až 75 stupňov, zvyšujúce sa po 5 stupňov. Pri výpočte boli použité udalosti z celej ohniskovej plochy.

PWISE: závislosť γ a pozície maxima spŕšky (všetky zenitové uhly)



Obr. 4–3: Závislosť pozície maxima spŕšky a chyby γ pre algoritmus PWISE.

4.2.3 Metóda hľadania cesty

Algoritmus a implementácia v systéme ESAF (označený ako `RobustModule`), ktorých autorom je S. Biktermanova (Biktemerova et al., 2013) vytvára množinu možných kandidátov cesty (sekvencia pixelov zoradená v čase a ležiaca v rovnakom smere), z ktorých je najlepšia vybraná. Pre tvorbu cesty algoritmus používa princípy Kalmanového filtra (Kalman, 1960)

Metóda hľadania cesty (*ang. Track Finding Method*) pre každú snímku (množina pixelov s rovnakým GTU) vyberá pixely s vysokou hodnotou. Potom sa algoritmus pokúša prepojiť všetky možné páry pixelov medzi dvoma snímkami do segmentov cesty. Teda, pokúša sa prepojiť všetky páry bodov, ktoré spĺňajú kritéria vzdialenosti, trvania a deviácie z čiary cesty. Ak bod spĺňa všetky kritéria je pridaný do cesty. Ak cesta obsahuje aspoň dva pixely, je interpolovaná s čiarou v každom kroku. Táto čiara je používaná v kritériu deviácie z čiary cesty. Kandidáti cesty nie sú vyberaní len z dvoch za sebou nasledujúcich momentiek, ale môžu byť vybraní z viacerých predchádzajúcich GTU (odporúča sa aspoň 5). Vychádza sa z predpokladu, že pixely pozadia sú náhodne distribuované, preto pravdepodobnosť nájdenia cesty sa s jej zvyšujúcou dĺžkou znižuje. Napriek tomu, pixel pozadia môže byť občas pridaný do cesty signálu, a tak narušiť túto cestu. Problém môže byť vyriešený skopírovaním cesty pred pridaním bodu - vzniknú dve cesty, rekonštrukcia môže pokračovať aj po pridaní nevhodne zarovnaného pixela. Na konci tohoto spracovania je veľká množina ciest, ktoré sú ale väčšinou krátke. Tvoria ich pixely šumu, ktoré sa náhodne usporiadali do cesty. Krátke cesty sú aj tie, ktoré boli pokazené započítaním pixelu šumu do rekonštruovanej cesty. Cesta s najväčšou sumou hodnôt pixelov je vybraná ako zrekonštruovaná.

5 UV pozadie Zeme

UV pozadie pozorované z orbity Zeme v situácii, keď je Mesiac za horizontom, sa skladá z troch základných zložiek: nočná žiara (preklad z anglického airglow) vrchnej vrstvy atmosféry, zodiakálne svetlo (niekedy nazývané aj protisvit) a integrované svetlo hviezd. Z týchto troch zložiek má v bezoblačných podmienkach nad oceánom najväčší podiel z celkového množstva svetla pri pozorovaní z orbity v nadir móde nočná žiara (Bobík, 2013).

Tvorba modelov UV pozadia je založená na kombinácii rôznych faktorov, z ktorých sa pozadie skladá. Hodnoty vychádzajú z experimentálnych meraní a fyzikálnych modelov (Leinert et al., 1998). Podstatným sledovaným faktorom je aj vlnová dĺžka sledovaných svetelných emisií. Úroveň nočnej žiary (svetelnej emisie atmosféry) sa výrazne mení s časom (závislosť od lokálneho času, sezóny a slnečnej aktivity) a geomagnetickou zemepisnou šírkou. Ďalším faktorom, ktorý je potrebné v modeloch zahrnúť je svetelné znečistenie spôsobené osvetlením miest, čo je opäť faktor závislý od geografickej pozície sledovaného miesta. Zodiakálne svetlo je spôsobené rozptylom slnečného svetla medziplanetárnymi časticami prachu a je modelované funkciou závislosti smeru pozorovania, vlnovej dĺžky a vzdialenosti od slnka. Integrované svetlo hviezd je závislé od jasnosti hviezd, ktorý závisí od galaktickej šírky¹⁰.

5.1 Simulácia UV pozadia

Je cieľom, aby triggerovací mechanizmus v UV pozadí neznamenal náhodné čiary pripomínajúce spŕšky, aby miera falošných triggerov (FTR - Fake Trigger Rates) bola nízka. Typicky sa pozadie simuluje na základe Poissonovej distribúcie. Pre gene-

¹⁰Galaktický koordinačný systém sú súradnice súradnicového systému založeného na rovine Galaxie a jej stredom je poloha Slnka.

rovanie pozadia za účelom jeho následnej analýzy boli urobené samostatné analýzy mimo softvérového rámca ESAF (Biktemerova et al., 2013). Pozadie bolo generované a následne spracovávané implementovaným spúšťačím mechanizmom - PTT a LTT spúšťačmi. Simulované bolo jedno PDM. Pozadie generované na základe Poissonovej distribúcie s priemerom 2.1 fotónov/pixel/GTU pre fotomultiplikátor M36. Falošné spršky zachytené spúšťačmi sú uložené v textových súboroch. Pre LTT spúšťač, ktorého výstup je pre túto diplomovú prácu zaujímavý, súbor má údaje organizované ako tabuľku so stĺpcami x, y, čas (GTU) a počet zaznamenaných fotoelektrónov (Pastirčák et al., 2012).

Tieto súbory sú generované vzhľadom na rozličné nastavenie. Vo všeobecnosti možno urobiť jednoduchý výpočet pre zistenie počtu riadkov jednej zaznamenatej udalosti $N_{lines} = L_{side}^2 \times N_{gtu}$, kde L_{side} je dĺžka strany PDM (počet pixelov), N_{gtu} je počet GTU zaznamenaných pre udalosť. V prípade, ktorý bude v ďalšej časti práce spracovávaný, je strana PDM 48 pixelov (3 EC = 6 PMT so stranou 8 pixelov) a trvanie udalosti spršky N_{gtu} je 256+1 GTU. Použitím týchto hodnôt dostaneme $48^2 \times 257 = 592128$. Oddelovačom medzi hodnotami sú dve znaky medzery, čo je vskutku zbytočné a zväčšuje výstupný súbor. Jeden riadok súboru (aj so znakom nového riadka má veľkosť 11 až 14 bajtov. Spočítanie počtu bajtov pre udalosť dáva výsledok 8381760 bajtov (približne 8MB) z čoho zbytočné medzery tvoria 1776384 (približne 1,7MB). Celkové veľkosti súborov sú v desiatkach gigabajtov.

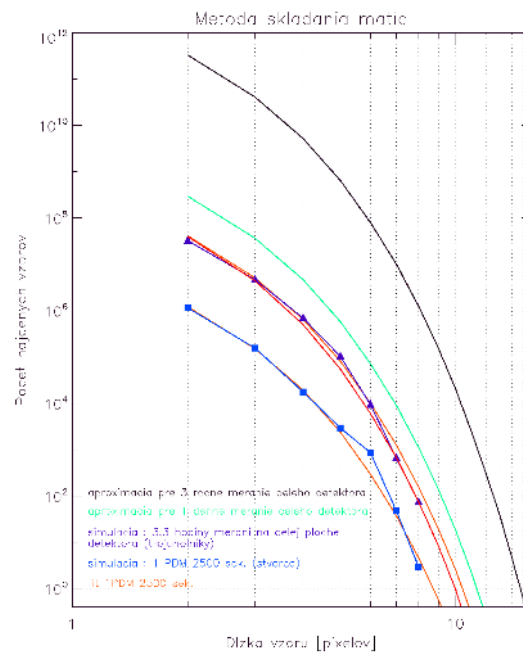
5.2 Hľadanie vzorov v simulácií UV pozadia

Mimo systému ESAF boli v simulácii pozadia už hľadané vzory. Hľadanie vzorov bolo realizované použitím Houghovej transformácie navrhnutej v diplomovej práci Miroslava Staroňa (Staroň, 2013). V spomenutej práci boli analyzované statické „snímky“ UV pozadia (v kombinácii s vneseným šumom - vplyvom detektora), ale najmä aj za sebou nasledujúce snímky vzdialené od seba čas jedného GTU (prezentované aj

v už spomenutom článku (Biktemerova et al., 2013)), ktoré viac vystihujú reálnu situáciu. Analýza za sebou nasledujúcich snímok bola vykonaná v prvom kroku skladaním snímok do jednej, ktorá obsahovala čiaru - sprišku, v druhom nájdením skonštruovanej čiary algoritmom Houghovej transformácie. Samozrejme, druhý krok úspešne nájde čiaru len v tom prípade, že sa čiaru podarilo skonštruovať v prvom kroku. Spájané snímky pochádzajú z výstupu LTT spúšťača prepúšťajúceho snímok, v ktorom bol aktivovaný, 15 predchádzajúcich a 15 nasledujúcich. Proces spájania snímok vychádza z transformácie geometrie sprišky v atmosfére na ohniskovú plochu detektora, vytvárajúc tabuľku so stĺpcami GTU a X . GTU bolo asociované so snímkom (maticou) a X so stĺpcom matice.

Pre za sebou nasledujúce snímky bola vypracovaná štatistika dĺžok nájdených sprišok v závislosti na počte vzorov pre rôzne dĺžky meraní. Túto závislosť možno aproximovať štatisticky motivovanou funkciou 5.1, kde N_p je počet nájdených vzorov, L_p je dĺžka vzoru a N_{px} je počet možných hodnôt jedného pixelu. Táto aproximácia konzervatívne odhaduje množstvo vzorov pre dlhšie vzory pri rôznych veľkostiach štatistiky. Obrázok 5–1 ilustruje túto závislosť pre rôzne veľkosti štatistiky, teda pre rôzne dlhé časové obdobia, z ktorých sú analyzované.

$$N_p(L_p) \sim \left(\frac{1}{N_{px}}\right)^{L_p} \quad (5.1)$$



Obr. 5–1: Závislosť počtu vzorov a dĺžok nájdených vzorov pre rôzne časové obdobia (Staroň, 2013).

6 Implementácia algoritmu Houghovej transformácie

Tento algoritmus bol implementovaný vo viacerých verziách so zvyšujúcou sa zložitou, stavajúc na základne jednoduchej Houghovej transformácie. Cieľom je implementovať metódu rozpoznávania vzorov, ktorá bude ponúkať lepšie výsledky ako aktuálne používaná metóda PWISE - aspoň v niektorých špecifických zenitových uhloch. Hlavným autorom algoritmov je Bc. Jozef Vasilko, ktorý na nich pracuje v rámci svojej diplomovej práce (Vasilko, 2015). V tejto práci sú algoritmy opísané vo forme ako boli implementované v ESAF-e.

Poskytnuté algoritmy boli implementované ako separátne aplikácie naprogramované v jazyku Java a MATLAB (prvé verzie) používajúce rozdielne údajové štruktúry, ktoré sú nevhodné pre implementáciu v softvérovom rámci ESAF a jazyku C++. Stanoveným cieľom bolo implementovať tieto algoritmy s ohľadom na minimalizovanie časových a pamäťových nákladov. Požiadavkou je, aby bol modul navrhnutý pre prácu v súčinnosti s modulmi `TrackDirectionModule2` a `PmtToShowerReco`, keďže tieto moduly sú aktuálne využívané pre rekonštrukciu s metódou PWISE (modul `PWISEModule`).

6.1 Základný princíp algoritmu

Každá priamku v obrazovom priestore je popísateľná dvoma parametrami (a, b) rovnice lineárnej funkcie $y = ax + b$. Priestor parametrov je teda definovaný parametrickou reprezentáciou popisujúcou čiary v obrazovom priestore na základe sklonu a posunutia. Problémom takejto reprezentácie je, že obe tieto parametre sú neobmedzené a nedovoľujú opis vertikálnej priamky. Použitie polárnych súradníc uvedené v rovnici je riešením, opisujúc priamku na základe uhla normály priamky a jej kol-

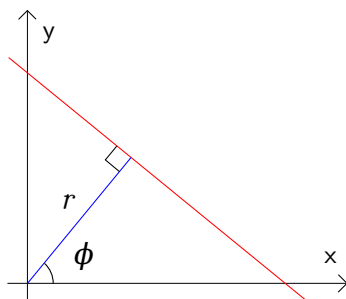
mej vzdialenosti od stredu súradnicovej sústavy (Duda et al., 1972). Každéj priamke je možné priradiť jedinečný bod (r, ϕ) parametrického priestoru práve vtedy, keď $\phi \in [0, \pi) \wedge r \in R$ alebo $\phi \in [0, 2\pi) \wedge r \geq 0$.

$$y = -\left(\frac{\cos\phi}{\sin\phi}\right)x + \frac{r}{\sin\phi} \quad (6.1)$$

Po úprave rovnice sa kolmá vzdialenosť r vypočíta pomocou rovnice 6.2.

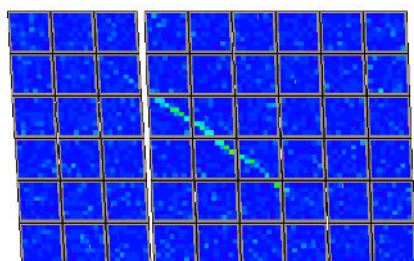
$$r = x \sin \phi + y \cos \phi \quad (6.2)$$

Obrázok 6–1 ilustruje parametrizáciu používanú pri Houghovej transformácii a obrázok 6–2 ilustruje vizualizáciu parametrického priestoru pri spracovaní spŕšky na obrázku 6–2a (zobrazené sú maximálne počty fotoelektrónov pixelov)¹¹.

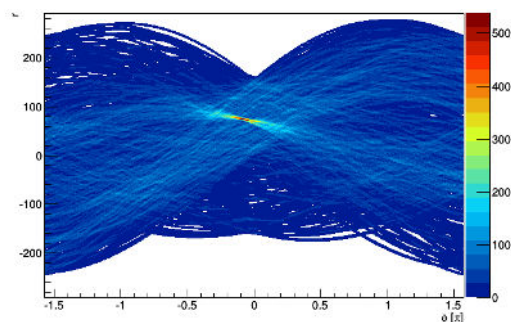


Obr. 6–1: Základný princíp popisu priamky v Houghovej transformácii.

¹¹Konkrétne, je to Houghova transformácia v prvom volaní `HoughLine` prvej verzie algoritmu - pozri časť 6.6.



(a) Spřška pred transformáciou.



(b) Houghova transformácia.

Obr. 6 – 2: Houghova transformácia spřšky.

Všeobecný algoritmus hľadania čiar na základe Houghovej transformácie je možné popísať následovne (Vasilko, 2015):

1. Nech je vstupom binárna matica $M \times N$, pričom body hrán nadobúdajú hodnotu 1 a zvyšné body 0. Nech r_d, ϕ_d sú vektory, ktoré budú obsahovať intervaly diskretizovaného parametrického priestoru $r \in [0, \sqrt{N^2 + M^2}], \phi \in [0, 2\pi)$.
2. Diskretizuj parametrický priestor, použijúc vhodné vzorkovanie s krokom $\delta r, \delta \phi$, na počet R a T intervalov.
3. Nech $A(R, T)$ je počítadlom bodov nájdených vo vstupnej matici pre jednotlivé priamky.
4. Pre každý bod $E(x, y) = 1$ a $h = 1, \dots, T$
 - $r = x \sin \phi_d(h) + y \cos \phi_d(h)$
 - nájdi taký index k , aby hodnota r_k bola čo najbližšia k r
 - inkrementuj $A(k, h)$
5. Podľa definovanej podmienky pre minimálny počet bodov rozhodni o existencii

priamok.

6.2 Vstupné a výstupné údaje modulu rekonštrukcie vzoru

Vstup je konfigurácia a pixely prednej elektroniky detektora, ktoré majú počet zaznamenaných fotónov viac ako nula. Údaje sú v module rekonštrukcie vzoru získane z objektu udalosti `RecoEvent` po jednom pomocou metódy `RecoEvent::GetRecoPixelData(Int_t)` alebo získaním referencie na celý zoznam pixelov pomocou `RecoEvent::GetRecoPixels()`. Získaný pixel je reprezentovaný triedou `RecoPixelData`.

PatternRecognition → **SelectedPixels** Smerník na údaje je zapísaný do „dátového“ objektu triedy `RecoGlobalData` prístupného pod kľúčom `PatternRecognition`, v ktorom sú uložené pod kľúčom `SelectedPixels`. Dáta sú objekt generickej triedy `vector<Int_t>`, kde členy sekvencie sú indexy, pomocou ktorých je možné získať údaje pixelu v danom GTU (trieda `RecoPixelData`) volaním metódy `RecoEvent::GetRecoPixelData(Int_t px_id)` objektu triedy `RecoEvent` (objektu udalosti). Výpis 4 ukazuje zjednodušený príklad získania týchto údajov (`fEv` je smerník na objekt triedy `RecoEvent`). Tieto údaje sú v procese rekonštrukcie využívané modulom `RecoTrackDirection2`.

Výpis 4: Získanie vybraných pixelov z globálnych údajov.

```
RecoGlobalData* gdPattReco = fEv->FindGlobalData("
    PatternRecognition");
(vector<Int_t>*) vector<Int_t> * selectedPixels = gdPattReco->
    GetObj("SelectedPixels");
```

Dáta modulu - CluPixels, Clusters `CluPixels` je smerník na rovnaký objekt ako už spomínaný `SelectedPixels`. `Clusters` je smerník na sekvenciu `vector<Int_t>`

`EusoCluster*>`, ktorá v aktuálnej verzii vždy obsahuje len jeden prvok - smerník na objekt triedy `EusoCluster` s rovnakými údajmi ako predošlé spomenuté. V aktuálne používaných moduloch (`TrackDirection2Module`, `PmtToShowerReco`) sa tieto údaje nevyužívajú.

Údaje zapísané do stromu objektu triedy `RecoRootEvent` Do výstupného súboru, ktorý údaje získané rekonštrukciou sprístupňuje pomocou stromu s koreňom objektu triedy `RecoRootEvent`, sa zapisujú štatistické informácie o rekonštrukcii. Cieľom je objekt triedy `RecoCulstering`.

- **Počet všetkých rozpoznaných pixelov** je počet prvkov zoznamu *SelectedPixels*. Získanie hodnoty: `fRecoRootEvent->GetRecoClustering().GetNumPoints()`.
- **Kontaminácia** je pomer množstva chybné rozpoznaných pixelov (čistý šum) ku počtu všetkých rozpoznaných pixelov (*SelectedPixels*) Získanie hodnoty: `fRecoRootEvent->GetRecoClustering().GetContamination()`.
- Pomer sumy vybraných skutočných signálov k sume všetkých skutočných signálov. Získanie hodnoty: `fRecoRootEvent->GetRecoClustering().GetSignalFraction()`.
- Pomer sumy vybraných skutočných signálov k sume všetkých skutočných, ktorých pixel mal celkový počet väčší ako limit nastavený v konfigurácii parametrom `KeHoughModule.fCountsThreshold`. Získanie hodnoty: `fRecoRootEvent->GetRecoClustering().GetSigFracOverThresh()`.

6.3 Prepis programov do jazyka C++

Ako bolo spomenuté, algoritmy boli implementované J. Vasilkom ako samostatná aplikácia. Na začiatku vývoja bol použitý jazyk MATLAB, neskôr bol použitý programovací jazyk Java, v ktorom boli implementované všetky nasledujúce verzie.

Faktory vyžadujúce použitie rozdielnych postupov ako pre prepis programu do jazyka C++ v softvérovom rámci ESAF sú:

- Na rozdiel od programovacieho jazyka Java *Garbage collector* v C++ nie je integrovaný, preto je potrebné spravovať dynamicky alokovanú pamäť, resp. priamo dynamicky nealokovať pamäť.
- Integrácia do systému ESAF určuje požiadavky na praktickosť použitia údajových typov - snaha používať existujúce dátové štruktúry a nevytvárať zbytočné kópie údajov.
- Vstupné údaje sú striktne len pre čítanie.
- Rozdielne rozhranie údajových štruktúr základnej knižnice.

Teoretickým cieľom úlohy je konštrukcia sémanticky ekvivalentného programu, ktorý má rovnakú denotáciu ako pôvodný program. Definícia denotácie je: „*Denotácia programu je výsledok vykonania programu bez ohľadu na to, v akom poradí a akým spôsobom sa vykonávajú jeho zložky v počítači*” (Steingartner, 2013). Porovnávanie implementácii bolo preto vykonané pomocou porovnávania výstupných údajov pri rovnakých vstupných.

Kontrola identickosti výsledkov programov bola vykonávaná pomocou ladiacich programov a kontrolných výpisov hodnôt premenných. A aj ak sa nejaká časť programu líšila algoritmicky, porovnané boli vstupy a výstupy časti. V prípade *Java* aplikácie bol ladiaci nástroj *jdb*, využitím rozhrania vývojového prostredia *NetBeans*. Vývoj

algoritmov pre ESAF bol vykonávaný pomocou vývojového prostredia *Eclipse CDT* a ako ladiaci nástroj bol využitý *gdb*¹². Pre umožnenie rýchlej kontroly výsledkov rekonštrukcie vzoru, najmä kontroly vybraných bodov, bol vytvorený modul rekonštrukcie `SelectedPixelsDebugModule`¹³.

6.4 „Nultá” verzia

„Nultá” verzia je jednoduchá, aplikujúca základný algoritmus na vstupné body. Program, z ktorého sa vychádzalo pri tejto implementácii bol napísaný v jazyku MATLAB. Keďže sa jednalo o prvé pokusy, podobnosť implementácií bola podstatná pre lepšie odhalenie problémov.

Záznam jedného pixela je v tomto programe reprezentovaný jedným riadkom matice, v ktorej stĺpce sú: pozícia x , pozícia y , hodnota pixela, GTU. V implementácii v module rekonštrukcie túto údajovú štruktúru reprezentuje trieda `RecoPixelData`, ktorá obsahuje všetky údaje daného pixela. V algoritme je unikátnosť pixela určená na základe jeho pozícií na osi x a y . V module rekonštrukcie je unikátnosť riešená na základe identifikátora pixela.

