

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Prototyp komponenty pro vizualizaci dat z částicových detektorů

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:

Jiří BUŇATA

Osobní číslo:

A21B0603P

Studijní program:

B3902 Inženýrská informatika

Studijní obor:

Informační systémy

Téma práce:

Prototyp komponenty pro vizualizaci dat z částicových detektorů

Zadávající katedra:

Katedra informatiky a výpočetní techniky

Zásady pro vypracování

1. Seznamte se s problematikou částicových detektorů a podobou dat která z nich jsou získávána.
2. Seznamte se s vhodnými metodami zobrazování dat, zaměřte se na metody přímého zobrazování objemových dat.
3. Vyberte alespoň dva postupy které by mohly být vhodné pro zobrazení získaných dat.
4. Navrhněte podobu vizuální komponenty pro zvolené metody.
5. Vybrané metody implementujte, s důrazem na možnost snadné konfigurace jejich parametrů.
6. Otestujte vytvořenou implementaci, zejména s ohledem na uživatelskou použitelnost. Vyhodnotěte přínosy a nevýhody zvolených metod.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Richard Lipka, Ph.D.**
Katedra informatiky a výpočetní techniky

Konzultant bakalářské práce: **Ing. Jan Broulím, Ph.D.**
ČVUT (Ústav Technické a Experimentální Fyziky)

Datum zadání bakalářské práce: **30. srpna 2022**

Termín odevzdání bakalářské práce: **4. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 4. května 2023

Jiří Buřata

Poděkování

Dkuji Ing. Janu Broulímovi, Ph.D. zadobré rady a vstícnost při validování postupu práce a následném testování. Dále dkuji Ing. Richardu Lipkovi, Ph.D. za jeho odborné vedení bakalářské práce a as, který mi v noval na konzultacích.

Abstract

The goal of this bachelor thesis is to develop a component for visualization of particle detector data. The main requirement for this component is to process this data and visualize them by using visualization method focused on rendering data in time. For experimental purposes, two visualization methods are implemented and are compared in the conclusion of the work and their gain is evaluated. These components can be used in other projects and applications for the purpose of real time data visualization.

The greatest contribution of the work lies in the creation of new ways of visualization of particle detector data with possibility to analyse their development in time.

Abstrakt

Cílem této bakalářské práce je vyvinout vizualizační komponentu pro data z čisticových detektorů. Hlavním požadavkem na komponentu je zpracování a vizualizace takových dat pomocí zvolené metody zaměřené na zobrazování dat v reálném čase. Vzhledem k experimentální povaze práce jsou implementovány dvě vizualizační metody, které jsou v závěru práce porovnány a je vyhodnocen jejich přínos. Obě metody jsou implementovány jako programové komponenty, které lze připojit do jiných projektů a aplikací sloužících k vizualizaci dat z čisticových detektorů.

Největší přínos práce spočívá ve vytvoření nových způsobů vizualizace dat z čisticových detektorů s možností analyzovat jejich vývoj v reálném čase.

Obsah

1	Úvod	8
2	Částicové detektory a data, která jsou z nich získávána	9
2.1	Částicové detektory	9
2.2	Data získaná z detektor	10
2.3	Formáty výstupních dat	13
2.3.1	Formát Time Frame	13
2.3.2	Formát Data Driven	14
3	Metody zobrazování dat z částicových detektorů	16
3.1	Souasné vizualiza ní metody	16
3.1.1	Vizualizace 2D dat	16
3.1.2	Vizualizace se zam ením na znázorn ní konkrétních jev v datech	19
3.1.3	Zobrazení dat v ase	20
3.2	Zvolení vhodných zobrazovacích metod pro komponenty	22
3.2.1	2D metoda	22
3.2.2	3D metoda	23
4	Návrh komponenty	26
4.1	Požadavky na komponentu	26
4.2	Základní pojmy	27
4.3	Volba technologie	27
4.3.1	AWT a Swing	28
4.3.2	JavaFX	28
4.4	Architektura komponenty	31
4.4.1	Popis základních struktur JavaFX	31
4.4.2	Navržená architektura	35
4.5	Komunikace mezi komponentou a aplikací, která ji využívá	36
4.6	Datový model	38
4.6.1	Uložení dat	38
4.6.2	Architektura datového modelu	42
4.6.3	Na ítání ze souboru a operace grouping	42
4.6.4	Propojení datového a vizualiza ního modelu	44
4.7	2D metoda	45
4.8	3D metoda	47

4.8.1	Výpo et sou adnic pro osu Z	49
4.8.2	Kamera	51
5	Implementace komponenty	53
5.1	Architektura	54
5.1.1	Architektura obecn	54
5.1.2	Hlavní t ídy komponenty	54
5.2	Datový model	56
5.2.1	Struktura a fungování datových operací	56
5.2.2	Uložení dat	57
5.2.3	Na ítání dat ze souboru	58
5.2.4	Grouping	59
5.2.5	D ležité hodnoty komponenty	62
5.3	Okno Settings	63
5.4	2D Vizualizace	63
5.5	3D Vizualizace	65
5.6	Barevný model	68
5.7	DataIndexBar	69
5.8	P edávání výsledk akcí v komponent	69
5.9	Demonstra ní aplikace	70
6	Testování	72
6.1	Programové testování	72
6.2	Uživatelské testování	72
7	Závěr	75
Literatura		77
I Příloha A - Uživatelská příručka		86
I.1	Vizualiza ní komponenta	86
I.1.1	P idání komponenty do projektu	86
I.1.2	Použití komponenty v projektu	87
I.2	Demonstra ní aplikace	92
I.2.1	P eklad a sestavení	92
I.2.2	Spušt ní	92
I.2.3	Vzhled aplikace a popis jednotlivých prvk	92
I.3	Ovládání	94
I.4	2D vizualizace	96
I.5	3D vizualizace	98
I.6	Nastavení parametr spole ných pro ob ásti	100

1 Úvod

Cílem této práce je navrhnut a vytvořit programovou komponentu, která bude umožnit zpracovat a vizualizovat data z ásticových detektorů. Motivací k vytvoření této bakalářské práce je vyvinout nový způsob vizualizace dat z ásticových detektorů s možností analyzovat vývoj dat v reálném čase. Aktuálně už existují nástroje na vizualizaci takových dat, ale v těchto nástrojích jsou data z konkrétních experimentů. Jedná se o experimentální práci, jejíž výsledkem bude vytvoření nové vizualizační metody pro data z ásticových detektorů, která umožní analyzovat jejich vývoj v reálném čase. Součástí výsledku bude i zhodnocení, zda vytvořená vizualizace je využitelná a jaké jsou její přínosy a nevýhody. Finálním produktem budou dvě komponenty. Každá bude implementovat jinou vizualizační metodu, a každá komponenta bude vytvořena demonstrativní aplikací komponentu využívající.

Nejprve je potřeba prozkoumat podobu a význam dat získávaných z ásticových detektorů. Následně prozkoumat možnosti vizualizace těchto dat. Na základě výsledků zkoumání budou vybrány dva zobrazení dat souběžně na zobrazení dat v reálném čase. Poté bude pro tyto metody navržena implementace pro konkrétní programové řešení. V rámci návrhu je nutné zohlednit, aby výsledný program byl použitelný jako samostatná komponenta a propojitelný do dalších projektů. Navržené řešení bude následně implementováno v konkrétní zvolené technologii a bude k tomu vytvořena aplikace využívající komponentu a demonstруjící její funkce. Aplikace bude využita k otestování zamýšleného řešení na správnost navrženého řešení a uživatelskou přivitost. Podmínkou je, aby komponenta a aplikace byla vytvořena v programovacím jazyce Java, minimální verze 8. Volba dalších technologií pro grafické uživatelské rozhraní a vizualizaci jsou na volbu uživatele práce.

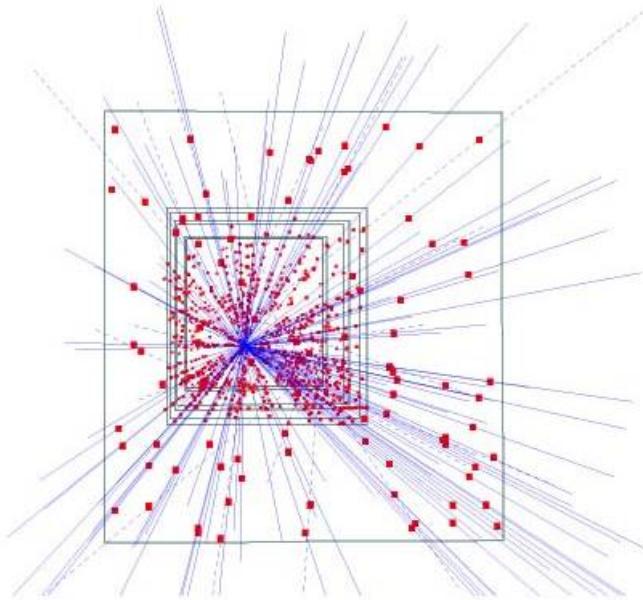
2 Částicové detektory a data, která jsou z nich získávána

Tato část je zaměřena na obecný popis čisticových detektorů. Nezaměřuje se na provedení detektoru z technologického i fyzikálního hlediska, popisována je jejich všeobecná funkcionality a využitelnost tak, aby tená získal ucelenou představu o tom, co jsou čisticové detektory a jaké informace z nich lze získávat. Pro tuto kapitolu byly použity informace původně zdroj zabývajících se čisticovými detektory nebo byly poskytnuty původně zadavatelem práce panem Ing. Broulímem Ph.D.

2.1 Časticové detektory

čisticový detektor je zařízení sloužící k detekci a identifikaci čistic a lze pomocí něj i určit jejich dráhu. Využívají se k fyzikálnímu výzkumu, například vesmíru i materiálu, velké využití mají i v lékařství, konkrétně v radiologii a tomografii. V rámci této práce bude uvažován takzvaný pixelový typ detektoru. Ten ukládá naměřená data jako jednotlivé pixely do rastrového obrázku. Pro představu lze detektor přirovnat k CCD čipu používaných v digitálních kamerách. [28] Když částice zanechá náboj, který je menší, tak ho detektor zaznamená. Zaznamenané náboje jsou reprezentovány množinou pixelů a ukládány do rastrového obrázku. Je to ilustrováno na obrázku 2.1. Na obrázku je zobrazeno 153 čistic letících tunelem. Uzavřené pixely reprezentují právě ty pixely, které byly zasaženy nabíjatou česticí. Černý rám, ve kterém jsou obsaženy, představuje as, ve kterém byly náboje naměřeny. Rám je na obrázku více, to znamená, že jsou zaznamenána data z několika asových známk. Detektor lze natočit pod úhlem v průseku tunelu od 0 do 90 stupňů, rozdílný úhel natočení pomoci zachytit různé vlastnosti častic se stejnou dráhou se pak při různých natočeních zaznamenají jinak. Například při natočení 0 stupňů se častic letící kolmo na detektor zaznamenají jako shluk pixelů podobný kruhu nebo tverci, při natočení blížící se 90 stupňů při stejném směru bude již ale zaznamenána jako táhlá řáda. Natočení detektoru lze využít například pro zaznamenávání druhů čistic. Některé detektory umí zaznamenat i energii, jakou časticí mají v momentu přechodu detektorem. Energie je reprezentována barvou pixelu. Barevná škála a význam jednotlivých barev závisí na každém detektoru.

Příkladem pixelového typu detektoru mohou být například Medipix a Timepix používané v oblasti fyzikálního výzkumu a vytvořené v Evropské organizaci pro jaderný výzkum (CERN). [9]

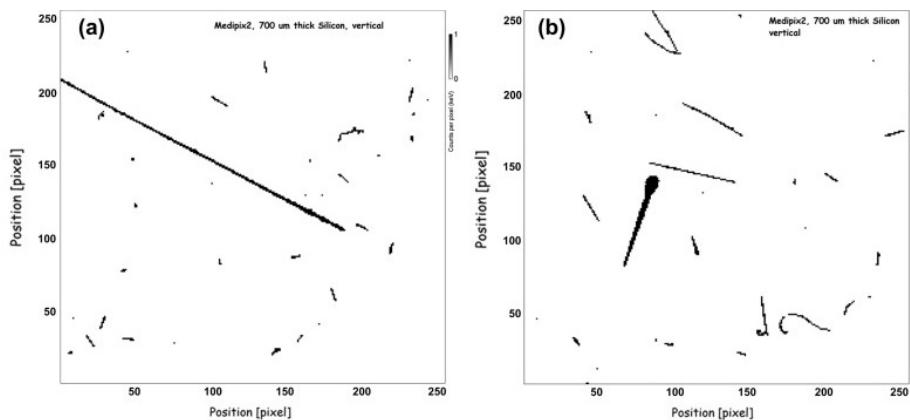


Obrázek 2.1: Znázornění pixelového detektoru [27]

2.2 Data získaná z detektorů

Tato část je zaměřena na obecný popis dat získávaných detektory, jak vypadají a jaký mají význam. K popisu jsou použity obrázky s jednoduchými možnostmi vizualizace, detailněji jsou pak současně metody vizualizace rozepsané v kapitole 3. Namířená data jsou ukládána jako pixely do 2D rastrového obrázku, kde jednotlivé hodnoty pixelů reprezentují energii v daném bodě. Obrázek může být tvercový nebo obdélníkový, podle konkrétního detektoru. Například Medipix1 používá 64x64 a Medipix2 256x256. [3] Nulová hodnota znamená, že v daném bodě nebyl zaznamenán žádný náboj. Ne-nulová naopak znamená, že v bodě byl náboj naměřen. Namířená energie je reprezentována barvou pixelu. Tato hodnota se nazývá *Time over Threshold*. Nutno poznat, že pouze reprezentuje danou energii v bodě, ale nelze ji použít přímo jako energii vyjadřitelnou ve fyzikálních jednotkách. Na to je potřeba ji převést, metoda výpočtu se liší podle konkrétního detektoru, tím se ale v této práci nebudeme zabývat. Pro vizualizaci použita samotná hodnota *Time over Threshold*, která poslouží k určení finální

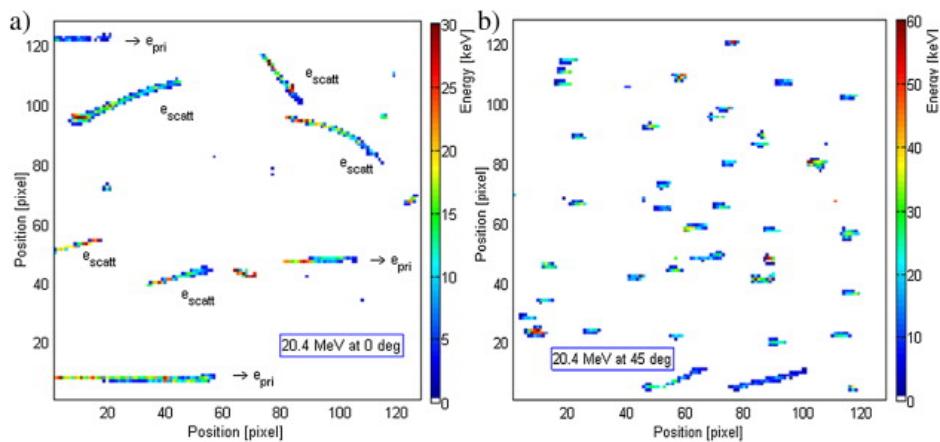
barvy vizualizovaného prvku. Tím v tří má hodnotu, tím v tří energii reprezentuje. Maximální možná hodnota též závisí na konkrétním detektoru. Poslední dležitou informací je asová známka udávající, v jakém asu byla data naměna. Nazývá se *Time of Arrival* a udává se v různých asových jednotkách, například v mikro, nano nebo pikosekundách. Množina pixelů s naměnou energií, které jsou sdruženy podle jejich vlastnosti, se nazývá *energy bin* nebo zkráceně *bin*. V rámci této práce budou sdružovány pixely podle asu. Bin tedy bude obsahovat pixely se stejnou asovou známkou. Způsob uložení dat je pak závislý na konkrétním formátu. Dále budou ukázány příklady na kterých dat a jejich popis.



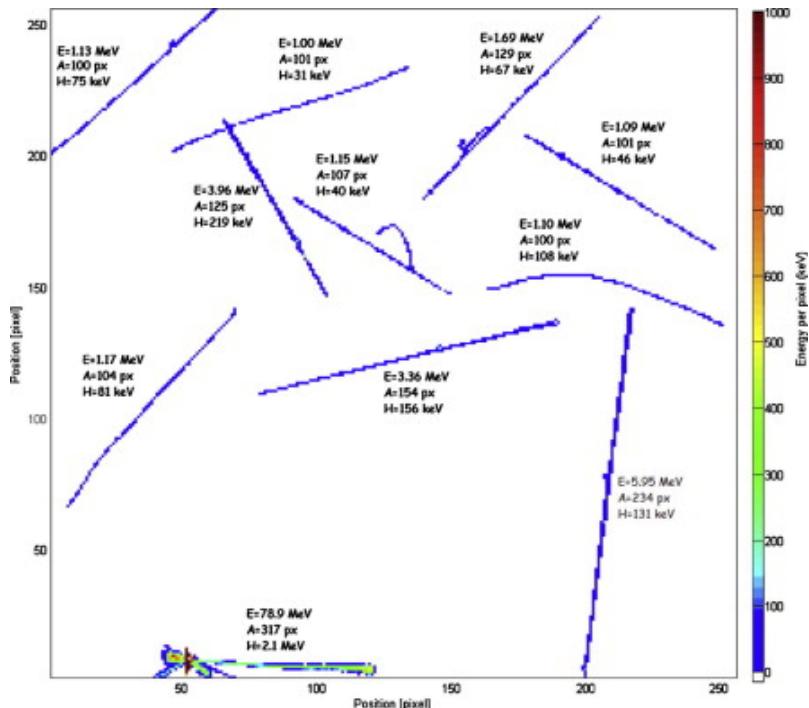
Obrázek 2.2: Příklad jednoduché vizualizace dat [12]

Na obrázku 2.2 se nachází obecný příklad vizualizace dat. Černé pixely znázorují náboje, které byly zaznamenány, a jejich polohu. V tomto příkladu není znázorněna energie, černá barva není v odstínech. Pouze udává informaci, že v daném místě byl naměněn náboj. U některých částic lze určit i jejich směr v detektoru. Například na levém obrázku lze vidět delší čáru, která vede zhruba z levého horního rohu do středu obrázku. To znamená, že její směr byl téměř rovnoběžný s detektorem.

Na obrázku 2.3 je příklad vizualizace, ve které je znázorněna i zaznamenaná energie. Pixely už nejsou černobílé, ale oboarveny dle dané barevné škály a jednotlivé barvy reprezentují energii. U obou obrázků je v modrého obdélníku označena energie, na jakou byly částice nabity, než byly prohnány detektorem a úhel natočení detektoru. Na levém obrázku je energie 20,4 MeV a natočení 0 stupňů a na pravém energie 20,4 MeV a natočení 45 stupňů.



Obrázek 2.3: Píklad vizualizace dat se znázornou energií [13]



Obrázek 2.4: Píklad vizualizace dat s popisem zaznamenaných ástic [12]

Na obrázku 2.4 jsou zobrazeny další píklady vizualizace dat. Energie je opřednána na barevnou škálu, v obrázku jsou sloučena data z mnoha snímků do jednoho. U zaznamenané ástice je vždy uvedena energie, jakou byla ástice prohnána detektorem (E), počet příslušných pixelů (A) a nejvyšší naměná energie ve shluku pixelů (H). U většiny zaznamenaných ástic byla pozice detektoru téměř rovnoběžná se směrem ástic, díky

tomu zaznamenaná data tvoří dlouhé řády.

Každý druh ástic má své specifické chování a zanechává specifické obrazce, podle kterých lze rozpoznat, která ástice byla zaznamenána. V tabulce na obrázku se nachází příklady takových obrazců a jakou ástici představují. Analýzu dat lze provádět manuálně pomocí které z dostupných vizualizačních metod (příklady uvedeny v kapitole 3) nebo existují strojové metody analýzy. Příkladem je online nástroj „Timepix Analysis Platform at School“ (TAPAS), vyvinutý na analýzu dat poskytnutých detektorem Timepix [11].

1) Dot		Photons and electrons (10keV)
2) Small blob		Photons and electrons (~100keV)
3) Curly track		Electrons (MeV range)
4) Heavy blob		Heavy ionizing particles with short range (alpha particles,...)
5) Heavy track		Heavy ionizing particles (protons, nuclei, Fe, ...)
6) Straight track		Energetic light charged particles (MIP, Muons,...)

Obrázek 2.5: Tabulka popisující jednotlivé obrazce vyskytující se v datech a popis, jaké ástice i jevy představují [15]

2.3 Formáty výstupních dat

V této části jsou popsány formáty, ve kterých lze ukládat data z detektoru. Jsou odvozené od toho, jakým způsobem byla data naměřena.

2.3.1 Formát Time Frame

Data pro tento formát jsou získávána asynchronicky, kdy v určité moment detektor provede snímek a zaznamená všechna data, které v jeden moment detekoval. Data snímku jsou znázorněna jako 2D rastrový obrázek, kde každý pixel popisuje, zda a jaká量 byla zaznamenána energie. Data v tomto formátu lze ukládat buď jako rastrový obrázek, nebo jako textovou matici, kde každá položka odpovídá hodnotě pixelu na konkrétní souřadnice. Barva pixelu určuje hodnotu *Time over Threshold*.

2.3.2 Formát Data Driven

Data pro tento formát jsou získávána takzvaným událostním méním. Tím je myšleno, že detektor reakce na událost a když ji namíří, zaznamená ji. Událostí je myšleno průchode částice detektorem. Data v tomto formátu jsou uložena v textové podobě po řádcích. Jedna řádka reprezentuje jeden pixel, ve kterém byla událost namířena. Obsahuje informace o souřadnicích daného pixelu, asu, kdy k události došlo a namířené energii. Formát zapsání řádky se může lišit v závislosti na detektoru, například v které hodnoty mohou být zadány přímo nebo je potřeba je z uvedených dat doložit, nebo může být použit jiný oddílovací hodnot. Pro tuto práci je použit formát pro *Timepix 3* [8], příklad se nachází na ukázce 1 a struktura vypadá následovně :

```
<coordinate>TAB<ToA>TAB<FToA>TAB<ToT>\n
```

1. **coordinate** – Tato hodnota určuje souřadnice pixelu v obrázku. Hodnota je jen jedna, konkrétní souřadnice X a Y je potřeba doložit. Reprezentuje index v maticovém modelu, kde jsou řádky poskládány za sebou do jednorozměrného pole. Výpočet se provede následovně :

- Souřadnice X: hodnota je celočíselná vydělena šířkou matice:

$$X = \frac{\text{coordinate}}{\text{matrix_width}} \quad (2.1)$$

- Souřadnice Y: s hodnotou je potřeba provést operaci dělení modulo:

$$Y = \text{coordinate} \pmod{\text{matrix_width}} \quad (2.2)$$

2. **ToA** – Time of Arrival, hrubá asová známka - hodnota, která následně slouží k výpočtu celkové asové známky (**TToA**).
3. **FToA** – Fine Time of Arrival, jemná asová známka - hodnota, která následně slouží k výpočtu celkové asové známky (**TToA**).
4. **ToT** – Time over Threshold, zaznamenaná energie v daném pixelu, hodnota reprezentuje jeho barvu.
5. **TAB** – Tabulátor, oddílovací jednotlivých hodnot.
6. **\n** – Symbol označující konec řádky.

65199	99848063	4	3
64945	99848063	5	1
65459	99848063	7	1
65461	99848064	11	8
65462	99848063	2	17
64431	99848063	0	1
65458	99848063	7	34
65201	99848063	5	10
64942	99848063	5	24
65457	99848063	5	32
64687	99848063	5	38
64688	99848063	5	13
64943	99848063	5	36
64944	99848063	5	28
64686	99848063	5	35

Ukázka kódu 1: Formát data driven

Z hodnot ToA a $FToA$ se ještě vypočítá celková asová známka, která udává, kdy k naměření události došlo. Nazývá se **Total Time of Arrival**(zkrácen $TToA$) a spočítá se podle vzorce:

$$TToA = ToA \times CONST_1 - FToA \times CONST_2 \quad (2.3)$$

$CONST_1$ a $CONST_2$ jsou konstanty udávané typem detektoru. Pro detektor *TimePix 3* je pro $CONST_1$ je použita hodnota 25000 a pro $CONST_2$ je 1562. Ta udává parametr „Krok jednoho binu“, který říká, z jak velkého asového rozsahu jsou naměřena data shromážděny do jednoho binu. Udává se v pikosekundách. [8]

3 Metody zobrazování dat z částicových detektorů

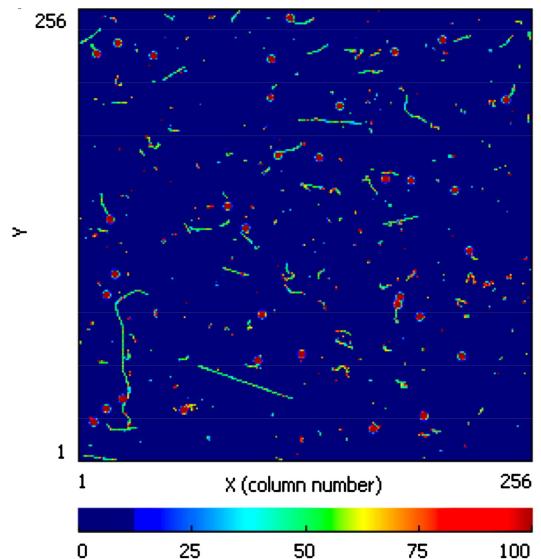
Tato kapitola je zaměřena na popis zobrazovacích metod dat z částicových detektorů. Nejprve jsou popsány příklady současných vizualizačních metod a následně jsou vybrány dvě vhodné zobrazovací metody, které budou implementovány v rámci této bakalářské práce.

3.1 Současné vizualizační metody

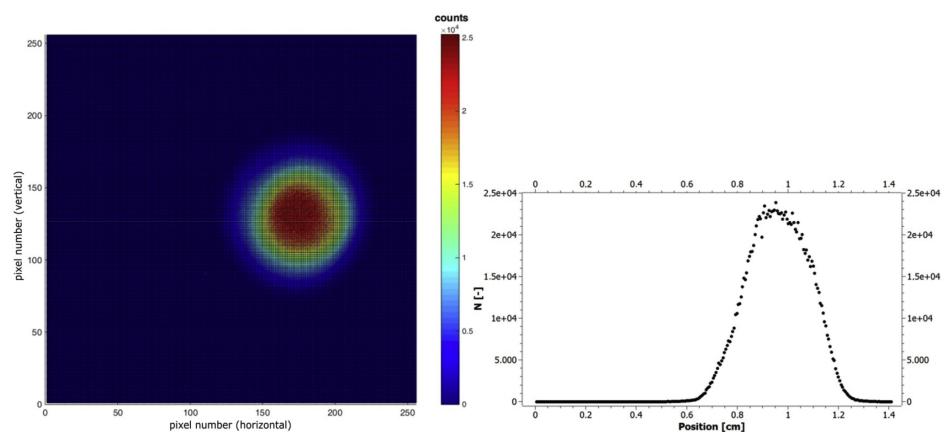
V této části jsou rozebrány současné vizualizační metody pro data z částicových detektorů a je zde uvedeno několik příkladů z výukových lánků. Vizualizace dat již byla lehce nařízena v kapitole 2, tam byl ale kladen důraz primárně na popis významu dat. Metody jsou rozděleny do tří kategorií: vizualizace 2D dat, vizualizace se zaměřením na konkrétní vlastnosti dat a vizualizace dat v reálném světě.

