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

Bakalářská práce

Prototyp komponenty pro vizualizaci dat z částicových detektorů

Plzeň 2023 Jiří Buňata

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 vizualizační 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. Vyhodnoťte 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

V Plzni dne 30. srpna 2022

Prohlášení

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

V Plzni dne 4. května 2023

Jiří Buňata

Poděkování

Děkuji Ing. Janu Broulímovi, Ph.D. za dobré rady a vstřícnost při validování postupu práce a následném testování. Dále děkuji Ing. Richardu Lipkovi, Ph.D. za jeho odborné vedení bakalářské práce a čas, který mi věnoval 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 částicových detektorů. Hlavním požadavkem na komponentu je zpracování a vizualizace takových dat pomocí zvolené metody zaměřené na zobrazování dat v 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 částicových detektorů.

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

Obsah

1 Úvod

Cílem této práce je navrhnout a vytvořit programovou komponentu, která bude umět zpracovat a vizualizovat data z částicových detektorů. Motivací k vytvoření této bakalářské práce je vyvinout nový způsob vizualizace dat z částicových detektorů s možností analyzovat vývoj dat v čase. Aktuálně už existují nástroje na vizualizaci takových dat, ale většinou zobrazují jen data z konkrétních časů. Jedná se o experimentální práci, jejíž výsledkem bude vytvoření nové vizualizační metody pro data z částicových detektorů, která umožní analyzovat jejich vývoj v 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 ke každé komponentně bude vytvořena demonstrativní aplikace komponentu využívající.

Nejprve je potřeba prozkoumat podobu a význam dat získávaných z částicových detektorů. Následně prozkoumat možnosti vizualizace těchto dat. Na základě výsledků zkoumání budou vybrány dvě zobrazovací metody s důrazem na zobrazování dat v 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 připojitelná do dalších projektů. Navržené řešení bude následně implementováno v konkrétní zvolené technologii a bude k němu vytvořena aplikace využívající komponentu a demonstrující její funkce. Aplikace bude využita k otestování zaměřenému na správnost navrženého řešení a uživatelské přívětivosti. Podmínkou je, aby komponenta a aplikace byla vytvořena v programovacím jazyce Java, minimálně verze 8. Volba dalších technologií pro grafické uživatelské rozhraní a vizualizaci jsou na volbě zhotovitele práce.

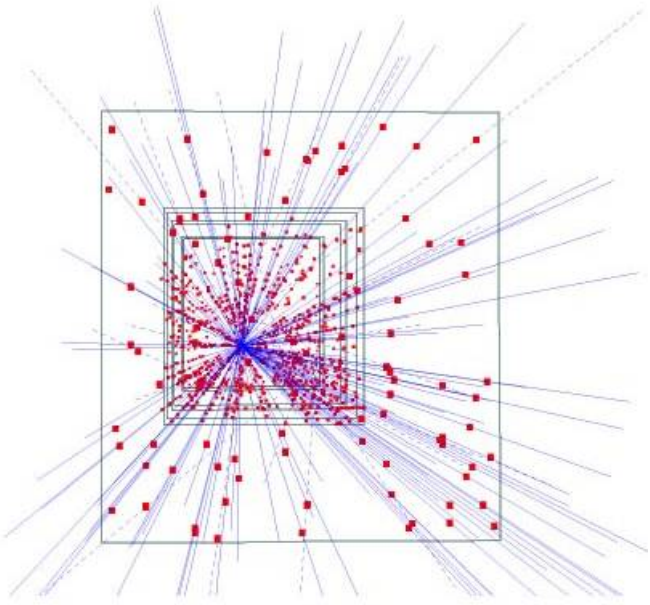
2 Částicové detektory a data, která jsou z nich získávána

Tato část je zaměřena na obecný popis částicových detektorů. Nezaměřuje se na provedení detektorů z technologického či fyzikálního hlediska, popisována je jejich všeobecná funkcionalita a využitelnost tak, aby čtenář získal ucelenou představu o tom, co jsou částicové detektory a jaké informace z nich lze získávat. Pro tuto kapitolu byly použity informace převzaté ze zdrojů zabývajících se částicovými detektory nebo byly poskytnuty přímo zadavatelem práce panem Ing. Broulímem Ph.D.