Efektivitu tejto implementácie znižuje napríklad zbytočné prechádzanie všetkých vzdialeností r pri napĺňaní akumuláčnej matice. Mierne zefektívnenia je možné dosiahnuť zlúčením tiel cyklov. Napríklad hľadanie maximálnej a minimálnej hodnoty v jednom prechode namiesto dvoch separátnych alebo hľadanie maximálnej hodnoty v akumuláčnej matici už počas pridávania - nie v samotnom prechode. Implementácia akumuláčnej matice pomocou hašovacích tabuliek potenciálne znižuje pamäťovú náročnosť - ukladanie len platných hodnôt, na druhej strane môže zvyšovať časovú náročnosť algoritmu, keďže hľadanie v hašovacej tabuľke je $O(\log n)$ operácia.

¹²Mierne rozdiely v hodnotách premenných typu dvojitej presnosti - `double` možno priradiť k mierne rozdielnym vstupným hodnotám spôsobených obmedzenou presnosťou výstupných hodnôt programu `MvEventPrinter` (pozri používateľskú príručku).

¹³Modul je opísaný v používateľskej príručke.

6.4.1 Algoritmus

Princíp algoritmu je podobný ako uvedený algoritmus Houghovej transformácie, s tým rozdielom, že sa spracovávajú pixely vo viacerých GTU, teda rovnaký pixel môže byť spracovaný viac krát. Základnou konfiguráciou programu je voľba prahu počtu zaznamenaných fotoelektrónov (*threshold*) na spracovávaných pixeloch, pre spracovanie sa vyberajú iba pixely s hodnotou nad touto hranicou. Druhým podstatným parametrom je voľba veľkosti jednotky dĺžky vzdialenosti r , označovaná dr . Podstatná je aj voľba počtu uhlov, ktoré sa budú overovať - ideálne by táto hodnota nemala byť veľmi vysoká z dôvodu urýchlenia spracovania, ale ani nízka, aby sa nestalo že spíška bude nerozpoznaná z dôvodu netestovania pre daný uhol.

1. **Posun pozícií pixelov do prvého kvadrantu** s cieľom zníženia rozsahu uhla pri Houghovej transformácii - posunuté pozície pixelov je potrebné uchovať do separátnej údajovej štruktúry s náhodným prístupom, čo je mapa využívajúca hašovaciu tabuľku, keďže kľúčom by bol identifikátor pixela (spracováva sa rozsah identifikátorov pixelov patriacich PDM, ktoré boli zvolené na základe predpokladu, že obsahujú záznam spíšky)
2. **Aplikácia Houghovej transformácie** na body nad prahom počtu zaznamenaných fotónov.
3. Výber všetkých pixelov (vo všetkých GTU) zo vstupných údajov, ktoré sú nad prahom počtu zaznamenaných fotónov. Výsledný zoznam je výsledkom rekonštrukcie vzoru

Výber bodov na základe Houghovej transformácie

1. Naplnenie zoradenej množiny (poľa) uhlov ϕ na základe vstupného parametra počtu uhlov. Označenie množiny *PHI_ARRAY*.

2. Naplnenie zoradenej množiny (poľa) vzdialeností r od stredu súradnicovej sústavy na základe rozsahu pozícií (maximálna vzdialenosť dvoch bodov) a minimálneho kroku δr . Označenie množiny R_ARRAY .
3. Konštrukcia akumuláčnej matice A reprezentujúcej Houghov priestor (v tejto verzii implementované ako hašovacia tabuľka obsahujúca hašovacie tabuľky). Hodnota a_{max} nastavená na 0 reprezentuje maximálnu hodnotu v akumuláčnej matici.
 - 3.1. Pre každý bod p zo vstupnej množiny bodov a pre každý uhol $\phi_i \in PHI_ARRAY$:
 - 3.1.1. Výpočet hodnoty r_1 , $r_1 = p.x \cos \phi + p.y \sin \phi$, kde $p.x, p.y$ sú X a Y pozície aktuálneho pixela, ϕ je aktuálny uhol:
 - 3.1.2. Ak $r \geq 0$: Pre každú dĺžku $r_2 \in R_ARRAY$: Ak $|r_1 - r_2| \leq dr$ kde r_1 je vypočítaná hodnota a r_2 je hodnota z poľa r_array , dr je parameter z konfigurácie `KeHoughModule.fDr`.
 - 3.1.2.1. Inkrementovanie hodnoty v akumuláčnej matici.
 - 3.1.2.2. Kontrola maximálnej hodnoty a_{max} v akumuláčnej matici.
4. Zadefinovanie množiny pixelov *HITS* typu - položky sú identifikátory pixelov na detektore.

6.4.2 Vylepšenie úspešnosti rekonštrukcie

Zlepšenie tejto metódy je možné dosiahnuť pridaním ďalšieho behu Houghovej transformácie, kde spracovávané parametre nie sú pozície osí x a y , ale kombinácia údajov jednej osi s časovým údajom, a potom prípadne kombinácia údajov druhej osi s časovým údajom. Táto metóda odstráni z rekonštruovanej sprišky body, ktoré síce boli

vybrané z pohľadu pozície x a y , ale nepatria do lineárne narastajúcej funkcie (rovná čiara), ktorou možno opísať závislosť údajov z osi x alebo y a času reprezentovaného pomocou GTU daného bodu. Pri tomto vylepšení sa do istej miery predpokladá, že rýchlosť pohybu bodu je približne konštantná. Ináč povedané, táto metóda nahradí jednu os časom, aplikuje na tieto údaje Houghovu transformáciu a vyberie body na maximálnej priamke - body nepatriace do tejto priamky sú nezapočítané - sú to body ležiace mimo hlavného pohybu bodov vo vstupných údajoch, teda predpokladá sa, že nepatria do spířsky, ktorá je lokalizovaná na jednej čiare. Po výbere bodov Houghovou transformáciou s údajmi z jednej z osí a GTU, možno aplikovať túto metódu aj na údaje z druhej z osí a GTU.

Zvolená metóda pre implementáciu pozostáva z dvoch behov Houghovej transformácie. Os použitá v druhom behu je zvolená na základe priemerného uhla ϕ vybraných čiar prvým behom transformácie. Uhly ϕ blízke 90 stupňov - vybrané čiary sú orientované najmä pozdĺž osi x , ináč sa použije os y .

Implementácia vylepšenia vyžaduje zmeny v „nultej“ verzii:

- V prípade druhého behu je potrebné vybrať pixely vo všetkých časoch, preto je potrebné vytvoriť novú vstupnú množinu pixelov.
- Ukladanie uzlov ϕ vybraných čiar.

6.5 Nové algoritmy hľadania čiar použitím Houghovej transformácie

V neskôr opísaných dvoch finálnych verziách algoritmov sa používajú tri verzie algoritmu Houghovej transformácie nazvané *HoughLine*, *HoughLine3D*, *HoughPlane*. Principiálne vychádzajú z už opísanej verzie, ale upravujú ju s cieľom dosiahnutia väčšej efektivity a lepšieho výsledku. Hlavnou myšlienkou posledných dvoch je už

opísaná snaha hľadať čiary na rovine určenej časom a jednou z osí.

Vstupom týchto funkcií sú rozsahy a minimálny krok pre kolmú vzdialenosť a uhly, taktiež aj „hrúbka” čiary, určujúca v akom rozsahu strednej vzdialenosti bude bunkám zvyšovaná hodnota. Výstupom funkcií je množina indexov vybraných pixelov, maximálna a minimálna kolmá vzdialenosť, maximálny a minimálny uhol (uhly).

6.5.1 Funkcia `HoughLine`

Vstupom tejto funkcie je zoznam pixelov, rozsah vzdialeností r , minimálny krok vzdialenosti, rozsah uhlov ϕ , minimálny uhol ϕ a dĺžka $size$, ktorá určuje rozsah rozdielov vzdialeností r , ktoré patria rovnakej priamke, teda zjednodušene „hrúbku” nájdenej čiary. Implementácia algoritmu funkcie (napríklad v metóde `HeHoughModuleV2`) taktiež umožňuje nastavenie funkcie pre získanie pozície X a Y, čo bude podstatné neskôr v algoritme *HoughLine3D*.

Algoritmus možno rozdeliť na dve základné časti: 1. *Napĺňanie akumuláčnej matice*, 2. *Výber priamok z akumuláčnej matice*. Rozmery akumuláčnej matice sú vypočítavané zo vstupných parametrov pre rozsah a minimálny krok. Základ výpočtu vychádza z rovnice $\lfloor |r_{koniec} - r_{začiatok}| / r_{krok} \rfloor$, konkrétny výpočet je ale ovplyvnený viacerými parametrami - napr. ošetruje sa prípad keď je $r_{začiatok} \leq r_{koniec}$, detailnejší popis je možné nájsť vo Vasilkovom opise algoritmu (Vasilko, 2015).

Napĺňanie akumuláčnej matice prebieha podobne ako v už opísanej verzii, s rozdielom, že indexy vzdialeností r sú vypočítané z hodnoty. V rozsahu medzi týmito vypočítanými indexmi akumuláčnej matice je potom bunkám pripočítaná hodnota počtu fotoelektrónov daného pixela, teda hodnota nie je len inkrementovaná o hodnotu 1 ako v predošlej verzii. Týmto spôsobom sa vybraným pixelom priradzuje váha, čo je využívané pridávaním korekcií počtov fotoelektrónov v neskoršom opise algoritmu. Pri zvyšovaní hodnoty sa tiež kontroluje nastavená hodnota bunky, či

nie je maximálna. V konkrétnej implementácii v jazyku C++ je akumulálna matica reprezentovaná objektom triedy `vector<Int_t>` a indexy buniek sú vypočítavané klasickým výpočtom $bunka = počet\ buniek\ v\ stĺpci \times riadok + stĺpec$, kde stĺpce reprezentujú indexy vzdialenosti r a riadky indexy uhlov ϕ . Samotné hodnoty r s ϕ sú v samostatných poliach. Vizualizácia tejto matice už je ukázaná na obrázku 6–2b.

Druhý krok funkcie *HoughLine* je výber čiar z akumuláčnej matice. V tomto kroku sa vyhľadávajú v matici bunky s už určenou maximálnou hodnotou a určuje sa maximálny a minimálny uhol a vzdialenosť nájdených priamok (v prípade, že maximálna hodnota je vo viacerých bunkách). Pre túto bunku sú potom prehladané všetky spracovávané pixely, pre ktoré je vypočítaná vzdialenosť r porovnaná s r danej bunky. A vybraný pixel je pridaný do výstupnej množiny. Tento krok by mohol byť odstránený, ak by sa pixel pridával hneď do zoznamu (alebo množiny) pixelov pri napĺňaní akumuláčnej matice. V takomto prípade by tento zoznam bol vždy vyprázdnený, ak by sa našla nová maximálna hodnota bunky. Nevýhodou takéhoto alternatívneho riešenia je viac zbytočných zápisov do pamäte.

6.5.2 Funkcia *HoughLine3D*

Princíp tejto funkcie spočíva v dvojnásobnom volaní funkcie *HoughLine*. Ako prvé je táto funkcia volaná na vstupných dátach a funkcia pre získanie koordinátu pozície pixela na osi y je nahradená funkciou pre získanie koordinátu z , ktorý je v skutočnosti GTU pixela. Ak nebol rozsah vzdialeností r pre druhý beh zadáný je vypočítaný nájdením maximálnej vzdialenosti koordinátov x a z . Následne je volaný druhý beh funkcie *HoughLine* so spomenutým rozsahom. A tu je zas koordinát x nahradený koordinátom z . A výsledok tejto funkcie *HoughLine* je potom výsledok *HoughLine3D*. Zjednodušene povedané funkcia *HoughLine3D* nájde čiaru na rovine x, GTU a na rovine y, GTU a výsledok funkcie *HoughLine3D* je prienik výsledkov z týchto rovín.

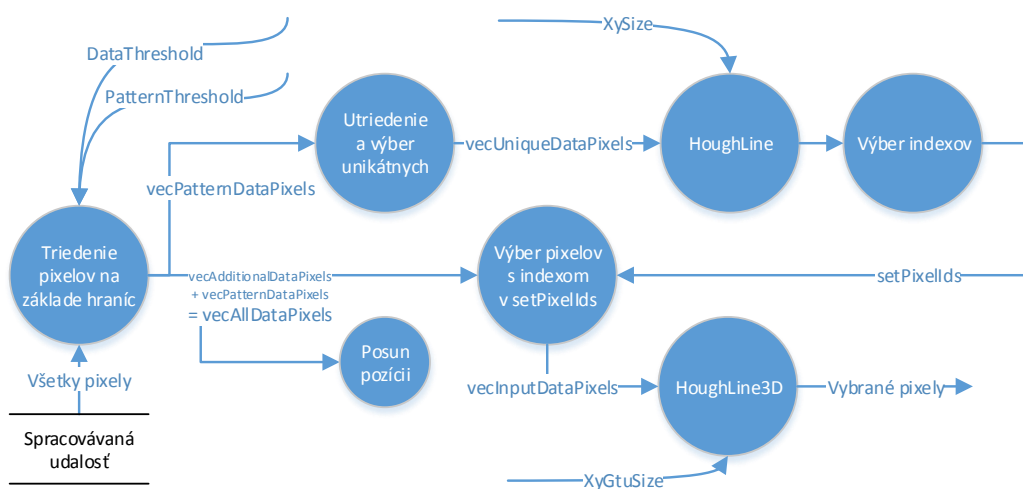
6.5.3 Funkcia *HoughPlane*

Táto funkcia je veľmi podobná funkcii *HoughLine*, rozdiel je, že výpočty sa vykonávajú pre priamku v 3D priestore - výpočty sú rozšírené o uhol θ . Rovnica 6.2 pre výpočet kolmej vzdialenosti je zmenená na rovnicu 6.3 a do všetkých cyklov je vnorený ďalší cyklus pre θ . Pridanie ďalšej dimenzie zvyšuje nielen časovú, ale aj pamäťovú náročnosť algoritmu, preto myšlienkou algoritmu opisovaného v ďalšej časti bolo znížiť počet pixlov, ktoré bude funkcia *HoughPlane* spracovávať.

$$r = i \sin \phi_d(h) \cos \theta_d(h) + j \sin \phi_d(h) \sin \theta_d(h) + k \cos \phi \quad (6.3)$$

6.6 Prvá verzia

Na základe princípu algoritmu *HoughLine3D* bola nanovo implementovaná jednoduchšia metóda. Táto metóda nebola stredobodom záujmu pri implementácii algoritmov a vychádzala najmä z časti kódu z druhej metódy. Obrázok 6 – 3 zjednodušene ilustruje kroky algoritmu a údaje medzi nimi vymieňané.



Obr. 6 – 3: Diagram prúdu dát - Prvá verzia algoritmu.

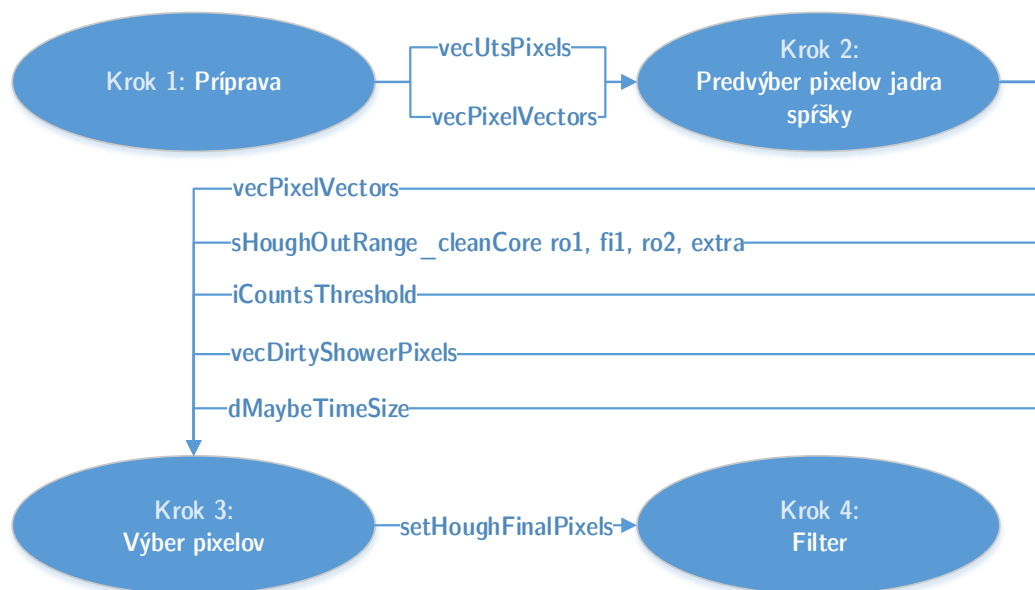
Zjednodušený opis algoritmu

1. Výber pixelov s počtom fotoelektrónov nad hranicou určenou konfiguračným parametrom `PatternThreshold` do zoznamu `vecPatternDataPixels`. Ak nie je nad touto hranicou, ale je nad hranicou určenou konfiguračným parametrom `DataThreshold`, výber do zoznamu `vecAllDataPixels`.
2. Zoznam `vecPatternDataPixels` je zoradený podľa identifikátora a počtu fotoelektrónov zostupne.
3. Pridanie `vecPatternDataPixels` a `vecAllDataPixels` do zoznamu `vecAllDataPixels`.
4. Pridanie unikátnych pixelov na základe identifikátora s najvyšším počtom fotoelektrónov pre daný pixel z `vecPatternDataPixels` do zoznamu `vecUniqueDataPixels`.
5. Posun pozícií pixelov v zozname `vecAllDataPixels` pre „vycentrovanie” spíšky v strede súradnicovej sústavy posunom pozícií vektorov v zozname `vecPixelVectors`.
6. Spracovanie pixelov zo zoznamu `vecUniqueDataPixels` pomocou funkcie *HoughLine* na rovine x, y .
7. Výber pixelov zo zoznamu `vecAllDataPixels`, ktorých identifikátory boli vo výsledku posledného behu funkcie *HoughLine*. Uchované v zozname `vecInputDataPixels`.
8. Spracovanie zoznamu `vecInputDataPixels` funkciou *HoughLine3D*.

6.7 Druhá verzia

Vo viacerých generáciách bola postupne navrhnutá druhá verzia algoritmu. Motiváciou bolo dosiahnutie lepších výsledkov rekonštrukcie. Metóda nie je založená čisto len na využití kombinácie čiar vybraných pomocou Houghovej transformácie, ale je rozšírená aj o iné postupy.

Na rozdiel od programovacieho jazyka Java, C++ je jazyk nižšej úrovne, pretože ponúka nižšiu úroveň abstrakcie inštrukčnej sady procesora. Programy sú prekladané do strojového kódu inštrukcii procesora a správa dynamicky pridelovanej pamäte musí byť súčasťou aplikácie. Tento fakt ovplyvňuje spôsob návrhu aplikácie. Preferuje sa priamo nepoužívať dynamickú alokáciu pamäte. Ak sa používa, je potrebné uchovať smerník na túto oblasť pamäte. Organizácia blokov implementovaného programu bola preto navrhnutá tak, aby sa nemusela priamo používať dynamická alokácia, ale tiež s cieľom minimalizácie pamäťových nárokov. Cieľom rozdelenia programu je aj oddelenie jeho logických častí, čo je obzvlášť praktické pri rozsiahlom zdrojovom kóde. Obrázok 6–4 ilustruje základné bloky programu a medzi nimi odovzdávané dáta.

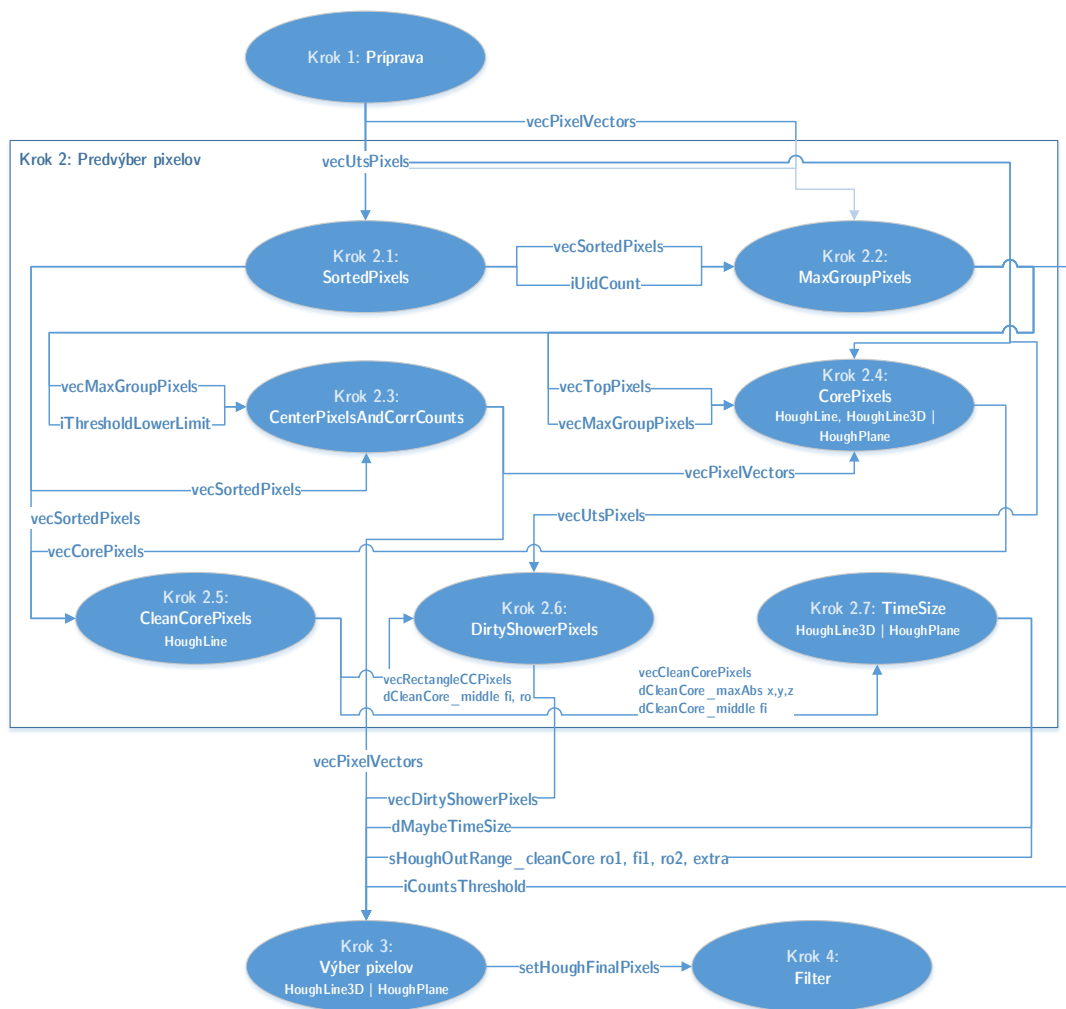


Obr. 6–4: Diagram prúdu dát - Základné bloky programu rekonštrukcie vzoru použitím Houghovej transformácie (druhá verzia).