3.1.1 Vizualizace 2D dat

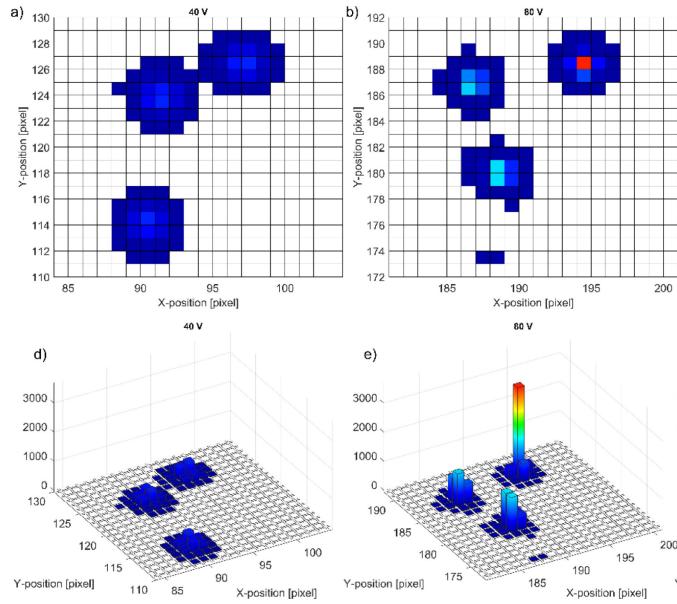
V této části jsou popsány možné vizualizační metody 2D dat. Není podstatné, zda data v obrázku jsou z jednoho i několika binů, i zda byla nějak upravována po výstupu z detektoru, ale vstupem pro vizualizaci musí být 2D rastrový obrázek. Nejdříve je uveden příklad obecné vizualizace podobné tomu uvedeným v kapitole 2 na obrázku 3.1. Jedná se o nejjednodušší formu vizualizace, kdy jsou jednotlivé pixelyobarveny podle barevné škály. Ta je hlavním prvkem sloužícím k tomu, jak rozpoznat jevy v datech. Existují metody, jak data dále zvýraznit. Typicky jsou zaměřeny na zvýraznění energie v jednotlivých pixelích. Jednou z možností je užití latencie v konkrétním bodě v obrázku a zobrazit energie v reálném světě. Znázorněno je to na obrázku 3.2. Na něm je vidět jak převodní vizualizace 2D, tak zobrazení reálného světa v konkrétní řádce vedle sebe. V tomto příkladu je pro zobrazení reálného světa použit 2D graf, kdy na osi X je pozice pixelu v řádce a na osi Y energie naměřená v daném bodě. Další možností podobnou k použití reálného světa je převedení 2D obrázku do 3D, kdy jako hodnota pro osu Z je použita naměřená energie pro konkrétní pixely. Příklady jsou na obrázcích 3.3 a 3.4. Na obou je pro porovnání vidět, jak data vypadají v převodní 2D podobě a jak ve vizualizované 3D podobě.



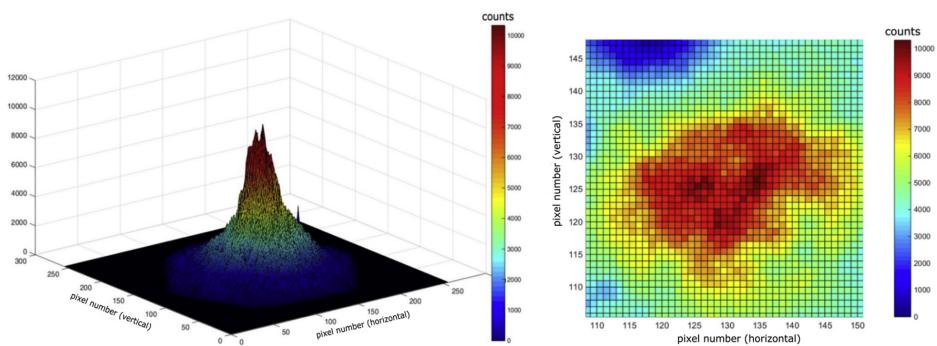
Obrázek 3.1: Píklad jednoduché 2D vizualizace. [15]



Obrázek 3.2: Píklad vizualizace ežem. [29]



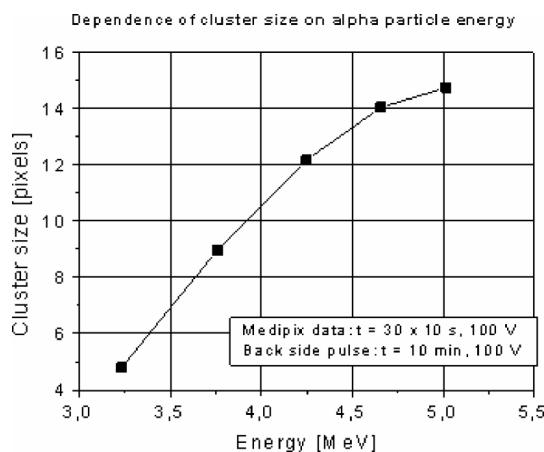
Obrázek 3.3: Příklad 3D vizualizace, na osách X a Y jsou pozice jednotlivých pixelů a osa Z reprezentuje energii pro jednotlivé pixely. [34]



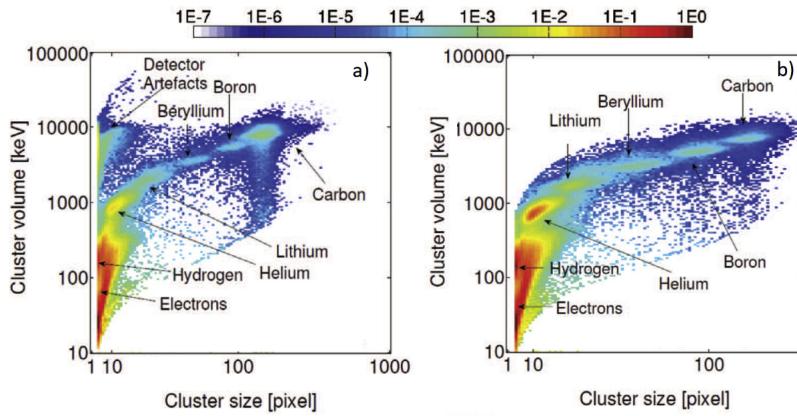
Obrázek 3.4: Příklad 3D vizualizace, na osách X a Y jsou pozice jednotlivých pixelů a na ose Z je vynesena energie pro jednotlivé pixely. [29]

3.1.2 Vizualizace se zaměřením na znázornění konkrétních jevů v datech

Tato část je zaměřena na možnosti vizualizace se zaměřením na konkrétní jevy v datech. Jsou zde uvedeny jednoduché možnosti vizualizace se zaměřením na konkrétní vlastnosti dat, aby bylo možné lépe hledat žádané jevy. Ve zdrojových datech jsou typicky naměny specifické hodnoty a ty jsou vizualizovány do požadované podoby. Příklady jsou uvedeny na obrázcích 3.5 a 3.6. Obrázek 3.5 je převzat z práce [6], kde bylo zpozorováno, že velikost clusteru pixel (shluku sousedních pixel) ; velikost clusteru udává počet pixel v n m) je závislá na energii zaznamenané alfa stice. [6] Tím byla energie alfa stice vyšší, tím více pixel se ve shluku nacházelo. Výsledky zde jsou zaznamenány v grafu na tomto obrázku, na ose X je vynesena energie a na ose Y počet pixel ve shluku. Obrázek 3.6 ukazuje zajímavou vizualizaci, která se také využije závislosti velikosti clusteru na energii. Jedná se o 2D graf, na ose X je velikost clusteru (udaná jako počet pixel pro daný cluster) a na ose Y „obsah“ clusteru (suma energií všech pixel pro daný cluster). Barevné jsou oddělena data z jednotlivých bin podle barevné škály. Popisky v grafu popisují jaké prvky jsou v jednotlivých clusterech zaznamenány. Tím, že se jedná o vizualizaci zde kolika samostatných bin, může se jednat i o možnost vizualizace dat zde kolika různých známek.



Obrázek 3.5: Graf znázorující závislost velikosti clusteru na energii zaznamenané alfa stice. [6]

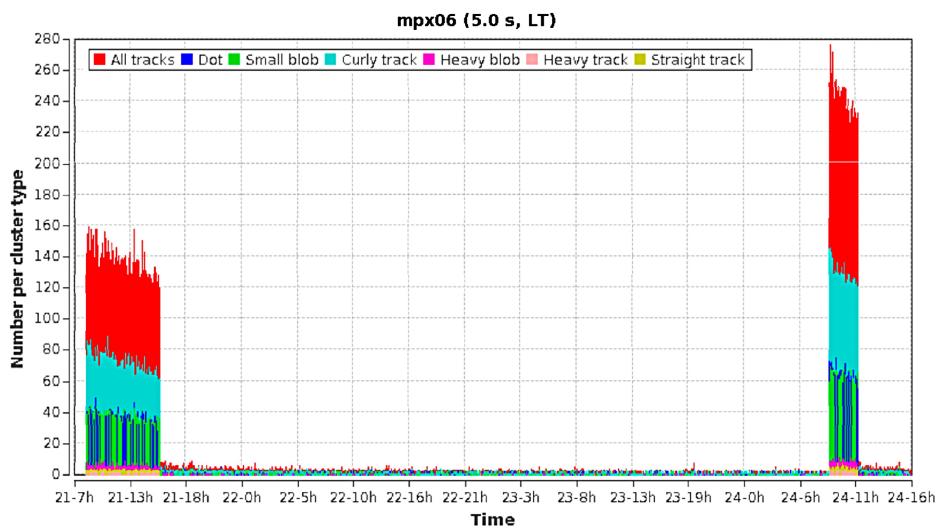


Obrázek 3.6: Graf znázorující závislost velikosti clusteru na obsahu clusteru (suma energií všech pixelů pro daný cluster). [32]

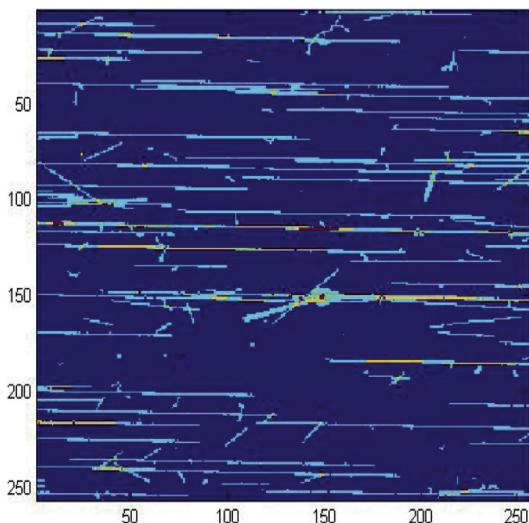
3.1.3 Zobrazení dat v čase

Tato část popisuje možnosti vizualizace s ohledem na zobrazení dat v čase. Cílem je zobrazit data z několika asových známek tak, aby bylo možné pozorovat jejich vývoj v čase. Jedna z možností je popsána v lánku *Measuring radiation environment in LHC or anywhere else, on your computer screen with Medipix* [15], z něhož byl převzat obrázek 3.7. V tomto lánku je popsáno mnoho, když byly analyzovány biny z několika asových známek. Pro každý bin byly analyzovány clustery a ty zařazeny do několika kategorií podle tvaru a velikosti, kategorie jsou odděleny barevně. Výsledky této analýzy byly vyneseny do grafu na obrázku 3.7. V tomto grafu je na ose X vynesen čas a na ose Y hodnoty, kolikrát se daný cluster v konkrétním čase nachází. Toto je jedna z možností zobrazení vývoje dat v čase, když se z několika dat prozkoumají pozorované jevy a vynesou se na časelnou osu. Příkazem dat na asové ose lze určit jevy na které se zaměřit a konkrétní biny, které je obsahují, pak hloubit ji prozkoumat. Pro příkaz jednotlivých bin může být použita následující metoda uvedená výše. Příklad je uveden na obrázku 3.8.

Další možnost jak vizualizovat data v čase je metoda, kdy jsou data z několika asových binů sloučena do jednoho binu. Energie z pokrývajících se pixelů se sčítají. Výsledkem je 2D obrázek, který lze vizualizovat například následující metodou uvedenou výše. Příklad je uveden na obrázku 3.8.



Obrázek 3.7: Graf ukazující typy cluster v binech pro konkrétní asové známky. Typy cluster jsou odd leny barevn podle legenda.[15]



Obrázek 3.8: 2D vizualizace, kdy jsou do jednoho obrázku slou ena data z n kolika asových známek. U pixel se stejnými sou adnicemi je jejich hodnota se tena. Pixely z pouze jedné asové známky jsou sv tle modré, z n kolika žluté až ervené. [14]

3.2 Zvolení vhodných zobrazovacích metod pro komponenty

V této ásti je rozebráno zvolení dvou vhodných zobrazovacích metod pro výsledné komponenty. Od zadavatele práce je dáno, aby jedna z metod byla ve 3D pro snazší zobrazení dat v ase. Jako druhá byla tím pádem zvolena 2D metoda za ú elem porovnání, jak velký rozdíl bude v p ínosu informace mezi dv ma dimenzemi. Vzhledem k experimentální podstat práce nelze dop edu ur it, která metoda bude ideální. Až na výsledcích práce bude ur eno, zda je zvolená metoda pro zobrazování daných informací vhodná a zda poskytuje pot ebné informace. Ob zvolené metody budou mezi sebou porovnány, zda ob poskytují stejné množství informací.

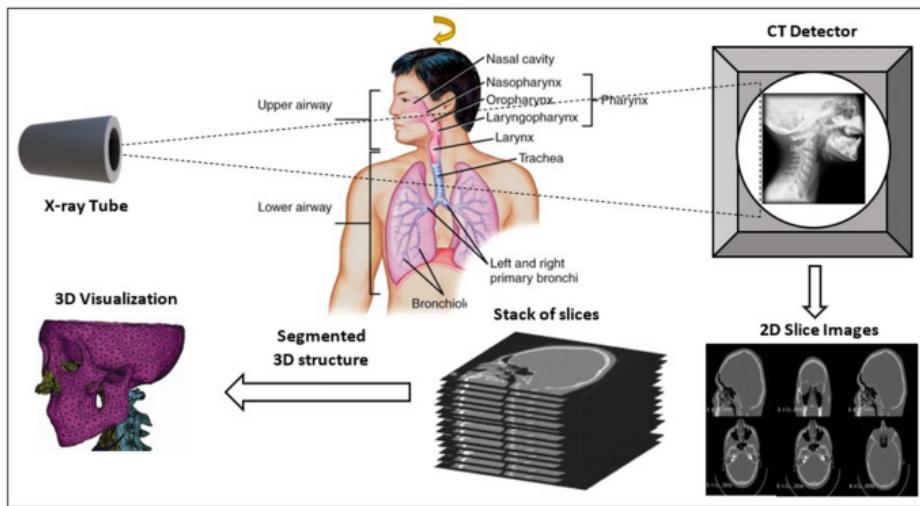
3.2.1 2D metoda

V této ásti je rozebrána volba vhodné zobrazovací 2D metody. Metoda musí um t ešit stejné problémy jako 3D metoda, ale jiným zp sobem. Ta je zam ena hlavn na p evod 2D dat do 3D podoby a jejich vynesení na ísel-nou osu. Vzhledem k chyb jící asové ose se 2D metoda zam í spíše na to, jak vizualizovat data z n kolika asových snímk ve 2D. Pro zobrazení v ase je zvolena již existující podoba popsána výše v ásti 3.1.2, kdy data z n kolika snímk budou promítnuta do jednoho snímku. P idanou hodnotou bude p idání funkcionality, aby p i slou ení nebyla ztracena informace o p vod-ních binech. Dosaženo toho bude pomocí zakomponování interaktivity, kdy bude možné kliknout na každý pixel a zobrazit informace o n m, jako je jeho energie, sou adnice a asové známky, ze kterých se skládá. V místech, kde budou kolidovat pixely s nam enou energíí z n kolika asových známk bu-dou, bude barva ur ena jinak. Na výb r je n kolik ešení, nap íklad: pixely v kolizi obarvit n jakou specifickou barvou; se íst barvy všech pixel , které jsou v kolizi; zvolit barvu jako po et as , které jsou v kolizi. Zvolena byla metoda se íst barvy všech pixel , aby bylo na pohled z etelné, že je v daném míst vyšší energie a že mohlo dojít ke slou ení energie z n kolika as . Konkrétn bude se tena hodnota ToT jednotlivých pixel a na základ té bude p i azena barva, jako pro jakýkoliv jiný pixel. Vzhledem k možnému v tšímu množství zdrojových dat bude zobrazována jen volitelná podmnožina dat. Vizualizace by byla jinak pravd podobn nep ehledná.

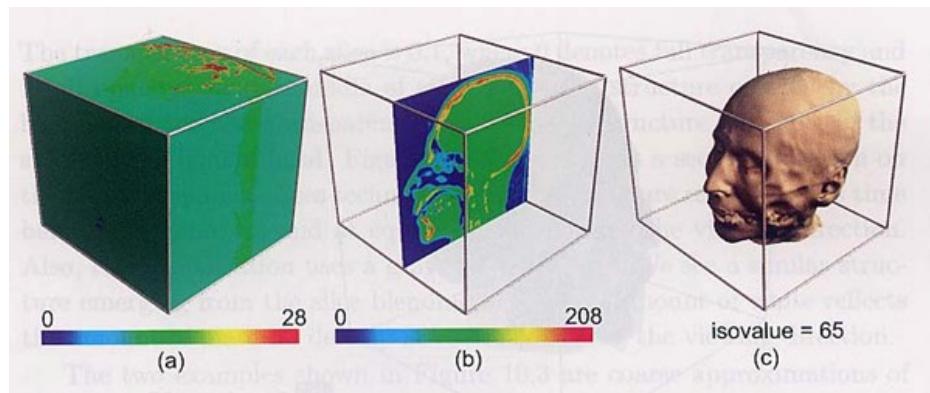
3.2.2 3D metoda

V této ásti je rozebrána volba vhodné zobrazovací 3D metody. Metoda musí um t ešít dva základní problémy: jak vhodn p evést zdrojové 2D snímky na 3D a jak snímky vynést chronologicky na asovou osu. Pro její zvolení se lze inspirovat v podobných úlohách zam ených na zobrazování objemových dat. Blízkým takovým kandidátem je technologie CT (computed tomography)[30], která se zam uje na zobrazení 3D vizualizací na základ nam ených 2D snímk . asto se používá v léka ství pro vizualizaci struktury ástí lidského t la nebo i pro vizualizaci a analýzu struktur materiál . CT sken nasnímá cílový objekt z n kolika úhl a z každého skenovaného úhlu vytvo í 2D snímek. Množina 2D snímk popisuje objekt z n kolika úhl a lze z nich tedy vytvo it 3D vizualizaci. Celý proces je znázorn n na obrázku 3.9. Na obrázcích 3.10 a 3.11 je poté vyobrazen proces, kdy z bloku 2D snímk je vytvo ena 3D vizualizace. Blok je zanalyzován vhodným algoritmem a jsou vybrána data, které je pot eba transformovat do 3D. Tyto jednotlivé prvky jsou následn transformovány do tzv. voxel , což jsou 3D ekvivalent pixel a jsou reprezentovány jako krychle. Z toho je i odvozen jejich název, pixel je složen z anglického slovního spojení „picture element“ (esky obrazový prvek), voxel ze slov „volume element“ (esky objemový prvek). Na obrázku 3.12 jsou pro porovnání znázorn ny rozdíly mezi rastrovou a vektorovou grafikou ve 2D a 3D. Tento zp sob, tedy použití voxelové grafiky, lze použít i pro 3D vizualizaci dat z ásticových detektor . Vstupem je také množina 2D snímk , které je pot eba vizualizovat ve 3D. S tím rozdílem, že metoda pro CT slouží k zobrazení v prostoru, data z detektor budou zobrazována v ase. Voxelová grafika je obecn astou technikou pro zobrazování reality, protože umož uje p ímý p evod z 2D rastrové grafiky do 3D, z jednotlivých voxel lze sestavit prostor podle požadovaných vlastností a každému voxelu lze p i adit specifické vlastnosti. V souasnosti jsou modelovány na softwarové vrstv pomocí polygon , v historii byly pokusy optimalizovat hardware na vykreslování voxel , ale v souasnosti je optimalizován na vykreslování polygon .

3D vizualizace tedy bude vytvo ena za pomoci voxelové grafiky. Jednotlivé biny/snímky budou p evedeny do 3D a umíst ny na osu Z, která poslouží jako asová osa. Jednotlivé snímky budou vytvo eny tak, že se vymodelují voxely odpovídající jednotlivým pixel m v obrázku. Jako souadnice lze použít jednotlivé hodnoty ToA každého snímku. Chronologicky na ni budou vyneseny jednotlivé snímky. Iov k, který pak bude data analyzovat, si bude moci v ase prohlédnout vývoj jednotlivých snímk . Vzhledem

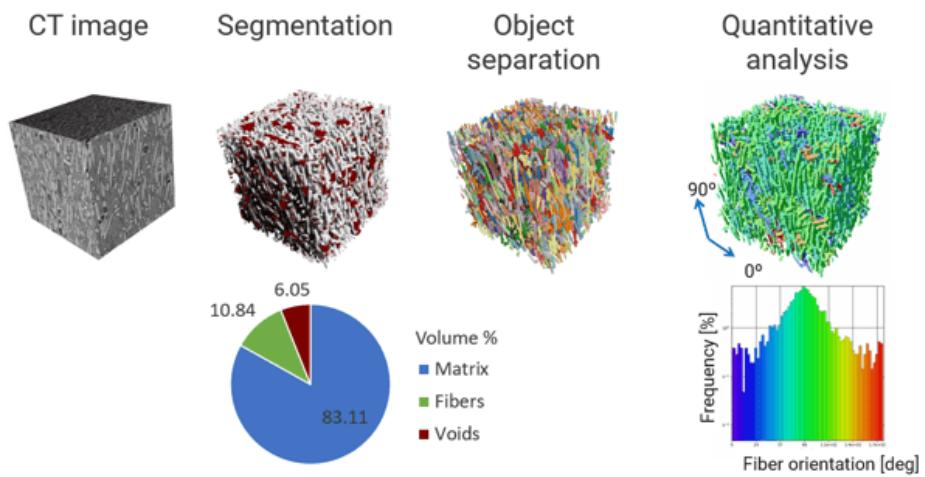


Obrázek 3.9: Proces vytvoření 3D obrazce pomocí použití technologie CT (computed tomography). [26]

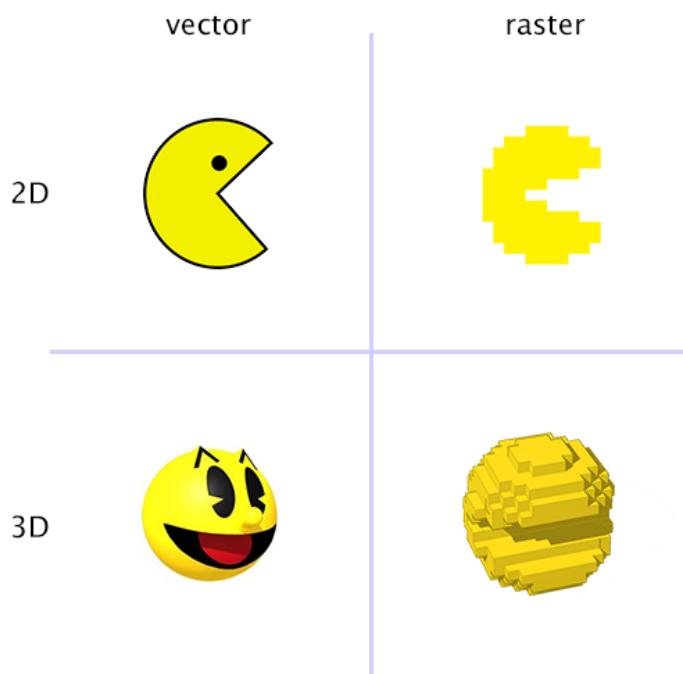


Obrázek 3.10: Znázornění konstrukce 3D obrazce ze zdrojových dat. a) množina namámených dat; b) záznam namámenými daty, 2D snímek; c) konstrukce 3D obrazce ze zdrojových dat [33]

k možnému v tisku množství zdrojových dat bude zobrazována jen volitelná podmnožina dat. Teoreticky bylo možné vykreslit najednou všechny pixely a uživatel by se v modelu pohyboval, ale to by bylo výpočetně náročné a vykreslení všech pixelů by mohlo udělat aplikaci neovladatelnou. Navíc je dobrý zvykem vykreslovat pouze ta data, která uživatel vidí, vykreslovat data na vzdálených souřadnicích osy Z bylo zbytečné.



Obrázek 3.11: Znázornení jednotlivých kroků vytvoření 3D vizualizace z množiny CT snímků. Za pomocí vizualizace je provedena analýza složení zkoumaného prvku. 3D objekty jsou vykreslovány pomocí voxelů. [31]



Obrázek 3.12: Znázornení rozdílu mezi vektorovou a rastrovou grafikou ve 2D a 3D. [16]

4 Návrh komponenty

Tato kapitola je zaměřena na návrh sestrojení samotné komponenty, která je podle této práce. Budou vyhotoveny dvě komponenty (pro 2D a 3D metodu), jejich návrh je v mnoha oblastech spolehlivý a liší se až na navržení konkrétní vizualizační metody.

4.1 Požadavky na komponentu

V této části jsou definovány požadavky na komponentu, podle kterých se odvíjí odvídajený další části návrhu. Obecně platí, že co nejméně spolehlivých vlastností pro obě komponenty bude slouženo do jednotné funkcionality.