Prvou časťou programu je príprava, kde sa prechádzajú všetky pixely spracovávanej udalosti a do zoznamu vektorov sa ukladá pozícia pixela. Taktiež sa inicializuje zoznam čísel pixelov (čísla od 0 po číslo posledného pixelu, pixely vo všetkých GTU), ktorý je následne zoradený podľa identifikátora pixela a počtu fotoelektrónov zoznamu. Zoradenie je realizované pomocou funkcie `std::sort`, teda používa sa algoritmus *introsort* s najhoršou zložitou $O(N \times \log_2 N)$ (Musser, 1997). V tejto metóde sa ukazuje aj jeden z rozdielov medzi implementáciami - pre referenciu pixela sa používa jeho číslo (poradové) v zozname všetkých pixelov (získaný pomocou `RecoEvent::GetRecoPixels()`). V Java verzii sa používa referencia na objekt poľa. K ukladaniu pozícií pixelov dochádza preto, aby sa neupravovali zdrojové údaje, ako je tomu v Java verzii. Výsledkom tohto kroku sú teda dva zoznamy - zoznam pozícií všetkých pixelov (`vecPixelVectors`) a zoznam čísel všetkých pixelov zoradený podľa identifikátora pixela a počtu fotoelektrónov (`vecUtsPixels`).

Nasledujúci blok v 6–4 nazvaný „2. Predvýber pixelov jadra spršky“ je najčlenitejší

z celého opisovaného procesu, preto je opísaný vo viacerých diagramoch. Obrázok 6–5 ilustruje prvú úroveň zanorenia.



Obr. 6 – 5: Diagram prúdu dát - Druhý blok programu rekonštrukcie vzoru použitím Houghovej transformácie (druhá verzia).

Krok 2.1 *SortedPixels* sa zakladá na triedení pixelov do skupín na základe počtu fotoelektrónov. Počet skupín je určený konfiguračným parametrom `NumDataLevel-Bins`. Pre každý na základe identifikátora (nie na základe času) unikátny pixel, je vybraný pixel s maximálnym počtom fotoelektrónov. Úlohu je možné riešiť jedným prechodom vďaka tomu, že sa pracuje so zoradenými údajmi z prvého kroku.

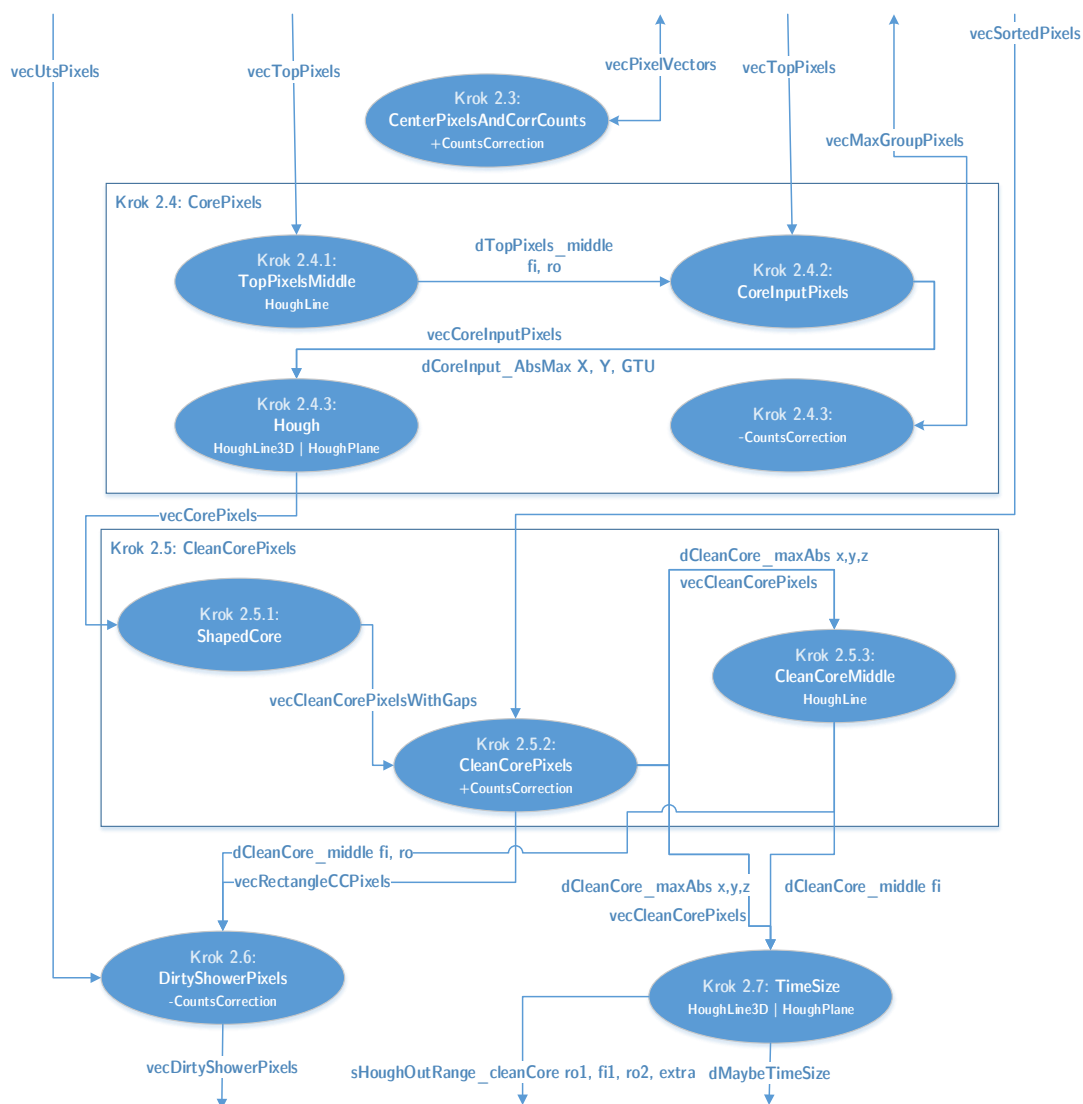
Výber maximálnej skupiny pixelov je vykonaný v kroku 2.2 *MaxGroupPixels*. Vytvára sa zoznam pixelov s maximálnymi počtami fotoelektrónov výberom určitého percentuálneho podielu pixelov nastaveného konfiguračným parametrom `PercentageDataLimit`. Po dosiahnutí tejto hranice je z tohto zoznamu vybraná maximálna skupina použitím funkcie *ShapedCore*. Ak maximálna skupina nebola vybraná a hranica počtu, ktorá sa každou touto iteráciou znižuje, je viac ako 0, proces sa opakuje.

Ako bolo spomenuté, funkcia *ShapedCore* vyberá maximálnu skupinu pixelov. Parametre ovplyvňujúce výber skupiny sú maximálna vzdialenosť medzi bodmi (`maxGap`) a minimálny počet pixelov skupiny (`minGroupSize`). V prvom kroku je inicializované pole (`bvecVisited`) s počtom prvkov rovnakému počtu prvkov spracovávaného zoznamu. Inicializuje sa aj štvorcová matica s riadkami a stĺpcami pre každý prvok zoznamu. Matica je v C++ implementácii zoznam typu `vector<Int_t>` s nastavenou dĺžkou druhej mocniny počtu prvkov spracovávaného zoznamu pixelov. Prvým krokom je naplnenie matice hodnotami `NEIGHBOURS`, pre blízke body a `DISTANT` pre vzdialené. Rovnaký bod je v matici nastavený na `DISTANT`. Konštrukcia matice umožní rozhodnúť pre každý bod vstupného zoznamu vzdialenosť k inému. Druhým krokom je prechádzanie matice po riadkoch a pre každý bod rekurzívne vyhľadanie množiny blízkych bodov realizované pomocou funkcie *GetNeighbours*. Navštívené body sú označené v poli `bvecVisited`, preto ak boli všetky prvky navštívené, hľadanie môže byť prerušené skôr. Funkcia *ShapedCore* teda identifikuje samostatné zhľuky pixelov vo vstupných dátach. Vybraná je skupina, ktorá má najväčší počet prvkov alebo, v prípade že viaceré majú rovnaký počet, je vybraná skupina s najväčšou sumou počtov fotoelektrónov.

Krok 2.3 *CenterPixelsAndCorrCounts* (z obrázku 6–5) spracováva výsledok predošlého kroku - zoznam pixelov s najväčším signálom `vecMaxGroupPixels` a hľadá maximálnu a minimálnu pozíciu nielen na osi x a y , ale aj na časovej osi reprezentovanej hodnotou GTU daného pixela. Následne sú z maxím a miním vypočítané stredy spŕšky na danej osi. Tieto hodnoty sú potom použité pre posunutie pozícií

pixelov (odpočítaním) v zozname pozícií všetkých pixelov spŕšky. Cieľom posúvania pozícií pixelov je pracovať s najnižšími vzdialenosťami pixelov od stredu súradnicovej sústavy (osi x , y , GTU). V tomto kroku sa taktiež vykonáva korekcia počtu fotoelektrónov pixelov v zozname `vecMaxGroup`.

Ďalší krok vyobrazený na obrázku 6–5 je *2.4 CorePixels*, ktorý podrobnejšie opisuje obrázok 6–6. Prvým pod-krokom kroku *2.4 CorePixels* je určenie parametrov priamky, na ktorej spŕška leží. Ako prvá je určená maximálna vzdialenosť od stredu súradnicovej sústavy určujúca rozsah vzdialeností použitej pri konštrukcii Houghovéhó priestoru. Následne sú body spracované algoritmom Houghovej transformácie - funkciou *HoughLine*, ktorej výsledok je minimálna a maximálna hodnota uhla a vzdialenosti z nájdených čiar. Spomenutá Houghova transformácia je vykonaná v rovine určenej osmi x a y . Zo spomenutých maximálnych a minimálnych hodnôt sú vypočítané priemerné hodnoty a toto je výsledok opisovaného kroku.



Obr. 6–6: Diagram prúdu dát - Bloky *CorePixels* a *CleanCorePixels* programu rekonštrukcie vzoru použitím Houghovej transformácie (druhá verzia).

Ďalším krokom na obrázku 6–6 je krok *2.4.2 CoreInputPixels* s cieľom výberu „obdĺžnikovej oblasti“ pixelov okolo priamky určenej v predchádzajúcom kroku. Šírka tejto oblasti je určená konfiguračným parametrom *CoreWidth*. V tomto kroku je v týchto vybraných bodoch hľadaná maximálna vzdialenosť od stredu súradnicovej sústavy na osiach x , y , GTU .

Posledné dva pod-kroky kroku 2.4 *CorePixels* sú aplikácia trojrozmernej Houghovej transformácie na pixely vybrané v predošlom kroku a odstránenie korekcie počtov fotoelektrónov na pixeloch maxima spŕšky aplikovanej v kroku 2.3 *CenterPixelsAndCorrCounts*. Použije sa funkcia *HoughPlane* alebo *HoughLine* v závislosti od konfigurácie.

Spracovanie pokračuje krokom 2.5 *CleanCorePixels*, kde je cieľom upresnenie výberu z predchádzajúceho bodu. Ako prvé sú zo zoznamu `vecCorePixels` vybraté blízke pixely (skupina s najvyšším počtom fotoelektrónov) funkciou `ShapedCore`, znova prefiltrované podmienkou vzdialenosti v pod-kroku 2.5.2 *CleanCorePixels* a uložené v zozname `vecCleanCorePixels`. Obe vzdialenosti v podmienkach sú konfigurovateľné. V tomto kroku (cykle) sa taktiež nájde maximálna vzdialenosť pixelov a na vybrané pixely sa aplikuje korekcia počtov fotoelektrónov. Podstatná časť, priamo neukázaná v obrázku, je vytvorenie zoznamu `vecRectangleCCPixels`, kde pixely sú vybrané zo všetkých unikátnych pixelov s maximálnym počtom fotoelektrónov, tie v obdĺžniku vymedzenom maximálnou a minimálnou pozíciou z `vecCleanCorePixels`. Zoznam unikátnych pixelov už bol vytvorený, je to `vecSortedPixels`. V pod-kroku 2.5.3 *CleanCoreMiddle* pomocou Houghovej transformácie funkciou *HoughLine* z `vecCleanCorePixels` určená stredná kolmá vzdialenosť r a stredný uhol ϕ .

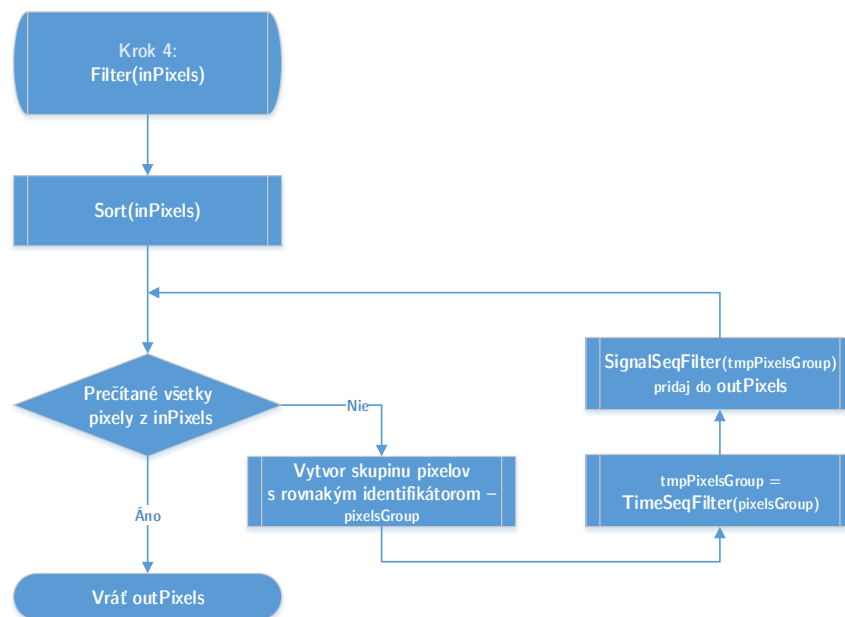
Cieľom kroku 2.6 *DirtyShowerPixels* je výber množiny tých identifikátorov pixelov zo zoznamu `vecRectangleCCPixels`, ktorých kolmá vzdialenosť vypočítavaná pri Houghovej transformácii je podobná strednej kolmej vzdialenosti z pod-kroku 2.5.3 *CleanCoreMiddle* pri výpočte so stredným uhlom z tohto pod-kroku. Maximálny rozdiel určuje konfiguračný parameter `CfgLinePatternSize`. Následne sú vybrané zo všetkých vstupných pixelov v zozname `vecUtsPixels` (z kroku 1. *Príprava*), zoradenom podľa identifikátora, pixely s identifikátormi nájdenými v tomto kroku. Vďaka tomu, že oboje sú rovnako zoradené vyfiltrovanie pixelov so správnym identifikátorom, je možné urobiť algoritmom podobným princípu algoritmu *sort-merge*

join.

Krok 2.7 *TimeSize* volá funkciu *HoughLine3D* alebo *HoughPlane* na údajoch zo zoznamu *vecCleanCorePixels* a z výsledku, ak je tak nakonfigurované parametrom *cfgTimeSizeEnabled*, je vypočítaná nová minimálna „hrúbka“ čiary pre finálny beh. Tým, že sa vychádza zo stredného uhla ϕ trojrozmernej houghovej transformácie je vypočítaná „hrúbka“ závislá od tretej - časovej dimenzie. Výstupom tohto kroku sú aj rozsahy uhlov a kolmých vzdialeností maximálnej priamky vybratých v Hough transformácii.

Finálne spracovanie použitím Houghovej transformácie nastáva v kroku 3. *Výber pixelov.* Vstupným zoznamom pixelov je *vecDirtyShowerPixels*, ktorý je podobne ako v kroku 2.1 *SortedPixels* rozdelený na pod-zoznamy na základe počtu fotoelektrónov pixela. Následne sú tieto pod-zoznamy postupne spracovávané trojrozmernou Houghovou transformáciou - *HoughLine3D* alebo *HoughPlane*. Rozsahy pre tieto transformácie sa postupne upravujú z výsledku predošlej transformácie. Začiatkový rozsah je výsledok z kroku 2.7 *TimeSize*. Znižovanie rozsahu umožňuje konštrukciu menších akumuláčnych matíc.

Úplne posledným krokom je aplikácia filtrov v *kroku 4*, ktorej princíp ukazuje vývojový diagram na obrázku 6–7. Ako prvé sú pixely zoradené podľa identifikátora. Potom sú skupiny pixelov s rovnakým identifikátorom spracované pomocou filtrov *TimeSeqFilter* a *SignalSeqFilter*. *TimeSeqFilter* vytvára zo vstupných pixelov skupiny, kde GTU pixelov za sebou postupne nasleduje bez medzery. Z týchto skupín je vybraná skupina s maximálnymi počtami fotoelektrónov smerom od najvyššieho GTU. *SignalSeqFilter* zoradí pixely podľa GTU a rozdelí ich na pravú a ľavú skupinu od pixela s najväčším počtom fotoelektrónov. V ľavej skupine potom musí počet fotoelektrónov s klesajúcim GTU klesať a v pravej so stúpajúcim GTU rovnako klesať. Vybrané sú pixely spĺňajúce túto podmienku.



Obr. 6 – 7: Vývojový diagram funkcie Filter.

Cieľom opísaného členenia je aj zníženie pamäťových nárokov, čo ale nemusí platiť pri primitívnych typoch z dôvodu optimalizácie kompilátora, ale ukončenie bloku volá deštruktor objektu a to je podstatné, pretože najviac pamäťového priestoru zaberajú práve zoznamy typu `std::vector<Int_t>`. Výpis 5 zjednodušene ukazuje metódu `KeHoughModuleV2::Process(RecoEvent* ev)`, kde si je možné všimnúť volania krokov spracovania a rozsah viditeľnosti premenných opísaných v diagramoch.

Výpis 5: Zjednodušený pohľad na metódu `KeHoughModuleV2::Process(...)`

```

Bool_t KeHoughModuleV2::Process( RecoEvent* ev ) {

    Int_t iThresholdLowerLimit( fCfgThresholdLowerLimit );
    Double_t dMaybeTimeSize( fCfgSize );

    vector<Int_t> vecDirtyShowerPixels;
    set<Int_t> setHoughFinalPixels;

    MinMaxRange sHoughOutRange_cleanCore_ro_1;
    MinMaxRange sHoughOutRange_cleanCore_fi_1;
  
```

```
MinMaxRange sHoughOutRange_cleanCore_ro_2;
MinMaxRange sHoughOutRange_cleanCore_extra;

Process_Step1_PixelVectorsAndUtsPixels (...);

{
    vector< vector<Int_t> > vecVecSortedPixels;
    vector<Int_t> vecTopPixels;
    vector<Int_t> vecMaxGroupPixels;
    vector<Int_t> vecCorePixels;
    vector<Int_t> vecCleanCorePixels;
    vector<Int_t> vecRectangleCCPixels;

    Int_t iUidCount(0);
    Double_t dCleanCore_maxAbsX(0);
    Double_t dCleanCore_maxAbsY(0);
    Double_t dCleanCore_maxAbsGtu(0);
    Double_t dCleanCore_middleFi;
    Double_t dCleanCore_middleRo;

    Process_Step2_1_SortedPixels (...);
    Process_Step2_2_MaxGroupPixels (...);
    Process_Step2_3_CenterPixelVecsAndCorrCounts (...);
    Process_Step2_4_CorePixels (...);

    {
        vector<Int_t> vecCleanCorePixelsWithGaps;
        ShapedCore (...);
        Process_Step2_5_1b_CleanCorePixels (...);
    }

    Process_Step2_5_2_CleanCoreMiddle (...);
    Process_Step2_6_DirtyShowerPixels (...);
    Process_Step2_7_TimeSize (...);
}

Process_Step3_FinalPixels (...);
Process_Step4_Filter (...);
}
```

6.8 Výsledky

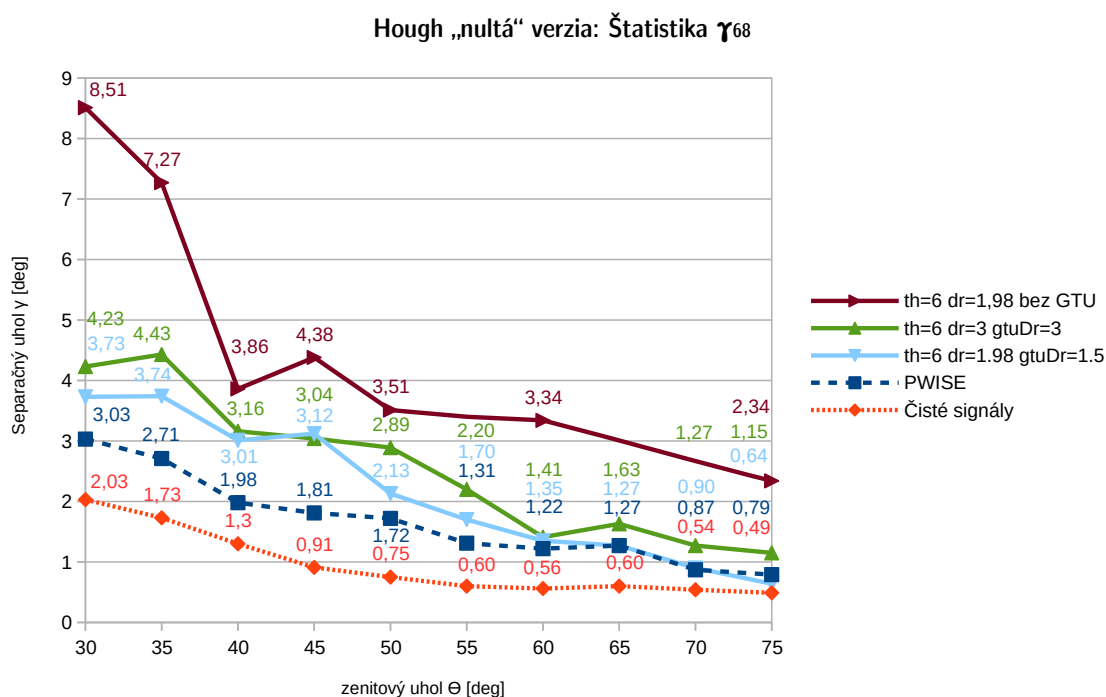
Vyhodnotenie algoritmov rekonštrukcie vzoru je zamerané najmä na výpočet štatistiky γ_{68} ¹⁴. Ukázalo sa, že podstatný údaj je aj počet zrekonštruovaných udalostí. Cieľom implementácie algoritmov do softvérového rámca ESAF bolo prekonať (aspoň v nejakom ohľade) aktuálne používaný algoritmus *PWISE*, preto sú dosiahnuté výsledky práve s ním porovnávané. Pre rekonštrukciu uhla bol použitý modul *Track-Direction2Module*, s analytickou aproximačnou metódou *AA1*.