- Musí být navržena tak, aby fungovala jako komponenta - program, který sám o sobě není spustitelný, ale dá se připojit do jiných projektů, které budou využívat jeho funkcionality.
- Technologie komponenty musí být kompatibilní s programovacím jazykem Java, minimální verze 8.
- Komponenta bude vyhotovena v technologii podporující GUI (Graphical User Interface) a vykreslování grafických objektů. Vizuálně bude tvořena pouze grafickými objekty, ale předejdě se připojit do GUI aplikací.
- Aplikace bude komunikovat s komponentou přes definované API [1], typickými akcemi budou: předání vstupních dat pro vizualizaci, úprava parametrů pro zpracování dat, úprava parametrů vizualizace. Informace o datech a vizualizaci předou získat, aby mohly být umístěny do okna aplikace. Ideálně v něj dynamické struktury, která se sama bude starat o aktualizaci informací, aby se aplikace samotná nemusela ptát na nové hodnoty.
- Komponenta bude reprezentována hlavní třídou, umístěnou instance této třídy do aplikace lze volat její funkcionality.
- Obě komponenty budou vizualizovat pouze část na tených dat, komponenta bude implementovat grafický ukazatel aktuální pozice zobrazeného okna v celkových datech. Ten předejde z komponenty získat a umístit do okna aplikace.

- Komponenta umožní provést operaci na vstupních datech nazývanou *grouping*, která umožní sloujit data z několika *bin* do jednoho. Tato operace umožní uživateli zobrazit více dat najednou.
- Pro obě komponenty bude implementována interaktivita pomocí myši, když povede na vizualizované události kliknout, zvýraznit je a vypíše se o informace o ní.

4.2 Základní pojmy

V této části jsou popsány pojmy, které jsou použity pro implementaci komponenty a jsou dležité k pochopení dalšího textu. Jedná se o fyzikální pojmy, do jisté míry už zmíněny v kapitole 2 a v zde jsou zopakovány. Fotonkový detektor udává naměřená data ve formě rastrového obrázku, barvy jednotlivých pixelů udávají, zda jak velká byla na tomto místě zaznamenaná energie. Obrázky jsou mimo tvercové, ale mohou být i obdélníkové. Data jsou pořizována ve dvou formátech. Prvním je TimeFrame, který udává zaznamenaná data v jeden konkrétní čas. Druhým je Data Driven, který je založený na bázi událostního měření, zaznamenaná data do souboru jakmile nastane událost. Tato událost odpovídá právě jednomu pixelu s naměřenou energií v rastrovém obrázku. Nadále budou nazývány jako *událost*, případně anglickým ekvivalentem *event*. Shluky události pro jeden konkrétní čas se nazývají *bin*. Zároveň se z nich dá sestrojit rastrový obrázek, proto bude shluk nazýván i pojmem *snímek*, případně anglickým ekvivalentem *image*.

4.3 Volba technologie

Tato část je zaměřena na výběr vhodné technologie, pomocí které bude komponenta realizována. Bude se jednat o technologii, která umožní využít GUI (Grafical User Interface) a vykreslování grafických komponent (ideálně 2D i 3D prvku). Lze použít dvě různé technologie, když každá implementovala jednu z těchto vlastností a následně byly spolu propojeny, ale pro jednoduchost bude vybrána taková, která umí obojí. Grafické komponenty budou použity k vykreslení vybrané vizualizační metody, přes které GUI bude využívána vizualizace ovládána. Ty sice nebudou součástí samotné komponenty, tu bude využít pouze vizualizační část, ale bude výhodné, aby je podporovala. Omezením při výběru je zadání, aby komponenta byla programovací jazyk Java. Zvolená technologie tedy musí být s tímto jazykem kompatibilní. Nejznámějšími možnostmi jsou technologie

gie AWT [2], Swing[35] a JavaFX[21]. V následujících odstavcích jsou tyto technologie popsány se zaměřením na tvorbu GUI a grafických prvků.[18]

4.3.1 AWT a Swing

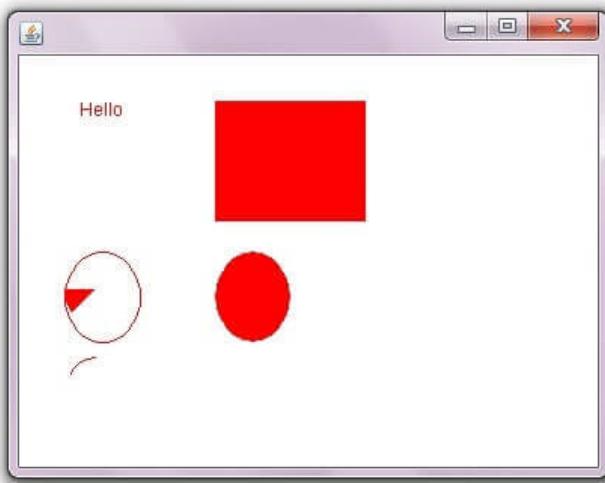
AWT a Swing jsou si velmi podobné, protože Swing vychází z AWT, takže byly sloučeny do společnosti.

AWT je z uvedených nejstarších technologií, provedená byla v roce 1995 spolu s první verzí Java. V současnosti už se ale téměř nepoužívá, protože už není dále vyvíjena. Tím pádem je zastarálá a neobsahuje pokročilé komponenty, které moderní GUI frameworky využívají. V současnosti se hodí pouze na menší a jednodušší aplikace. Díky svému odkazu se v Java stále vyskytuje a některé z nových frameworků používají některé funkce AWT. AWT umožňuje kreslení vlastní grafiky pomocí třídy `java.awt.Graphics` nebo `java.awt.Graphic2D`. Umožňuje pouze kreslení jednoduchých 2D obrazců, jakou jsou obdélníky, elipsy nebo polygony. AWT neumožňuje vykreslování 3D objektů. Proto by tento framework byl užitečný primárně pro implementaci 2D metod. Pro 3D by případně bylo potřeba vytvořit vlastní implementaci, která by počítala objekty ve třech dimenzích a následně je prováděla do 2D, kterou by pak následně vykreslovala třída `Graphics`.

Swing vychází z frameworku AWT, proveden byl v roce 1996. Jedná se o následníka AWT, funkcionalita je vystavována nad jádrem AWT. Opravuje a nahrazuje některé jeho funkce za lepší a přidává další funkcionalitu k existujícím funkcím. Swing byl dlouhou dobu populární, ale v současnosti již není vyvíjen a aktualizován. Zejména kvůli vývoji nových frameworků. Přestože je stále používán v mnoha projektech, zejména z historických důvodů v tom, které ho používají z minulosti a je poměrně složité přejít na jiný framework. Kreslení obrazců se zde provádí pomocí třídy `java.awt.Graphics`, právě protože Swing je vystavována nad jádrem AWT. Možnosti vizualizace jsou tedy stejné jako pro AWT. Ukázka kreslení v technologii Swing se nachází na obrázku 4.1.

4.3.2 JavaFX

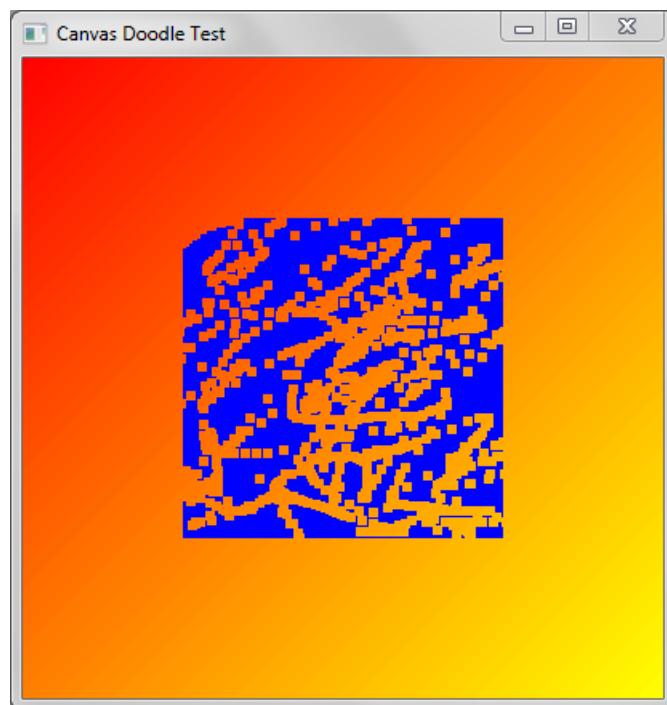
JavaFX patří k nejnovejším a nejlepším Java GUI frameworkům, provedená byla v roce 2008 jako součást JDK. Od JDK verze 11 a dále není JavaFX jeho součástí, ale je vydávána jako samostatná knihovna, kterou je potřeba do projektu importovat. JavaFX je stále vyvíjena a pravidelně



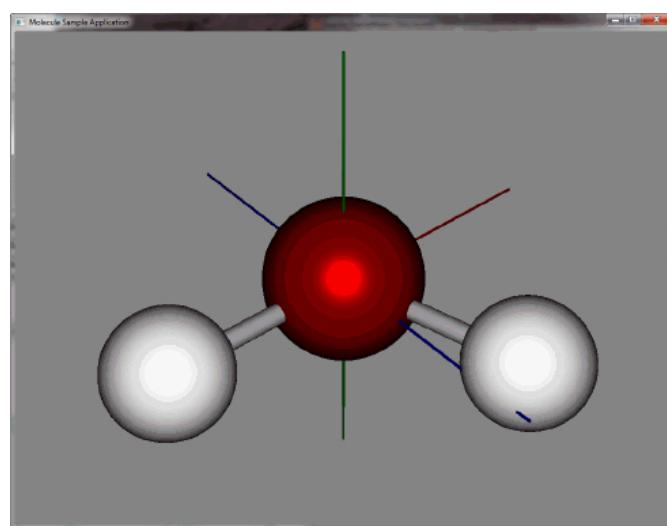
Obrázek 4.1: Píklad kreslení 2D objekt pomocí frameworku Swing [36]

jsou vydávány nové verze, což usnad uje rozvoj a údržbu aplikací. V těch se nových Java GUI aplikací vyvíjených v poslední dekád používají JavaFX. Tím že je nov jší a neustále vyvíjena, poskytuje modern jší a bohatší vzhled GUI než t eba AWT nebo Swing. Umož uje snadnou práci s technologiemi jako CSS nebo FXML. JavaFX umož uje vykreslování 2D i 3D grafických komponent. 2D obrazce lze vykreslit n kolika zp soby. Zaprvé lze vytvá et samostatné 2D objekty, které jsou umis ovány p ímo do scény (lze použít p eddefinované jako obdélník i kruh apod., tak i lze vytvoit vlastní pomocí t ídy javafx.scene.shape). Zadruhé lze použít t ídu javafx.scene.canvas.Canvas, která vytvo í „plátno“, do kterého lze tvoit libovolné obrazce. Píklad použití metody Canvas je na obrázku 4.2. Zat etí lze ješt využít komponenty pro vykreslení p ímo p edaných obrázk (vytvo it uvnit aplikace jako obrázek v n jakém standardizovaném formátu (jpg, png,...)), napíklad javafx.scene.image.ImageView. Tyto metody lze použít pro 2D vizualizaci, napíklad pro slou ení více snímk do jednoho a následn ho vykreslit. Lze ho vykreslit do plátna jako nový snímek, nebo ho vytvo it uvnit aplikace jako obrázek v n jakém standardizovaném formátu (jpg, png,...) a následn ho vykreslit pomocí knihovní funkce. Pro vykreslování 3D obrazc lze použít podobný zp sob jako pro 2D, kdy jsou obrazce vytvo eny jako samostatné objekty (krychle, kvádr, koule,...) a následn jsou umist ny do scény. asto se vykreslují do samostatné scény vložené do p vodní scény (javafx.scene.SubScene), protože 3D scéna bude mít jiné vlastnosti než scéna s 2D komponentami. Píklad vykreslení 3D objekt se nachází na obrázku 4.3. Pípadn by bylo samoz ejm možné také využít

Canvas, kdy by byly 3D objekty vypoítány pomocí vlastní implementace a následně vykresleny do Canvas.



Obrázek 4.2: Příklad kreslení pomocí techniky Canvas JavaFX [7]



Obrázek 4.3: Příklad jednoduché JavaFX 3D aplikace [25]

Pro tvorbu komponenty byla zvolena technologie JavaFX kvůli existujícím možnostem pro vytváření 3D komponent a kvůli pokraujícímu vývoji

této technologie. Bude tedy potřeba importovat knihovnu JavaFX například znými verzemi Java. Pro samotnou komponentu by to neměl byt velký problém, protože ta sama o sobě spustitelná nebude, bude importována do jiného projektu a kompatibilitu bude muset ověřit až v rámci tohoto projektu. Bude to ale potřeba vyvystavit u demonstrační aplikace, která je součástí této práce.

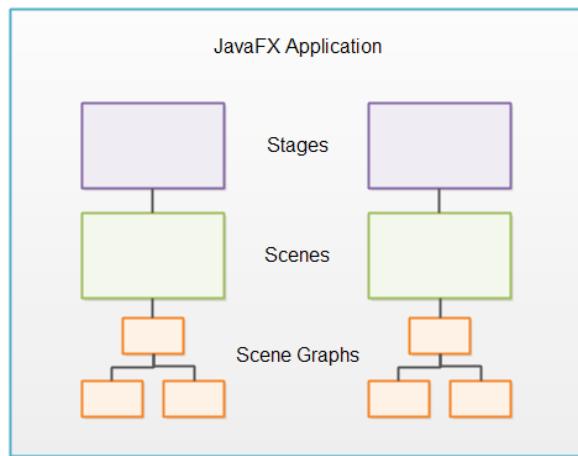
4.4 Architektura komponenty

Tato část je zaměřena na navržení architektury a struktury komponenty. Při návrhu je potřeba vzít v potaz, že jde o implementaci komponenty a ne samostatného spustitelného programu. Komponenta bude program, který je možné importovat do jiného programu a funkce komponenty lze využívat voláním metod definovaného API [1]. Při návrhu je potřeba počítat s tím, aby veškerá funkcionalita byla zapožata a byla dostupná pouze přes toto API. Proto bude výjimka jedna třída komponenty jako hlavní třída, jejíž všechny metody ho budou reprezentovat. Komponenta bude navržena jako vlastní grafický prvek JavaFX (jako jsou tlačítka, okna pro výpis, atd.), který lze přímo umístit do scény jako *Node*. Snadné řešení jak toho dosáhnout je, aby komponenta dělala odstranění jaké existující grafické komponenty JavaFX. Seznam všech komponent lze nalézt pod zdrojem [23].

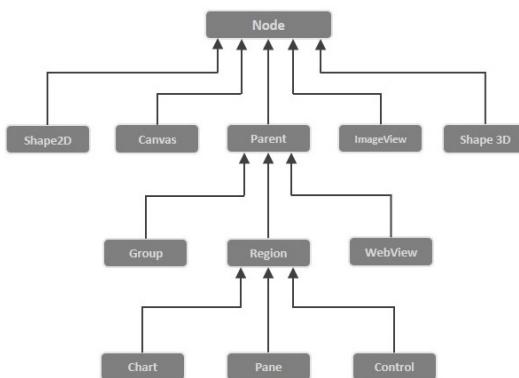
4.4.1 Popis základních struktur JavaFX

Pro lepší pochopení, jaké místo bude v hierarchii JavaFX komponenta zaujímat, jsou v následujících odstavcích popsány základy architektury JavaFX. Předem tedy třídy není poskytnout žádná instrukce na pracování s JavaFX, spíše se zaměřuje na obecné popsání chování k snazšímu pochopení navržené komponenty. Na obrázku 4.4 se nachází znázornění složení JavaFX aplikace. V základu se skládá ze tří hlavních prvků: **Stage**, **Scene** a **SceneGraphs**. **Stage** je vnitřní rám JavaFX aplikace, typicky představuje okno. Pokud má aplikace více oken, každé okno má svou Stage. V kódu je reprezentováno objektem *Stage*. Každá JavaFX aplikace má výchozí Stage, označovanou jako *primaryStage*, kterou vytvoří přímo framework JavaFX a uživatelská aplikace ho využívá. Pro zobrazení dalších oken pak aplikace může vytvořit další Stage a uchovávat jejich instance. **Scene**, neboli scéna, je prostor, který slouží k zobrazování objektů samotného GUI. Všechny prvky, které je potřeba zobrazit, jsou umístěny do scény. Stage umožňuje zobrazovat pouze jednu scénu, scény je možné v rámci Stage vyměňovat během běhu aplikace. V programu je scéna reprezentována objektem *Scene*.

Scene Graph jsou všechny grafické prvky, které mají být ve scéně zobrazeny (tlačítka, textová okna, menu,...). Na obrázku 4.5 se nachází znázornění tzv. *Node*. *Node* jsou nazývány jednotlivé komponenty, které jsou umisťovány do *scene graph*. Všechny komponenty JavaFX díky třídě *javafx.scene.Node*. Z toho vyplývá, že aby navržena vizualizace komponenta fungovala správně jako komponenta JavaFX, musí od této třídy také dělat. Toho lze dosáhnout tak, že bude komponenta dělat už od nějaké existující JavaFX komponenty a umístit na do scény jako *Node*. [23]



Obrázek 4.4: Znázornění hierarchie komponent JavaFX [23]



Obrázek 4.5: Znázornění hierarchie komponent JavaFX [24]

Příklad použití se nachází na ukázce kódu 2. Jedná se o jednoduchý "Hello World" program v JavaFX. Vytváří okno se dvěma grafickými prvky, tlačítkem a textem. Při stisknutí tlačítka „Say 'Hello World'“ se zobrazí a skryje

text s nápisem „Hello World!“. Na tomto krátkém příkladu je znázorněno využití prvků JavaFX popsaných výše. Nejprve metoda `main()` zavolá hlavní metodu JavaFX `launch()`, která spouští funkce samotného GUI. Dále je zavolána metoda `start()`, která inicializuje okno a všechny prvky JavaFX. Nejprve je inicializována `Stage primaryStage`, jejíž instanci již vytvořil samotný framework JavaFX. Následně se inicializují všechny kontrolní prvky (`controls component`), tlačítka (`Button`) a text (`Label`). Poté je inicializován prvek umístěný (`layout component`) `VBox root` (Vertical Box; prvky jsou umístěny vertikálně). Nakonec je inicializována scéna `Scene`, do které je umístěn layout komponenta `VBox`. Scéna je poté vložena do `Stage` a celé okno je zobrazeno.

K vhodnému propojení *API* a JavaFX komponenty lze vyhradit jednu třídu, která bude splňovat parametry obou těchto prvků. Bude se jednat o tzv. hlavní třídu komponenty. Při použití komponenty bude instance této třídy vkládána do scény. Tato hlavní třída bude mít definovány všechny metody pro ovládání komponenty a bude dělat od existujících JavaFX komponenty typu *layout*, aby bylo možné do ní umístit další prvky. Díky ní od komponenty typu *controls* se nehodí, protože ty slouží k poskytování konkrétní funkcionality.

```

public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        // stage
        primaryStage.setTitle("Hello World!");

        // controls components
        Label label = new Label("Hello World!");
        label.setVisible(false);
        Button button = new Button("Say Hello World");
        button.setOnAction(event -> {
            label.setVisible(!label.isVisible());
        });

        // layout component
        VBox root = new VBox();
        root.getChildren().addAll(button, label);
        root.setAlignment(Pos.CENTER);
        root.setSpacing(5);

        // scene
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}

```

Ukázka kódu 2: Vytvoření Hello World aplikace v technologii JavaFX

4.4.2 Navržená architektura

V této ásti je rozebrána navržená architektura komponenty. Jak bylo zmíněno v předešlé ásti, komponenta bude obsahovat hlavní třídu, která bude řídit veškeré operace komponenty. Bude navržena podle architektury *Model-View-Controller*, z toho vyplývá, že bude rozdělena na tři hlavní ásti: *Model* - obsahuje data a logiku aplikace, *View* - zobrazení uživateli, tato ásta bude provádět samotnou vizualizaci a *Controller* - ovladače pro uživatele, který manipuluje s modelem, zde bude reprezentován hlavní třídou komponenty. Tato hlavní třída bude řídit operace mezi datovým modelem a vizualizací. Například datovému modelu zadá požadavek na zobrazení nových dat a až tuto operaci dokončí, předá hlavní třídě vizualizaci požadavek na vizualizaci nových dat.

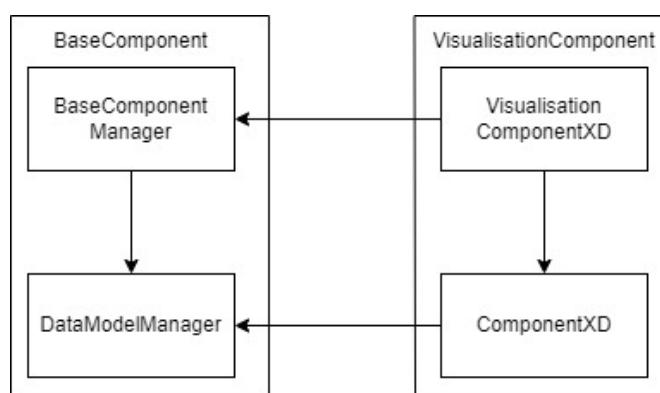
Vizualizace samotná data od datového modelu pouze přebírá a upravuje podle svých potřeb. Pro získání dat z datového modelu budou definovány konkrétní metody, které poskytnou data v určeném formátu. Vizualizace tak nikdy nebude sahat do datového modelu přímo. Tyto metody také lze označit jako součást ásti *Controller*.

Ásta komponenty musí být navržena tak, aby byla nezávislá na zvolené vizualizační metodě. To je výrazné usnadnění pro vývoj, protože nebude potřeba spolehnout kód upravovat duplicitně pro každou komponentu zvlášť. Navíc bude snazší případná implementace další vizualizační metody, kdy bude potřeba implementovat pouze vizualizační funkce, protože datové funkce již budou existovat a lze je využívat. Tato spolehná ásta bude nazývána jako **BaseComponent**, tedy základní ásta komponenty. Spolehně budou primárně operace s daty, proto do této spolehné ásti bude zařazen datový model a z hlavní třídy operace pracující s daty. Balíček **VisualisationModel** pak bude kompletně vymezitelný podle vizualizační metody. Při rozdělení komponenty na dvě ásty bude potřeba i rozdělit hlavní třidu komponenty, protože některé funkce se dotýkají *BaseComponent* a některé *VisualisationComponent*. Funkce *BaseComponent* budou vždy stejné, ale funkce *VisualisationComponent* se budou lišit podle konkrétní vizualizační metody. Z toho důvodu bude vytvořena sekundární hlavní třída, která bude dělit od převodní hlavní třídy (v též stanou operace *BaseComponent*) a bude obsahovat funkcionality pro vizualizaci. Tato sekundární třída bude touto třídou, jejíž instanci bude uživatel přidávat do programu. Vizualizační ásta komponenty se bude nazývat **VisualisationComponent2D** pro 2D metodu a **VisualisationComponent3D** pro 3D metodu. Vizualizační operace a sekundární hlavní třída spadají do ásti **VisualisationComponent2D** nebo **VisualisationComponent3D**.

sationComponent3D. Tato ást je závislá na zvolené vizualiza ní metod .

Diagram architektury komponenty se nachází na obrázku 4.6. Diagramy datového modelu a vizualizací se nachází níže v p íslušných ástech. T ída BaseComponentManager je hlavní t ídou ásti komponenty *BaseComponent*, t ída DataModel Manager je hlavní t ídou datového modelu.

Vi sual i sati onComponentXD (za x se doplní 2 nebo 3 podle toho, zda se jedná konkrétn o 2D nebo 3D metodu) bude hlavní t ídou vizualiza ní ásti a zárove hlavní t ídou celé komponenty. ComponentXD pak bude hlavní t ídou samotné vizualizace, bude poskytovat a idit veškeré vizualiza ní ope race.



Obrázek 4.6: Navržená architektura komponenty. Obrázek byl vytvo en v programu draw.io. [10]

4.5 Komunikace mezi komponentou a aplikací, která ji využívá

V této ásti je rozebrán zp sob, jakým budou data importována i exportována z komponenty a jak budou upravovány její parametry. Jedná se o vstupní data pro vizualizaci, parametry detektoru, jehož data budou vizualizována (rozm ry vstupních dat, konstanty pro výpo et asových známek v datech) a parametry vizualizace. K ur ení zp sob importu a exportu je pot eba vzít v potaz navrženou architekturu v ásti 4.4.2. Podle navržené architektury bude komponenta navržena rozd lena na dv ásti, *BaseComponent* a *VisualisationComponentxD*. Z tohoto rozd lení lze ur it i zp sob nahrávání dat a informací. Je pot eba vzít i v potaz, jaký zp sob importu a exportu by byl uživateli p íjemný. Nap íklad bude vhodné, aby úpravy parametr vizualizace byly viditelné ihned.

ášt *BaseComponent* bude stejná pro jakoukoliv vizualizaní metodu. Zpříjemnění nastavení dat a parametrů by neměly být nijak závislé na konkrétní implementaci zobrazovací metody. V této části budou uchovávána vstupní data pro vizualizaci, parametry detektoru (využijí se při zpracování vstupních dat) a další parametry, například zapnutí operace *Grouping* na vstupních datech i zapnutí logování. Vstupní data jsou zadána v textovém souboru, který bude předán do komponenty zvnějšku přes aplikaci, využívající komponentu. Ta ho předá obslužným operacím v datovém modelu, které spustí na útah. Soubor bude předán rovnou ve formě instance třídy `java.io.File` přes hlavní API komponenty, protože výběr souboru bude obstarávat demonstrační aplikace.

Parametry závislé na detektoru nebudou importovány takto zvnějšku, protože jsou klíčové pro fungování komponenty a navíc vždy musí obsahovat nějakou hodnotu. Lze je udržovat separátně v některém konfiguračním souboru a na ústřední stránce když o nich bude vyžádáno, ale to by bylo uživatelsky nepříjemné. Také budou udržovány přímo v komponentě v jedné třídě určené k držení těchto hodnot. Aby byla aktualizace univerzální pro všechny komponenty, bude prováděna přes samostatné zobrazitelné okno, které uživatel zobrazí a použije v případě potřeby aktualizace těchto hodnot. V rámci tohoto okna bude rovnou zajistit na validaci přesdaných hodnot.

U parametrů části *VisualisationComponentxD* je potřeba zvolit jiný způsob, protože vizualizační metoda bude vymezitelná a navíc je potřeba mít její parametry tak, aby se ve vizualizaci projevily ihned a uživatel na první pohled poznal rozdíl. Možnost přes samostatné okno je také možná, ale ne-musí být pro uživatele tak pohodlná. Místo toho lze prvky pro aktualizaci vystřídit mimo komponentu, aby je obstarávala aplikace a hlavní třída pouze poskytne metody pro nahrání nových hodnot. Nové hodnoty předávají se aplikaci zadat přes `TextFieldy` umístěné do scény, po stisknutí tlačítka pro aktualizaci budou předány příslušným metodám uvnitř komponenty, které je zvalidují, nahrají a aktualizují vizualizaci.

Ještě je potřeba určit, jakým způsobem exportovat informace o načtených datech a vizualizaci, aby mohly být vypsány v okně aplikace, například do `javafx.scene.control.TextArea`. Komponenta je bude ukládat do nějaké dynamické struktury, aby se aktualizace hodnot propojily do aplikace a ta se naopak nemusela ptát, zda k aktualizaci došlo. Budou tedy ukládány do `javafx.beans.property` (Integer nebo String, podle potřeby) z toho důvodu, že na nich lze umístit *listener*, který bude detektovat změny hodnot v `Property`, případně lze na ně nasadit *binding*. Jednotlivé hodnoty budou

odd leny specifickým odd lova em a identifikovány specifickými kódy a aplikace sama je převede do podoby iteln jí pro uživatele.