6.8.1 „Nultá” verzia

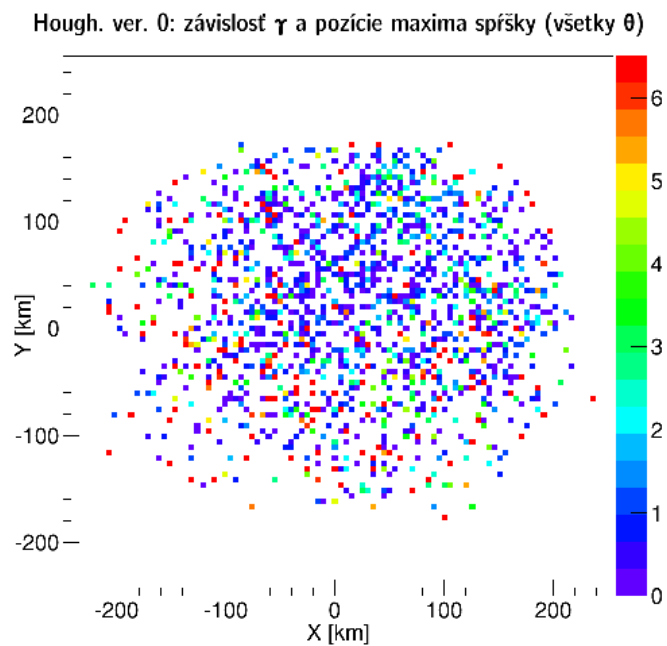
Implementácia „nulte” verzie algoritmu vo veľkej miere slúžila pre zoznámenie sa s princípom rekonštrukcie vzoru (čiary) použitím Houghovej transformácie a pre zoznámenie sa so štruktúrou programu rekonštrukcie softvérového rámca ESAF. Pre získanie pochopenia závislosti parametrov algoritmu od jeho výsledkov bola štatistika γ_{68} vypočítaná pre viaceré kombinácie parametrov - rôzne minimálne zrná vzdialenosti priamok dr a rôzne hranice počtu fotoelektrónov.

Obrázok 6–8 je porovnanie štatistík γ_{68} pre analyzované konfigurácie. Zelená čiara naznačuje výsledky pre najjednoduchšiu verziu algoritmu. Je možné pozorovať, že aplikácia druhého behu algoritmu dosiahla vylepšenie výsledkov. Výsledky sú porovnané s algoritmom *PWISE*, ktorý je vyobrazený dva-krát. *PWISE reference* sú údaje z článku *Performances of JEM-EUSO: angular reconstruction* (Biktemerova et. al, 2013) a *PWISE analysis* sú výsledky z analýzy vykonanej v rámci tejto práce. Obrázok 6–9 vyobrazuje chybu γ v závislosti od pozície maxima spířsky.

¹⁴Pre výpočet štatistiky zrekonštruovaných udalostí bola vytvorená trieda *MvRecoStatistics* opísaná v používateľskej príručke.

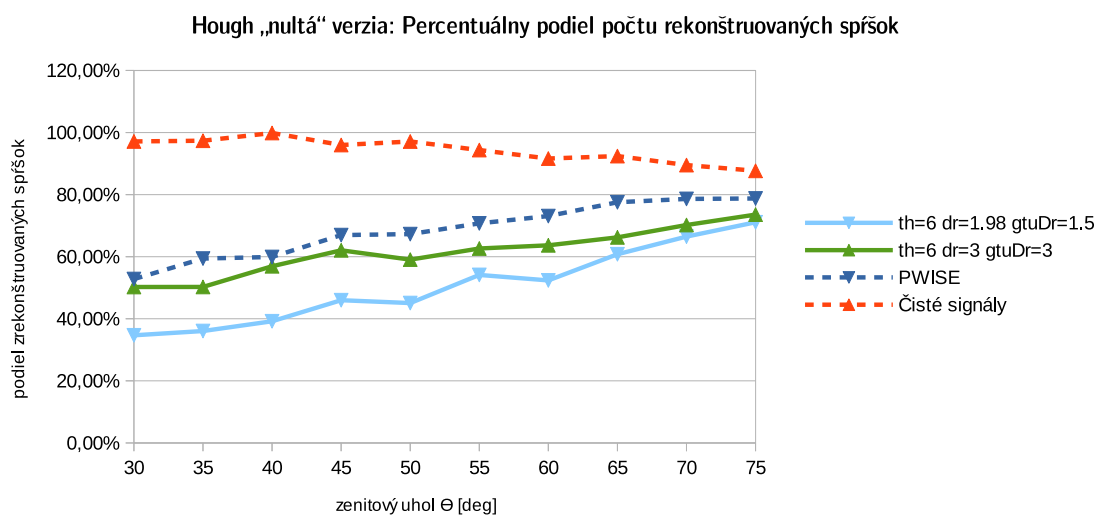


Obr. 6–8: Porovnanie štatistík γ_{68} rôznych konfigurácií „nultej“ verzie algoritmu.



Obr. 6–9: Závislosť pozície maxima spřšky a chyby γ pre „nultú“ verziu algoritmu.

Analýza ukázala, že táto verzia nedosahuje lepšie výsledky ako algoritmus PWISE. Problémom tejto „nulte“ verzie bol aj nízky počet zrekonštruovaných udalostí. Obrázok 6–10 porovnáva percentuálny podiel udalostí, ktoré boli akceptované modulom `TrackDirection2Module`. Tento modul, na základe nastavenia parametrom `TrackDirection2Module.fNumPointsMin`, obmedzuje minimálny počet pixelov pre inicializáciu jeho procesu.



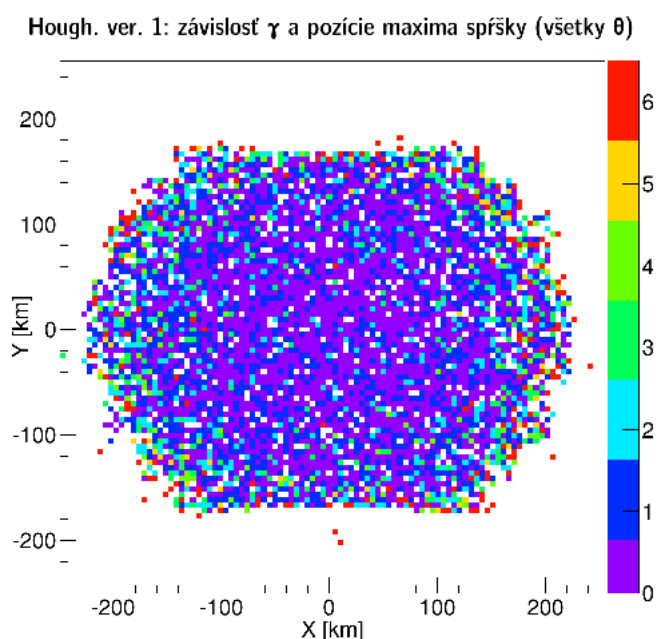
Obr. 6 – 10: Porovnanie percentuálneho podielu zrekonštruovaných spršok rôznych konfigurácií „nulte“ verzie algoritmu.

6.8.2 Prvá verzia

Ak pre iné spomenuté algoritmy, aj pomocou tejto implementácie algoritmu bola urobená rekonštrukcia približne 1500 udalostí spršok pre uhly od 30° do 75° . Obrázok 6–11 ilustruje závislosť pozície maxima spršky a odchýlky γ . Štatistika γ_{68} je v grafe na obrázku 6–12. Z pohľadu rekonštrukcie uhla zo simulovaných spršok sa ukazuje aj táto verzia úspešnejšia ako algoritmus PWISE. Výpis 6 ukazuje konfiguráciu použitú v tejto analýze. V tejto konfigurácii boli vybrané len pixely nad hranicou 5 fotoelektrónov.

Výpis 6: Konfigurácia modulu KeHoughModuleV1 použitá pri rekonštrukcii.

```
KeHoughModuleV1.DataThreshold = 5
KeHoughModuleV1.PatternThreshold = 5
KeHoughModuleV1.FiStep = 0.1
KeHoughModuleV1.RoStep = 0.1
KeHoughModuleV1.XyGtuSize = 2
KeHoughModuleV1.XySize = 4
```



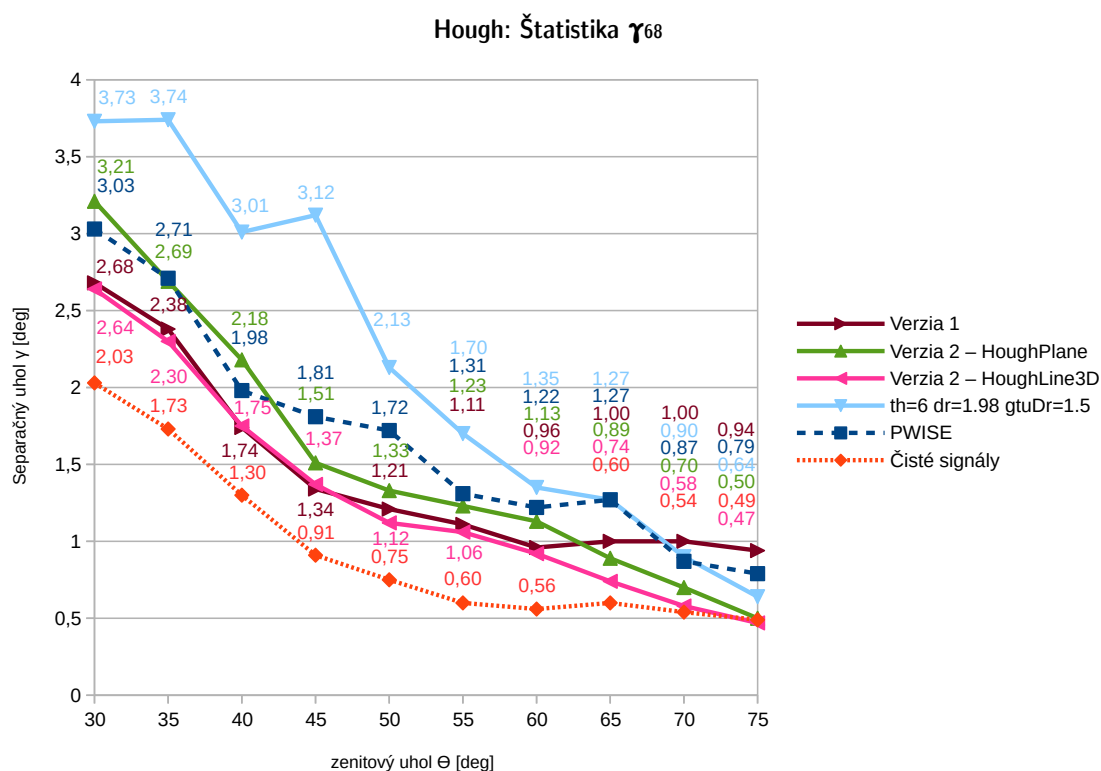
Obr. 6 – 11: Závislosť pozície maxima spřšky a chyby γ prvej verzie algoritmu

6.8.3 Druhá verzia

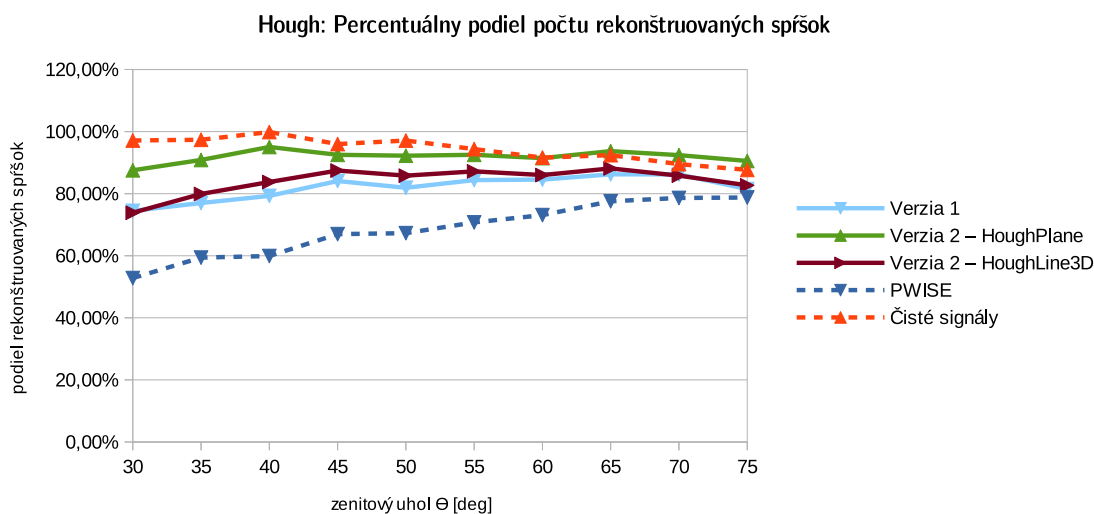
Cieľom implementácie druhej verzie algoritmu bolo na základe skúseností získanej z „nulte“ verzie implementovať algoritmus, ktorý by už dosahoval lepšie výsledky ako PWISE. Ako už bolo opísané táto verzia má dva „módy“ Houghovej transformácie - *HoughPlane*, *HoughLine3D*.

Obrázok 6–12 ukazuje porovnanie γ_{68} s algoritmom PWISE - výsledky sú veľmi podobné, prípadne lepšie. Vlastnosťou druhej verzie algoritmu je, že je zrekonštru-

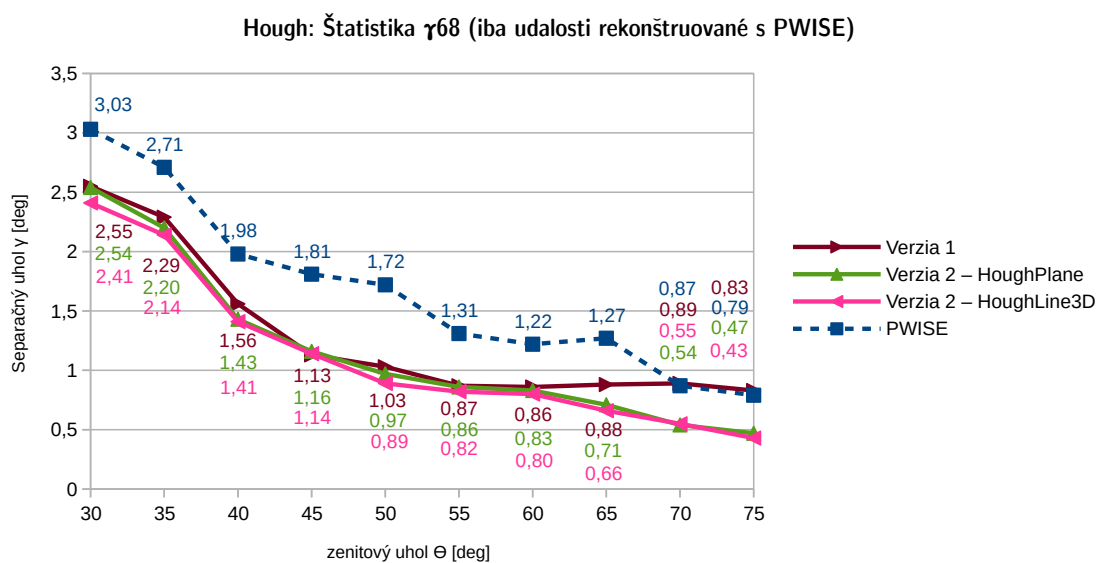
ovaný veľký podiel z rekonštruovaných udalostí, čo je vyobrazené na obrázku 6–13. Väčšina tých, ktoré sú zrekonštruované naviac, v porovnaní s algoritmom PWISE, sú udalosti s maximom na okraji ohniskovej plochy a tie aj zhoršujú štatistiku γ_{68} . Toto ilustruje obrázok 6–15a. Pri použití len udalostí, ktoré zrekonštruoval aj PWISE sa dosiahnú lepšie výsledky ukázané na obrázku 6–14.



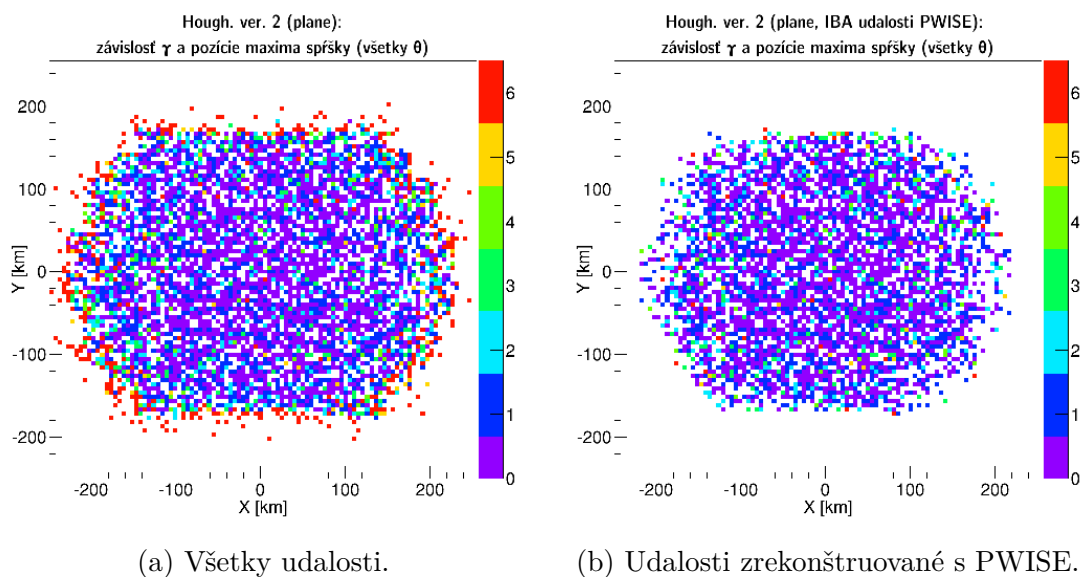
Obr. 6–12: Porovnanie štatistík γ_{68} prvej a druhej verzie algoritmu s algoritmom PWISE.



Obr. 6 – 13: Porovnanie počtu zrekonštruovaných udalostí prvej a druhej verzie algoritmu s algoritmom PWISE.



Obr. 6 – 14: Porovnanie štatistík γ_{68} prvej a druhej verzie algoritmu s algoritmom PWISE, pri použití udalostí zrekonštruovaných algoritmom PWISE.



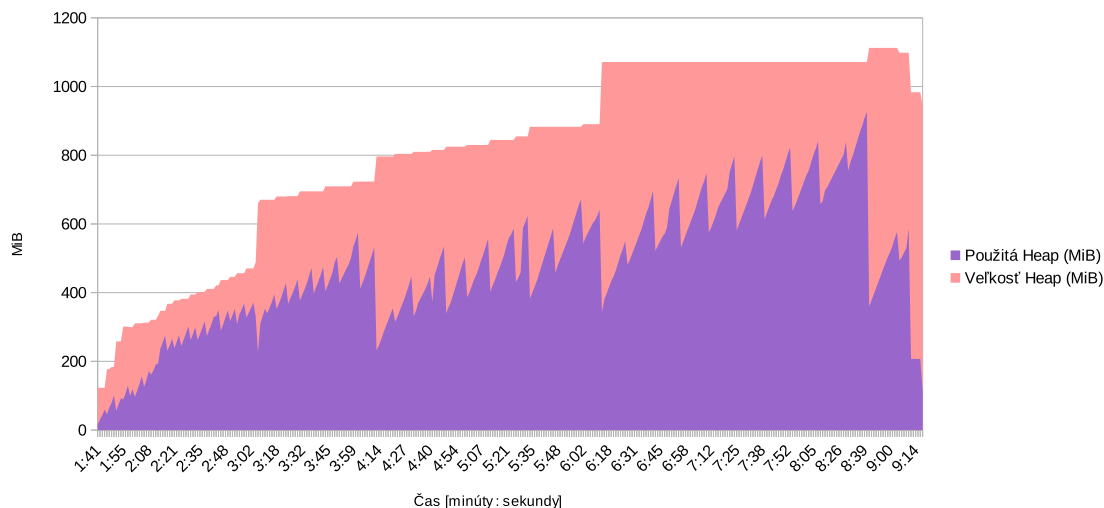
Obr. 6–15: Závislosť pozície maxima spřšky a chyby γ druhej verzie algoritmu (mód *HoughPlane*).

6.8.4 Porovnanie implementácií

Jedným z očakávaných benefitov implementácie algoritmu v programovacom jazyku nižšej úrovne bola vyššia rýchlosť spracovania a nižšie pamäťové nároky.