4.6 Datový model

V této části je popsán návrh datového modelu, zpřímo vztahující se k uložení dat a datových operacím.

4.6.1 Uložení dat

V této části je rozebrán návrh zpřímo vztahující se k uložení dat z detektorů určených pro vizualizaci. Jako vstupní formát dat bude uvažován pouze *data driven*, protože data budou vynášena v reálném čase a přesně na to je určen. K určení vhodné datové struktury na uložení *data driven* formátu je potřeba zopakovat, jaké informace udává a v jaké formě. Udává informace o pixelech, na kterých byl naměřen zkoumaný náboj. Informace jsou udávány v textovém souboru po řádkách a každá jednotlivá řádka reprezentuje jeden pixel (*data driven event*) s naměřeným nábojem. První informací jsou souřadnice pixelu, druhou je asová známka a třetí naměřená energie v pixelu. Asová známka může být stejná pro více pixelů, protože v jednom čase může být naměřený náboj na více pixelech. Z eventů v jednom čase lze zkonstruovat bin, pro který lze vytvořit rastrový obrázek. Požadavky a kritéria na strukturu pro uložení dat jsou následující:

- Nezávislost na zvolené vizualizační metodě
- Minimální paměťová náročnost a redundance
- Všechny eventy musí být jednoznačně identifikovatelné
- Uchovávat seznam eventů pro jednu konkrétní asovou známku - reprezentovat jako bin
- Biny budou vzestupně seřazeny vzestupně podle asových známk
- K binu může být potřeba přistupovat podle indexu (pro předaný index bude vrácen bin, který je na daném indexu umístěn) a podle asové známky přidelené k binu

Je tedy potřeba určit, jak ukládat eventy pro jeden konkrétní bin a následně jak ukládat samotné biny. Eventy je možné ukládat do matice, aby odpovídaly struktuře rastrového obrázku. To ale bude zbytečné památovat náročné, protože bude potřeba alokovat památku pro prvky, pro které nebyla zaznamenaná energie. Lepší bude zvolit strukturu, která uloží pouze hodnoty se zaznamenanou energií. Pro event lze vytvořit i id uvedenou jako nositel jeho hodnot, množinu eventů pro bin lze uložit jako seznam těchto id. Nepředpokládá se úprava tohoto seznamu po nařízení dat, takže může být seznam uložen jako statické pole. Předpokládá se, že soubor se vstupními daty může obsahovat až miliony jednotek záznamů, proto bude potřeba pro parametry eventu zvolit co nejméně památkové datové typy. Z toho dle vodu jsou dále popsány jednotlivé informace, jaké je potřeba ukládat a jaké probíhaly zvoleny datové typy. Pro určení vhodného datového typu byla použita tabulka 4.1, která popisuje velikost, minimální a maximální hodnoty celoúsečníků datových typů v Java.

1. **Souřadnice pixelu v obrázku** – ve formátu data driven jsou souřadnice uloženy v jedné hodnotě a hodnoty souřadnic X a Y je potřeba doplnit. Kvůli přesnosti a snazšímu používání budou v datovém modelu uloženy souřadnice X a Y samostatně jako dvě proměnné. Jedná se o celou úsečníku proměnné, maximální hodnoty jsou závislé na délce hran obrázku, ze kterého je lze získat. V tabulce 4.2 jsou popsány rozlišení obrázků detektoru Medipix. Nejnovější Medipix4 z roku 2021 má maximální délku hrany 1690. Do datového typu byte se nevejdě, ale do typu short ano a zbyde ještě hodnota prostoru pro případné rozšíření. Byl tedy zvolen datový typ short. Kdyby byly souřadnice uloženy pouze v jedné hodnotě, bylo by potřeba použít typ int. Obě řešení jsou tedy stejně památkové, ale rozdíl lení do dvou proměnných umožní následnou snadnou práci.
2. **Time over Threshold (ToT)** – naměřené energie v eventu, poslouží k určení barev pixelu nebo krychle při vizualizaci. Hodnota bude převzata rovnou ze vstupních dat a nebude se dále nijak přepočítávat. Až následná vizualizace a barevný model na základě této hodnoty určí barvu, kterou pravku přidá. Hodnota může nabývat i desítek tisíc, u typu short může hrozit přetečení, u typu int je dostatečně buď pro rozšíření. Byl tedy zvolen datový typ int.
3. **Time of Arrival (ToA)** – asová známka, odkáván vysoké hodnoty. Pro určení datového typu bylo provedeno měření na poskytnutých testovacích datech, maximální naměřená hodnota se vešla pouze

do datového typu long. Ten byl tedy pro tuto hodnotu zvolen.

Datový typ	Velikost	Minimum, Maximum
byte	1 byte	Min: -128 Max: 127
short	2 byty	Min: -32,768 Max: 32,767
int	4 byty	Min: -2,147,483,648 Max: 2,147,483,647
long	8 byt	Min: -9,223,372,036,854,775,808 Max: 9,223,372,036,854,775,807

Tabulka 4.1: Celosíselné datové typy v Java, jejich velikost a minimální a maximální hodnoty, které do nich lze uložit.[20]

	Medipix	Medipix2	Medipix3	Medipix4
Rok vzniku	1997	2005	2013	2021
Rozlišení	64 x 64	256 x 256	256 x 256 128 x 128	320 x 320 1690 x 160

Tabulka 4.2: Přehled možností nastavení rozlišení dat pro čisticové detektory typu Medipix.[3]

Pro každou souadnici X a Y byl vybrán datový typ short (2x2 byty), pro ToT typ int (4 byty) a pro ToA typ long (8 byt), celkem tedy jedna událost zabere 16 byt.

Bin lze reprezentovat třídou, jejími parametry bude ToA a pole s eventy daného binu. K binu m je potřeba přistupovat primárně podle indexu. Tím se jako ideální datové struktury pro uložení seznamu binů nabízí pole, dynamické pole a spojová seznam. Opět se nepodkládá, že po návštěvě budou biny přidávány i odstraňovány, takže se použije statické pole. Biny v tomto poli budou seřazeny vzestupně dle ToA, aby byly pro vizualizaci poskytovány již seřazené. Pro event bude třída pojmenována jako Event a ponese informace o souadnicích a hodnotách ToT. Pro bin bude třída pojmenována jako BinData a uchovává pole událostí binu a dvě asové známky. Dvě jsou

```

// class representing event
class Event{
    private short x, y;
    private int ToT;
}

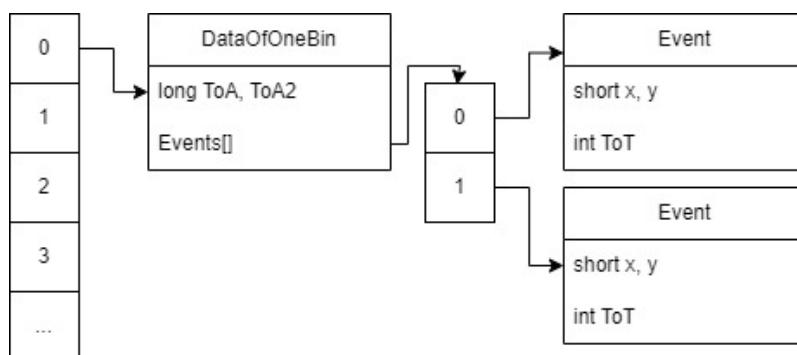
// class representing bin/image
class BinData{
    private long ToA, ToA2;
    private Event[] eventsInBin;
}

// array of bins representing data model
BinData[] dataModel;

```

Ukázka kódu 3: Struktury pro uložení dat ur ených k vizualizaci

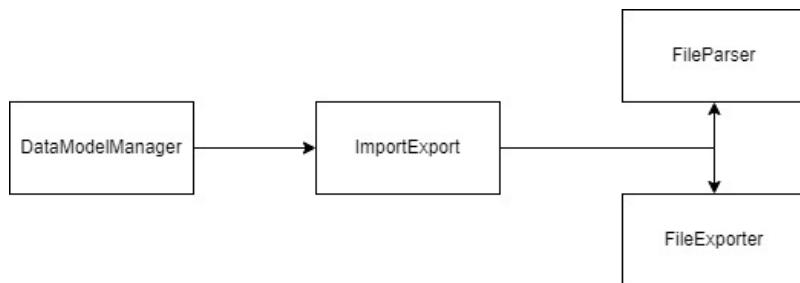
z d vodu operace *grouping*, popsané v části 4.6.3. Při ní dochází ke sloučení n kolika bin do jednoho, je pak uchovávána asová známka nejnižšího a nejvyššího binu a jsou z nich pořítnány souřadnice pro osu Z ve 3D metod. Pokud operace grouping není na datech provedena, hodnoty obou asových známk jsou shodné. Znázornění se nachází na ukázce kódu 3 a na obrázku 4.7. Při použití takové struktury pro uložení dat lze každý event jednoznamenat identifikovat podle indexu binu, ve kterém je uložen, a dvojicí souřadnic X a Y.



Obrázek 4.7: Uložení dat pro vizualizaci v datovém modelu. Obrázek byl vytvořen v programu draw.io. [10]

4.6.2 Architektura datového modelu

V této ásti je popsán návrh architektury datového modelu. Datové operace, které bude pot eba obsloužit jsou: uchovávání a poskytování vstupních dat, uchovávání dat pot ebných k chodu komponenty, import vstupních dat, export uložených dat. Datový model bude mít hlavní t ídu zvanou DataModel Manager, jejím úkolem je ředit a distribuovat datové operace. Zárove v ní budou uchovávána na tená data a vizualizace bude volat ve ejné metody této t ídy pro jejich získání. Datové operace budou distribuovány do více t íd v rámci balíku datového modelu. Dále bude existovat t ída pro řízení a provedení veškerých operací importu a exportu, bude nazvána ImportExport. Na řítaní ze souboru bude provedeno p es t ídu FileParser (popsáno je v ásti 4.6.3), export do souboru p es t ídu FileExporter.



Obrázek 4.8: Navržená architektura datového modelu. Obrázek byl vytvo en v programu draw.io. [10]

4.6.3 Načítání ze souboru a operace grouping

V této ásti je popsán návrh implementace na řítaní dat ze souboru a operace *grouping*. *Grouping* je operace, kdy jsou data z daného po tu bin slou eny do jednoho binu. Ve vizualizaci to umožní uživateli prohlédnout v tší množinu dat najednou. Po et bin , které mají být slou eny budou volitelné uživatelem. P i slou ení budou k výslednému binu p i azeny dv asové známky, nejnižší a nejvyšší z originálních bin a to z toho d vodu, že na základ asových známek budou totiž bin m p i azovány sou adnice osy Z. Kdyby byla p id lena pouze jedna hodnota (nejnižší, pr m r, apod.), p i operaci *grouping* by se ztratily detailní informace o vzdálenostech mezi biny a sou adnice osy Z by pak mohly být nep esné. Tím, že se použijí nejnižší a nejvyšší hodnota, stále p jde ur it vzdálenosti mezi biny stejn dob e jako když operace provedena nebude. *Eventy*, které jsou ve slou eném binu v kolizi (mají shodné sou adnice), budou slou eny do jednoho, jehož hodnota *ToT* bude se tena z jednotlivých event v kolizi. Tato operace bude ešena

už na úrovni *parseru*, do datového modelu bude předána přímo upravená verze. Výhodou je nižší paměťová náročnost při sloučení bin, nevýhodou, že při úpravě parametr sloučení bude potřeba data naštít znovu. Samotná operace *grouping* bude provedena až po dokončení naštítání, aby byla zajištěno, že jsou data kompletní a seřazena. Pro vstup do *groupingu* již musí být biny seřazeny podle asových známek.

Pro implementaci naštítání ze souboru je jednoznačně definováno, jak má vypadat struktura, ve které mají být data z *parseru* předána. Pro snadné nahrazení metody pro naštítání bude použito rozhraní, které bude definovat hlavu k metodě, která bude pro naštítání volána. Vstupním parametrem bude soubor, který má být nařazen a vrácen bude strukturu s nařízenými daty. Požadavky na datovou strukturu jsou podobné jako na samotný datový model. Hlavním požadavkem je, aby obsahovala každou asovou známkou obsaženou v souboru a pro ni seznam událostí. Bylo by možné tedy zvolit stejnou strukturu jako pro data v datovém modelu, ale zároveň je na strukturu snadno použitelnou i pro samotné naštítání. Datový model používá statická pole, ale ty jsou pro naštítání ze souboru nevhodná, protože dopředu není známo, kolik binů i eventů je v souboru obsaženo. Pro přidání každého dalšího prvku by bylo potřeba pole alokovat znovu. Zvolena bude jiná struktura splňující požadavky a tím je například implementace struktury mapa ($\text{Map} < \text{K}, \text{V} >$) [19]. Jako klíč je použita asová známka a jako hodnota je data reprezentující bin, pro kterou seznam událostí bude implementován dynamickou strukturou, aby bylo možné prvky přidávat. Na výběr je zde několika implementací mapy, všechny splňují dané požadavky a lze z nich následně transformovat data do datového modelu. Nicméně, když bude přidán ještě požadavek na seřazení dat v mapě pro následnou snazší konverzi do datového modelu, lze použít implementaci **SortedMap**. V této lze se adit klíč a vzestupně i sestupně. V mapě budou klíče seřazeny vzestupně, aby bylo možné provést transformaci dat do struktury datového modelu mohla rovnou iterovat nad mapou. Struktura pak bude vypadat takto: `SortedMap<ToA, BinData>`.

Ještě je potřeba zvolit dynamickou strukturu pro ukládání eventů k binům. V Java lze použít připravenou implementaci dynamického pole (`ArrayList<E>`) nebo spojového seznamu (`LinkedList<E>`). `ArrayList` je lepší použít pokud se budou data pouze přidávat a bude se k nim přistupovat, ale ne modifikovat a místnost jejich pořadí. `LinkedList` je naopak lepší, když se data budou modifikovat a místnost jejich pořadí. [17] Při naštítání budou data do struktury pouze přidávána, nebude se místnost jejich pořadí. Úprava místnosti následovat při operaci *grouping*, která bude provedena až po skončení

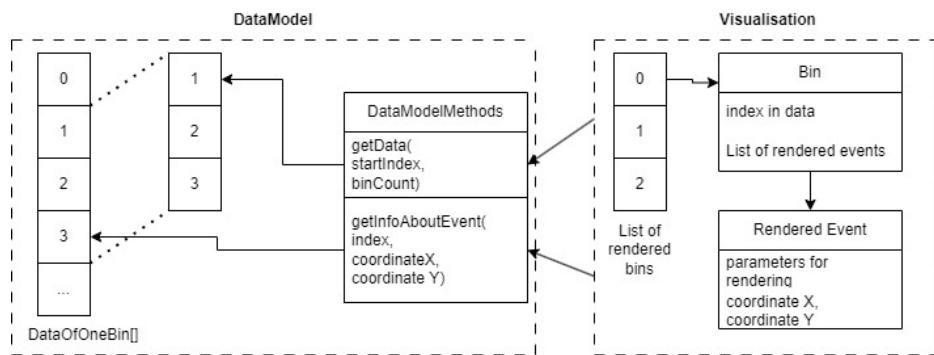
na útání, kdy budou data kompletní. Při ní budou seznamy eventů sloučeny do sebe a následně z nich budou odstraněny všechny eventy v kolizi a budou nahrazeny jedním sloučeným. Právě kvůli operaci *grouping* bylo rozhodnuto, že bude použit `LinkedList<E>`, protože ováni bude potreba méně režie než pro sloučení polí `ArrayList<E>` a dojde k odstranění prvků ze seznamu, u `ArrayList<E>` by pro to bylo potreba alokovat nové pole o menší velikosti.

4.6.4 Propojení datového a vizualizačního modelu

Tato část se zabývá návrhem propojení datového a vizualizačního modelu. Předem bylo stanoveno, že vizualizační model připotřebuje vizualizovat nová data, si je vyžadá od datového modelu přes definované metody. Je potreba vyřešit dva hlavní problémy, například v jakém formátu a podle jakých parametrů budou data předávána vizualizaci a zadruhé jak ve vizualizaci data označit tak, aby mohla být zpětně identifikovatelná (například při kliknutí myší na vizualizovaný event bude potreba zobrazit všechny dostupné informace o tomto eventu, takže se bude potreba zeptat datového modelu). Pro snazší pochopení je finální propojení znázorněno na obrázku 4.9.

Pro určení formátu předávání dat vizualizaci je potreba vzít v potaz, jakým způsobem vizualizace bude data potrebovat a jakým způsobem budou data uložena v datovém modelu. V sekci 4.6.1 bylo stanoveno, že data budou uložena do pole objektů reprezentujících binární sekvence vzestupně podle asových známek. K jednotlivým binárním sekvencím se bude přistupovat primárně podle indexu. Vizualizace (2D i 3D) bude vždy vyžadovat konkrétní část z celkových dat, které následně zobrazí. Bude znát tedy hranice tohoto okna, které chce zobrazit. Místož tedy datovému modelu poskytnout rovnou počáteční a koncový index tohoto okna. Vizualizace budou pracovat rovnou s velikostí zobrazeného okna, takže bude snazší, když datový model bude vracet data na základě počátečního indexu a počtu binárních sekvencí, které má vrátit. Aby byla poskytnutá data použitelná pro jakoukoliv vizualizační metodu, budou vrácena ve stejném formátu, v jakém jsou uložena a vizualizace vezme pouze ta data, která bude potrebovat. Vraceno tedy bude podpole celkových dat o velikosti žádaného počtu binárních sekvencí, který byl v originálním poli na daném indexu. Vrácena bude plná kopie původních dat, aby vizualizační model nemohl přepsovat data v datovém modelu. Dvě vizualizační metody zvolené pro tento práci budou vyžadovat: souadnice jednotlivých eventů (poslouží k určení souadnic jednotlivých pixelů i voxelů) a indexy jednotlivých binárních sekvencí.

Tímto způsobem si vizualizace získá data, která chce aktuálně vykreslit. Je ještě potřeba určit, že v obecném směru, kdy bude chtít vizualizace zjistit informace o konkrétním pixelu i voxelu. Lze to využít jednoduše tak, že si vizualizace uloží ke každému vykreslenému objektu všechna získaná data a nebude se na ně muset dotazovat datového modelu. Tato varianta ale není preferována, protože bude docházet k vyšší paměti na paměti a hlavně k redundancii dat. Je tedy potřeba určit systém, jakým označit vykreslený event tak, aby ho datový model dokázal jednoznačně identifikovat. V této části 4.6 bylo stanoveno, že každý event lze identifikovat indexem binu, ve kterém event leží a jeho souřadnicemi X a Y. Každý vykreslený bin si musí uložit informaci, jaký index představuje v celkových datech a každý vykreslený event si musí ponechat informaci o souřadnicích X a Y. Ty sice už nemusí mít smysl, protože budou použity k určení jeho polohy, mohou ale být předmětem ústřívané podle požadavků dané vizualizace. Originální souřadnice si tedy každý event ponechá, budou uloženy v datovém typu short, jako v povodních datech. Při žádosti o více informací o konkrétním eventu tedy vizualizace předá datovému modelu index binu, ve kterém se Event nachází a jeho povodní souřadnice.

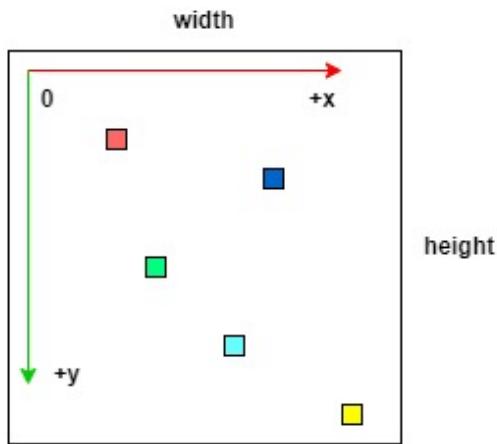


Obrázek 4.9: Navržené propojení datového modelu a vizualizace. Obrázek byl vytvořen v programu draw.io. [10]

4.7 2D metoda

V této části je rozepsán návrh implementace 2D metody. Typ 2D metody byl zvolen v této 3.2.1 jako metoda, kdy budou sloučena data z několika binů do jednoho. Pixely z různých binů, které jsou v kolizi (mají shodné souřadnice), budou sloučeny do jednoho a jeho hodnota ToT bude součtem všech ToT jednotlivých pixelů. U této metody se ve vizualizaci ztrátí informace o tom, z jakých asových známků data pocházejí. Proto bude potřeba

doimplementovat funkciu, ktorá tyto informace obstará jiným zp slobom. Ta bude zahrnuta do interaktivitu vizualizace pomocí myši, kdy na každý pixel p jde kliknout a zobrazit si o n m informace, v etn asových známech, ze kterých byl pixel složen. Tato interaktivita bude implementována odchytáváním udalostí myši **MouseEvent** a detekci, zda uživatel kliknul na konkrétní pixel, informace budou následn získány z datového modelu metodu popsanou v ásti 4.6.4. Znázorn ní navržené 2D vizualizace je znázorn n na obrázku 4.10.

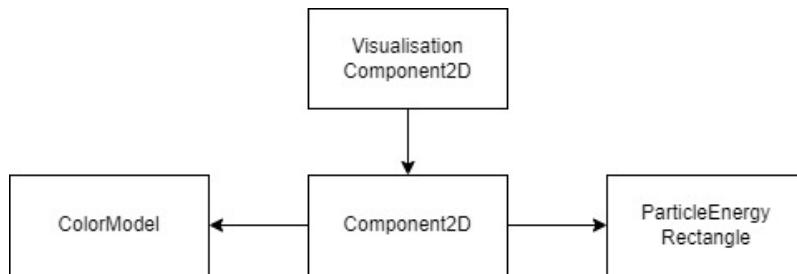


Obrázek 4.10: Navržené 2D vizualizace. Obrázek byl vytvo en v programu draw.io. [10]

Pro implementaci lze zvolit ze dvou základních možností, zmín ných v sekci 4.3.2, zda bude vykreslování provedeno p es t ídu *Canvas* nebo pomocí 2D objekt *JavaFX*. *Canvas* dává programátorovi v tší volnost s vykreslením, ale zárove si musí více funkcí implementovat sám, jako interaktivitu pomocí myši. P i zachycení **MouseEvent** p i stisku tla ítka myši je poteba získat její sou adnice a spo ítat, zda a na který tverec reprezentující udalost bylo kliknuto a následn ho zvýraznit. Ješt lze zvolit metodu s použitím tzv. image mapy, tedy neviditelné pole s interaktivními prvky nad pixely s energií. P i kliknutí na p íslušné pole se spustí pole pro zvýrazn ní p íslušeného pixelu. Vykreslení pomocí 2D objekt *JavaFX* (jakoLine, Rectangle d dících od t ídy j *javafx.scene.shape.Shape*) je snazší na implementaci, v etn implementace interaktivitu. Pro 2D objekty *JavaFX* lze p ímo nastavit detekci kliknutí a následné akce. Ješt je možné použít komponentu j *javafx.scene.image.ImageView*, která slouží k p ímému zobrazování rastrových obrázk , k vykreslení se jí p edá p ímo instance *javafx.scene.image.Image*. Do okna se pak umístí jako b žná GUI

komponenta. Implementováno bude ešení s vykreslením pomocí JavaFX 2D objekt a to hlavn z d vodu jednoduší implementace interaktivnosti. U n j je poteba zvolit, jak p esn mají být vykreslovány. Každý jednotlivý event bude reprezentován t ídu Rectangle, kterému lze přiřadit barvu a detekci interaktivnosti. Tato implementace mže mít nevýhodu nároku jeho výpočtu na výkonu při v tím množství vykreslených objekt , pokud bude event v jednom binu v tísni množství.

Na obrázku 4.11 se nachází návrh architektury 2D vizualizace. Na vrchu se nachází hlavní t ída vizualiza ní ásti, VisualisationComponent2D, která v sob drží instanci na t ídu Component2D. To je hlavní t ída vizualizace samotné, která provádí veškerou vizualizaci. Vykresluje jednotlivé eventy jako ParticleEnergyRectangle, který díky t ídě JavaFX 2D Rectangle a navíc obsahuje parametry potřebné pro zpracovou identifikaci eventu. T ída ColorModel slouží k provedení barev jednotlivým eventem na základ jejich hodnoty T_{OT} .



Obrázek 4.11: Navržená architektura 2D vizualizace. Obrázek byl vytvo en v programu draw.io. [10]

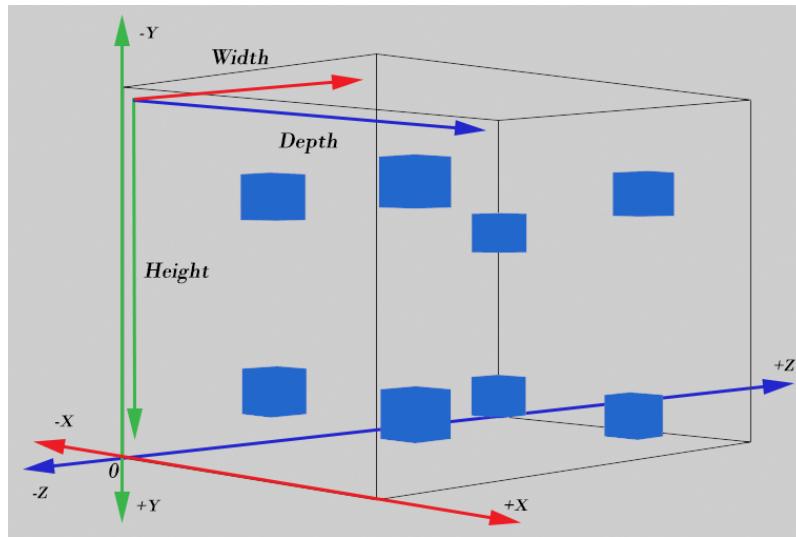
4.8 3D metoda

V této ásti je rozepsán návrh implementace 3D metody. V ásti 3.2.2 byla zvolena jako metoda, kdy bude použita tzv. voxelová grafika. Osa Z se použije jako asová osa, biny budou za sebe poskládány chronologicky a jednotlivé eventy budou transformovány na voxely. Na výbr jsou dv základní možnosti implementace. První je využít predefinovaných 3D objekt JavaFX jako je Box a umisťovat je přímo do scény na souadnice scény. Veškeré transformace a vykreslení pak provede sám framework JavaFX. Druhá, složit jí, je implementovat vlastní ešení. Souadnice, objekty a transformace bude vypoítávat vlastní ást a budou zobrazovány do scény napíklad pomocí vykreslení do Canvas. První ešení umožuje i jednoduší implementaci

interaktivit, protože lze nastavit detekci kliknutí pímo jednotlivým 3D objektům. Pro druhou variantu je to složit jí, protože je potřeba implementovat vlastní způsob. Při detekci kliknutí bude potřeba získat souřadnice myši a spočítat, zda a na kterou krychli bylo kliknuto. Druhá varianta je náročnější na výrobu, ale při vhodném zvoleném řešení může být lépe optimalizováno oproti vykreslování předdefinovaných 3D objektů JavaFX. To je zamezeno spíše na vykreslování objektů s podporou GUI. Zvolena bude první metoda, tedy použít definované 3D objekty JavaFX, hlavně z dudu jednodušší implementace.

V JavaFX se vykreslení 3D objektů provádí do samostatné scény, protože promítání 3D objektů vyžaduje jiné chování kamery a nasvícení. U 2D scén je kamera pouze v jedné pozici, u 3D scén ji ale může být potřeba nastavit. Pro separátní scénu se používá třída SubScene, kterou lze vložit do libovojí Sceny. V JavaFX 3D funguje klasický souřadnicový systém s osami X, Y a Z. Na obrázku 4.12 se nachází znázornění os X a Y v JavaFX a umístění vykreslovaných dat v souřadnicovém systému. Osa X je rostoucí směrem doprava, osy Y rostoucí směrem dolů. Stejný směr platí i pro souřadnice dat, po kterém ale bude posunutý nahoru o výšku vstupních dat. Na osách X a Y budou vynášeny jednotlivé krychle reprezentující eventy. Jejich souřadnice jsou k dispozici již v datech, mohou být tedy rovnou použity i pro vizualizaci. Souřadnice osy Z je potřeba nastavit. Všechny krychle jednoho binu budou umístěny do komponenty Group (layout komponenta pro umístění množiny prvků), Group s krychlemi tedy bude reprezentovat jeden bin. Souřadnice Z budou pro jeden bin společné, souřadnice bude přidána pímo Group reprezentující bin, takže se posunou i všechny krychle které Group obsahuje. Při vykreslování je potřeba vztít v potaz, že osa Y v JavaFX ve výchozím nastavení směřuje směrem dolů.

Opět je na výběr několik možností, jak z voxelů *bin* sestavit. Tím, že se jedná o provedení z pixelů na voxelu, můžeme vykreslit všechny voxely, i ty u kterých nebyla naměřena energie. To by ale mohlo být výpočetně náročné. Například při délkách hran 256 pixelů je obsah roven $65 \cdot 536 = (256 \times 256)$. Výpočetní výkon by pak ale mohl být zbytečně vytížen vykreslováním voxelů bez energie. Navíc by bylo potřeba vyřešit přesnost jednotlivých voxelů, aby byly dobře viditelné i další biny. Dále je možnost vykreslovat pouze voxelů se zaznamenanou energií. Takové řešení se zdá jako nejjednodušší a díky 3D prostoru možná i nejvhodnější, protože nebude potřeba vyřešit problém s přesností voxelů s nezaznamenanou energií. Zvoleno tedy bylo řešení vykreslovat pouze eventy se zaznamenanou energií.

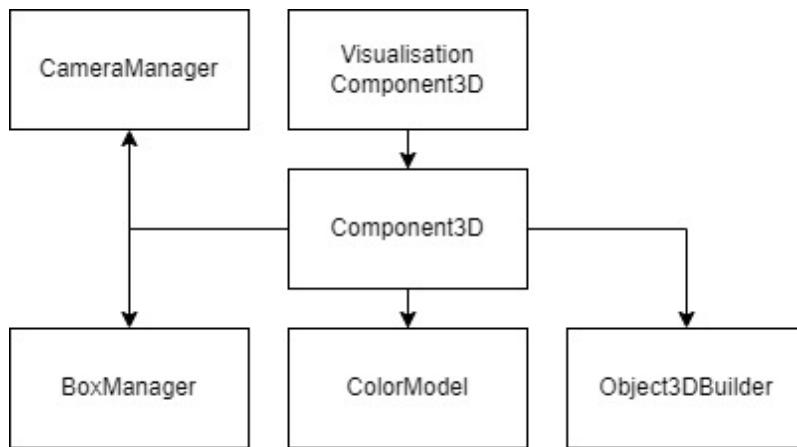


Obrázek 4.12: Navržené 3D vizualizace. Návrh byl vytvořen v programu blender. [4]

Architektura této 3D vizualizace se nachází na obrázku 4.13. Je rozdělena tak, aby každá část obstarávala jakou z hlavních funkcionalit 3D vizualizace. Hlavní část vizualizace níjmá střední komponenty je VisualisationComponent3D, která v sobě drží instanci na Component3D. To je hlavní část vizualizace samotné, bude v sobě držet data vizualizace a mít k distribuovat vizualizaci operace. Tato BoxManager se zaměří na předení event do voxelů a jejich vykreslení, ColorModel bude reprezentovat barevný model písmazující barvy jednotlivých voxelů a Object3DBuilder obstará vytvoření a manipulaci všech ostatních 3D objektů. CameraManager je manažer poskytující kamery a operace pro ni.

4.8.1 Výpočet souřadnic pro osu Z

Osa Z reprezentuje asovou osu, souřadnice se dají spočítat podle jednotlivých binů. Ke každému binu je přidělena asová známka, ToA . Na základě této lze vypočítat souřadnice. Nelze použít výchozí hodnoty ToA přímo jako souřadnice, protože hodnoty ToA jsou vysoké a mohou být daleko od sebe. Dva vedle sebe ležící biny by pak mohly být v nedosažitelné vzdálenosti a uživatel by musel kamery dlouho posunovat. Navíc ToA prvního binu nebývá hodnota 0. Místo toho budou souřadnice počítány pomocí ToA jednotlivých binů a spočítané souřadnice budou přidovány indexem místem binu. Bylo by možné zvolit jednoduché řešení a všechny biny mít přidruženou souřadnici tak, aby byly od sebe vzdáleny konstantní vzdálenost. Tím by ale uživatel na



Obrázek 4.13: Diagram tříd 3D vizualizace. Obrázek byl vytvořen v programu draw.io. [10]

první pohled nedokázal rozeznat rozdíly v řadě mezi jednotlivými biny a jak jsou od sebe vzdálené. Následně jevy mohou být rozprostřeny po řadě několika ToA a později se to právě tím, že tyto ToA budou souběžně blízké. Současně tedy musí zachovat vizuální vzdálenosti od sebe.

Současně tedy budou spočítány podle následujícího algoritmu. Vezme se minimální rozdíl ToA mezi dvěma binami nalezený v aktuálních na těchto datech a prohlásí se za jednu jednotku míry, protože menší vzdálenost se v datech nevyskytne. Výsledné hodnoty vzdálenosti budou násobky této minimální jednotky. Následně budou biny iterovány a spočítají se vzdálenosti mezi nimi jako rozdíl jejich ToA . Jak je popsáno v části 4.6, třída reprezentující bin bude mít dvě hodnoty ToA kvůli operaci *grouping* a to právě kvůli určování současných osy Z. Pokud byla operace provedena, první ToA značí nejnižší a ze sloupu ených bin druhá ToA nejvyšší. Vzdálenosti mezi dvěma binami jsou počítány jako rozdíly k sobě bližších současně, tedy druhá ToA i -tého binu a první ToA binu na indexu $i + 1$. Pokud operace provedena nebyla, hodnota obou ToA je shodná a výpočet zafunguje stejně. Takto se spočítají všechny vzdálenosti mezi biny a následně budou porovnány s minimálním rozdílem ToA mezi biny tak, že jsou tímto minimem vydleny podle vzorce 4.1. Zároveň bude ještě zaveden parametr maximální povolené vzdálenosti mezi daty pro případ, kdyby i po provedení byla vzdálenost příliš vysoká a pro vizualizaci nepoužitelná. Pokud bude vypočítaná vzdálenost vyšší než definované maximum, bude použito právě toto maximum. Podle tohoto algoritmu budou spočítány vzdálenosti mezi jednotlivými biny. Následně ještě potřeba ještě současně konkrétním binem. Současně

prvního binu bude stanovena jako 0. K následujícím bin m se vždy p i te jejich vzdálenost od posledního binu.

$$final_range = \frac{init_range}{MIN_range} \quad (4.1)$$

4.8.2 Kamera

Kamera se chová rozdíln pro 2D a 3D prvky GUI, takže pro **SubScene** s 3D vizualizací bude vytvo ena nová samostatná kamera. Aplikace pak bude mít dv kamery, jednu výchozí pro hlavní scénu obsahující komponentu a další 2D prvky a druhou vytvo enou ist pro 3D prvky a umíst nou do *SubScene* komponenty. Kameru lze umístit do scény jako jakýkoliv jiný prvek JavaFX. Pro 3D kameru je pot eba nastavit n kolik parametr rozdílných od 2D kamery, podrobný popis se nachází pod odkazem [22]. Na ukázce kódu níže jsou zobrazeny dva možné konstruktory pro kameru. Druhý konstruktor Perspecti veCamera(**bool** *ean* fi xedEyeAtCameraZero) byl p edstavený v Java 8 a obsahuje navíc parametr fi xedEyeAtCameraZero. Ten ur uje, zda má kamera z stat na stejné pozici, když dojde ke zm n promítané oblasti nebo velikosti okna. P i nastavení na *true* bude o ka kamery umíst na na sou adnice (0, 0, 0) svého sou adnicového systému. P i výchozím nastavení *false* má kamera definovaný sou adnicový systém s definovaným po átkem v levém horním rohu scény jako pro 2D. P i zm n velikosti okna dojde ke zm n pozice kamery tak, aby z stala v levém horním rohu okna. Takové nastavení se hodí pro 2D rozložení, ale ne pro 3D. U 3D naopak je pot eba, aby kamera z stala na míst a zobrazovala prvky stále stejn . Pro 3D kameru je tedy pot eba nastavit tento parametr jako *true*.

Dalším parametrem kamery je **Field of View** (esky zorné pole). Udává, jak velký úhel má kamera zabírat, úhel je udán ve stupních. Udává, jak velký úhel má kamera zabírat, úhel je udán ve stupních. Hodnota bude nastavena na 30° , která podle dokumentace t ídy JavaFX *Camera* odpovídá módu *Telephoto*. Posledními d ležitými parametry jsou **Near Clip** a **Far Clip**. Zná í o íznutí pohledu tak, aby byl z n ho zobrazen jen požadovaný objem. Znázorn ní t chto parametr se nachází na obrázku 4.14.

```

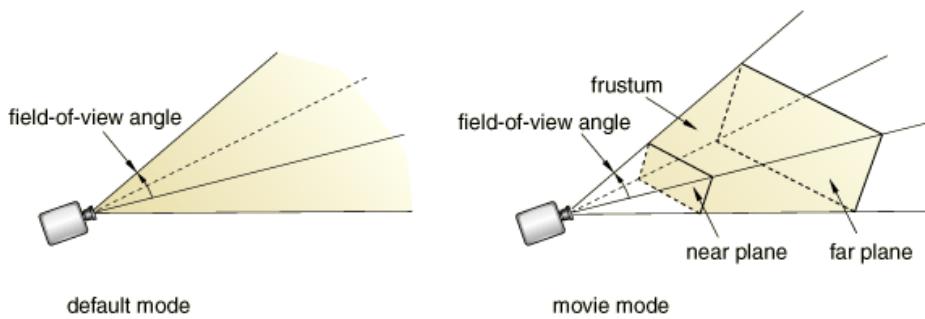
PerspectiveCamera()
PerspectiveCamera(boolean fixedEyeAtCameraZero)

Camera camera = new PerspectiveCamera(true);
scene.setCamera(camera);

camera.setFieldOfView(double value);
camera.setNearClip(double value);
camera.setFarClip(double value);

```

Ukázka kódu 4: Znázornení nastavení kamery v JavaFX 3D



Obrázek 4.14: Ukázka parametrů kamery Field of View, Near Clip a Far Clip. [5]

5 Implementace komponenty

Tato kapitola se vnuje konkrétnímu popisu vyhotovené implementace 2D a 3D komponent a jejich příslušných demonstračních aplikací. Celkem byly vyhotoveny dvě komponenty, jedna pro 2D a druhá pro 3D metodu, a pro každou komponentu demonstrační aplikace. Kapitola je strukturována na popis dležitých funkcionalit, v nich kterých případech je možné vidět a zdrojový kód. Popis funkcionalit je zaměřený na konkrétní popis algoritmu a fungování jednotlivých procesů, podrobný popis jednotlivých funkcí je pak obstarán prostřednictvím dokumentačních komentářů ve zdrojových souborech komponent a aplikací. Tyto jsou popsány slovně i znázorněny pomocí UML (Unified Modeling Language) diagramu. [37]

Obecné parametry vyhotovené implementace:

- Obě komponenty jsou rozděleny na dvě části, **BaseComponent** a **VisualisationComponent**. *BaseComponent* obstarává datové operace a je shodná pro obě komponenty. *VisualisationComponent* obstarává vizualizační operace.
- Data a parametry komponenty lze upravovat direktně, podle typu parametru. Parametry části *BaseComponent* lze upravovat přes samostatné zobrazitelné okno nazvané „Component Settings“. Jsou to parametry nezávislé na vizualizační metodě. Parametry vizualizační metody pak lze upravovat zvnějšku komponenty, přes prvky aplikace. Změny ve vizualizaci se projeví ihned.
- Data a parametry o na těchto datech a vizualizaci jsou ukládány do přes JavaFX StringProperty v definovaném formátu. Tento Properties je několik, každá pro jiný typ dat. Lze je z komponenty exportovat a přidat do příslušného GUI pole aplikace.
- Komponenta dle odyssey JavaFX Group, lze umístit do scény jako běžnou JavaFX komponentu.
- Vizualizace v komponentách se ovládá pomocí myši a klávesnice (konkrétní klávesy jsou definovány v uživatelském příručce), lze posouvat vizualizované okno a v případě 3D komponenty lze klávesami ovládat i kamery a posouvat je ve vizualizaci. Myší lze zvýraznit konkrétní vykreslený event a u 3D vizualizace lze ovládat i kamery.
- Při vývoji komponent a aplikací byl použit nástroj *Maven*.

5.1 Architektura

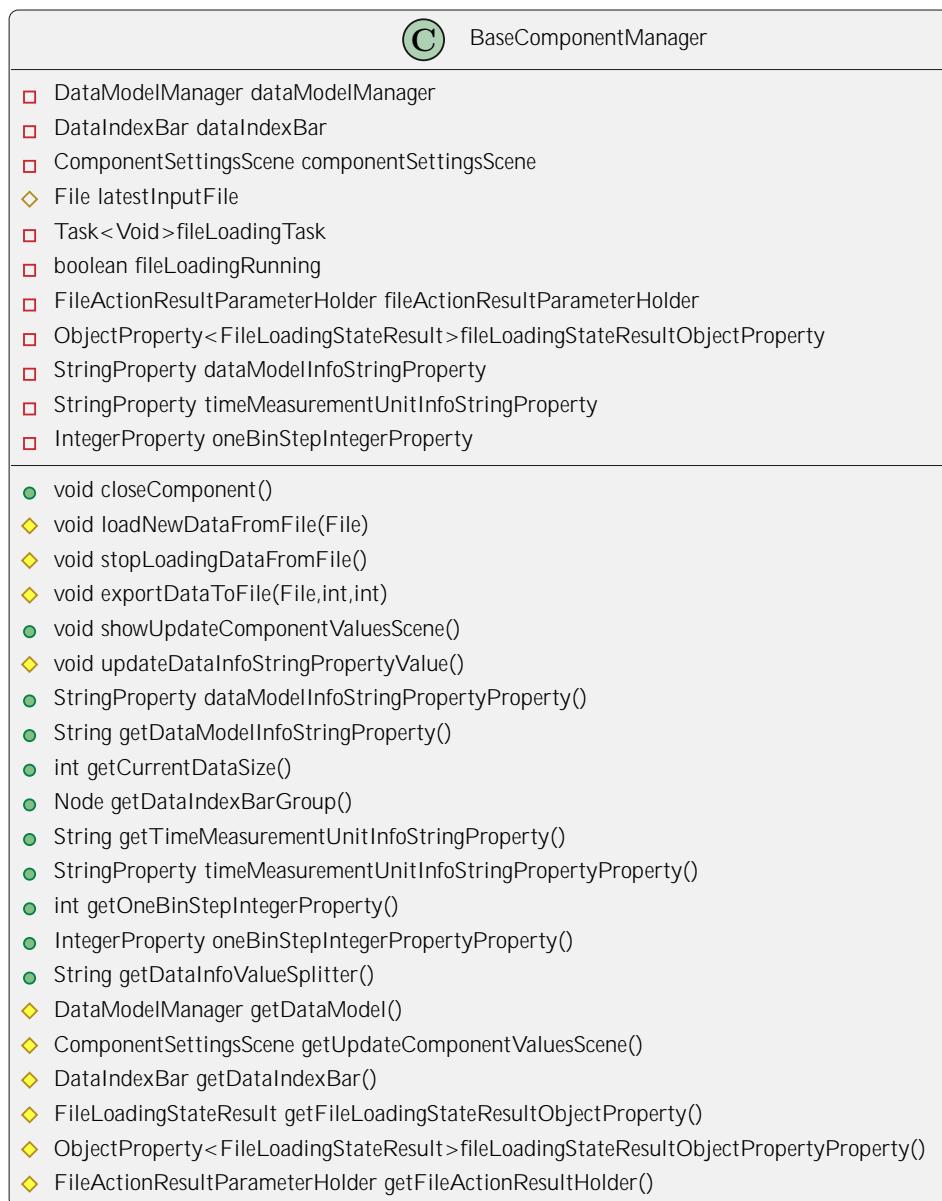
Tato část je zaměřena na popis architektury vyhotovené komponenty.

5.1.1 Architektura obecně

Architektura komponenty byla vyhotovena podle návrhu v sekci 4.4 a diagramu 4.6. Komponenta je rozdělena na dvě části, **BaseComponent**, poskytuje hlavní datové operace, a **VisualisationComponent**, poskytuje hlavní vizualizaci. *BaseComponent* je společná pro obě komponenty a pokrývá veškerou funkcionalitu nezávislou na vizualizaci metod. Každá část má svou hlavní třídu, řídící všechny operace dané části. K *BaseComponent* patří hlavní třída nazvaná jako *BaseComponentManager*. K *VisualisationComponent* patří podle konkrétní komponenty *VisualisationComponent2D* nebo *VisualisationComponent3D*. *BaseComponent* poskytuje všechny operace společné pro jakoukoliv vizualizaci metod, konkrétně: uložení dat a datové operace jako načítání ze souboru (část 5.2), uchování dalších dležitých hodnot komponenty a jejich aktualizace (část 5.3), grafický ukazatel aktuální pozice zobrazeného okna v celkových datech (část 5.7; lze exportovat mimo komponentu a uložit do scény; jedná se o část vizualizace, ale takový ukazatel je užitečný nezávisle na implementované vizualizační metodě, protože je implementován v *BaseComponent*) a několik JavaFX Property udávající informace o načtených datech, zadánych parametrech komponenty a aktuální index počtu vizualizovaného indexu. Část *VisualisationComponent* je implementovaná specifiky pro každou metodu, jejich struktury jsou si podobné, aby byly jednodušší na porozumění. Obecné obě poskytují získání *Node* obsahující vizualizaci, ve které metody pro úpravy vizualizace, manažera obsluhy události z klávesnice (na úpravu vizualizace apod.) a JavaFX Property udávající informace o vizualizaci a o aktuálně vybraném eventu.

5.1.2 Hlavní třídy komponenty

Hlavní třída části **BaseComponent** se nazývá *BaseComponentManager* a dříve od téží JavaFX Group. Hlavní třída části **VisualisationComponent** má název pro 2D komponentu určen jako *VisualisationComponent2D* a pro 3D komponentu *VisualisationComponent3D*. Instanci tříd *VisualisationComponent2D* nebo *VisualisationComponent3D* umístí programátor jako grafický prvek *Node* do scény i v jakém *layout* komponenty JavaFX. V demonstrační aplikaci je umístěno do komponenty *BorderPane* do středové části nazývané *MiddlePane*.



Obrázek 5.1: UML diagram třídy `BaseComponentManager`

Na obrázku 5.1 se nachází UML diagram třídy `BaseComponentManager`. Z metody je zde zmíněn konstruktor `BaseComponentManager()` se třemi parametry: `boolean calculateZAxis` - indikátor, zda má být v datovém modelu spořitány souřadnice pro osu Z. Nastavení je závislé na konkrétní vizualizaci metod, pro 3D je zapnuté, pro 2D vypnuto. Druhý parametr `boolean turnOnLogging` - indikátor, zda má být zapnuto logování do konzole pro všechny operace komponenty. Komponenta navíc loguje ve vlastním

formátu. Tento parametr `bool canIndex` indikuje, zda má být vytvořen grafický ukazatel pozice vizualizovaného okna v celkových datech. Programátor nemusí chtít tento ukazatel využívat a může si zajistit zobrazení této informace jinou cestou. Další dležitou metodou této části je `void LoadNewDataFromFile(File inputFile)`, která slouží ke spuštění na čtení ze souboru. V této metodi se pro to vytvoří samostatné vlákno, které je uloženo do parametru `Task<Void> fileLoadingTask`. Po dokončení na čtení je výsledek uložen do příslušné struktury a hlavní čítačka vizualizace pak zahájí následné operace. Je povoleno pouze jedno aktivní vlákno, pokud uživatel zažádá o další na čtení, je jeho požadavek odmítnut.

Hlavní čítače visualizací `onComponent2D` a `onComponent3D` obsahují oba dva konstruktory, jeden s prednastavenými parametry komponenty a druhý dává programátorovi možnost nastavit si všechny parametry sám (detailní popis se nachází v uživatelském průřezu, která je součástí příloh práce). Dále obsahuje všechny metody obstarávající funkcionality komponenty: načtení dat, úprava parametrů a získání `DataIndexBaru` a `Properties` obsahující informace a parametry komponenty.

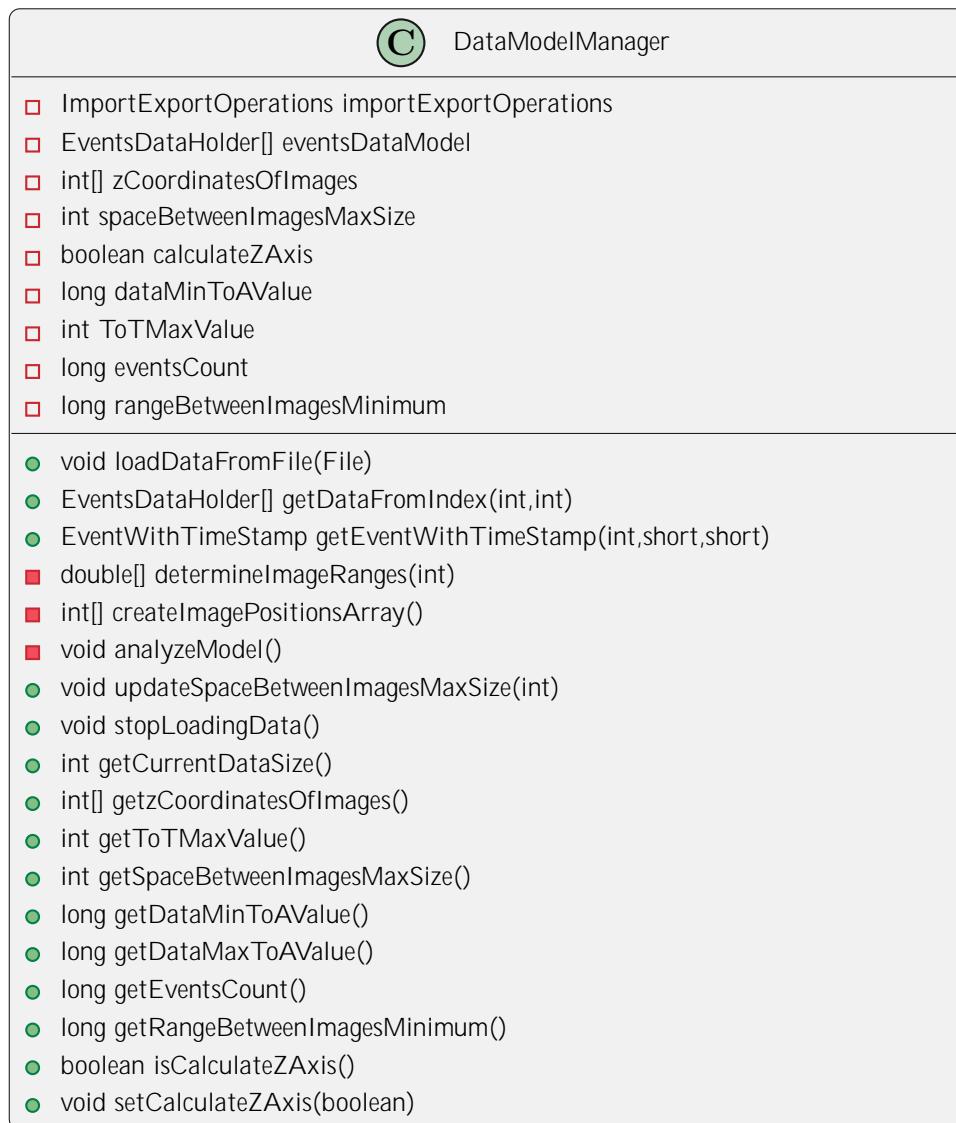
5.2 Datový model

Tato část je zaměřena na popis balíčku s datovými operacemi `DataOperations` a všech datových operací, které komponenta provádí. Uložení dat je vymodelováno podle návrhu v části 4.6.

5.2.1 Struktura a fungování datových operací

Hlavní čítače datového modelu je čítačka `DataModelManager`, znázorněna na obrázku 5.2. Tato čítačka v sobě uchovává uložená data pro vizualizaci a slouží jako zprostředkovatel datových operací. Operace jsou sama prováděny nebo distribuuje jejich provedení. Základní funkcionalita, kterou poskytuje: načtení dat ze souboru, spočítání a uložení pozic pro osu Z (volitelné podle použité vizualizační metody), spočítání statistických údajů na tených datech, poskytnutí dat podle počtu indexu a požadovaného binárního indexu, poskytnutí konkrétní události podle indexu a souřadnic X a Y (tato trojice parametrů jsou minimální nutné parametry k jednoznačné identifikaci konkrétní události), uložení indexu binárního podle jedné asové známky a metody pro aktualizaci kterých parametrů datového modelu. Při získání dat podle indexu nejsou vráceny reference do struktur s uloženými daty, ale jsou vytvořeny.

eny jejich kopie. Je tak u in no z bezpe nostních d vod , aby vizualiza ní metody nemohly modifikovat uložená data.