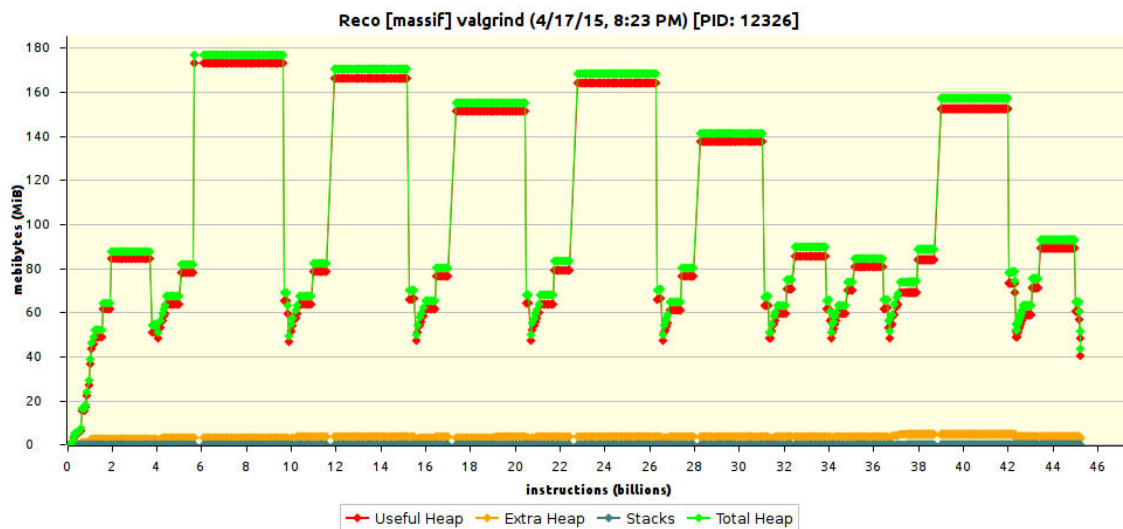
Sledovanie využitia pamäte programom je praktické sledovať pomocou profilovacích nástrojov. Implementácia v programovacom jazyku Java bola analyzovaná použitím nástroja integrovanom vo vývojovom prostredí *NetBeans*. Obrázok 6–16 ilustruje využitie pamäte virtuálnym strojom *Java Virtual Machine* (skratka *JVM*) pri spracovávaní desiatich udalostí pre algoritmus v druhej verzii. Najväčší podiel použitého priestoru zaberajú akumulčné matice Houghovej transformácie a postupne vytvárané zoznamy pixelov, čo dobre ilustruje rastúci charakter grafov. Keďže správa pamäte v Jave sa spolieha na *Garbage collection* pamäť je čistená v závislosti od potreby, čo logicky vytvára aj určitú réžiu. Na druhej strane, virtuálny stroj *JVM* interpretuje medzikód s cieľom vysokej efektivity a využíva viaceré dostupné procesorové jadrá aj pre spracovanie sériových programov (napríklad paralelne vlákno

pre *Garbage collection*).



Obr. 6 – 16: Využitie pamäte virtuálnym strojom JVM pri spracovaní 10 udalostí v implementácii druhej verzie algoritmu v programovacom jazyku Java.

Profílovanie požívania pamäte programom napísanom v jazyku *C++* bolo vykonané pomocou nástroja *Valgrind*, presnejšie, požívanie pamäte sa sledovalo nástrojom *Massif*. V diagrame na obrázku 6 – 17 je možné sledovať podobnú vlastnosť stúpania množstva alokovaného priestoru programom. Na rozdiel od Java verzie je pamäť uvoľňovaná programom, teda po výbere vzoru (spracovaní udalosti spŕšky) je príslušný priestor hneď uvoľnený.



Obr. 6–17: Využitie pamäte pri spracovaní 10 udalostí v implementácii druhej verzie algoritmu v programovacom jazyku C++.

Výsledok prekladu programu a efektívnosť tohto prekladu v kompilovanom (prekladanom) programovacom jazyku je určená prekladačom. Pre preklad C++ programov softvérového rámca sa používa prekladač `g++`, ktorý je rozhraním pre systém prekladačov *GNU Compiler Collection* - skratene *GCC*. Štandardne, cieľom prekladača *GCC* je redukovať cenu kompilácie a zabezpečiť, aby ladený program produkoval očakávané výsledky. Rýchlosť spracovania programu je možné zvýšiť použitím optimalizácií, ktoré vykonávajú globálnejšie transformácie na programe a aplikujú náročnejšie algoritmy pre analýzu. Knížnica `libclustering.so` softvérového rámca *ESAF*, obsahujúca triedy (moduly) pre rekonštrukciu vzoru, bola skompilovaná s použitím rozličných úrovni optimalizácie¹⁵. Tabuľky 6–1 a 6–2 porovnávajú priemerné namerané časy spracovania pre moduly rekonštrukcie vzoru použitím houghovej transformácie v rozličných optimalizačných úrovniach (analýza urobená na 100 udalostiach pre každý zenitový uhol). Štatistika bola vypočítaná pre rekonštrukcie simulácií rôznych zenitových uhlov, keďže počet bodov spŕšky spravidla so zvyšujúcim sa zenitovým uhlom stúpa. Tabuľka 6–1 je pre druhú verziu algoritmu

¹⁵Presný popis optimalizácií sa nachádza v dokumentácii *GCC* na adrese: <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>.

a tabuľka 6–2 pre prvú verziu algoritmu - variant B. Porovnávané rekonštrukcie v oboch tabuľkách používali štandardnú konfiguráciu, teda v prípade druhej verzie algoritmu sa vykonávala houghova transformácia metódou *HoughPlane*. Nižší výkon prvej verzie možno vysvetliť, tým že pri jej implementácii sa nemenila štruktúra programu na efektívnejšiu, čo môže byť predmetom ďalšej práce.

Tabuľka 6 – 1: Porovnanie priemerných časov spracovania druhej verzie algoritmu (v milisekundách)

zenitový uhol θ	Java	bez optimalizácie	-O1	-O2	-O3	-Os
30	777	944	491	525	480	594
45	948	1075	575	597	563	677
60	1260	1395	773	787	750	883
75	3043	2554	1441	1515	1441	1649

Tabuľka 6 – 2: Porovnanie priemerných časov spracovania prvej verzie algoritmu (v milisekundách)

zenitový uhol θ	Java	bez optimalizácie	-O1	-O2	-O3	-Os
30	106	196	194	202	203	204
45	128	236	235	241	232	232
60	200	372	377	376	360	368
75	608	956	969	1004	952	958

Provanie bolo vykonané na rovnakom počítači pod rovnakým operačným systémom. Operačný systém bol virtualizovaný Linux - distribúcia Ubuntu vo verzii 14.04.2 LTS 64-bit. Pre tento virtualizovaný systém boli sprístupnené 4 logické jadrá procesora Core i7-4702MQ a 4GB operačnej pamäte (celková operačná pamäť počítača 8GB - proces nemal dôvod odkladať pamäť na pevný disk). Použitý virtualizačný softvér bol VirtualBox 4.3.26. Hostovský operačný systém bol Windows 8.1 64-bit.

7 Prepis knižníc generovaných UV pozadí do dátovej štruktúry softvérového rámca ESAF

Pre analýzu generovaných údajov (pozri časť 5.1.) v balíku ESAF je potrebné použiť výsledky simulácie pozadia ako vstup do programu Reco. Pre riešenie problému sa ponúkajú dve možnosti:

- Konverzia textových súborov do *.root* súborov so štruktúrou rovnakou ako výstupné súbory simulácie - strom triedy softvérového rámca ROOT *TTree*, ktorý vo vetvách obsahuje triedy asociované s triedou *EEvent*.
- Vstupný modul rekonštrukcie schopný priamo čítať údaje z textových súborov.

Konverzia textových súborov do *.root* súborov. Výhoda tohto postupu je, že sa nevykonávajú žiadne zásahy do zdrojových kódov ESAF-u. Úloha by bola riešená buď samostatným programom využívajúcim knižnice ESAF-u a ROOT-u, alebo jednoduchšie, makrom pre softvérový rámec ROOT - riešenia sa v podstate líšia len metódou kompilácie (priame volanie *gcc* alebo použitie *CINT ROOT interaktívnej konzoly*) a spúšťaním (priame spustenie spustiteľného súboru alebo načítanie zdieľanej knižnice a následné spustenie požadovanej funkcie)¹⁶. Hlavná nevýhoda riešenia je potreba transformácie desiatok gigabajtov dát reprezentujúcich vygenerované UV pozadie v textových súboroch na *.root* súbory, ktorých veľkosť bude síce menšia (najmä z dôvodu neuchovávaní pixelov s nulovým počtom zaznamenaných fotónov), ale stále to bude zabraný diskový priestor. Zbytočnosť vytvárania *.root* súborov sa podčiarkuje tým, že požiadavkou pre prevod údajov je spracovanie týchto údajov v rekonštrukcii. Konverzia textových súborov do *.root* súborov, teda pridáva zbytočný medzikrok. Dôvod nepraktickosti riešenia je aj potreba pridávať informácie o detektore (napr. údaje o geometrii). Množina týchto údajov je pomerne veľká,

¹⁶Pozri opis softvérového rámca ROOT v časti 3.1.

a preto by bolo potrebné spracovávať veľkú časť parametrov konfigurácie a ukladať ich do tohoto súboru.

Vstupný modul rekonštrukcie. Motiváciou k riešeniu použitím vstupného modulu je, že nie je potrebné transformovať dáta do iného formátu. Taktiež, pre načítanie parametrov detektora je možné použiť už existujúce metódy v softvérovom rámci ESAF. Navyše, odstraňuje sa potreba medzikroku. Preto je toto riešenie zvolené ako, to na ktorom sa bude pracovať.

Formát textových súborov. Ako už bolo uvedené v časti 5.1 údaje v textových súboroch sa nachádzajú organizované ako 4 „stĺpce“ oddelené 2 znakmi medzery. Údaje popisujú, každý pixel na PDM v každom GTU, dokopy $48 * N_{GTU}$ záznamov pre jednu udalosť, kde N_{GTU} je počet GTU v danej udalosti. Jeden súbor obsahuje viacero takýchto udalostí. Je podstatné poznamenať, aké údaje sa v stĺpcoch nachádzajú, je to: x, y, GTU, počet fotónov zaznamenaných na pixeli. Identifikácia pixela je určená celočíselnou pozíciou od pravého horného rohu PDM - indexy 0 až 47. Pre stručnosť toto usporiadanie sa v práci bude označovať schéma 0-47.

7.1 Prevod súradníc na identifikátor pixela

Prevod súradníc zo schémy 0-47 na identifikátory pixelov je možné realizovať dvoma metódami:

- Výpočtom (odvodením) identifikátora z pozície.
- Priradením identifikátora odvodením z pozície použitím mapy pixelov ohniskovej plochy.

7.1.1 Výpočet identifikátora z pozície

Výpočet identifikátora z pozície je založený na určení riadka a stĺpca elementárnej bunky (EC), riadka a stĺpca pixela v PMT a pozície PMT v EC. Určené indexy sa potom vynásobia počtom pixelov v danom prvku. Tento výpočet je určený pre aktuálne usporiadanie prvkov ohniskovej plochy. Na obrázku 7–1 je zobrazená organizácia PDM, PMT, a potom samotných pixelov na ohniskovej ploche. Obrázok 7–1a ilustruje celú ohniskovú plochu detektora, kde jeden štvorec reprezentuje jeden PDM s číslom v strede štvorca. Obrázok 7–1b ukazuje výsek z priblíženého pohľadu na PDM s číslami 1, 2, 12, 13 - stred ohniskovej plochy. Štvorce v tomto obrázku reprezentujú jednotlivé PMT, rozličná farba odlišuje PDM. Červený rámček zvýrazňuje jednu EC - štvoricu PMT, v rámci ktorej sú PMT usporiadané po stĺpcoch, samotné EC sú potom usporiadané po riadkoch. Posledný obrázok 7–1c ilustruje jednu EC, kde štvorce reprezentujú pixely s identifikátorm (číslom pixela) v štvorci. Pixely v PMT sú usporiadané po stĺpcoch, vyznačené červeným obdĺžnikom. Pre iné PDM (makrobunka, v ESAF označované *MacroCell*) ako 1 je potrebné pripočítať násobok počtu pixelov v jednom PDM.

Ak súradnice pixela v schéme 0-47 sú $x \in \langle 0, 47 \rangle, y \in \langle 0, 47 \rangle$ a PDM_{id} je identifikátor PDM. Potom identifikátor pixela je určený funkciou $UID(x, y, PDM_{id})$ na rovnici 7.6. Určenie čísla EC pre ďalšie výpočty v rozmedzí 1 až 6 ukazujú rovnice 7.1 a 7.2.

$$EC_{stlpec}(x) = \lceil \frac{(x+1)}{16} \rceil \quad (7.1)$$

$$EC_{riadok}(y) = \lceil \frac{(y+1)}{16} \rceil - 4 \quad (7.2)$$

Výpočet čísiel PMT určujú rovnice 7.3 a 7.4

$$PMT_{stlpec}(x) = \begin{cases} x - |EC_{stlpec}(x) - 1| * 16 - 8 \\ \text{Ak } x \in \langle 16n + 8, 16n + 15 \rangle \text{ pre } n \in \langle 0, 2 \rangle \\ x - |EC_{stlpec}(x) - 1| * 16 \end{cases} \quad \text{inak} \quad (7.3)$$

$$PMT_{riadok}(y) = \begin{cases} EC_{riadok}(y) * 16 - |y - 48| - 8 \\ \text{Ak } y \in \langle 16n + 8, 16n + 15 \rangle \text{ pre } n \in \langle 0, 2 \rangle \\ EC_{riadok}(y) * 16 - |y - 48| \end{cases} \quad \text{inak} \quad (7.4)$$

Index PMT v rámci EC je v rovnici 7.5.

$$PMT_{index}(x, y) = \begin{cases} 0 & \text{Ak } x, y \in \langle 16n + 8, 16n + 15 \rangle \text{ pre } n \in \langle 0, 2 \rangle \\ 1 & \text{Ak } x \in \langle 16n + 8, 16n + 15 \rangle \text{ pre } n \in \langle 0, 2 \rangle \\ 2 & \text{Ak } y \in \langle 16n + 8, 16n + 15 \rangle \text{ pre } n \in \langle 0, 2 \rangle \\ 3 & \text{inak} \end{cases} \quad (7.5)$$

$$\begin{aligned}
UID(x, y, PDM_{id}) = & \\
& (PDM_{id} - 1) * 2304 \\
& + (EC_{riadok}(y) - 1) * 768 + (EC_{stlpec}(x) - 1) * 256 \quad (7.6) \\
& + PMT_{index}(x, y) * 64 + PMT_{stlpec}(x) * 8 \\
& + PMT_{riadok}(y) + 1
\end{aligned}$$

7.1.2 Priradenie identifikátora odvodením z pozície pixela použitím mapy ohniskovej plochy

Myšlienka tejto metódy pozostáva z „prekrytia“ dvoch matíc - matica indexov a matica pixelov (pixel je reprezentovaný n-ticou (x, y, uid)). Ak n je počet rôznych pozícií na osi x a m je počet rôznych pozícií na osi y , maticu pozícií pixelov možno vytvoriť napríklad nasledujúcim postupom. Uvažuje sa, že sa spracovávajú údaje len pre jedno PDM.

1. Do usporiadanej množiny $xpositions$ sa vložia všetky rôzne pozície pixelov na osi x .
2. Do usporiadanej množiny $ypositions$ sa vložia všetky rôzne pozície pixelov na osi y .
3. Všetky pixely, unikátne na základe identifikátora, sú vložené do matice M o veľkosti $n \times m$, tak že poradie (index) v množine $xpositions$ určuje index stĺpca a poradie (index) v množine $ypositions$ určuje index riadku.

Tento postup predpokladá, že množina pixelov, z ktorej sa „zoradená“ matica vytvára je vyfiltrovaná obsahujúca pixely konkrétneho jedného PDM. Výhodou konštrukcie takéhoto mapovania je, že metóda nie je zameraná len na jeden typ usporiadania pixelov v PDM. Problémom, na ktorý by bolo potrebné pri implementácii

myslieť je, že pozície pixelov netvorí presne štvorcovú maticu, preto by ich bolo potrebné zaokrúhľovať.

7.2 Informácie potrebné pre rekonštrukciu

Okrem samotných záznamov pixelov na prednej elektronike detektora ukladaných v objektoch triedy `RecoPixel`, je potrebné s udalosťou `RecoEvent` asociovať pravdivé informácie o udalosti, napríklad skutočnú energiu spršky, skutočný zenitový uhol, skutočnú výšku maxima spršky, používané na výpočet štatistických informácií pre porovnanie s výsledkom rekonštrukcie. Veľmi dôležitými informáciami, bez ktorých sa nie je možné v rekonštrukcii zaobísť, sú informácie o detektore, najmä pozície pixelov, na ktorých bol zaznamenaný nejaký fotón, potrebné napríklad pre algoritmy rekonštrukcie vzoru. Informácie o aktivovaných spúšťačoch môžu byť tiež požadované, napríklad modul rekonštrukcie `PmtToShowerReco` vyžaduje, aby spúšťač `CCB_LTT` na PDM, ktorej patrí spracovávaný pixel, bol aktivovaný.

V prípade vstupu s `.root` súborom výstupu simulácie tieto informácie sú vstupným modulom `RootInputModule` čítané z tohto súboru. Na druhej strane, keď tieto informácie nie sú dostupné, je ich potrebné dodať konfiguráciou.

7.3 Implementácia

Riešenie tejto úlohy bolo založené na implementácii vstupného modulu rekonštrukcie, čo vyžadovalo prekonanie komplikácií spôsobených nevhodným návrhom softvérového rámca ESAF. Pre účely generovania testovacích údajov bolo rozšírené makro `MvEventPrinter` o možnosť opačného prepočtu. Tento modul bol do ESAF-u pridaný pod názvom `TxtFeeInputModule`.

7.3.1 Výpočet identifikátora z pozície

Pre implementáciu v module bola zvolená prvá uvažovaná možnosť, teda určenie pozície prepočtom. V budúcnosti môže byť ďalšia metóda doimplementovaná. Výpis 7 ukazuje implementáciu prepočtu v programovacom jazyku C/C++.

Výpis 7: Výpočet identifikátora pixela zo schémy 0-47 - C/C++ implementácia.

```

Int_t CalcUid(int x, int y, int macrocellId) {
    Int_t ec_col_n, ec_row_n, pmt_off_n, pmt_row_loc, pmt_col_loc;
    Short_t pmt_row_lt8, pmt_col_lt8;

    // vypocet EC
    ec_col_n = CeilNint((x+1)/16.);
    ec_row_n = Abs(CeilNint((y+1)/16.) - 4);

    // vypocet riadku a stlpca v~PMT
    pmt_col_loc = x - ((ec_col_n-1)*16);
    pmt_row_loc = (ec_row_n*16)-Abs(y-48);

    pmt_col_lt8 = (pmt_col_loc<8);
    pmt_row_lt8 = (pmt_row_loc<8);

    if(!pmt_col_lt8) pmt_col_loc -= 8;
    if(!pmt_row_lt8) pmt_row_loc -= 8;

    // vypocet cisla PMT v~EC
    pmt_off_n = (pmt_col_lt8*2) + pmt_row_lt8;

    return
        (macrocellId-1)*2304 +
        (ec_row_n-1)*768 + (ec_col_n-1)*256 +
        pmt_off_n*64 + (pmt_col_loc)*8 + (pmt_row_loc) + 1;
}

```

Opačný prepočet. Pre účely testovania bola implementovaná aj metóda opačného prepočtu z identifikátorov pixelov na schému 0-47. Takto je možné overiť funkcionality vstupného modulu tým, že sa do formátu záznamov UV pozadí transfor-

muje spráška zo simulácie. A ak sú výsledky pre obe vstupné metódy rovnaké, dá sa predpokladať, že vstupný modul pracuje správne. Samozrejme, pre správne overenie funkčnosti je potrebné, aby vstupný súbor, pre tento vstupný modul rekonštrukcie, mal rovnaký formát ako textové súbory generovaného pozadia. Výpis 8 ukazuje implementáciu prepočtu v programovacom jazyku *C/C++*.

Výpis 8: Výpočet pozícií x, y v schéme 0-47 z identifikátora pixela - *C/C++* implementácia.

```
void CalcXY47(int uid, int * x47, int * y47) {
    uid = uid - ((uid-1)/2304)*2304;

    int row3 = (uid-1)/768;
    int rowsec = uid - row3*768;

    int col3 = (rowsec-1) >> 8;
    int colsec = rowsec - (col3 << 8);

    int quadrant = (colsec-1) >> 6 ;
    int quadsec = colsec - (quadrant << 6);
    int octet = (quadsec-1) >> 3;
    int octetNum = quadsec - (octet << 3);

    *x47 = (col3 << 4) + 8*(quadrant<2) + octet;
    *y47 = 47-((row3 << 4) + (((quadrant%2)+1) << 3) ) + octetNum;
}
```

7.3.2 Chýbajúce údaje pre rekonštrukciu

Hlavný chýbajúci údaj v rekonštrukcii je mapa pixelov sprístupňovaná pomocou objektu parametrov behu reprezentovaného triedou *RunParameters*. Riešením tohto problému bolo vytvorenie objektu triedy *EusoElectronics*, ktorý túto konfiguráciu načítava a potom použitím objektov tried *ERunParsPixelMapFiller* a *ERunParsPmtsFiller* naplní parametre behu. Súbor *GNUmakefile* musel byť patrične upravený, aby sa použili potrebné hlavičkové súbory. Ostatné doplnkové parametre sú nastaviteľné v konfigurácii vstupného modulu. Podstatná hodnota, na ktorú treba

myslieť pri nastavovaní konfiguračných parametrov, je reťazec bitov „slovo spúšťačov“, ktoré musí v sebe obsahovať číslo 8192 indikujúce aktiváciu spúšťača `CCB_LTT`.