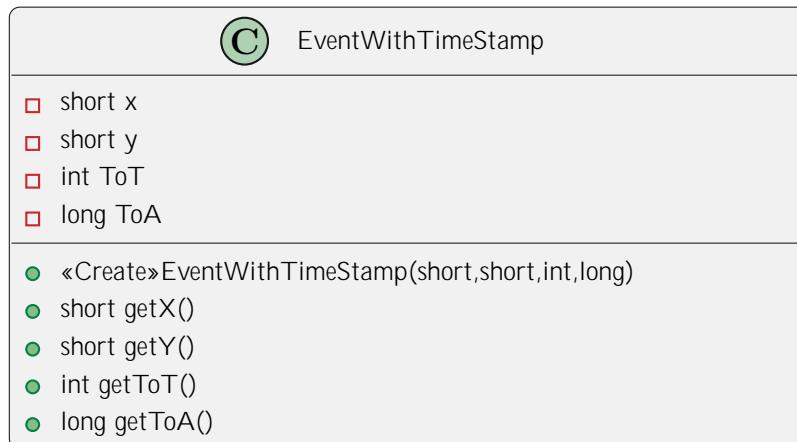


Obrázek 5.2: UML diagram t ídy DataModelManager

5.2.2 Uložení dat

V této ásti je popsán zp sob uložení dat ur ených k vizualizaci, které jsou uchovávány ve t íd DataModel Manager. K jejich uložení se používají t ídy, které jsou uložené v balíku BaseComponent. DataOperati ons. EventRepresentati ons:

EventWithoutTimeStamp (UML diagram na obrázku 5.4), EventWithTimeStamp (UML diagram na obrázku 5.3) a BinData (UML diagram na obrázku 5.5). Jednotlivé eventy jsou reprezentovány třídami EventWithoutTimeStamp nebo EventWithTimeStamp. Použití se rozlišuje podle toho, zda je potřeba přidat ke konkrétnímu eventu i adresovou známku, ToA. BinData slouží jako reprezentant jednoho binu, celkové uložení dat je reprezentováno jako pole tříd BinData se zadaných vzestupně podle ToA jednotlivých binů.



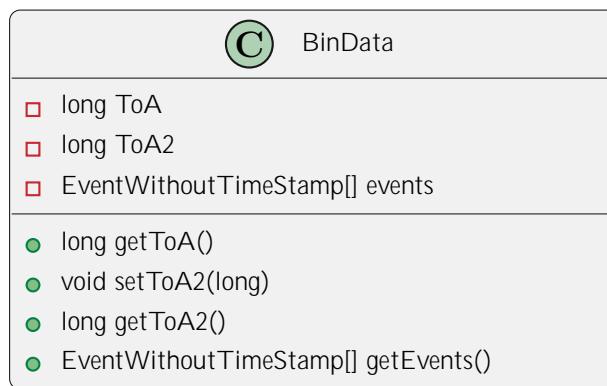
Obrázek 5.3: UML diagram třídy EventWithTimeStamp



Obrázek 5.4: UML diagram třídy EventWithoutTimeStamp

5.2.3 Načítání dat ze souboru

Na obrázku 5.6 se nachází UML diagram tříd poskytující na čítání ze souboru. Skládají se z abstraktní třídy DataDrivenFileParser, rozhraní IFileParser a třídy FileParser, všechny tyto třídy se nachází v balíku



Obrázek 5.5: UML diagram třídy BinData

DataOperations. Třída FileParser implementuje rozhraní IFileParser a díl od abstraktní třídy DataDrivenFileParser.

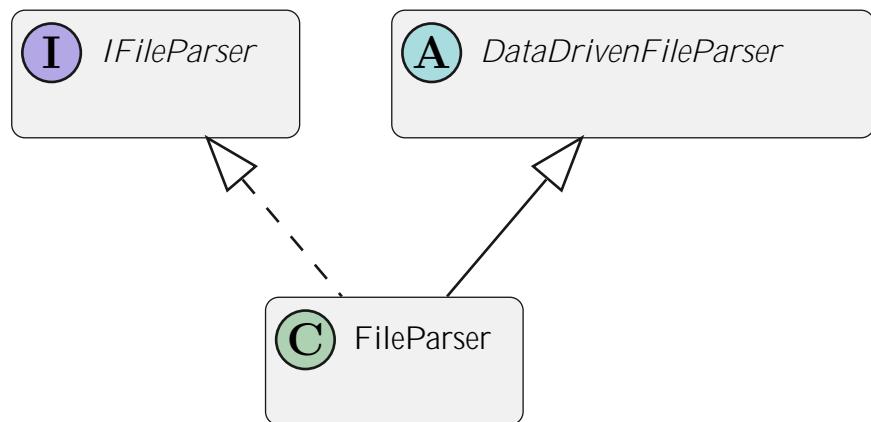
Instance rozhraní IFileParser je umístěna jako parametr třídy ImportExport. Rozhraní je zavedeno z toho důvodu, aby byly znění metody na čítání dat snadno zaměnitelné. Rozhraní definuje metodu pro načítání dat

`IoadDataDrivenDataFromFile(File)`, která vždy musí vrátit načíta data ve struktuře `SortedMap<Long, BinData>`, ve které jsou uloženy jednotlivé biny. Jako klíč je použita `ToA` binu a jako hodnota je třída `BinData` reprezentující jeden bin. Dále rozhraní ještě definuje *getter* a *setter* parametru, zda má být provedeno na čítání dat (při požadavku o provedení od uživatele je zavolán práv *setter*, který nastaví hodnotu na `true`).

`DataDrivenFileParser` obsahuje pouze jednu metodu určenou ke konverzi ádky s eventem ze vstupního souboru do třídy `EventWithoutTimeStamp`. Tato třída je abstraktní, protože její funkce je použitelná pro jakoukoliv implementovanou metodu na čítání ze souboru. Po skončení na čítání je na získaných datech provedena operace *grouping*, pokud to tak uživatel zvolil, a následně jsou data vrácena.

5.2.4 Grouping

V této části je popsána implementace operace **grouping**, která provádí sloučení dat z kolika binů do jednoho. Operaci je implementována ve třídě `GroupingTool`, je volitelná a uživatel si může zvolit, z kolika snímků budou data sloučována. Pokud zvolí hodnotu 1, *grouping* neprobíhne. Finálnímu binu jsou přidány dvě asové známky, nejnižší a nejvyšší ze sloučených binů. Jsou použity dvě hodnoty následného určování souřadnice Z ve 3D vizualizaci. Při konfliktu eventů (eventy ze sloučených binů mají shodné



Obrázek 5.6: UML diagram tří file parseru

sou adnictví) dojde k jejich sloučení do jednoho eventu, výsledná hodnota ToT je součtem ToT všech jednotlivých událostí. Dochází k ní okamžit po dokončení na čítání dat ze souboru, aby bylo zajištěno, že jsou všechna na tená data seřazena.

Funkce operace **grouping** je znázorněna na ukázce kódu 5, funguje následovně: Vytvoří se nová instance `SortedMap<Long, BinData>`, do které budou ukládána nová upravená data, původní mapa zůstane zachována. Nová mapa se nazývá `newMap`, původní originální `Map`. Iteruje se nad původní mapou, nová data jsou ukládány do nové mapy, podle hodnoty počítadla počítadla `counter` se pozná, zda se mají data aktuálního binu přidat do existujícího binu v nové mapě i zda bude vytvořen nový. Po dokončení této iterace jsou v nové mapě uloženy nové biny, ještě je potřeba provést sloučení událostí v konfliktu. Operace provedena iterace nad daty a všechny eventy v binách jsou prozkoumány a případně sloučeny.

```

SortedMap<Long, BinData> originalMap,
SortedMap<Long, BinData> newMap;
int binCount
int counter = binCount;

// iterate over original map
for(Long ToA : originalMap.keySet()){
    if(counter == binCount){
        // get current bin from original Map
        // and create new bin in newMap containing
        // events of bin from original map

        counter = 1;
    }
    else{
        // get current bin from originalMap and last
        // created bin from newMap
        // add events from original bin to bin in new map

        counter++;
    }
}

// iterate over newMap
for(Long ToA : newMap.keySet()){
    // merge events in conflict
}

```

Ukázka kódu 5: Znázorní algoritmu operace *grouping*

5.2.5 Důležité hodnoty komponenty

V balíku DataOperations se nachází třída ComponentParameters obsahující dležité hodnoty komponenty. Jedná se o hodnoty nezávislé na vizuální metodě, například parametry detektoru, ze kterého byla data získána nebo z kolik snímků má být sloučováno při operaci *grouping*. Tyto hodnoty jsou uloženy jako ve ejné statické proměnné a je dovoleno je mít pouze skrze okno vytvořené samotnou komponentou. Jedná se o okno definované ve třídě ComponentSettingsScene, popsané v části 5.3. Konkrétně jedná o sílu a výšku vstupních dat (`imageWidth`, `imageHeight`), konstanty sloužící k výpočtu *ToA* při návštěvě (`ToAConstant1`, `ToAConstant2`), et cetera znázorňující komentáře v vstupním souboru (`lineCommentMark`), počet udávající kolik binů má být sloučeno při operaci *grouping* (`binCount`) a oddělovač hodnot ve `StringProperties` pro uložení informací, které si mohou programátor z komponenty exportovat (`dataInfoValueSplitter`). Tato hodnota je *final*, její změna není žádoucí. Znázornění se nachází v ukázce kódu 6.

Všechny proměnné obsahují nějakou hodnotu, žádná nesmí být prázdná. Hodnota všech řízených proměnných musí být kladná, et cetera `lineCommentMark` nesmí být prázdný. Úpravy lze provést kdykoliv, nicméně v těch funkci pracujících s těmito hodnotami je vytvořena tak, aby se změny neprojevily hned, ale těeba až pro další datovou sadu. Proto je ideální hodnoty mít před načtením nové datové sady.

```

public class ComponentParameters {
    public static int imageWidth = 256;
    public static int imageHeight = 256;
    public static long ToAConstant1 = 25000;
    public static long ToAConstant2 = 1562;
    public static String lineCommentMark = "#";
    public static int binCount = 1;
    public static final String dataInfoValueSplitter = "|";
}

```

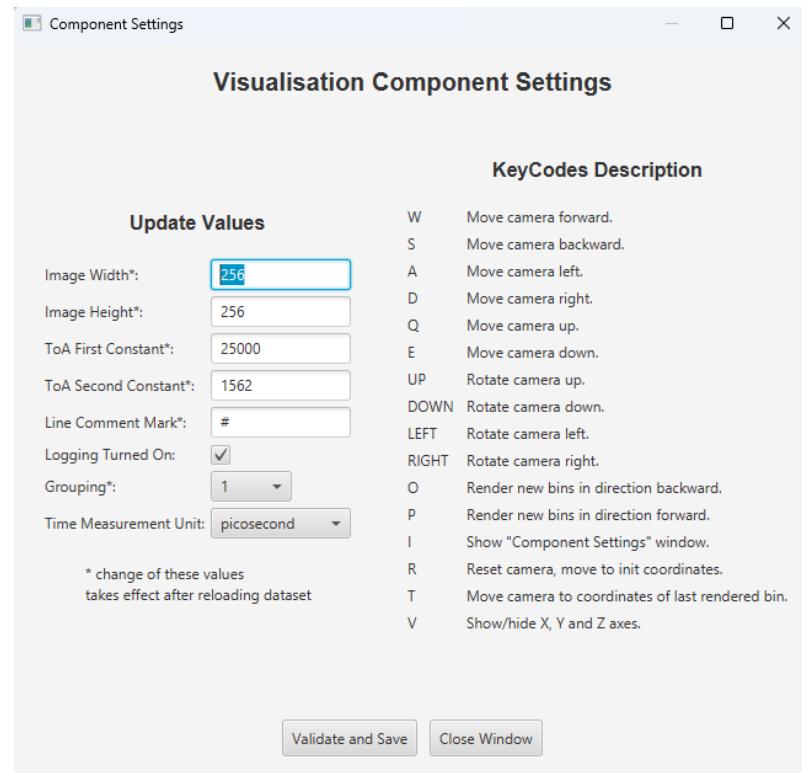
Ukázka kódu 6: Uložení parametr komponenty

5.3 Okno Settings

V této části je popsána implementace okna pro aktualizaci parametr části *BaseComponent*. Konkrétně se jedná o tzv. dležité hodnoty komponenty (popsané v části 5.2.5), zapnutí logování a nastavení jednotky asu pro vstupní data. Dále pak ukazuje seznam kláves k ovládání vizualizace s popisem jejich významu. Tento seznam je do scény nahrán ze třídy *KeyEventHandler*, která se stará o obsluhu událostí z klávesnice. Toto okno je definováno ve třídě *ComponentSettingsScene* v balíku *BaseComponent.CustomComponentScenes*. Tato třída má jako hlavní parametr Stage *importantComponentValuesWindow*. Jedná se o Stage, která reprezentuje okno pro update hodnot komponenty. Základní Stage *primaryStage* je použit pro okno demonstrativní aplikace. *ImportantComponentValuesWindow* je nová Stage reprezentující nové okno. Modalita okna je stanovena na *Modality.APPLICATION_MODAL*, to znamená, že při jeho zobrazení je responzivní pouze toto okno. Díky tomu například není potřeba kontrola, aby bylo najednou vytvořeno pouze jedno toto okno, protože okno aplikace je zablokováno. Layout komponenty je dle třídy *BorderPane*. Okno je ukázáno na obrázku 5.7.

5.4 2D Vizualizace

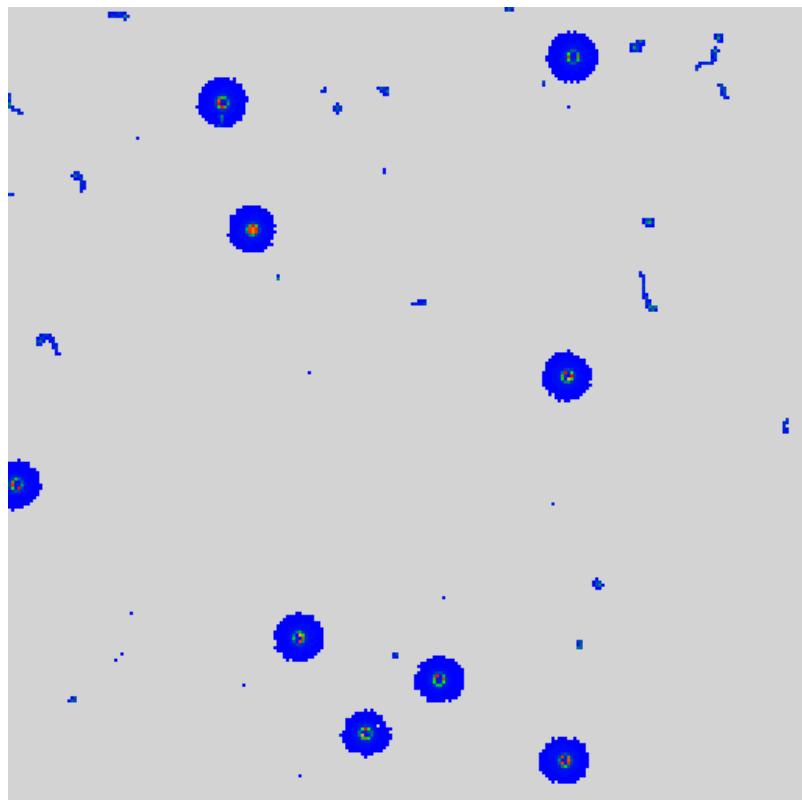
V této části je popsána implementace 2D vizualizací metod. Podle návrhu jsou sloužovány biny z rozšíření daného uživatelem do jednoho binu a tento výsledný je vykreslen jako 2D obrázek. Implementace je vytvořena v části komponenty VisualisationComponent a ukázka vizualizace na obrázku 5.8. Nejdříve je popsána vizualizace z vizuálního hlediska a následně z programového.



Obrázek 5.7: Okno s nastavením parametr komponenty

Vizualizace je založena na tom, že data z jednotlivých bin jsou slou ena do jednoho a ten je vykreslen jako 2D obrázek. Rozp tí, kolik bin bude zobrazeno, ur uje uživatel. Klávesami O a P m že posouvat s vizualizovaným oknem dop edu a dozadu. Rychlosť vykreslování dalších bin lze nastavit a lze vykreslit data na konkrétním indexu nebo asové známce. Jednotlivé eventy jsou vykresleny jako tverce, barva je jim p id lena barevným modelem, vychází z hodnoty ToT p íslušného eventu. Na každý tverec lze kliknout a ten se následn zvýrazní (jeho barva je zm n na na r žovou). Informace o vybraném eventu jsou uloženy do p íslušné StringProperty pomocí které aplikace do p íslušného TextArea tyto informace vypisuje.

Hlavní t ídu 2D vizualiza ní ásti je t ída VisualisationComponent2D, p es tuto t ídu jsou provád ny všechny akce k vizualizaci. Samotná vizualizace je pak provád na p es jednu t ídu, Component2D. Jednotlivé tverce reprezentující události jsou vymodelovány pomocí t ídy ParticuleEnergyRectangle, která dí od JavaFX 2D t ídy Rectangle a obsahuje navíc p vodní X a Y sou adnice eventu.



Obrázek 5.8: Znázornení 2D vizualizace

Nejprve je vykreslen obýejný Rectangle, který poslouží jako pozadí, na které budou eventy vykreslovány. Následně při získání požadavku na vykreslení dat jsou z modelu získána data k vizualizaci. Nejprve je potřeba všechny binární sloužit a se řídit hodnoty ToT eventů se shodnými souřadnicemi. Toho je dosaženo tak, že získané eventy jsou postupně umístovány do matice (odpovídá rastrovému obrázku), souřadnice matice jsou shodné se souřadnicemi eventů. Pokud už na nějaké pozici nějaký event umístíme, je k nimmužena hodnota ToT aktuálního eventu. Když jsou všechny eventy do matice umístěny, tak matice odpovídá rastrovému obrázku, který má být vykreslen. Projdou se všechny prvky matice a pro každý, ve kterém je zaznamenán event je vytvořen Partikel eEnergyRectangle. Ten získá barvu z barevného modelu podle jeho výsledné hodnoty ToT a je umístěn do Rectangle reprezentujícího pozadí na příslušné souřadnice.

5.5 3D Vizualizace

V této části je popsána implementace 3D vizualizace metodou. Podle návrhu byla použita tzv. voxelová grafika, kdy eventy v datech jsou vymodelovány

lovány jako krychle(voxely). Implementace je provedena v ásti komponenty *VisualisationComponent* a ukázka vizualizace na obrázku 5.9. Nejd íve je popsána vizualizace z funkcionálního hlediska a následn z programového.

Vizualizace je založena na tom, že data z jednotlivých bin jsou poskládána chronologicky za sebou. Hranice snímk jsou vyzna eny obalovým kvádrem, jeho ší ka a výška odpovídá ší ce a výše vstupních dat. Data jsou vykreslována od po átku sou adnicového systému. Vykreslena je pouze ást celkových dat, velikost vykreslené ásti je definována uživatelem a lze s oknem hýbat dop edu/dozadu klávesami. Hloubka obalového kvádru je nastavena na sou adnici posledního vykresleného binu. Jednotlivé eventy jsou vykresleny jako voxely/krychle pomocí t ídy JavaFX Box, jejich sou adnice odpovídají sou adnicím eventu. Krychle mají p id lenou barvu barevným modelem, vychází z hodnoty *ToT* p íslušného eventu. Na každou krychli lze kliknout, ta se následn zvýrazní (její barva je zm n na na r žovou) a v její pozici je zobrazen osový k íž, který zvýrazní její pozici. Informace o vybrané krychli jsou uloženy do p íslušné *StringProperty* pomocí které aplikace do p íslušného *TextArea* tyto informace vypisuje. P es definované klávesy lze posouvat a otá et kamenu ve vizualizaci. Vizualizovat lze ve dvou módech, které se liší v tom, jak se bude model vizualizace chovat p i posunu zobrazeného okna dop edu a dozadu.

- **STATIC** – zobrazené okno z stane „na míst“. Sou adnice Z prvního binu bude vždy 0, poslední sou adnice se aktualizuje podle sou adnice posledního binu.
- **MOVABLE** – p i posunu sou adnice Z sou asných bin z stanou stejné, nové biny se p idají na konec a ze za átku se uberoou. P i posunu dat je pak pak pot eba i posunout kamenu ve sm ru tvo ení dat.

Jednotlivé eventy jsou zobrazeny ve form voxel /krychlí. Jsou zobrazeny pouze eventy se zaznamenanou energíí. Krychle je vytvarována pomocí JavaFX 3D objektu Box. Krychle v sob nesou informaci o originální X a Y sou adnici, aby mohla být zp tn identifikovatelná v datovém modelu. Z toho d vodu je definován objekt *ParticleEnergyBox*, který dí od t ídy Box a navíc obsahuje jako dva parametry tyto sou adnice. Bin je reprezentován t ídou *Group*, také kv li zp tné identifikace musí v sob nést ílo indexu v datech, takže je definován objekt *BinGroup*, který dí od t ídy Group a má jako parametr práv ílo indexu. Parametry v *ParticleEnergyBox* a *ImageGroup* jsou *final*. Do seznamu prvk *Children* *BinGroup* je povoleno

Obrázek 5.9: Znázornení 3D vizualizace

umí sestavovat pouze objekty `Parti cl eEnergyBox`. Souadnice `Z` jsou pouze v binu. Na ukázce kódu 7 se nachází ukázka metody `buidParti cl eEnergyBox(EventWithoutTimeStamp eventToBuild)` k sestavení jedné krychle `Parti cl eEnergyBox` podle jedaného eventu. Na tomto příkladu je dobré znázornit, jak pracovat s objekty a souadnicemi. Lokální proměnné `x`, `y` a `z` defaultní slouží k uložení souadnic a hodnoty `ToT`. Souadnice `X` a hodnota `ToT` se ponechá bez změny, souadnice `Y` je potřeba doplnit. Jak bylo znázorněno na obrázku 4.12, osa `Y` v JavaFX 3D scéně směuje dolů. Data jsou vykreslována do horní části, ale osa `Y` ve zdrojových datech též směuje dolů. Je tedy potřeba k hodnotě `Y` přidat výšku vstupních dat. Tak se boxy v pořátku souadnic objeví v horním okraji obalového rámu, z hlediska dat stále budou mít souadnici 0. Následně je vytvořena instance `Parti cl eEnergyBox`, jako parametry musí být zadány originální souadnice `X` a `Y` a velikost hrany krychle. Velikost hrany je stanovena na hodnotu 1 pro všechny krychle, hodnota je nemnitelná.

```

Parti cl eEnergyBox buil dParti cl eEnergyBox(
    EventWi thoutTi meStamp eventToBui ld){

    fi nal int center = 0;
    int pi xel DefaultCol or;
    double x, y;

    pi xel DefaultCol or = eventToBui ld.getToT();
    x = eventToBui ld.getX();
    y = eventToBui ld.getY()
        - ImportantComponentVal ues.imageHei ght * boxSi ze;

    Parti cl eEnergyBox parti cl eEnergyBox =
        new Parti cl eEnergyBox(boxSi ze,
            eventToBui ld.getX(), eventToBui ld.getY());
    parti cl eEnergyBox.setMateri al (new PhongMateri al (
        col orModel.determi neCol orByPi xel Val ue(
            pi xel DefaultCol or))));

    parti cl eEnergyBox.setTransl ateX(center + x * boxSi ze);
    parti cl eEnergyBox.setTransl ateY(center + y * boxSi ze);
    parti cl eEnergyBox.setTransl ateZ(center);

    return parti cl eEnergyBox;
}

```

Ukázka kódu 7: Znázorní sestavení jednoho voxelu

Objekty 3D jsou uchovávány v jednotlivých Group podle významu, jednotlivé Group jsou uloženy do SubScene. Tyto prvky jsou v SubScene uchovány trvale, mní se pouze jejich obsah, tedy hodnoty seznamu prvků (Children) obsažených v Group. Celkem je Group použit pro obalový rám, pro všechny vykreslené krychle, pro osový kříž zvolené krychle, osy.

5.6 Barevný model

V této části je popsána funkce ColorModel reprezentující barevný model přiřazující barvy eventům ve vizualizaci. Je shodný pro 2D a 3D metodu. Vojená metoda třídy determineColorByPixelValue(int pixelValue) slouží

k p id lení barvy na základ p edané hodnoty ToT eventu. P i azení funguje na principu procentuálního rozd lení všech možných barev pro na tená data. T ída uchovává pole objekt `Color` s p edgenerovanými hodnotami dlouhé 101 prvk (0 - 100 v etn), minimální a maximální hodnotu ToT . Minimální je vždy 0, maximální hodnota je aktualizována p i na tení nových dat a je získána práv z dat. Hodnota ke vrácení je ur ena tak, že je spo ítán procentuální podíl p edané ToT na celku. Tento procentuální podíl se použije jako index do pole s barvami a barva uložená na pozici je vrácena.

5.7 DataIndexBar

V této ásti je popsána implementace grafického ukazatele pozice vykresleného okna v celkových datech. Nazývá se **DataIndexBar**, lze ho exportovat jako `Node` z komponenty a umístit do scény. Jeho ukázka se nachází na obrázku 5.10. Je složen ze dvou JavaFX 2D objekt `Rectangle`. První v ervené barv znázor uje celý soubor a druhý v modré barv je umíst n p es ervený a znázor uje pozici zobrazeného okna. Délka erveného obdélníku je volitelná. Levý okraj je index 0, pravý je poslední index. K aktualizaci dochází tak, že t ída udržuje referenci na `IntegerProperty` s hodnotou aktuální pozice zobrazeného okna, jeho hodnotu aktualizuje vizualizace. `ColorModel` na tuto referenci umístil `listener` a p i jeho aktualizaci je ur ena nová pozice modrého obdélníku. Pozice je ur ena tak, že je porovnána nová hodnota oproti maximu v datech s délkou erveného obdélníku. Procentuální podíl je použit jako finální pozice.

Obrázek 5.10: Ukazatel pozice aktuáln zobrazeného okna *DataIndexBar*

5.8 Předávání výsledků akcí v komponentě

Pro p edávání výsledk a stav o akcích komponenta využívá vlastní definované výjimky jazyka Java. Jsou použity primárn když nastane nestandardní situace, jako jsou nevalidní p edané vstupy, chyby p i vizualizaci, apod. Jsou navrženy tak, aby p edávali informace o nastalé akci samotnému programu a ten s nimi mohl následn pracovat a zárove , aby poskytovali informace užite né pro uživatele. Jsou rozd leny na dv základní skupiny.

První jsou výjimky používané základní sdílenou částí **BaseComponent**, nachází se v balíku BaseExceptions. Druhou skupinou jsou výjimky v části implementující konkrétní vizualizační metodu **VisualisationComponentxD**, nachází se v balíku VisualisationExceptions. Specializuje se na pokrytí akcí spojených s vizualizací.

Standardně jsou odchycovány v hlavních částech obou částí, **BaseComponentManager** a **VisualisationComponentxD**, kde je poskytnuta následná obsluha výjimky. V těsnou se jedná o zobrazení okna *Alert* s popisem, k jaké situaci došlo a případně, jak jí vyřešit.

5.9 Demonstrační aplikace

V této části je popsán popis implementace demonstrační aplikace, která využívá komponentu a předvádí její funkcionality. Pro každou komponentu je vyhotovena samostatná demonstrační aplikace, jsou ale skoro shodné. Liší se pouze v importu samotných komponent a v prvcích pro aktualizaci vizualizace.

Aplikace je tvořena jedním oknem, přímo vychází Stage přímo z primaryStage, rozložení prvků v okně je dán layout komponentou BorderPane. Znázornění aplikace v etapě zobrazení rozložení GUI prvků se nachází na obrázku 5.11. V horní části (TopPane) se nachází tlačítka pro import a export dat pro vizualizaci, v levém horním rohu se nachází menu s tlačítky pro ukončení aplikace a pro zobrazení okna „Nastavení komponenty“ (viz. 5.3). V prostrední části (MiddlePane) se nachází komponenta samotná. Pro obě varianty je reprezentována šedým tvercem, do kterého je vizualizace vykreslována. V levé části (LeftPane) se nachází prvky pro úpravu vizualizace, v těsné blízkosti je složena z Label uvedující konkrétní prvek, TextField pro zadání hodnoty a tlačítka pro nahraní hodnoty do komponenty. Jsou umístěny do mřížky GridPane. Komponenta tyto hodnoty validuje, pokud jsou v pořádku, tak je nahraje do vizualizace, pokud ne, zobrazí Alert okno s popisem chyby. Na pravé straně (RightPane) jsou TextArea pro výpis informací o komponentě. Ve spodní části (BottomPane) se nachází ukazatel aktuálně vizualizovaného okna v datech, DataIndexBar. Ten je exportován z komponenty a zasazen do scény, o veškerou jeho funkcionality se stará komponenta samotná.

TextArea pro zobrazení hodnot komponenty jsou vyplňeny pomocí StringProperty exportovaných z komponenty určených pro zaznamenávání

informací. Komponenta do nich zadává informace v předepsaném formátu, aplikace na ně umístila *listener* a při aktualizaci hodnot je přeloží do formátu příslušného pro uživatele a zobrazí do příslušných `TextArea`. Každé tlačítko v aplikaci po jeho stisknutí zavolá obslužnou metodu, která převeze parametry pro akci a zavolá příslušnou metodu v API komponenty. Parametry si metoda bude vytáhne z příslušných prvků (z `TextField` pro aktualizaci vizualizace), nebo jejich získání sama poskytne (např. při zadání na čítání ze souboru metoda pro uživatele otevře okno pro vybrání souboru (přes `FileChooser`).

Obrázek 5.11: Ukázka rozložení prvků v demonstrační aplikaci

6 Testování

V této kapitole je rozebráno testování komponent. Díl se na dvě hlavní části, programové testování a uživatelské testování. V rámci programového testování je otestováno, zda výpočetní operace komponenty fungují správně a očekávaně. V rámci uživatelského testování byly obecně demonstrativní aplikace poskytnuty zadavateli práce panu Ing. Broulímovi Ph.D. Primárním cílem bylo zjistit, zda jsou vizualizační metody funkční, splňují zadání a zobrazují hledaná data. Dále byly aplikace otestovány z hlediska uživatelské přívlastnosti a zároveň byly obecně zvolené vizualizační metody spolu porovnány.

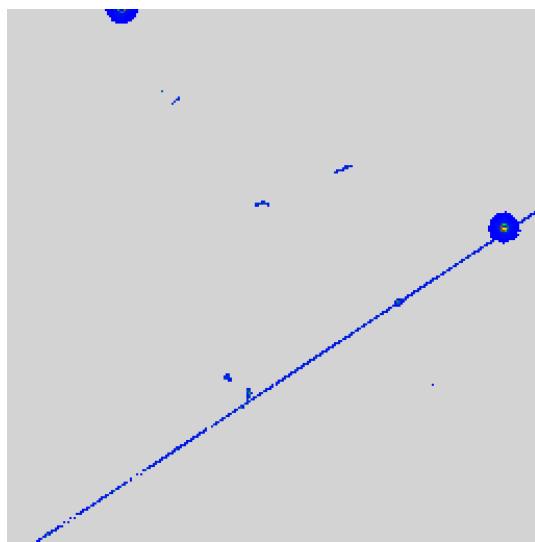
6.1 Programové testování

Programové testování je zaměřeno na testování výpočetních operací, pokryto je pomocí Java Unit testů. Výpočetní operace jsou na útahy ze souboru poskytované třídou DataModelManager a v některé funkce třídy Utils. Unit testy jsou obsaženy ve dvou třídách, DataModelManagerTest a UtilsTests. Chování jednotlivých funkcí bylo testováno tak, že jim byl předán vstup s očekávanou strukturou a následně bylo ověřeno, zda vrátily očekávaný výsledek. Testováno bylo i chování při předání nevalidních vstupů. Pro toto testování byla vygenerována vlastní testovací data. Nejdříve je otestováno, zda jsou data nařízena a uložena správně. Na to byl vygenerován soubor s daty se specifickými vlastnostmi. Obsahuje třítiny, eventy každého typu i jiný obrazec. Poté na tento jsou data zkонтrolována, zda jsou ve správných binárních správně uloženy patří obrazce. Následně je otestována i operace *grouping*, zda jsou obrazce správně sloučeny. Na toto byl vytvořen soubor se specifickým pořadem asových známek, aby bylo možné výsledek snadno ověřit. Následně byly ověřeny všechny chybné stavy při na útahy podchycované výjimkami, pro které z nichž testy byly vygenerovány testovací soubory se specifickou chybou.

6.2 Uživatelské testování

Pro uživatelské testování byly poskytnuty demonstrativní aplikace zadavateli práce panu Ing. Broulímovi Ph.D. Spolu s aplikací mu byla poskytnuta i uživatelská příručka, která je součástí přílohy k této bakalářské práci. Nejdříve bylo zhodnoceno, zda vizualizační metody zobrazují požadovaná

data. Testování bylo provedeno na n kolika datových sadách, nap íklad z ob servato e nacházející se 2 300 m nad mo em nebo z atomového krytu. Vý sledky jsou pro ob vizualiza ní metody pozitivní, v obou se poda ilo najít požadované jevy. Hlavním cílem bylo najít v datech tzv. miony, jedná se o kosmické zá ení a projevují se jako dlouhé rovné áry. Nalezeny byly ve všech poskytnutých datových sadách. P íklad se nachází na obrázcích. Zají-mavé bylo i pozorování, že se v datové sad z observato e miony vyskytují ast ji než v sad z atomového krytu. Toto pozorování i potvrzuje správnost vizualizace, protože taková hustota výskytu t chto jev odpovídá umíst ní detektor .



Obrázek 6.1: Zobrazení mionu ve 2D komponent .

Dále se zadavatel zam il na otestování uživatelské p ítvosti aplikací. Vyhodnotil, že ovládání p es klávesnicí je intuitivní a je s ním spokojen. Ovlá-dání vizualizaci myší je také v po ádku, ale m l požadavek, zda by rychlost otá ení modelu v 3D vizualizaci mohla být nastavitelná. Do komponenty byla následn úprava rychlosti p idány, v aplikaci se upravuje pomocí p i-daného šoupátka. Kriti t ji se vyjád il k intuitivnosti n kterých vizuálních prvk aplikace, protože nebylo podle p vodního pojmenování úpln jasné, jaké parametry p esn upravují a p ípadn , k emu p esn parametry slouží. Podle jeho poznatk byly tyto prvky ješt upraveny.

K 3D vizualiza ní metod samotné poznamenal, zda by šlo upravit na svícení scény. P i prohlížení voxel pod n kterými úhly jsou n které voxely stínované a vizuáln by pro n j bylo p íjemn jší, aby stínování bylo vypnuto.

Obrázek 6.2: Zobrazení mionu ve 3D komponentě.

Nasvícení tedy bylo ještě dodatečně upraveno nastavením zdroje světla, aby posobil ze všech stran, čímž bylo odstraněno stínování voxelů.

Při testování ještě zachytily chybu 3D komponenty, kdy se při znovu načtení nějaké datové sady při prvním kliknutí na voxel vyskočilo okno s popisem vnitřní odchycené výjimky, která by ale správně při běžném fungování neměla nastávat. Tuto chybu se podařilo nasimulovat a byla následně opravena.

Na závěr ještě zhodnotil užitečnost obou vizualizačních metod. 3D vizualizaci vyhodnotil jako povedenou, splnila zadání a daří se v ní nacházet hledané obrazce. Oproti 2D vizualizaci má velkou výhodu v přidání třetí dimenze, kdy lze vizuálně prohlédnout do dalších řasových známek a porovnat vývoj. V 2D vizualizaci ohodnotil jako funkční, ale není tak užitečná jako 3D. Zkoumané jevy pomocí ní hledat také, ale ne tak efektivně a navíc má velkou nevýhodu v tom, že není na první pohled poznat řasové známky jednotlivých eventů. Celkově 3D vizualizaci vyhodnotil jako povedenou a lze s ní nadále pracovat a případně rozširovat. 2D vizualizaci vyhodnotil tak, že oproti 3D není příliš užitečná, není a díky možnosti používat 3D metodu není potřeba 2D používat. Maximálně by mohlo použít jako doplněk k 3D komponentě, kdyby ve 2D vizualizaci mohlo například exportovat vybraná množina dat do 2D obrázku.

7 Závěr

Cílem této práce bylo vytvořit prototyp komponenty pro vizualizaci dat z ľásticových detektorů. V rámci práce byly zvoleny dvě zobrazovací metody vhodné pro zobrazování dat z ľásticových detektorů v řase a pro každou metodu byla vyhotovena samostatná komponenta. Cíl práce se podařilo splnit a zadavatel je s vytvořenými metodami vizualizace spokojen.

V první části práce jsem se věnoval režerzi ľásticových detektorů společně s jejich formáty výstupních dat. Cílem této části bylo seznámit se s obecným fungováním detektorů a primárně porozumět formátům, v jakých jsou udávány, aby následně mohla být zvolena správná metoda jejich zpracování a uložení.

V druhé části práce jsem se věnoval analýze metod na zobrazování týkajících dat. Nejdříve byly zanalyzovány současné metody vizualizace, následně byly zanalyzovány možnosti vizualizace objemových dat a na základě týkající analýzy byly zvoleny dvě vhodné zobrazovací metody k implementování. Jako první byla zvolena 3D metoda, kdy osa Z poslouží jako časová osa a jednotlivá data jsou vynesena v řase. Jako druhá byla zvolena 2D metoda, kdy jsou data z několika časových známků sloužena do jedné.

Poté jsem se věnoval návrhu a implementaci samotných komponent. K implementaci byla použita technologie JavaFX. Při implementaci jsem narazil na dvě hlavní potíže. První bylo zvolení vhodného způsobu uložení dat tak, aby operace datového modelu byly dobře použitelné pro vizualizace. Druhou bylo zvolení vhodného algoritmu pro přidávání souřadnic u 3D vizualizace, aby byl použitelný nezávisle na zvolené datové sadě.

V poslední části této práce se věnuji testování obou komponent. Byly otestovány nejprve jednotkovým testováním a následně bylo provedeno uživatelské testování se zadavatelem práce. Hlavním výsledkem testování je, že obě komponenty jsou funkční a zobrazují požadované jevy. 3D metoda byla vyhodnocena jako užitečná, efektivní a do budoucna použitelná.

Práci bylo možné rozšířit přidáním možnosti analyzovat několik datových sad najednou, zaznamenaných ve stejném časovém rozmezí na různých detektorech. To by bylo možné použít ke zkoumání jevů, které byly zaznamenány ve stejný čas na více detektorech. Nejjednodušší způsob jak toho dosáhnout bylo zobrazení několika komponent najednou, kdy by každá zobrazovala jinou datovou sadu. Všechny vizualizace bylo procházet najed-

nou společným ovládáním, data by tak byla zobrazována ze stejných řas..

Literatura

- [1] What is an API? [online]. IBM, 2023. [cit. 11. b°ezna 2023]. Dostupné z: <https://www.ibm.com/topics/api>.
- [2] AWT Official Documentation [online]. 2020. [cit. 5. února 2023]. Dostupné z: <https://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html>.
- [3] Ballabriga , R. Campbell , M. Llopart , X. An introduction to the Medipix family ASICs. *Radiation Measurements* 2020, 136, s. 106271. ISSN 1350-4487. doi: <https://doi.org/10.1016/j.radmeas.2020.106271>. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S1350448720300354>.
- [4] blender [online]. 2023. [cit. 16. dubna 2023]. Dostupné z: <https://www.blender.org/>.
- [5] Camera modes and view terminology[online]. 2017. [cit. 23. b°ezna 2023]. Dostupné z: <https://abaqus-docs.mit.edu/2017/English/SIMACAECAERefMap/simacae-c-viwcamera.htm>
- [6] Campbell , M. et al. Charge collection from proton and alpha particle tracks in silicon pixel detector devices. In 2007 IEEE Nuclear Science Symposium Conference Record 2, s. 1047 1050, 2007. doi: 10.1109/NSSMIC.2007.4437190.
- [7] Working with Canvas [online]. 2013. [cit. 20. 2. 2023]. Dostupné z: <https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm>.
- [8] Timepix3 [online]. CERN, 2023. [cit. 20. dubna 2023]. Dostupné z: <https://kt.cern/technologies/timepix3>.
- [9] Medipix and Timepix projects [online]. CERN, 2023. [cit. 25. dubna 2023]. Dostupné z: <https://medipix.web.cern.ch/medipix-and-timepix-projects>.
- [10] draw.io [online]. 2023. [cit. 16. dubna 2023]. Dostupné z: <https://drawio-app.com/>.
- [11] Furnell , W. et al. First results from the LUCID-Timepix spacecraft payload onboard the TechDemoSat-1 satellite in Low Earth Orbit. *Advances in Space Research* 2019, 63, 5, s. 1523 1540. ISSN 0273-1177. doi: <https://doi.org/10.1016/j.asr.2018.10.045>. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0273117718308330>.

- [12] Granja , C. Pospisil , S. Quantum dosimetry and online visualization of X-ray and charged particle radiation in commercial aircraft at operational flight altitudes with the pixel detector Timepix. Advances in Space Research 2014, 54, 2, s. 241 251. ISSN 0273-1177. doi: <https://doi.org/10.1016/j.asr.2014.04.006>. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0273117714002208>
- [13] Granja , C. et al. Energy loss and online directional track visualization of fast electrons with the pixel detector Timepix. Radiation Measurements 2013, 59, s. 245 261. ISSN 1350-4487. doi: <https://doi.org/10.1016/j.radmeas.2013.07.006>. Dostupné z:<https://www.sciencedirect.com/science/article/pii/S1350448713002874>
- [14] Heijne , E. H. History and future of radiation imaging with single quantum processing pixel detectors.Radiation Measurements 2021, 140, s. 106436. ISSN 1350-4487. doi: <https://doi.org/10.1016/j.radmeas.2020.106436>. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S1350448720302146>
- [15] Heijne , E. H. et al. Measuring radiation environment in LHC or anywhere else, on your computer screen with Medipix.Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment2013, 699, s. 198 204. ISSN 0168-9002. doi: <https://doi.org/10.1016/j.nima.2012.05.023>. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0168900212005104> . Proceedings of the 8th International Hiroshima Symposium on the Development and Application of Semiconductor Tracking Detectors.
- [16] Jan , M. Pixels and voxels, the long answer[online]. 2016. [cit. 8. ledna 2023]. Dostupné z:<https://medium.com/retronator-magazine/pixels-and-voxels-the-long-answer-5889ecc18190>
- [17] ArrayList vs LinkedList in Java [online]. 2022. [cit. 14. února 2023]. Dostupné z: <https://www.geeksforgeeks.org/arraylist-vs-linkedlist-java/>
- [18] Best Java GUI Frameworks[online]. 2022. [cit. 12. února 2023]. Dostupné z: <https://csveda.com/best-java-gui-frameworks/>
- [19] Java Map Interface [online]. 2021. [cit. 12. února 2023]. Dostupné z: <https://www.javatpoint.com/java-map>
- [20] Primitive Data Types [online]. 2022. [cit. 28. ledna 2023]. Dostupné z: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

- [21] JavaFX Official Site [online]. 2022. [cit. 5. února 2023]. Dostupné z: <https://openjfx.io/>
- [22] 3 Camera [online]. Oracle, 2023. [cit. 23. března 2023]. Dostupné z: <https://docs.oracle.com/javase/8/javafx/graphics-tutorial/camera.htm#CJAHFAHB>
- [23] JavaFX Overview [online]. Jenkov, 2023. [cit. 18. března 2023]. Dostupné z: <https://jenkov.com/tutorials/javafx/overview.html>
- [24] JavaFX - Application [online]. Tutorials Point, 2023. [cit. 18. března 2023]. Dostupné z: https://www.tutorialspoint.com/javafx/javafx_application.htm
- [25] JavaFX: Working with JavaFX Graphics [online]. 2014. [cit. 3. února 2023]. Dostupné z: <https://docs.oracle.com/javase/8/javafx/graphics-tutorial/javafx-3d-graphics.htm>
- [26] Maken , P. Gupta , A. 2D-to-3D: A Review for Computational 3D Image Reconstruction from X-ray Images [online]. Springer, 2023. [cit. 15. dubna 2023]. Dostupné z: <https://link.springer.com/article/10.1007/s11831-022-09790-z>
- [27] <https://medipix.web.cern.ch/our-story> [online]. 2022. [cit. 10. 12. 2022]. Dostupné z: <https://medipix.web.cern.ch/our-story>
- [28] Úvod do techniky CCD čipů [online]. 2011. [cit. 10. listopadu. 2022]. Dostupné z: <https://www.gxccd.com/art?id=303&lang=405>
- [29] Proke² , R. Trojek , T. Musílek , L. Determination of X-ray tubes radiation beam characteristics with semiconductor pixel detectors. Radiation Physics and Chemistry 2020, 172, s. 108771. ISSN 0969-806X. doi: <https://doi.org/10.1016/j.radphyschem.2020.108771>. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0969806X19308679>
- [30] X-RAY COMPUTED TOMOGRAPHY: A BRIEF INTRODUCTION [online]. RIGAKU, 2021. [cit. 15. dubna 2023]. Dostupné z: <https://imaging.rigaku.com/learning/x-ray-computed-tomography-brief-introduction>
- [31] WHAT IS MICRO CT? [online]. RIGAKU, 2021. [cit. 15. dubna 2023]. Dostupné z: <https://imaging.rigaku.com/learning/micro-ct>
- [32] Rosenfeld , A. et al. Medipix detectors in radiation therapy for advanced quality-assurance. Radiation Measurements 2020, 130, s. 106211. ISSN

1350-4487. doi: <https://doi.org/10.1016/j.radmeas.2019.106211>.

Dostupné z: <https://www.sciencedirect.com/science/article/pii/S1350448719304974>

- [33] X-RAY COMPUTED TOMOGRAPHY: A BRIEF INTRODUCTION [online]. SliderPlayer, 2023. [cit. 15. dubna 2023]. Dostupné z: <https://slideplayer.com/slide/8978642/>
- [34] Sommer, M. et al. High-energy per-pixel calibration of timepix pixel detector with laboratory alpha source. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 2022, 1022, s. 165957. ISSN 0168-9002. doi: <https://doi.org/10.1016/j.nima.2021.165957>. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0168900221009049>
- [35] Swing Official Documentation [online]. 2022. [cit. 5. února 2023]. Dostupné z: <https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html>
- [36] Displaying graphics in swing[online]. 2022. [cit. 20. 2. 2023]. Dostupné z: <https://www.javatpoint.com/Graphics-in-swing>
- [37] UML Official Site [online]. 2022. [cit. 10. března 2023]. Dostupné z: <https://www.uml.org/>

Seznam obrázků

2.1	Znázornění pixelového detektoru [27]	10
2.2	Příklad jednoduché vizualizace dat [12]	11
2.3	Příklad vizualizace dat se znázorněnou energií [13]	12
2.4	Příklad vizualizace dat s popisem zaznamenaných částic [12]	12
2.5	Tabulka popisující jednotlivé obrazce vyskytující se v datech a popis, jaké částice či jevy představují [15]	13
3.1	Příklad jednoduché 2D vizualizace. [15]	17
3.2	Příklad vizualizace řezem. [29]	17
3.3	Příklad 3D vizualizace, na osách X a Y jsou pozice jednotlivých pixelů a osa Z reprezentuje energii pro jednotlivé pixely. [34]	18
3.4	Příklad 3D vizualizace, na osách X a Y jsou pozice jednotlivých pixelů a na ose Z je vynesena energie pro jednotlivé pixely. [29]	18
3.5	Graf znázorňující závislost velikosti clusteru na energii zaznamenané částice. [6]	19
3.6	Graf znázorňující závislost velikosti clusteru na obsah clusteru (suma energií všech pixelů pro daný cluster). [32]	20
3.7	Graf ukazující typy clusterů v binech pro konkrétní časové známky. Typy clusterů jsou odděleny barevně podle legenda. [15]	21
3.8	2D vizualizace, kdy jsou do jednoho obrázku sloučena data z několika časových známek. U pixelů se stejnými souřadnicemi je jejich hodnota seftena. Pixely z pouze jedné časové známky jsou svítě modré, z několika řluté až žlavené. [14] .	21
3.9	Proces vytvoření 3D obrazce pomocí použití technologie CT (computed tomography). [26]	24
3.10	Znázornění konstrukce 3D obrazce ze zdrojových dat. a) mnoha naměřených dat; b) řez naměřenými daty, 2D snímek; c) konstrukce 3D obrazce ze zdrojových dat [33]	24
3.11	Znázornění jednotlivých kroků vytvoření 3D vizualizace z mnoha CT snímků. Za pomocí vizualizace je provedena analýza složení zkoumaného prvku. 3D objekty jsou vykreslovány pomocí voxelů. [31]	25
3.12	Znázornění rozdílu mezi vektorovou a rastrovou grafikou ve 2D a 3D. [16]	25

4.1	Pøíklad kreslení 2D objekt· pomocí frameworku Swing [36]	29
4.2	Pøíklad kreslení pomocí t°ídyCanvasJavaFX [7]	30
4.3	Pøíklad jednoduché JavaFX 3D aplikace [25]	30
4.4	Znázorn¥ní hierarchie komponent JavaFX [23]	32
4.5	Znázorn¥ní hierarchie komponent JavaFX [24]	32
4.6	Navr°ená architektura komponenty. Obrázek byl vytvo°en v pro- gramu draw.io. [10]	36
4.7	Ulo°ení dat pro vizualizaci v datovém modelu. Obrázek byl vytvo°en v programu draw.io. [10]	41
4.8	Navr°ená architektura datového modelu. Obrázek byl vytvo- °en v programu draw.io. [10]	42
4.9	Navr°ené propojení datového modelu a vizualizace. Obrázek byl vytvo°en v programu draw.io. [10]	45
4.10	Navr°ené 2D vizualizace. Obrázek byl vytvo°en v programu draw.io. [10]	46
4.11	Navr°ená architektura 2D vizualizace. Obrázek byl vytvo°en v programu draw.io. [10]	47
4.12	Navr°ené 3D vizualizace. Návrh byl vytvo°en v programu blender. [4]	49
4.13	Diagram t°íd 3D vizualizace. Obrázek byl vytvo°en v pro- gramu draw.io. [10]	50
4.14	Ukázka parametr· kamery Field of View, Near Clip a Far Clip. [5]	52
5.1	UML diagram t°ídy BaseComponentManager	55
5.2	UML diagram t°ídy DataModelManager	57
5.3	UML diagram t°ídy EventWithTimeStamp	58
5.4	UML diagram t°ídy EventWithoutTimeStamp	58
5.5	UML diagram t°ídy BinData	59
5.6	UML diagram t°íd le parseru	60
5.7	Okno s nastavením parametr· komponenty	64
5.8	Znázorn¥ní 2D vizualizace	65
5.9	Znázorn¥ní 3D vizualizace	67
5.10	Ukazetel pozice aktuáln¥ zobrazeného okna DataIndexBar	69
5.11	Ukázka rozlo°ení prvk· v demonstra£ní aplikaci	71
6.1	Zobrazení mionu ve 2D komponent¥.	73
6.2	Zobrazení mionu ve 3D komponent¥.	74
I.1	Ukázka vzhledu DataIndexBar	89
I.2	Vzhled demonstra£ní aplikace.	94

I.3	Rozložení prvků v demonstrační aplikaci	95
I.4	2D vizualizace	97
I.5	3D vizualizace	99
I.6	Okno s nastavením komponenty	101

Seznam tabulek

4.1	Celo£íselné datové typy v Java, jejich velikost a minimální a maximální hodnoty, které do nich lze ulo®it.[20]	40
4.2	P°ehled mo®ností nastavení rozli®ení dat pro £ásticové detek- tory typu Medipix.[3]	40

Seznam ukázek kódu

1	Formát data driven	15
2	Vytvoření Hello World aplikace v technologii JavaFX	34
3	Struktur pro uložení dat určených k vizualizaci	41
4	Znázornění nastavení kamery v JavaFX 3D	52
5	Znázornění algoritmu operace grouping	61
6	Uložení parametr komponenty	63
7	Znázornění sestavení jednoho voxelu	68
I.1.1	Konstruktory obou komponent	88

I P°íloha A - U°ivatelská p°íru£ka

U°ivatelská p°íru£ka popisuje pou°ívání vizualiza£ní komponenty a demonstra£ní aplikace. U°ivatelská p°íru£ka je rozd¥lena na dv¥ hlavní £ásti: Vizualiza£ní komponenta - zam¥ení na p°idání vizualiza£ní komponenty do projektu z programátorského hlediska, Demonstra£ní aplikace - popis jak spustit a ovládat aplikaci. Komponenta a aplikace jsou vytvo°eny v technologii Java, GUI a grafické komponenty ve frameworku JavaFX. Vyvinuty byly ve verzích Java a JavaFX 8, pou°itelné jsou ve v²ech verzích od 8 vý²e. K datu vyhotovení byla nejaktuáln¥jí verze Java i JavaFX 19. P°i pou°ití verze Java 11 a vy²ích je nutné framework JavaFX p°ipojit ke komponent¥£i aplikaci extern¥.

Komponenta a aplikace se nachází v adresá°i Aplikace_a_knihovny, který obsahuje celkem 2est adresá°.

- ^ base_component - Základní £ást komponenty poskytující operace s daty. Pou°ijte p°i implementaci vlastní vizualiza£ní metody.
- ^ component_2D komponenta s 2D vizualiza£ní metodou.
- ^ component_3D komponenta s 3D vizualiza£ní metodou.
- ^ application_2D aplikace vyu°ívající 2D komponentu
- ^ application_3D aplikace vyu°ívající 3D komponentu
- ^ tests jednotkové testy pou°ity k testování výpo£etních operací komponenty.

I.1 Vizualiza£ní komponenta

V této £ásti se nachází popis práce s vizualiza£ními komponentami, jedná o balí£ky component_2D a component_3D.