7.3.3 Trieda `RecoEventBuilder`

Komplikácia pri implementácii tohto vstupného modulu bola nevhodná implementácia triedy `RecoEvent` a jej asociovaných tried. Problémom je neexistencia *setter* metód pre nastavenie hodnôt členských premenných týchto tried. Miesto toho, trieda `RootInputModule` je priateľská trieda týchto tried. Čo síce splňa požiadavku, aby údaje udalosti boli modifikované len vstupným modulom, daná implementácia, ale napevno dovoľuje len použitie jedného vstupného modulu a program takto stráca abstrakciu.

Pridanie ďalšej priateľskej triedy bolo riešenie vyžadujúce minimálnu úpravu existujúcich zdrojových kódov. Pre zovšeobecnenie, táto trieda nie je priamo vstupný modul, ale „medzikrok“. Navrhnutá trieda `RecoEventBuilder` ponúka pohodlnú formu nastavenia údajov spršky a následne vytvorenie objektu triedy `RecoEvent`. Čo popisuje návrhový vzor *staviteľ* (*ang. builder*). Praktickou vlastnosťou tejto triedy je aj to, že zaobahuje konštrukcie objektov a nastavovanie hodnôt členských premenných do kratšieho, diskutabilne¹⁷ prehľadnejšieho volania metód. Taktiež sa vykonáva zoradenie pixelov prednej elektroniky (trieda `EFee`), s cieľom zvýšenia konzistencie, podľa GTU a identifikátora pixela, čo je praktické, keďže napríklad pixely v simulácii pozadia sú v inom poradí.

7.3.4 Čítanie údajov z vstupného súboru

Očakávaný formát vstupného súboru bol parametrizovaný, teda s možnosťou nastavenia stĺpcov čítaných zo súboru. Navyiac čítanie je pripravené pre spracovanie

¹⁷Pri väčšom zaobalení v metóde s viacerými parametrami, možno polemizovať, že z prvého pohľadu nie je jasné, čo metóda robí.

súborov s desatinnými číslami a použitie rôznych oddelovačov medzi hodnotami. Na základe toho, či sa medzi stĺpcami nachádza identifikátor pixela bolo čítanie súboru rozdelené na priame a nepriame. V nepriamom čítaní sa pre určenie identifikátora používal výpočet už uvedený vo výpise 7. Keďže súbory simulovaného pozadia obsahujú viacero konkatenovaných potenciálne rozpoznateľných udalostí. Každá z týchto udalostí je zoradená vzostupne podľa stĺpcov x , y , GTU . Keďže koniec potenciálnej udalosti nie je označený špeciálnym znakom, je potrebné porovnávať hodnoty aktuálneho a predošlého riadku. Ak sú hodnoty aktuálneho riadku nižšie ako predošlý, aktuálny riadok je uchovaný, ostatné riadky patria do udalosti ďalej spracovávanej rekonštrukciou. Každý prečítaný záznam, s vyšším alebo rovným počtom fotoelektrónov ako hranica nastavená konfiguračným parametrom `TxtFeInputModule.fCountsThreshold`, je uložený ako objekt triedy `RecoPixelData`. Tento objekt je potom využívaný v moduloch rekonštrukcie. Doplnkovo, pre každý záznam je vytvorený aj objekt triedy `RecoCellHit`, vytvorený pri volaní metódy `RecoCellInfo::AddHit`.

7.4 Výsledky

Ako prvé, v rámci testovania modulu, bola skutočná nasimulovaná spíška pretransformovaná do formátu výstupu simulácie pozadia použitím prevodu uvedeného vo výpise 8¹⁸. Porovnanie výsledkov ukázalo rozdiel v rekonštruovaných energiách v treťom desatinnom rade, ostatné vypočítané hodnoty boli rovnaké.

S cieľom identifikácie tohto problému bol preskúmaný zdrojový kód modulu `PmtToShowerReco`. Skúmanie ukázalo, že rozdiel je spôsobený použitím náhodnej hodnoty pri rekonštrukcii, presnejšie pri výpočte priepustnosti atmosféry. Vychádzajúc od metódy `PmtToShowerReco::Process(RecoEvent*ev)`, v ktorej sa volá me-

¹⁸Táto možnosť bola pridaná do triedy (makra) `MvEventPrinter` opísaného v používateľskej príručke.

tóda `PmtToShowerReco::AtmosphericLoss(...)` zodpovedná za výpočet absorpčnej charakteristiky atmosféry. Tu nastáva volanie metódy `PmtToShowerReco::Transmission(...)`, kde sa pre každé políčko (anglicky bin) spektra (trieda popisujúca spektrum je `EsafSpectrum`) v rámci každého GTU, simuluje fotón (trieda `SinglePhoton`). Vlnová dĺžka tohto fotónu je nastavená použitím metódy `EsafSpectrum::GetLambda(Int_t i)`, kde sa v rozmedzí i -tého políčka vypočíta vlnová dĺžka použitím náhodnej hodnoty. Výsledkom tohto opisu teda je, že pri porovnávaní výsledkov rekonštrukcie sa korektne môžu výsledky rekonštrukcie energie odlišovať v nižších desatinných rádoch.

Výsledky rekonštrukcie testovacej udalosti boli porovnané s rekonštrukciou z `.root` súboru až na spomenutý problém s energiou boli rovnaké. Porovnal sa zrekonštruovaný zenitový uhol θ (modul `TrackDirection2Module`), výška maxima spršky v metroch $Hmax$, slant hĺbka spršky $Xmax$, pozícia jadra spršky. V ďalšom kroku bolo potrebné overiť čítanie viacerých udalostí v súbore za sebou, bez vedomosti o počte riadkov udalosti. Toto bolo realizované konkaténáciou viacerých udalostí. A znova porovnaním výsledkov. Opísané overenie umožnilo, aby mohol byť spracovaný súbor simulácie pozadia s dostatočnou istotou správnosti čítania vstupného súboru.

Modul bol nasadený na spracovanie 2087 záznamov pozadia, ktoré prešli simulovanou schémou spúšťačov. Aj keď sa nejedná o veľkú vzorku, spracovanie takýchto „falošných udalostí“ približne ukazuje výsledky, ktoré by sa dosiahli pri väčších vzorkách. Tieto údaje boli spracované algoritmom PWISE a algoritmami Houghovej transformácie, ktoré boli implementované v rámci tejto práce. Tabuľka 7–1 porovnáva počty udalostí „falošných spršok“, ktoré našli viac ako 10 pixelov, teda boli akceptované ďalším modulom rekonštrukcie `TrackDirection2Module`.

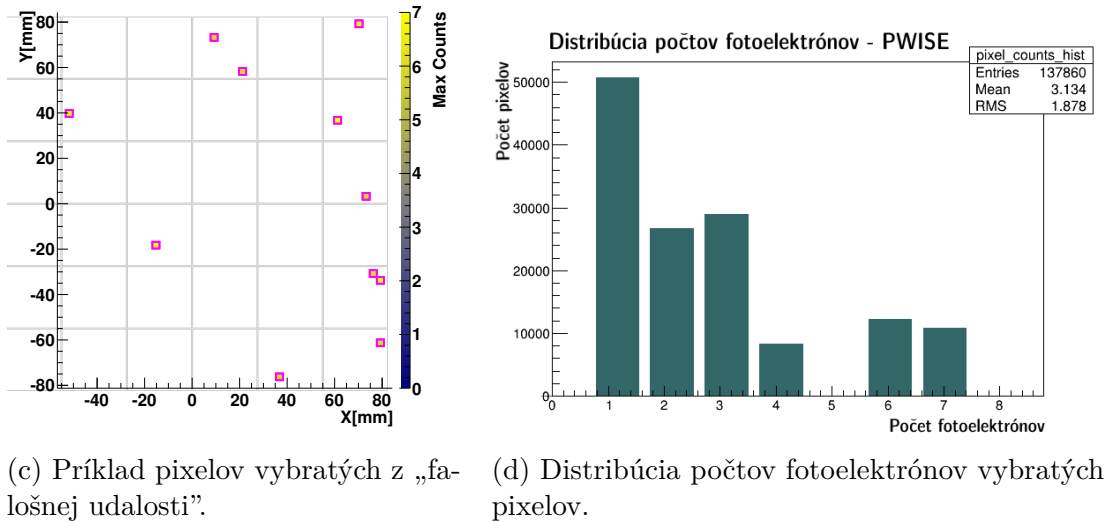
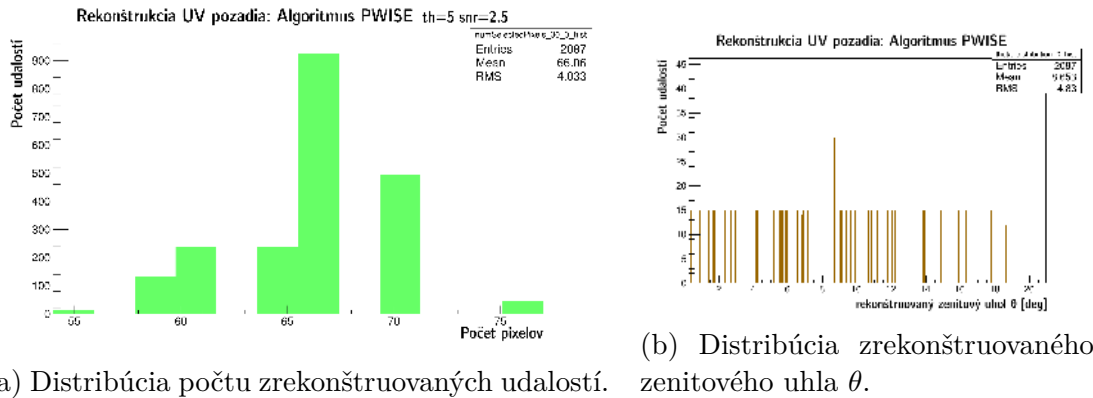
Algoritmus PWISE pri štandardne použíwanej konfigurácii ¹⁹, ktorá bola analyzovaná

¹⁹Pozri opis algoritmu PWISE v časti 4.2.2.

i v tejto práci, nenašla v týchto dátach žiadne vzory. V rámci experimentu bola znižovaná hodnota parametrov `PWISEModule.fThreshold` a `PWISEModule.fSNRejection`. Konfigurácia `PWISEModule.fThreshold = 5` a `PWISEModule.fSNRejection = 2.5` vybrala viac ako 10 pixelov vo všetkých „falošných spráškach“ spracovávanej množiny dát. Obrázok 7–2a ukazuje distribúciu počtov vybraných pixelov pre túto množinu dát a obrázok 7–2b distribúciu zrekonštruovaných zenitových uhlov θ . Typický výsledok rekonštrukcie pomocou algoritmu PWISE pre analyzovaný šum je na obrázku 7–2c²⁰. Distribúcia počtov fotoelektrónov vybraných pixelov je na obrázku 7–2d²¹.

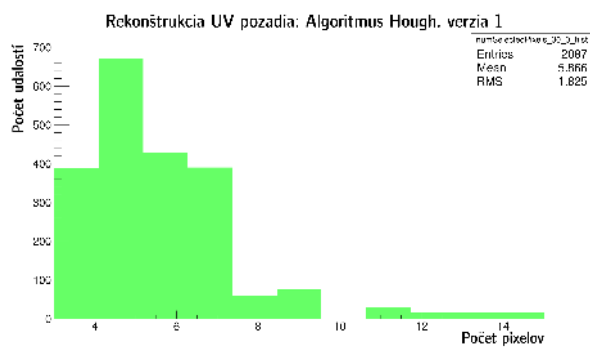
²⁰Pre vygenerovanie tejto vizualizácie bolo použité makro `RecoInfo.C` opísané v používateľskej príručke.

²¹Pre vygenerovanie histogramov bola použitá trieda (makro) `MvRecoStatistics` opísané v používateľskej príručke.

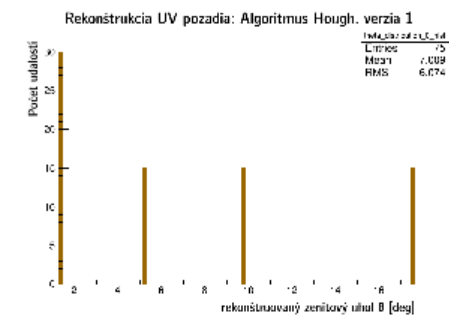
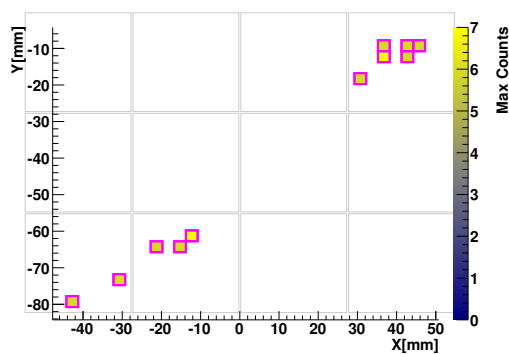


Obr. 7 – 2: Výsledky rekonštrukcie „falošných udalostí” pri algoritme PWISE.

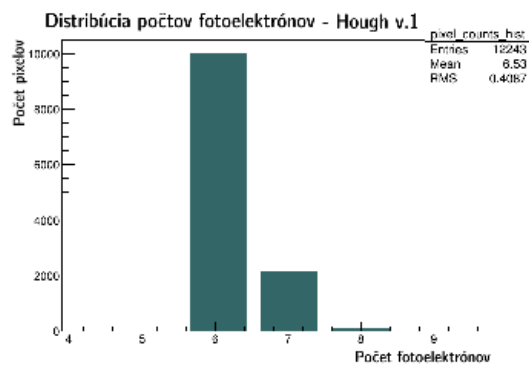
Rekonštrukcia spršok algoritmom Houghovej transformácie ukázala problematickú stránku implementovaných algoritmov. Ich analýza ukázala, že tieto algoritmy nachádzajú vzory vo veľkej časti udalostí spršiek, čo sa ukázalo negatívne pri rekonštrukcii šumu. Prvá verzia našla viac ako 10 pixelov v približne 3,6% z analyzovaných „falošných udalostí”. Druhá verzia algoritmu v každej udalosti našla aspoň 3 body, kde približne polovica udalostí bola akceptovaná modulom `TrackDirection2Module`. Je potrebné poznamenať, že žiadny z nájdených vzorov nebol rekonštruovaný ako sprška so zenitovým uhlom θ viac ako 20° . Obrázky 7–3 a 7–4 vizualizujú získané výsledky.



(a) Distribúcia počtu zrekonštruovaných udalostí.

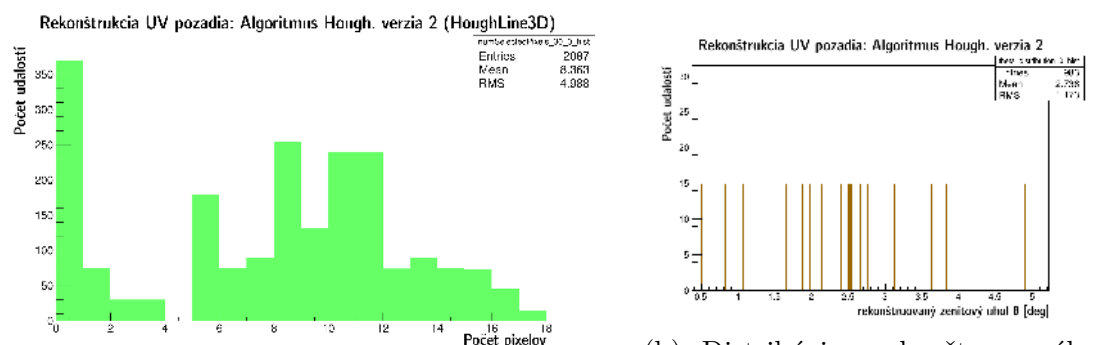
(b) Distribúcia zrekonštruovaného zenitového uhla θ .

(c) Príklad pixelov vybratých z „falošnej udalosti“.

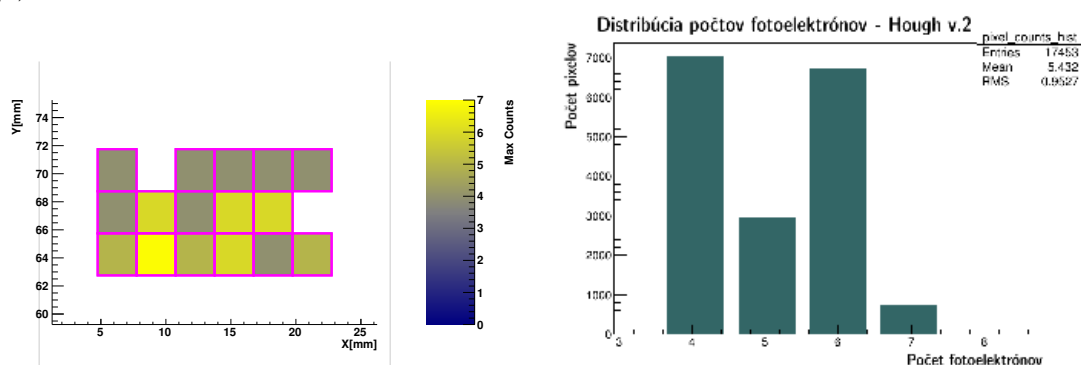


(d) Distribúcia počtov fotoelektrónov vybratých pixelov.

Obr. 7–3: Výsledky rekonštrukcie „falošných udalostí“ pri prvej verzii algoritmu Houghovej transformácie.



(a) Distribúcia počtu zrekonštruovaných udalostí.

(b) Distribúcia zrekonštruovaného zenitového uhla θ .

(c) Príklad pixelov vybratých z „falošnej udalosti“.

(d) Distribúcia počtov fotoelektrónov vybratých pixelov.

Obr. 7 – 4: Výsledky rekonštrukcie „falošných udalostí“ pri druhej verzii algoritmu Houghovej transformácie (*HoughLine3D*).

Tabuľka 7 – 1: Porovnanie výsledkov rekonštrukcie pozadia pri použití vstupného modulu *TxtFeeInputModule*.

Rekonštrukcia vzoru	Konfigurácia	Počet pixelov > 10
PWISE	fThreshold=8 fSNRejection=5	0
PWISE	fThreshold=8 fSNRejection=3	0
PWISE	fThreshold=6 fSNRejection=3	0
PWISE	fThreshold=5 fSNRejection=3	0
PWISE	fThreshold=5 fSNRejection=2.5	2087
Hough verzia 2 (<i>HoughLine3D</i>)	rovnaká ako analýza v časti 6.8.3	983
Hough verzia 1	rovnaká ako analýza v časti 6.8.2	75

8 Rozšírenie možností simulácie UV pozadia

Simulácia UV pozadia sa vykonáva pri simulácii GTU čipu prednej elektroniky (4 PMT, 1 EC) reprezentovaného triedou `FrontEndChip` - metóda `FrontEndChip::Gtu()`²². Simulácia je realizovaná generovaním náhodného čísla na základe *Poissonovej distribúcie* využitím metódy ponúkanej softvérovým rámcom `ROOT TRandom::Poisson(Double_t)`. Parametrom tejto metódy je stredná hodnota, ktorá je v tomto prípade vypočítaná ako $\mu = \text{trvanie_GTU} \times \text{úroveň_nočnej_žiary}$, kde *úroveň_nočnej_žiary* je externá hodnota, ktorú je cieľom nastaviť. Táto hodnota je pridelená čipu prednej elektorniky - objektu triedy `FrontEndChip`, pred procesom simulácie v metóde `EusoElectronics::BuildBackgroundChipDist()` volanej v metóde `EusoElectronics::Build()`, teda pred samotným procesom simulácie.

Metóda `EusoElectronics::NightGlowRate(const TVector3& pos, const TVector3& norm, Double_t pxsize, Double_t pde)` je „centrom záujmu“ problematiky určenia úrovne nočnej žiary v softvérovom rámci ESAF. Parametre tejto metódy sú:

pos - vektor pozície na ohniskovej ploche, prakticky je to priemerná pozícia PMT na čipe,

norm - vektor normály určujúcej orientáciu, prakticky orientácia prvého PMT na čipe,

pxsize - veľkosť pixela, rovnako získané z prvého PMT,

pde - kvantová efektivita PMT.

Výsledok volania tejto metódy je hodnota, ku ktorej je pripočítaná *úroveň tmavého šumu PMT* (prvé v čipe) a táto hodnota je nastavená ako *úroveň_nočnej_žiary*.

²²Časť 3.2 stručne opisuje proces simulácie. Ešte podrobnejšie je proces opísaný v používateľskej príručke.

8.1 Model distribúcie UV pozadia na citlivej ohniskovej ploche detektora

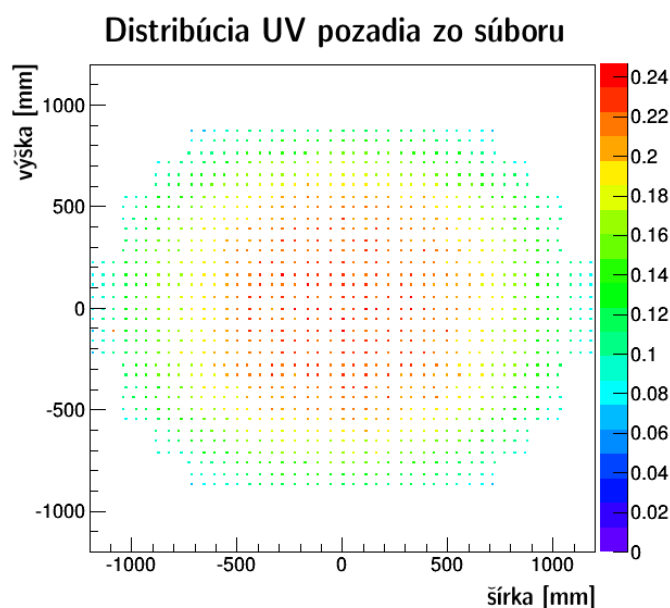
Cieľom tejto úlohy je implementovať do softverového rámca ESAF možnosť použitia externého súboru popisujúceho úroveň UV pozadia na základe pozície, zdrojom týchto údajov sú externé simulačné programy.

Z implementačného hľadiska je potrebné upraviť tieto existujúce metódy triedy `EusoElectronics`:

- `EusoElectronics::Build()` - kontrola parametra konfigurácie rozšírená o novú možnosť, načítanie súboru popisujúceho úroveň UV pozadia,
- `EusoElectronics::NightGlowRate(...)` - výber hodnoty pre požadovanú pozíciu, prípadne PMT alebo EC (pre čip prednej elektroniky).

8.1.1 Súbor distribúcie UV pozadia na ohniskovej ploche

Súbor pre použitie v simulácii obsahuje mapu bodov, ktorá asocjuje pozíciu na ohniskovej ploche k intenzite UV pozadia. Problémom je potreba priradiť tieto pozície k pozíciám na ohniskovej ploche a keďže sa očakáva, že pozície nie sú identické musí sa realizovať interpolácia. Obrázok 8–1 vizualizuje body prečítané z tohto súboru. Pri hľadaní bodov treba myslieť aj na to, že pozície v súbore nie sú v presnej matici - pozície bodov v „stĺpcoch” sú v rozmedzí 20mm , keďže matica je projekcia z 3D modelu.



Obr. 8 – 1: Distribúcia intenzity UV pozadia zo súboru.

8.1.2 Metódy interpolácie

Priradenie správnej hodnoty pre hľadanú pozíciu je realizované interpoláciou najbližších bodov definovaných v súbore modelu. Pre aktuálnu verziu sa uvažujú techniky *najbližší sused* a *bilineárna interpolácia*.

Najbližší sused Technika najbližší sused (anglicky *Nearest-neighbor*) je jednoduchá v prípade, že najbližšie body boli nájdené. Princíp výpočtu je použitie Pytagorovej vety pre výpočet vzdialenosti interpolovaného a zdrojového bodu, kde dve hrany trojuholníka sú rozdiely koordinátov x a y .

Bilineárna interpolácia Trochu zložitejší princíp je bilineárna interpolácia. Skladá sa z dvoch lineárnych interpolácií - jednej na osi x , druhej na osi y . Miernym rozdielom oproti štandardnej je v tomto nasadení mierne nerovnomerná pozícia bodov, čo by korektne vyžadovalo zložitejší výpočet. Pre aktuálnu verziu ale bola zvolená bili-

neárna interpolácia pre regulárnu mriežku a bolo rozhodnuté zanedbať malé rozdiely v pozíciách. V budúcnosti, ak to bude potrebné, je možné implementovať pokročilejšiu techniku. Keďže sa očakáva aj nerovnomerné rozloženie bodov (najmä na okraji ohniskovej plochy), v prípadoch, kde sa interpolovaný bod nachádza mimo štvoruholníka (prípadne trojuholníka) tvoreného najbližšími bodmi používajú sa existujúce hodnoty. Rovnice 8.1, 8.2 ukazujú výpočet interpolácie hodnoty na osiach x a rovnica 8.3 výpočet interpolovanej hodnoty. V prípade riešeného problému premenné Q_{11} , Q_{12} , Q_{21} , Q_{22} reprezentujú intenzity v okolitých bodoch, ktorých pozície na osi x určujú premenné x_1 , x_2 a na osi y premenné y_1 , y_2 . Premenné x a y určujú pozíciu interpolovaného bodu.

$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} Q_{11} + \frac{x - x_1}{x_2 - x_1} Q_{21} \quad (8.1)$$

$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} Q_{12} + \frac{x - x_1}{x_2 - x_1} Q_{22} \quad (8.2)$$

$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \quad (8.3)$$

8.1.3 Implementácia

Pre implementáciu ďalšieho modelu UV pozadia v softvérovom rámci ESAF bola potrebná úprava už spomenutých zdrojových súborov. Pre zlepšenie implementácii ďalších modelov UV pozadia je pridané rozhranie všeobecne definujúce tento model, ktoré reprezentuje abstraktná trieda `NightGlowModel`. Táto implementácia umožňuje „modularizáciu” modelov UV pozadia. Najpodstatnejšími metódami definovanými rozhraním sú metódy `NightGlowModel::NightGlowRate(...)` analogické s tými už implementovanými v triede `EusoElectronics`. Cieľom bola aj minimalizovanie úprav existujúceho kódu. V metóde `EusoElectronics::Build()` bolo pridané volanie metódy `NightGlowModel::Init()` a vytvorenie inštancie triedy je realizo-

vané, pre ESAF štandardne použitím „továrničky“ triedy typu *singleton*²³, `ElectronicsFactory`. V budúcnosti by táto abstraktná trieda mohla byť presunutá do inej logicky vhodnejšej knižnice, keďže ale generovanie UV pozadia je aktuálne pod knižnicou elektroniky, nachádzajú sa tu aj tieto triedy.

Implementácia tohto modelu je v triede `ExternalNightGlowModel`. V metóde `ExternalNightGlowModel::Init()` sa vykonáva načítanie súboru do operačnej pamäte - veľkosť súboru je prakticky len zopár desiatok kilobajtov. A záznam je reprezentovaný ako trojica: pozícia na osi x , pozícia na osi y a intenzita. Z dôvodu, že potrebná interpolácia je vykonávaná pri spustení simulácie pre celú ohniskovú plochu - každý čip prednej elektroniky (trieda `FrontEndChip`), je cieľom urýchliť vyhľadávanie položiek. Pre tento účel sú záznamy zoradené podľa hodnoty x a y a je vytváraný jednoduchý index hodnoty x podobný princípu primárneho indexu. Blok má variabilnú veľkosť, keďže záznam do indexu sa pridáva po určitom počte rozdielnych hodnôt x . Počet položiek s rovnakou hodnotou x nie je definovaný, no v spracovávanom súbore sa najčastejšie pre jednu hodnotu x nachádza jeden záznam, tri hodnoty x však majú až 33 záznamov. Vyhľadanie je realizované lineárne, keďže položiek nie je veľmi veľa. V prípade potreby je možné v budúcnosti implementovať binárne vyhľadávanie.

Priebeh vyhľadania položky je možné opísať v piatich krokoch:

1. Vyhľadanie najbližšej nižšej hodnoty x v indexe, hľadaním prvej vyššej hodnoty.
2. Vyhľadanie najbližšej nižšej hodnoty x , hľadaním prvej vyššej hodnoty.
3. Nájdenie nižšej a vyššej hodnoty y pre x z predošlého kroku.
4. Postupné znižovanie hodnoty x za cenu nájdenia podobnejšej hodnoty y . Index sa znižuje pokiaľ je hodnota x vzdialenosť od najbližšej x pod určeným limitom.

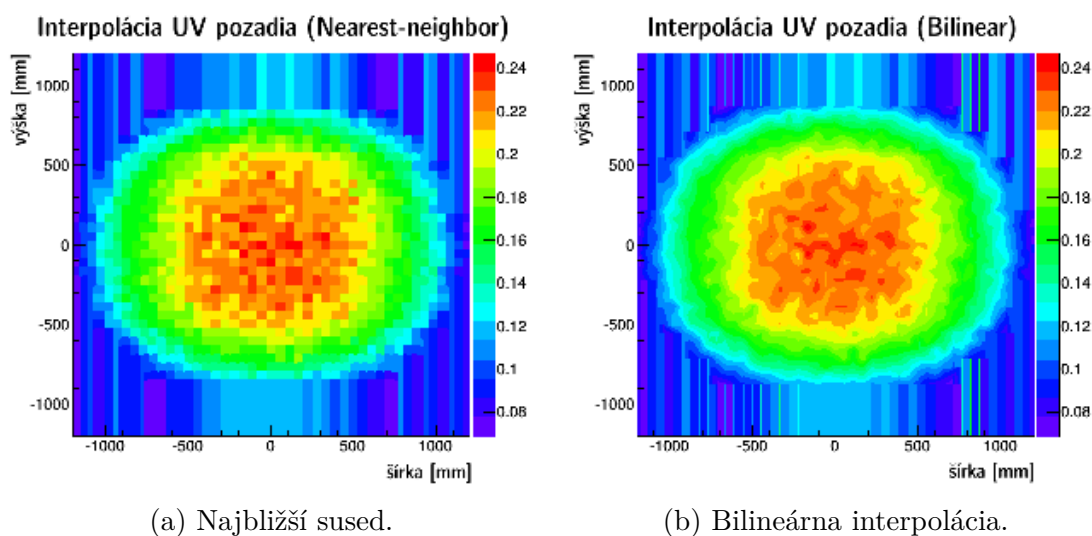
²³Návrhový vzor s cieľom umožniť vytvorenie iba jednej inštancie objektu danej triedy.

Analogicky pre vyššiu hodnotu x .

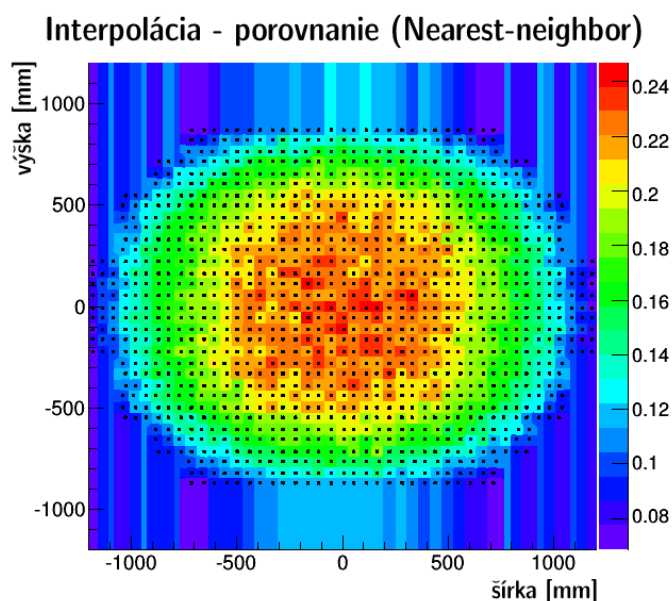
5. Hľadanie vo „vzdialenejšom stĺpci” ak nižšia a vyššia hodnota x je menej ako limit použitý v predošlom kroku.

8.1.4 Výsledky

Interpolácia je vyhodnotená interpoláciou intenzity pre celú plochu ohniskovej plochy. Vizualizácie na obrázku 8–2 boli vytvorené zaznačením výsledku interpolácie do dvojrozmerného histogramu. Interpolované boli pozície v rozmedzí od -1200 do 1200 s inkrementom 2 . Obrázky ukazujú, že základný princíp interpolácie je správny. Obrázok 8–3 ukazuje prekrytie bodov zo súboru a interpolovaných hodnôt pri metóde *Nearest-neighbor*.



Obr. 8–2: Vizualizácia interpolácie údajov zo súboru distribúcie UV pozadia na ohniskovej ploche.



Obr. 8–3: Distribúcia intenzity UV pozadia zo súboru - porovnanie interpolácie metódy *najbližší sused* s bodmi zo súboru.

Testy ukázali, že použitie indexovania pre vyhľadanie hodnoty bolo správne rozhodnutie. Testovalo sa použitím rovnakej metódy ako pre spomenutú vizualizáciu, s tým rozdielom, že uvedený výpočet bol realizovaný viac krát pre zvýšenie štatistiky. Tabuľka ukazuje porovnanie priemerných časov potrebných pre vykonanie prepočtov pre rovnaký výsek ohniskovej plochy. Výpočet bol skúšaný s viacerými veľkosťami indexovaného bloku (počet rôznych hodnôt x) a ukázalo sa, že okolo veľkosti bloku 20 sa pri použití tejto vstupnej množiny dosiahnú najlepšie výsledky. Tabuľka 8–1 ukazuje namerané časy spracovania.

Tabuľka 8–1: Časy spracovania interpolácie pre rovnakú oblasť FS pri použití rôznych veľkostí indexovaného bloku.

Veľkosť bloku	Priemerný čas [sekunda]
3	1,28
5	1,28
10	1,15

Pokračuje na ďalšej strane

Tabuľka 8–1 – Pokračovanie z predošlej strany

Veľkosť bloku	Priemerný čas [sekunda]
15	1,12
20	1,09
25	1,09
30	1,12
50	1,17
100	1,26
200	1,55
<i>bez indexovania</i>	3,95

8.2 Model UV pozadia

Komplikovanejšia úloha je implementácia modelu UV pozadia, teda modelu UV pozadia planéty, ktorý na základe geografickej pozície, nad ktorou sa detektor nachádza a času, v ktorom sa tam nachádza určí miestnu hodnotu UV pozadia. Následne je potrebné získať model distribúcie UV pozadia na citlivej ohniskovej ploche detektora (teda rovnaký výsledok, aký je použitý pre vstup v predošlom probléme) pre dané miestne pozadie. Komplikácia je, že model sa môže skladať z viacerých zdrojov popisujúcich menšie oblasti v rôznych časoch.

Ponúkajú sa dve cesty riešenia:

- implementácia samostatného programu, generujúceho model distribúcie UV pozadia na ohniskovej ploche detektora,
- implementácia rozšírenia existujúceho kódu softvérového rámca o možnosť voľby geografickej pozície a času pre simuláciu.

Riešenie použitím samostatného programu. Vstupné údaje programu by boli: databáza modelov pozadí pre rôzne geografické pozície a časy, geografická pozícia a čas. Výstup programu model distribúcie UV pozadia na ohniskovej ploche, ktorý by bolo možné použiť ako vstup pri použití funkcionality implementovanej v predošlej časti 8.1.

Riešenie rozšírením existujúceho kódu. Táto cesta je veľmi podobná ako predchádzajúca, ale zdrojový kód programu sa integruje do softvérového rámca ESAF. Výhodou môže byť, že nie je potrebné vytvárať extra súbory popisujúce distribúciu UV pozadia na ploche detektora, ale model sa vytvorí pred začatím procesu simulácie a uchová sa len ako objekt v operačnej pamäti. Čas, pozícia a cesta k databáze modelov pre vytváraný model by boli parametre konfigurácie. Toto riešenie sa javí používateľsky praktickejšie, ak sa uvažuje že daný model bude používaný len v softvérovom rámci ESAF.

Riešenie tejto úlohy má tri podúlohy: program pre generovanie testovacieho, implementácia metódy pre interpoláciu, integrácia do systému ESAF.

8.2.1 Špecifikácia modelu

Vyžadovaný model UV pozadia sa skladá z 528 „časopriestorových máp”, kde každá mapa je geografická mriežka s bunkami veľkosti 5 stupňov pre geografickú dĺžku a rovnako 5 stupňov pre geografickú šírku. Bunky sú centrovane od pozície 0°0°. Mapa pokrýva všetky dĺžky a šírky od -52,5 až 52,5, teda 21 šírkových „pasov”. Z pohľadu časovej dimenzie mapy sú definované v rozmedzí 11 rokov, pre každý rok sú vybrané štyri dni a deň (noc) má definovaných 12 hodín od 19:00 do 6:00. Intenzita je približne periodická v rámci dňa a v rámci roka.

8.2.2 Generovanie testovacích dát

Keďže zatiaľ neexistujú reálne dáta pre použitie v tomto modeli, je potrebné pre účely testovania nasimulovať dáta, ktoré sa budú v rámci testovania aproximovať. Pre určenie intenzity sa odvodila jednoduchá rovnica - základná myšlienka pre generovanie. Rovnica 8.4 ukazuje princíp výpočtu intenzity I , na ktorého základe môže generovanie testovacích dát prebiehať. Premenné Y , D , H , M reprezentujú rok, deň v roku, hodinu a uvedenom prípade aj minútu (M), čo zvyšuje členitosť grafu. Reálny model má najmenšiu časovú periódu hodinu. Konštanta b určuje základnú úroveň a konštanty k_1 , k_2 , k_3 , k_4 váhy jednotlivých časových periód.

$$I = b + P_y + (Y - 2000) * k_1 + (D - 1) * k_2 + H * k_3 + M * k_4 \quad (8.4)$$

Algoritmus generovania záznamov Myšlienkou tohto algoritmu je generovanie všetkých kombinácií dĺžky n hodnôt z n množín. Každá množina má rozdielnu dĺžku a je nutné, aby poradie prvkov v množinách bolo dodržané aj v poradí vytvorených kombinácií. Keďže sa pracuje s časom, pridáva sa aj potreba aspoň jednoduchšej korekcie „nižšej“ úrovne v prípade „pretočenia“ vyššej.

Príklad Máme 2 zoradené množiny $A = \{1, 2, 5\}$, $B = \{3, 4, 2\}$ potom chceme získať tieto množiny kombinácií ich prvkov v uvedenom poradí:

$$R = \{\{1, 3\}, \{1, 4\}, \{2, 2\}, \{2, 3\}, \{2, 4\}, \{3, 2\}, \{5, 3\}, \{5, 4\}, \{6, 2\}\}.$$

V tomto príklade sa ukazuje cielená funkcionalita algoritmu. Je možné pozorovať, že v kombinácii s číslom 2 z množiny B je k číslu z množiny A pripočítané 1. Príklad bol vybraný tak, aby nenastala jedna akceptovaná vlastnosť (chyba) algoritmu, možnosť straty zoradenia v prípade, že prvý prvok zoradenej množiny B je menej ako posledný, ktorý nie je najväčším prvkom danej množiny.

Princíp algoritmu Zjednodušene si je možné algoritmus predstaviť takto - úrovne sú organizované do stĺpcov, kde riadky sú jednotlivé položky zoznamov. Postupným znižovaním úrovne (stĺpca) a zvyšovaním indexu položky (riadku) sa čítajú položky od pravého horného okraja. Ak sa došlo na poslednú položku v úrovni, prejde sa na ďalšiu položku v nižšej úrovni a čítanie sa vráti k pravému hornému okraju. Princíp algoritmu je podrobnejšie vysvetlený v systémovej príručke.

8.2.3 Interpolácia hodnoty pozadia z modelu

Ako bolo spomenuté, záznamy v modele sú len v určitých rokoch, dňoch a hodinách. Preto bolo cieľom navrhnúť univerzálny algoritmus, ktorý by umožnil získať úroveň pozadia z modelu pre akýkoľvek čas. Je potrebné myslieť na to, že sa neinterpolujú najbližšie body, ale „najpodobnejšie” body. Komplikáciou algoritmu bola aj snaha navrhnúť ho univerzálne pre akékoľvek úrovne periodicity.

Princíp algoritmu Prvým krokom je nájdenie najbližšej nižšej a vyššej hodnoty, k vstupnej časovej známke prechodom všetkých položiek. Ak je v tomto kroku nájdená rovnaká hodnota ako hľadaná, výsledok je táto hodnota. Druhým krokom je volanie funkcie *FindInterpolate* pre obe nájdené hodnôty.

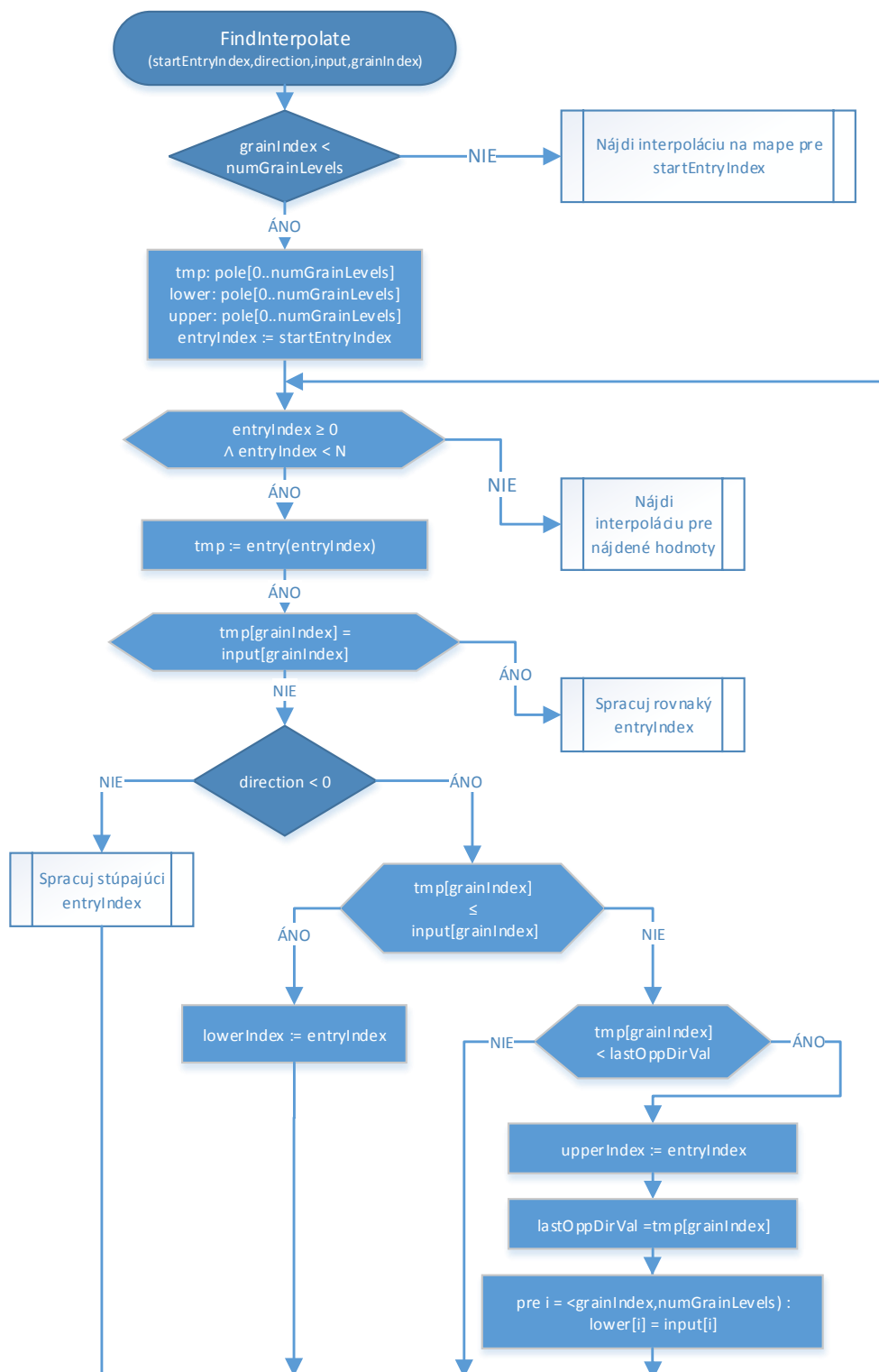
Pri návrhu tejto verzie algoritmu bolo cieľom minimalizovať počet prechodov prvkami a priamo ne byť viazaný na potrebu kopírovania všetkých záznamov do operačnej pamäte, keďže sa predpokladá, že celý model môže byť priveľký pre uloženie v operačnej pamäti.