I.1.1 P°idání komponenty do projektu

Postup p°idání komponenty do projektu je stejný pro ob¥ varianty komponenty, 2D a 3D, rozdíl je aº v pou°ívání jejich funkcionalit. Komponenta

je k dispozici ve formě nespustitelného archivu .jar pojmenovaného component_2D.jar a component_3.jar. Pro použití přidejte archiv do vašeho JavaFX projektu projektu. Možností přidání do projektu je například Eclipse nebo IntelliJ IDEA to bývá umožněno nebo lze použít nástroje pro sestavování projektů jako Maven nebo Gradle. Při vývoji komponent a demonstračních aplikací byl použit Maven.

I.1.2 Použití komponenty v projektu

Po přidání do projektu má programátor přístup ke dvěm třídám komponenty, ale využívá pouze jednu. Jedná se o hlavní třídu komponenty, VisualisationComponent2D pro 2D komponentu a VisualisationComponent3D pro 3D komponentu. Tato třída dědí od layout komponenty JavaFXGroup. Přidejte tedy instanci této třídy do scény jako bývoucí JavaFX komponentu. K dispozici jsou dva konstruktory. První s přednastavenými hodnotami a druhý s možností, aby si programátor všechny parametry nastavil sám. Znázorněny jsou na ukázce I.1.1 a dále je pak popsán význam jejich parametrů. Následuje popis výjivých metod obou komponent, které lze využít k jejich ovládání. Pokud se u některých nepovede akce dokončit (např. kvůli předání nevalidní hodnoty), zobrazí komponenta okno Alert s popisem chyby.

```

VisualisationComponent3D( boolean calculateZAxis,
    boolean initializeDataIndexBar,      boolean turnOnLogging);
VisualisationComponent3D( int initSubSceneHeight,
    int initSubSceneWidth,
    int initTimeWindow, String initVisualisationMode3DCode,
    boolean calculateZAxis,   boolean initializeDataIndexBar,
    boolean turnOnLogging);

VisualisationComponent2D( boolean calculateZAxis,
    boolean initializeFileIndexBar,      boolean turnOnLogging);
VisualisationComponent2D( boolean calculateZAxis,
    boolean initializeFileIndexBar,      boolean turnOnLogging,
    int initImageViewHeight,   int initImageViewWidth,
    int initTimeWindow);

```

Ukázka kódu I.1.1: Konstruktory obou komponent

Parametry konstruktoru:

- ^ **boolean calculateZAxis** Zda se mají v datovém modelu pořítat souřadnice bin. pro osu Z. Tento parametr je závislý na použití vizualizační metod. Pro 2D komponentu je potřeba zadat false (komponenta by validně fungovala i s hodnotou true, ale vypořítané souřadnice by zbytečně zabírali paměť), pro 3D komponentu true.
- ^ **boolean initializeDataIndexBar** Zda se má vytvořit instance DataIndexBar. Jedná o ukazovátko pozice aktuálně zobrazeného okna v datech, které lze z komponenty získat a umístit do scény. Ukázka se nachází na obrázku I.1.
- ^ **boolean turnOnLogging** Zda má být zapnuto během běhu aplikace logování. Komponenta využívá vlastní logovací formát: datum a čas vypsání, soubor a metoda, ze které bylo logování voláno a samotná zpráva. Toto logování lze zapnout i vypnout.
- ^ **int initSubSceneHeight** pořáteční výška SubScene, do které bude 3D vizualizace vykreslena.
- ^ **int initSubSceneWidth** pořáteční šířka SubScene, do které bude 3D vizualizace vykreslena.
- ^ **int initTimeWindow** pořáteční velikost časového okna, tedy počet, kolik bin. má být najednou vykresleno.

- ^ `int initImageViewHeight` pořáteční výška okna, do které bude 2D vizualizace vykreslena.
- ^ `int initImageViewWidth` pořáteční šířka okna, do které bude 2D vizualizace vykreslena.
- ^ `String initVisualisationMode3DCode` Jaký 3D vizualizační mód má být použit. Na výběr jsou dva módy, Static a Movable blížeji popisníe v části 3D vizualizace. Jako parametr metody zadejte "etzcový kód určující, jaký mód má být použit. Kód STATIC pro mód Static, kód MOVABLE pro kód Movable Pokud je půedaný kód nevalidní, automaticky je nastaven módStatic.

Obrázek I.1: Ukázka vzhledu DataIndexBar

Seznam metod, které poskytuje objekt komponenty:

1. `void loadNewDataFromFileAction(File inputFile)` - Metoda pro načítání dat ze souboru. Parametr inputFile je vstupní soubor, ze kterého budou data načteny. Soubor musí být textový a data musí být uložena v data driven formátu. Tuto funkci typicky volá tlačítko pro započatí načítání.
2. `void stopLoadingDataFromFileAction()` - Metoda pro předčasné zastavení načítání ze souboru. Načítání může trvat jednotky desítek vteřin a uživatel ho může chtít předčasně zastavit, například z důvodu zpatně zvoleného vstupního souboru. Po zavolení je načítání neprodleně ukončeno.
3. `void setKeyEventHandler(KeyEvent keyEvent)` - Metoda pro zpracování události z klávesnice. Do scény aplikace umístěte odchytávání událostí z klávesnice KeyEvent a předávejte je této metodi, která zajistí, že komponenta událost správně zpracuje.
4. `void showDataAtIndexAction(int newIndex)` - Zobrazení dat položující binem s půedaným indexem. Předchozí zobrazené okno bude smazáno a budou vykreslena nová data na půedaném indexu. Hodnota indexu musí být v rozsahu nula až počet_bin_v_datech).

5. `void showDataAtTimestampAction(long timeStamp)` - Zobrazení dat pořínající binem s pořadanou řasovou známkou (ToA). Pokud se v datech nenachází bin s pořadanou ToA, je použit bin s nejbližší hodnotou ToA k pořadané. Pořadcozí zobrazené okno bude smazáno a budou vykreslena nová. Pořadaná řasová známka může mít jakoukoliv hodnotu, vždy bude v datech nalezena nejbližší.
6. `void updateTimeWindowAction(int newTimeWindow)` Nastavení nové hodnoty parametru TimeWindow, tedy pořtu, kolik binů má být na jednou vykresleno. Nová hodnota musí být kladné číslo.
7. `void updateImagesToMoveCountAction(int newImagesToMoveCount)`
 - Nastavení nové hodnoty udávající, o kolik binů se má okno posunout pořadí jednom zavolání akce posunu. Tedy, kolik binů má být pořadí posunu pořidáno na konec dat a kolik binů má být na začátku odebráno. Nová hodnota musí být kladná.
8. `void updateMaxPixelColorValueAction(int newMaxPixelColorValue)`
 - Nastavení nové hodnoty maximální hodnoty ToT v datech. Hodnota je využita pro vypořítání barev v barevném modelu. Pořadí snížování hodnoty dojde ke k pořazení výrazných barev jednotlivým krychlím čtverců. Nová hodnota musí být kladné číslo.
9. `IntegerProperty oneBinStepIntegerProperty()` - Getter pro získání IntegerProperty s aktuální hodnotou parametru krok jednoho binu.
10. `StringProperty timeMeasurementUnitInfoStringProperty()`
 - Getter pro získání StringProperty s aktuální hodnotou nastavené jednotky řasu pro aktuální data.
11. `StringProperty dataModelInfoStringProperty()` - Getter pro získání StringProperty obsahující informace o parametr datového modelu a aktuálně načtených datech.
12. `Node getDataIndexBarGroup()` - Getter pro získání Group obsahující DataIndexBar.
13. `IntegerProperty dataIndexIntegerProperty()` - Getter pro získání IntegerProperty obsahující informace o aktuálním pořazením indexu, od kterého jsou vykresleny data.

Seznam metod, které poskytuje pouze 2D komponenta:

1. StringProperty `chosenSquareInfoStringProperty()` - Getter pro získání StringProperty obsahující informace o aktuálně zvoleném čtverci.
2. StringProperty `visualisation2DInfoStringProperty()`
- Getter pro získání StringProperty obsahující informace o aktuálním nastavení 2D vizualizace.

Seznam metod, které poskytuje pouze 3D komponenta:

1. `void updateZAxisScaleAction(int scaleValuePercentage)` - Nastavení nové hodnoty mýtíka osy Z. Hodnota musí být zadána v percentech v intervalu (0 - 100>. Defaultně je hodnota nastavena na 100%.
2. `void updateMaxSpaceBetweenImagesAction(int newSpaceBetweenImages)`
- Nastavení nové hodnoty maximální meze mezi dvěma snímky. Tato hodnota se používá pro výpočet souřadnic osy Z. Vzdálenost mezi dvěma body nesmí být výšší než tato hodnota. Hodnota musí být kladná.
3. `void updateSpeedForwardAction(int newSpeed)`- Nastavení nové hodnoty rychlosti jakou se kamera pohybuje ve směru dopředu / dozadu. Hodnota musí být kladná.
4. `void update3DVisualisationModeAction(String mode)` - Nastavení 3D vizualizačního módu pomocí kódu v řetězcové podobě. Kód "STATIC" pro mód Static, kód "MOVABLE" pro kód Movable. Změna se projeví až po zobrazení nových dat.
5. `void updateSubSceneSize(int subSceneWidth, int subSceneHeight)`
- Aktualizace řídky a výšky SubScene. Změna se projeví okamžitě. Hodnoty musí být kladné.
6. `void moveCameraWithDataAction(boolean moveCamera)` Nastavení hodnoty, zda se má požítí akce showDataAtTimestampAction(long timeStamp) nebo showDataAtTimestampAction(long timeStamp) na pozici nově vykreslených dat poesunout i kamera. Použitelné pouze pro mód Movable, požaduje mód Static kamera zůstává na místě.
7. StringProperty `chosenBoxInfoStringProperty()` - Getter pro získání StringProperty obsahující informace o aktuálně zvolené krychli.

8. StringProperty [visualisation3DInfoStringPropertyProperty\(\)](#)
 - Getter pro získání StringProperty obsahující informace o aktuálním nastavení 3D vizualizace.

I.2 Demonstrační aplikace

I.2.1 Příklad a sestavení

Aplikaci lze přeložit a sestavit na operačním systému Windows, který má nainstalovaný JDK minimálně verze 8 a Maven minimálně verze 3. Maven je použit k sestavení aplikace. K příkladu se přesuňte do kořenového adresáře příslušné aplikace, application_2D nebo application_3D. V adresáři se nachází dvě položky, složka `src` se zdrojovými soubory aplikace a komponenty a XML soubor `pom.xml`, což je hlavní soubor programu Maven, který de nuje scénáře pro sestavení aplikace. Ve výchozím stavu obsahuje informace potřebné pro aplikaci a pro její export. Pokud používáte Java 8, 9, 10, můžete rovnou sestavit aplikaci. Pokud používáte Java 11 a vyžíbáte, ve kterých již není zakomponována JavaFX defaultně v Java, můžete ji přidat jako dependency do `pom.xml`. Pro příklad otevřete v kořenovém adresáři příkazovou řádku/terminál a zadejte příkaz `mvn clean install`, čímž spustíte příklad a sestavení. Po jeho dokončení se vygeneruje složka `/target`, ve které se nachází spustitelný `.jar` soubor s aplikací pojmenovaný `Particle_Detector_Visualization_App-jar-with-dependencies.jar`.

I.2.2 Spuštění

Aplikace se spustí spustitelným `.jar` souborem `Particle_Detector_Visualization_App-jar-with-dependencies.jar` nacházejícím se po vygenerování v adresáři `target`. Spustí se dvojklikem myší na konkrétní archiv (nemusí fungovat vše, může být potřeba ještě povolit v operačním systému) nebo z příkazové řádky (funguje vše). Aplikace ke spuštění nevyžaduje žádné další argumenty. Příkaz spuštění vypadá následovně:

```
...>java -jar Particle_Detector_Visualization_App-jar-with-dependencies.jar
```

I.2.3 Vzhled aplikace a popis jednotlivých prvků

Po spuštění se otevře okno aplikace. Rozložení scény obou aplikací je velice podobné, liží se pouze v prvcích pro úpravu vizualizace a v komponentech umís-

téhle uprostřed. Podobu okna můžete vidět na obrázku I.2 a na obrázku I.3 jeho rozložení vrstek, spodek, prostředek a levou a pravou část. V horní části se nachází tlačítka sloužící k importu a exportu dat, jejich podrobný popis se nachází níže. V levém horním rohu lze pod položkou File zobrazit menu se dvěma položkami Exit a Component settings popis taktéž níže. Ve střední části se nachází samotná komponenta. Jedná se o samostatnou část, ve které jsou vykreslena zobrazovaná data. Ve spodní části se nachází ukažatel aktuální pozice zobrazovaných dat v celku. Na levé části se nachází ovládací prvky pro úpravu vzhledu nebo chování vizualizace. Výdy je uveden popis prvku, text eld k zadání nové hodnoty a tlačítko, po jejímž stisknutí je hodnota z text eld provedena na číslo a provedána komponentě. Pokud je hodnota nevalidní, komponenta zobrazí okno Alert s popisem. U některého prvku je úprava pomocí checkboxu, změna se projeví ihned po zakliknutí. Význam jednotlivých prvků je popsán níže u popisů vizualizací. V pravé části okna se nachází textové oblasti sloužící pro výpis informací o načtených datech. V první se nachází aktuální hodnota kroku jednoho binu, ve druhé aktuální početní index vizualizovaných dat, ve třetí statistické informace o aktuálně načtených datech, ve čtvrté informace o aktuální vizualizaci a v páté informace o aktuálně vybrané krychli/čtverci.

Popis tlačítek v horní části aplikace:

1. Load Data Driven File slouží ke spuštění načítání dat ve formátu data driven ze souboru. Po stisknutí se otevře souborové okno operačního systému, vyberte textový soubor se vstupními daty. Po vybrání bude soubor provedán komponentě, ta provede jeho validaci a pokud je v pořádku, spustí načítání. To bývá trvá jednotky vteřin, u velkých souborů i desítky. Po úspěšném dokončení načítání se zobrazí okno informující o úspěšném nahrání dat do komponenty, po kliknutí na tlačítko OK se okno zavře a do komponenty budou vykreslena počítání data. Pokud býhem k načítání došlo k nějaké chybě, je ukončeno a zobrazí se okno s popisem chyby.
2. Stop Loading Data Driven File slouží k provedení ukončení načítání dat ze souboru. Po kliknutí bude načítání provedeno a zobrazí se okno informující o úspěšném provedení.
3. Export Data To File slouží k exportu aktuálně zobrazeného okna do textového souboru. Po kliknutí se otevře souborové okno operačního systému, vyberte adresář, do kterého má být export proveden. Po vybrání bude adresář provedán komponentě, která vezme aktuálně načtená data a exportuje je do nově vytvořeného textového souboru,

jeho^o jméno je slo^{enino} ze jména datové sady a pořáteřního a koncového indexu exportovaného okna. Po skončení se zobrazí okno s informací, ^{že} export byl dokončen.

4. Reload Data znovu se načte poslední načtený soubor.
5. File - Exit slouží k ukončení aplikace. Po stisknutí je zavolána funkce v komponentě pro ukončení operací komponenty a následně je celé okno uzavřeno.
6. File - Component Settings slouží k zobrazení okna pro úpravu základních hodnot komponenty, podrobný popis okna níže. Lze zobrazit i stisknutím klávesy I.

Obrázek I.2: Vzhled demonstrační aplikace..

I.3 Ovládání

V této části je popsáno ovládání komponenty. Komponenta se ovládá buď pomocí GUI prvků ve scéně (jejich význam popsán výše) nebo hlavní klávesnicí a myší, které slouží primárně pro ovládání vizualizací. Následuje popis ovládání pomocí klávesnice a myši.

Seznam kláves:

1. 'ipka doleva - rotace kamery doleva, pouze 3D komponenta

Obrázek I.3: Rozložení prvk. v demonstrační aplikaci.

2. 'ipka doprava - rotace kamery doprava, pouze 3D komponenta
3. 'ipka nahoru - rotace kamery nahoru, pouze 3D komponenta
4. 'ipka dol- - rotace kamery dol-, pouze 3D komponenta
5. W - posun kamery dop°edu, pouze 3D komponenta
6. S - posun kamery dozadu, pouze 3D komponenta
7. A - posun kamery doleva, pouze 3D komponenta
8. D - posun kamery doprava, pouze 3D komponenta
9. Q - posun kamery nahoru, pouze 3D komponenta
10. E - posun kamery dol-, pouze 3D komponenta
11. O - vykreslení dalších bin. ve směru dop°edu
12. P - vykreslení dalších bin. ve směru dozadu
13. T - posun kamery na souřadnici Z posledního vykresleného binu, pouze 3D komponenta
14. R - posun kamery na její první souřadnice, pouze 3D komponenta
15. V - zobrazit/schovat osu X, Y a Z (mohou pomoci v orientaci ve vizualizaci), pouze 3D komponenta

16. I - zobrazit okno pro úpravu základních hodnot komponenty

Ovládání my²ⁱ:

1. Kliknutí levým tlačítkem my²ⁱ - kliknutím na vykreslený čtverec/voxel tento prvek zvýrazníte a v pravé části okna se vypíší informace o eventu, který tento prvek reprezentuje. Pokud kliknete do pozadí, zvýraznění se smaže.
2. Posunutí levým tlačítkem my²ⁱ - podržením levého tlačítka my²ⁱ a následným táhnutím my²ⁱ můžete otáčet zobrazeným modelem. Pouze 3D vizualizace.
3. Posunutí pravým tlačítkem my²ⁱ - podržením pravého tlačítka my²ⁱ a následným táhnutím my²ⁱ můžete posunovat kamerou ve směru dopředu či dozadu. Pouze 3D vizualizace.
4. Scrolling - Scrollováním můžete posunovat kamerou ve směru dopředu či dozadu. Pouze 3D vizualizace.
5. Upravení rychlosti otáčení kamery při použití levého tlačítka - Rychlosť můžete upravit čoupátkem v levé části scény označeném: New Mouse Rotation Speed

I.4 2D vizualizace

V této části jsou popsány vlastnosti 2D vizualizace, znázorněna na obrázku I.4. 2D vizualizace funguje jako sloučení několika snímků z několika časů do jedné scény. Je vykreslována pomocí čtverců. Pozadí je tvořeno bílým čtvercem, jednotlivé události s namázenou energií jako čtverec s odpovídající barvou. Jejich X a Y souřadnice odpovídají zdrojovým souřadnicím z událostí. Na libovolný čtverec lze kliknout, následně se zvýrazní a napravo od komponenty se v příslušné oblasti vypíší informace o události, kterou čtverec reprezentuje. Kliknutím mimo krychli se zvýraznění čtverec smaže. Data jsou uložena jako seznam bináře seřazených chronologicky. Vždy je vykreslena jen část z uložených dat, vykreslení lze čoupat danými klávesami ve směru dopředu či dozadu. Pozice vykreslení lze zadat i přímo pomocí index či časovou známkou počátečního binu.

Dále jsou popsány jednotlivé parametry vizualizace, které lze upravovat z levé části scény.

1. Time Window hodnota udávající, kolik bin- má být najednou vykresleno. Po aktualizaci je vizualizace ihned p°ekreslena. Hodnota musí být kladná.
2. Space Between Images maximální mo°ná vzdálenost mezi dv¥ma biny. P°i ur°ování sou°adnic Z je pou°ito toto maximum, vzdálenost mezi dv¥ma biny nesmí být v¥tží ne° tato hodnota. Po aktualizaci je vizualizace ihned p°ekreslena. Hodnota musí být kladná.
3. Render Speed kolik bin- má být vykresleno p°i pohybu okna dop°edu £i dozadu. Platí pro oba vizualiza£ní módy. Po aktualizaci je vizualizace ihned p°ekreslena. Hodnota musí být kladná.
4. Max Pixel Color udávající maximální po°nou hodnotu ToT v datech, pomocí této hodnoty jsou spo£ítány barvy pro krychle. Lze upravit a zadat jinou hodnotu ne° je v datech, £ím° model p°ebarvit.
5. Color Range znázorn¥ní barevné 2kály pou°ívané k obarvení £tverc-. Zleva stoupající.

Obrázek I.4: 2D vizualizace

I.5 3D vizualizace

V této části jsou popsány vlastnosti 3D vizualizace, znázorněné na obrázku I.5. Ta je vykreslována do samostatné JavaFX scény původně souborné 3D. Má vlastní kameru, se kterou lze hýbat a otáčet klávesami. Události s naměřenou energií jsou vykresleny jako krychle a jsou umístěny na X a Y souřadnice odpovídající souřadnicím originálních událostí. Osa Z reprezentuje časovou osu, rozmístění bin. po ose Z odpovídá jejich dobu. Vykreslená data jsou obalena kvádrem značícím jejich hranice. Na libovolnou krychli lze kliknout, zvýrazní se a napravo od komponenty se v počtu oblasti vypíší informace o události, kterou krychle reprezentuje. Kliknutím mimo krychli se zvýrazní krychle sama. Data jsou uložena jako seznam bin. seřazených chronologicky. Vždy je vykreslena jen část z uložených dat, vykreslení lze rozepdat danými klávesami ve směru dopředu či dozadu. Pozice vykreslení lze zadat i přímo pomocí index či časovou značku počátečního binu.

Dále jsou popsány jednotlivé parametry vizualizace, které lze upravovat z levé části scény.

1. 3D VisualisationMode data ve 3D lze vizualizovat ve dvou módech, STATIC a MOVABLE. STATIC znamená, že při vykreslování nových bin. vizualizované okno zůstane na místě a mění se souřadnice Z jednotlivých bin. v pořadovaném směru. Kamera zůstane na místě. MOVABLE znamená, že při vykreslování nových bin. zůstanou souřadnice Z existujících bin. stejně, ale posune se vizualizované okno v pořadovaném směru. Je pak potřeba ještě posunout v tomto směru i kameru. Pro projevení změny módu je potřeba znova načíst vstupní data.
2. Time Window hodnota udávající, kolik bin. má být najednou vykresleno. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná.
3. Space Between Images maximální možná vzdálenost mezi dvěma bin. Při určování souřadnic Z je použito toto maximum, vzdálenost mezi dvěma bin. nesmí být větší než tato hodnota. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná.
4. Camera Speed rychlosť posunu kamery ve směru dopředu či dozadu. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná. V ostatních směrech je rychlosť pohybu kamery konstantní.

5. Render Speed kolik bim má být vykresleno při pohybu okna dopředu až dozadu. Platí pro oba vizualizační módy. Po aktualizaci je vizualizace ihned vykreslena. Hodnota musí být kladná.
6. Move Camera With Data zda se má kamera pohybovat spolu s vykreslováním nových dat, platí pouze pro mód MOVABLE. Po aktualizaci je vizualizace ihned vykreslena. Při posunu na konkrétní index nebo časovou známkou je kamera posunuta nezávisle na této hodnotě.
7. Max Pixel Color udávající maximální povrchovou hodnotu ToT v datech, pomocí této hodnoty jsou spočítány barvy pro krychle. Lze upravit a zadat jinou hodnotu než je v datech, tím model vybarvit.
8. Color Range znázorňuje barevné zakály používané k obarvení čtverců. Zleva stoupající.

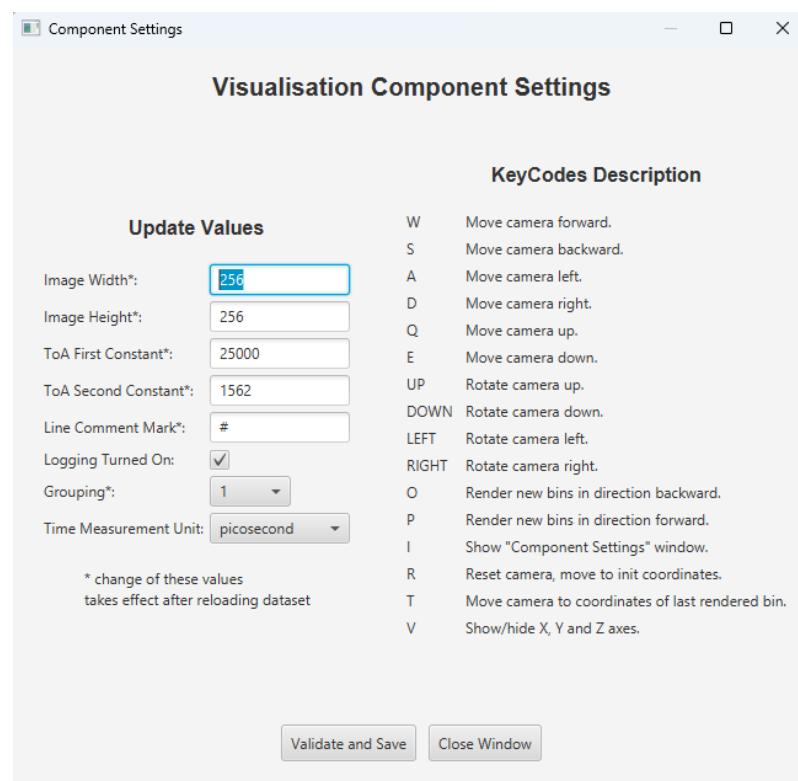
Obrázek I.5: 3D vizualizace

I.6 Nastavení parametrů společných pro obě části

Nastavení parametrů společných pro obě části (nezávislých na zvolené vizualizační metodě) se provádí přes samostatné okno, které lze zobrazit dvěma způsoby. Zaprve přes menu v levém horním rohu scény aplikace *File - Component Settings*, zadruhé stisknutím tlačítka */*. Podoba se nachází na obrázku I.6, obsahuje prvky pro aktualizaci daných hodnot. Nalevo se nachází popis prvku a napravo pozice pro zadání hodnoty. Při zobrazení okna se položky napravo naplní aktuálními hodnotami. Prvních pět hodnot se zadávají přes textfieldy a aktualizují se stisknutím tlačítka *Save and Validate*. Po jeho stisknutí jsou všechny zadané hodnoty zvalidovány. Pokud jsou validní, jsou aktualizovány, pokud ne, zobrazí se okno *Alert*. Ostatní hodnoty se zadávají přes checkbox nebo combobox. Aktualizují se ihned po zadání.

Dále jsou popsány jednotlivé hodnoty k aktualizaci a jejich význam.

1. **Image Width** – šířka vstupních dat. Hodnota musí být kladná.
2. **Image Height** – výška vstupních dat. Hodnota musí být kladná.
3. **ToA First Constant** – první konstanta k vypočítání ToA. Hodnota musí být kladná.
4. **ToA Second Constant** – druhá konstanta k vypočítání ToA, zároveň udává krok jednoho binu. Hodnota musí být kladná.
5. **Line Comment Mark** – řetězec udávající znak komentáře ve vstupních datech. Hodnota může být jakýkoliv řetězec, pouze hodnota nesmí být prázdná.
6. **Logging Turned On** – zda má být zapnuto logování v komponentě, když se loguje do konzole.
7. **Grouping** – kolik binů má být sloučeno při operaci grouping, označované též jako rebinování. Pokud je hodnota „1“, ke groupingu nedochází.
8. **Time Measurement Unit** – asová jednotka, ve které byla naměnena vstupní data. Není použita k dalším výpočtem, pouze k zobrazení aktuální hodnoty kroku jednoho binu v okně aplikace.



Obrázek I.6: Okno s nastavením komponenty