Funkcia FindInterpolate Vstupnými parametrami funkcie *FindInterpolate* sú: začiatkový index, od ktorého bude čítať položky (`startEntryIndex`), smer čítania (`direction`), hľadaný dátum `arrTimeGrainsInput`, a „časové zrno” (napr. rok, deň,

hodina, `grainIndex`), ktoré sa porovnáva. Myšlienkou, na ktorej je táto funkcia založená je nájdenie najpodobnejších hodnôt k hľadanej hodnote na úrovni určenej vstupným parametrom. Ak je nájdená nižšia a vyššia najpodobnejšia hodnota, funkcia je rekurzívne volaná s vyššou úrovňou „časového zrna“ pre obe nájdené. Vstupom pre rekurzívne volanie je nájdený čas, ktorého vyššie úrovne ako aktuálna sú nastavené na hľadaný čas. Volania funkcie sa takto vnárajú, až pokiaľ je porovnávaná úroveň vyššia (pri indexovaní od 0 rovná) ako počet úrovní, v tomto prípade je medzi bodmi urobená lineárna aproximácia. Hodnoty pre aproximáciu sú prečítané z mapy v pozícii vo vstupe algoritmu. Parameter smeru čítania určuje, či sa index porovnáwanej položky pri hľadaní zvyšuje alebo znižuje, tento smer ostáva zachovaný aj v rekurzívnych volaniach.

Komplikácia, ktorú musí algoritmus riešiť je, že napríklad hodiny sú pre deň v modeli definované v rozmedzí 19:00 až 6:00, začínajúc na dni 15 a končiac na dni 16. Hľadaná hodina je 23. Teda aj keď je pre nižšiu úroveň bližší deň číslo 16, cieľom je nájsť záznam z dňa 15, pretože vtedy je najpodobnejšia hodina definovaná. Riešenie teda vyžaduje možnosť pokračovať v hľadaní, aj keď už zjavne najbližšia hodnota bola nájdená.

Obrázok 8–4 v zjednodušenej podobe ilustruje hľadanie najpodobnejšieho záznamu. V diagrame sú oproti skutočnej verzii vynechané napríklad kontroly pokračovania alebo hľadanie jednej úrovne nižšie.

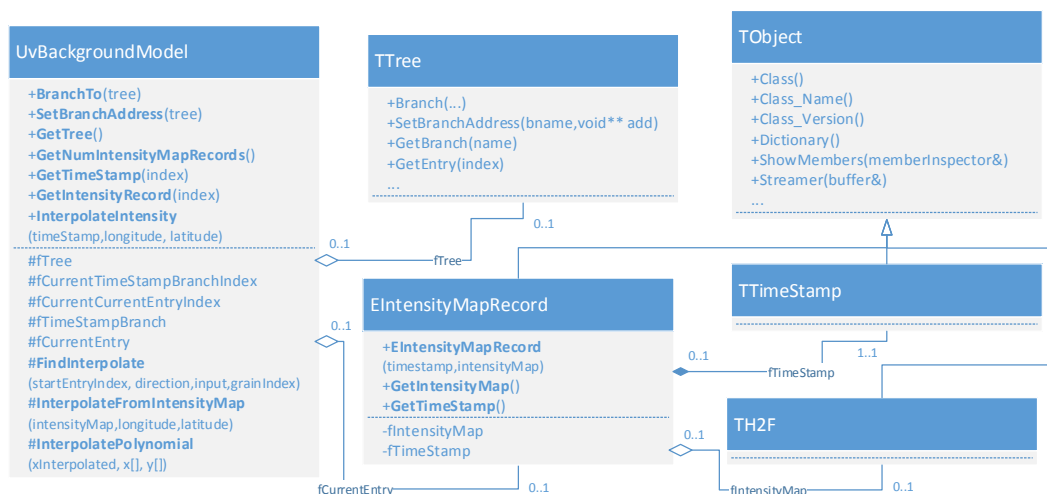


Obr. 8–4: Vývojový diagram v zjednodušenej podobe ilustrujúci funkciu *FindInterpolate*.

8.2.4 Implementácia

Návrh modelu Pre implementačný návrh modelu bola využitá stromová štruktúra, resp. zoznam stromových štruktúr, softvérového rámca ROOT - trieda `TTree`. Výhodou tohto rozhodnutia je odstránenie potreby riešenia serializácie objektu pri jeho ukladaní do súboru. Tento princíp je pre programy založené na ROOT-e typický, využívaný napríklad aj pre uchovávanie výstupu simulácie a rekonštrukcie ESAF-u - triedy `EEvent`, `RecoEvent`. V prípade modelu UV pozadia je touto triedou trieda `EIntensityMapRecord`. Trieda má dve dôležité členské premenné - časovú známku `fTimeStamp` a mapu `fIntensityMap` reprezentovanú dvojrozmerným histogramom triedy `TH2I`. Použitie triedy histogramu najmä zjednodušilo implementáciu a ponúklo existujúcu metódu interpolácie pozície. Ak by bola potreba, stále je možné naprogramovať vlastnú metódu, čítaním dát z tohto histogramu, nevyžadujúc konverziu súborov modelu. Prácu s modelom, najmä vyhľadávanie záznamov, zaobahuje trieda `UvBackgroundModel`. Algoritmus bol implementovaný zatiaľ ako metóda v tejto triede, v prípade potreby je ho možné osamostatniť. Obrázok 8-5 ilustruje tieto triedy použitím diagramu tried.

Tieto triedy boli pridané do knižnice detektora softvérového rámca ESAF. K detektoru boli pridané preto, lebo v aktuálnej verzii je simulácia pozadia riešená na tomto mieste. Polohu a čas, pre ktoré sa bude pozadie simulovať je možné nastaviť v konfigurácii.

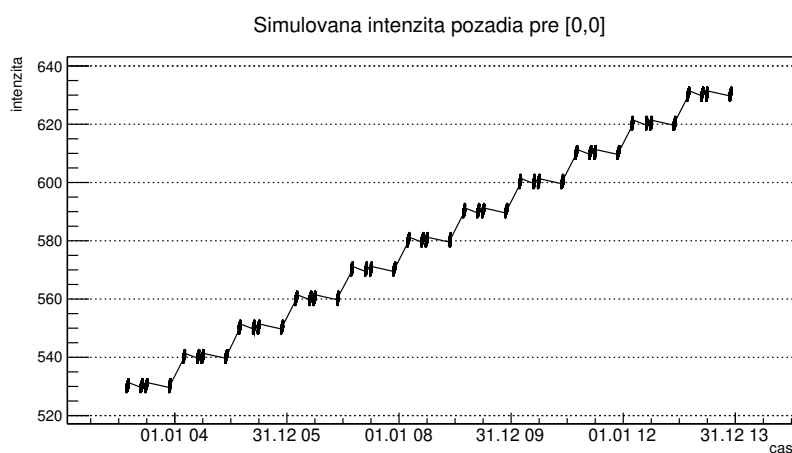


Obr. 8 – 5: Diagram tried navrhnutých pre reprezentáciu UV modelu pozadia.

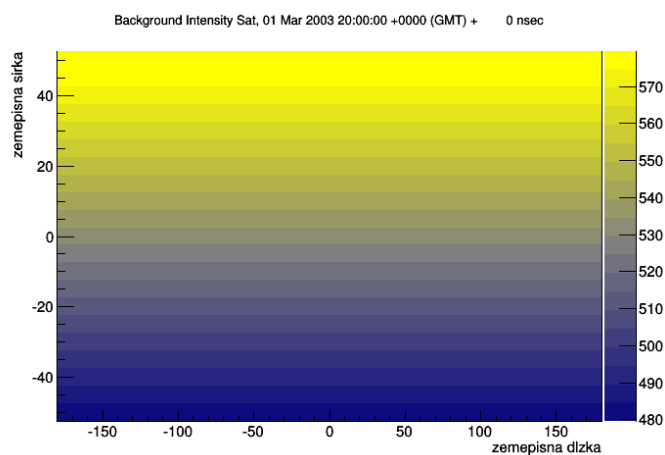
Model UV pozadia v ESAF Použitie tohto modelu je zaobalené v triede `UvBgNightGlowModel` implementujúcej abstraktnú triedu `NightGlowModel` opísanú v časti 8.1.3. Proces interpolácie nastáva pri inicializácii modulu v metóde `UvBgNightGlowModel::Init()`, teda pri inicializácii simulácie.

Generovanie testovacích záznamov Pre generovanie testovacích záznamov bolo vytvorené makro `GenerateFakeUvBg.C`. Záznamy sú pridávané ako položky zoznamu stromov reprezentovaného triedou `TTree`, použitím opísanej organizácie. Tento zoznam položiek je zapísaný do výstupného súboru.

Obrázok 8–6 ilustruje graf tejto funkcie vygenerovaný z údajov vygenerovanými pri použití rovnice 8.4. Intenzita v rámci jednej mapy sa mení lineárne s pozíciou y , teda geografickou šírkou. Toto ilustruje obrázok 8–7



Obr. 8–6: Intenzita pozadia generovaného testovacieho modelu závislá od času.

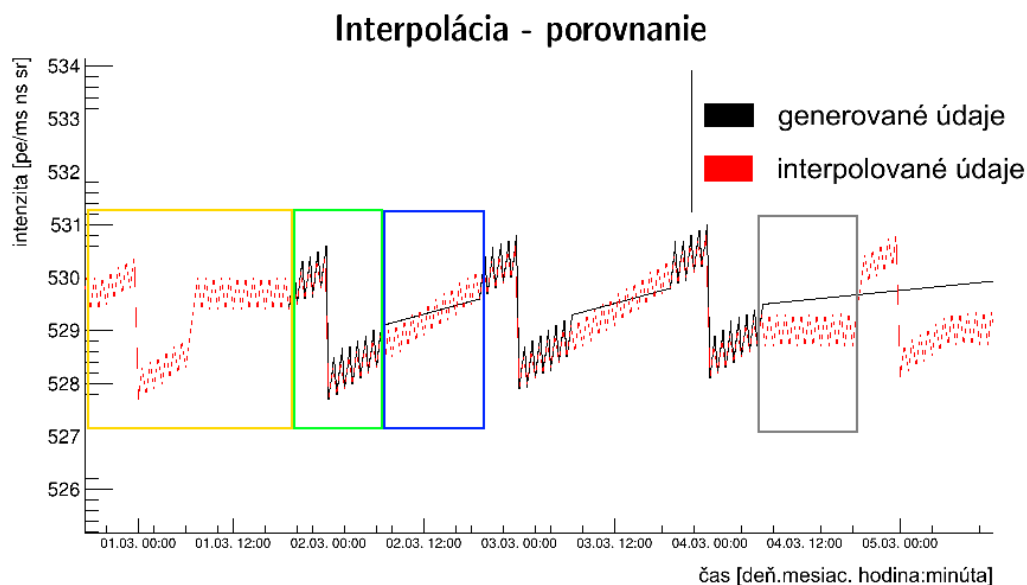


Obr. 8–7: Intenzita pozadia generovaného testovacieho modelu závislá od geografickej pozície.

8.2.5 Výsledky

Keďže reálne dáta neboli dostupné, bola vyhodnotená implementácia algoritmu interpolácie použitím generovaných údajov z obrázku 8–6. Interpolované boli časy s rozdielom 30 minút. Obrázok 8–8 ukazuje 4 dni. Z tohto obrázku je možné pozorovať vlastnosti algoritmu. Pre „znáročnenie“ úlohy boli použité štyri „časové zrná“ - rok, deň v roku, hodina, minúta.

- Do hodnoty 1.3. 19:00, kde neexistuje nižšia hodnota pre interpoláciu, sú použité najpodobnejšie vyššie záznamy (žltá oblasť na obrázku).
- V bodoch, kde sa nachádzajú skutočné údaje, výsledok je rovnaký (zelená oblasť na obrázku).
- V rozmedzí 2.3. 6:00 až 2.3. 19:00 sa interpolujú hodnoty medzi dvoma najpodobnejšími známymi hodnotami - nižšou a vyššou²⁴ (modrá oblasť na obrázku).
- Navrhovaný algoritmus v akrátnej podobe nedáva úplne optimálne výsledky, ak sú dve úrovne neznáme napríklad v časovom rozmedzí 4.3. 7:00 až 5.3. 19:00 (sivá oblasť na obrázku).



Obr. 8–8: Interpolácia intenzity na generovaných údajoch.

²⁴V skutočnosti v tomto konkrétnom časovom rozmedzí nie je správne simulovať sprišky, keďže v tomto čase je deň.

9 Záver

Hlavnou časťou práce bola implementácia, analýza a testovanie modulov rozpoznávania obrazcov spíriek zaznamenaných na prednej elektronike detektora. Postupne boli implementované viaceré inkrementálne vylepšované verzie algoritmov. V tejto práci boli opísané štyri takéto verzie. Prvá základná verzia (označovaná ako „nultá“) slúžila najmä ako nástroj pre pochopenie problematiky a nedosahovala akceptovateľné výsledky. Vylepšenie, ktoré bolo pridané do tohto algoritmu je založené na analýze závislosti pozície a *GTU* pixela. Získané skúsenosti boli využité pri implementácii dvoch finálnych verzií algoritmov. Tieto moduly rekonštrukcie dosiahli vyššiu alebo porovnateľnú účinnosť ako pri metódach doteraz používaných JEM-EUSO kolaboráciou pri analýze spířok tvorených UHECR časticami. Stanovený sekundárny cieľ implementovať algoritmy rekonštrukcie efektívnejšie ako v dodaných Java aplikáciách bol splnený pri druhej verzii algoritmu, ktorá bola „stredobodom záujmu“ práce.

Pre analýzu výsledkov rekonštrukcie boli vytvorené programy, ktorých výstupom je výpočet a vizualizácia štatistík. Tieto a iné spravidla makrá pre softvérový rámec ROOT, zlepšili použiteľnosť ESAF-u, keďže pred touto prácou takéto nástroje neexistovali.

Prínosom sú aj implementované moduly aplikácie modelu UV pozadia vyvíjaného na UEF SAV. Modul umožňuje prenos výsledkov modelu do ESAF-u a ich následné využívanie používateľmi softvérového rámca, do ktorého bola pridaná abstraktná reprezentácia modelu UV pozadia využiteľná aj pre implementácie ďalších modulov. Práca na moduloch bude ďalej pokračovať a navrhnuté metódy interpolácie budú ďalej vylepšované.

Úloha prepisu knižníc generovaných UV pozadií bola riešená viacúčelovým vstupným modulom rekonštrukcie. Pri riešení tohto problému bola vytvorená metóda prepočtu

súradníc pixelov na identifikátory, ktorá je širšie aplikovateľná aj mimo ESAF. Navyiac tento modul umožňuje čítanie dát prednej elektroniky detektora JEM-EUSO z rôznych textových súborov - možnosť výberu stĺpcov a ich poradia. Ďalším prínosom implementácie je vyriešenie problému tvorby vstupných modulov rekonštrukcie pridaním triedy `RecoEventBuilder`. Analýza šumu použitím tohto modulu ukázala vlastnosť implementovaných algoritmov rekonštrukcie vzoru pomocou Houghovej transformácie, ktorých si musí byť používateľ vedomý ²⁵.

Implementované moduly budú ešte intenzívne testované v inštaláciách na počítačovom klastru SAV UEF a niektoré, najmä modul rekonštrukcie vzorov využívajúci Houghovu transformáciu, budú odovzdané do globálneho repozitára softvérového rámca ESAF. Neskôr, potom ako bude modul UV pozadia, vyvíjaný na pracovisku SAV UEF vytvorený, bude aj tento model spolu s príslušným modulom ESAF-u odovzdaný do globálneho repozitáru. Všetky „vedľajšie“ moduly a makrá sú umiestnené a inštalované na počítačoch klastra a sú využívané pri práci so softvérovým rámcom ESAF.

Budúcnosť využitia ESAF-u v kolaborácii JEM-EUSO je otázna. Kolaborácii bolo umožnené používať softvérový rámec Offline využívaný v experimente Auger zameraný na rovnakú fyziku ako experiment JEM-EUSO. Systém Offline má viaceré výhody, vrátane aktívneho vývojárskeho tímu, preto existuje snaha o prechod naň. Ale aj napriek tomu, že ESAF možno nemá jasnú budúcnosť, implementované algoritmy bude možné pretransformovať do softvérového rámca Offline, keďže v mnohom, vrátane využívaného programovacieho jazyka C++ a používania softvérového rámca ROOT, sú si podobné.

²⁵Tieto algoritmy majú pri nastavení, ktoré je vhodné pri rekonštrukcii spršok, schopnosť vybrať pixely aj zo šumu. Pozri časť 7.4.

Literatúra

- ADAMS J.H. et. al. 2013. *An evaluation of the exposure in nadir observation of the JEM-EUSO mission*. In: *Astroparticle Physics*, 2013, vol. 44. pp. 76–90.
arXiv:1305.2478.
- BERAT, C. et al. 2010. *ESAF: Full Simulation of Space-Based Extensive Air Showers Detectors*. In: *Astroparticle Physics*, 2010, vol. 33, no. 4, pp. 221–247.
arXiv:0907.5275
- BIKTEMEROVA, S. et al. 2013. *Pattern recognition and direction reconstruction for JEM-EUSO experiment*. In: *Proceedings of the 33rd International Cosmic Ray Conference*. Rio de Janeiro, 2013. pp. 63–66. arXiv:1307.7071v1
- BIKTEMEROVA, S. et al. 2013. *Simulations and the analysis of fake trigger events background in JEM-EUSO experiment*. In: *Proceedings of the 33rd International Cosmic Ray Conference*. Rio de Janeiro, 2013. pp. 59–62. arXiv:1307.7071v1
- BIKTEMEROVA, S. et al. 2013. *Performances of JEM-EUSO: angular reconstruction*. In: *Experimental Astronomy*. Springer Netherlands, 2014. 25 s.
ISSN: 0922-6435
- BLASCHKE, F. 2009. *Analýza korelovaných sprsiek kosmického záženia*.
Diplomová práca. Opava: Slezská univerzita v Opavě FPU, 2009. 103 s.
- BOBÍK, P. 2013. *UV spectrum on the Earth night side – ver. 1*. Interný dokument.
Košice: Ústav experimentálnej fyziky SAV, 2013.
- DAŇO, I. - OSTERTAGOVÁ, E. 2010. *Numerické metódy, pravdepodobnosť a matematická štatistika*. Prvé vydanie. Košice: EQUILIBRIA, 2010. 166 s.
ISBN 978-80-89284-56-6
- DUDA R. O. - HART P. O. 1972. *Use of the Hough Transformation To Detect Lines*

-
- and Curves in Pictures* In: Communications of the ACM, 1972, vol. 15, no. 1, pp.11–15
- EBISUZAKI T. et al. 2010. *Report on the phase A study 2010*.
Riken: JEM-EUSO Collaboration, 2010. 437 s.
- FENU, F. 2013. *A Simulation Study of the JEM–EUSO Mission for the Detection of Ultra–High Energy Cosmic Rays*. Dizertačná práca.
Tübingen: Eberhard Karls Universität Tübingen, 2013.
- GRIEDER, K.F. Peter 2010. *Extensive Air Showers. Volume I*. Prvé vydanie.
New York: Springer, 2010. 1118 s. ISBN 978-3-540-76941-5
- GUZMAN, A. et al. 2013. *The Peak and Window Searching Technique for the EUSO Simulation and Analysis Framework: Impact on the Angular Reconstruction of EAS* In: Astroparticle Physics, 2013, vol. 409, no. 1
- HÜFNER, S. 2003. *Photoelectron Spectroscopy: Principles and Applications*.
New York: Springer, 2012. 662 s. ISBN 3-540-41802-4.
- KALMAN, R. E. 1960. *A New Approach to Linear Filtering and Prediction Problems*. In: Journal of Basic Engineering, 1995, vol. 82, no. 1, pp. 35–45
- LEINERT, CH. et al. 1998. *The 1997 reference of diffuse night sky brightness*. In: Astronomy and Astrophysics Supplement, 1998, vol. 127, pp.1–99
- MATO, P. et al. 2014 *Root - Data Analysis Framework*. [online]. [cit. 20.12.2014].
Dostupné na internete: <<http://root.cern.ch>>
- MERNIK, T. et al. 2012. *Angular Reconstruction and Pattern Recognition: Expected spatial resolution performance of JEM-EUSO*. 12th JEM-EUSO international collaboration meeting at ICRR & RIKEN. Prezentácia.
-

- MUSSER, D. R. 1997. *Introspective Sorting and Selection Algorithms*. In: Software Practice and Experience, 1997, vol. 27, no. 8, pp. 983–993
- PASTIRČÁK, B. - FENU, F. et al. 2012. *Fake trigger background simulation*. JEM-EUSO Simulation meeting Madrid Spain. Prezentácia. Madrid, 2012.
- PAVLLAVICINI, M. - THEA, A. 2004. *Euso Simulation and Analysis Framework: User Guide*. Interný dokument.
- STAROŇ, M. 2013. *Algoritmy pre rozpoznávanie obrazcov pri štúdiu kozmického žiarenia ultravysokých energií v rámci JEM-EUSO experimentu*. Diplomová práca. Košice: Technická univerzita v Košiciach FEI, 2013. 56 s.
- STEINGARTNER W. 2013. *Sémantika programovacích jazykov: 9. prednáška*. Prezentácia. Košice: Technická univerzita v Košiciach FEI, 2013.
- VASILKO, J. 2015. *Aplikácia Houghovej metódy rozpoznávania obrazcov pre detekciu sprášok tvorených časticami ultravysokých energií*. Diplomová práca. Košice: Technická univerzita v Košiciach FEI, 2015.

Zoznam príloh

Príloha A Používateľská príručka

Príloha B Systémová príručka (len na CD)

Príloha C CD médium

Príloha D Článok popisujúci prácu