2.1 Částicové detektory

Částicový detektor je zařízení sloužící k detekci a identifikaci částic a lze pomocí něj i určit jejich dráhu. Využívají se k fyzikálnímu výzkumu, například vesmíru či materiálů, 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 čipům používaných v digitálních kamerách. [28] Když částice zanechá náboj, který je měřen, tak ho detektor zaznamená. Zaznamenané náboje jsou reprezentovány množinou pixelů a ukládány do rastrového obrázku. Je to ilustrováno na obrázku 2.1. Na něm je zobrazeno 153 částic letících tunelem. Červené pixely reprezentují právě ty pixely, které byly zasaženy nabitou částicí. Černý rám, ve kterém jsou obsaženy, představuje čas, ve kterém byly náboje naměřeny. Rámů je na obrázků více, to znamená, že jsou zaznamenána data z několika časových známek. Detektor lze natočit pod úhlem vůči ose tunelu od 0 do 90 stupňů, rozdílný úhel natočení může pomoci zachytit různé vlastnosti. Částice se stejnou dráhou se pak při různých natočeních zaznamenají jinak. Například při natočení 0 stupňů se částice letící kolmo na detektor zaznamenají jako shluk pixelů podobný kruhu nebo čtverci, při natočení blížící se 90 stupňům při stejném směru bude již ale zaznamenána jako táhlá čára. Natočení detektoru lze využít třeba pro zaznamenávání drah částic. Některé detektory umí zaznamenat i energii, jakou částice měla v momentě průchodu detektorem. Energie je reprezentována barvou pixelu. Barevná škála a význam jednotlivých barev závisí na každém detektoru.

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

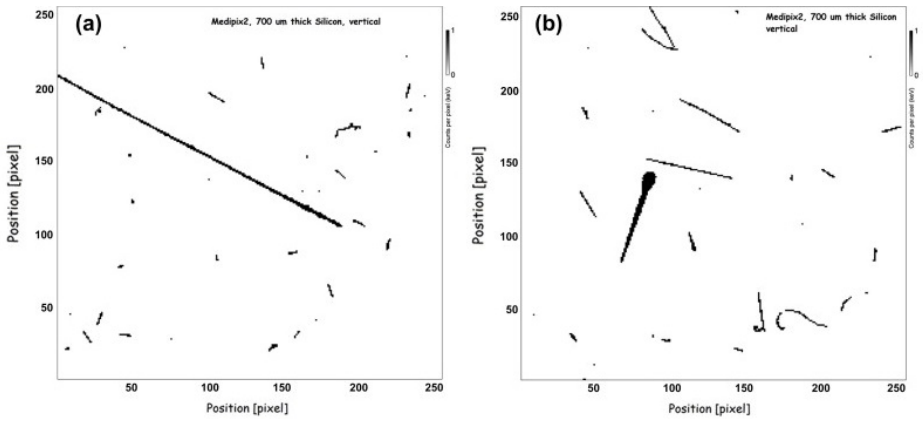


Obrázek 2.1: Znázornění pixelového detektoru [27]

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 současné metody vizualizace rozebrány v kapitole 3. Naměřená data jsou ukládána jako pixely do 2D rastrového obrázku, kde jednotlivé hodnoty pixelů reprezentují energii v daném bodě. Obrázek může být čtvercový nebo obdélníkový, podle konkrétního detektoru. Například Medipix1 používá 64x64 a Medipix2 256x256. [3] Nulová hodnota znamená, že v daném bodě nebyl zaznamenám žádný náboj. Nenulová naopak znamená, že v bodě byl náboj naměřen. Naměřená energie je reprezentována barvou pixelu. Tato hodnota se nazývá *Time over Threshold*. Nutno poznamenat, že pouze reprezentuje danou energii v bodě, ale nelze ji využít přímo jako energii vyjádřitelnou ve fyzikálních jednotkách. Na to je potřeba ji přepočítat, metoda výpočtu se liší podle konkrétního detektoru, tím se ale v této práci nebudeme zabývat. Pro vizualizaci postačí samotná hodnota *Time over Threshold*, která poslouží k určení finální

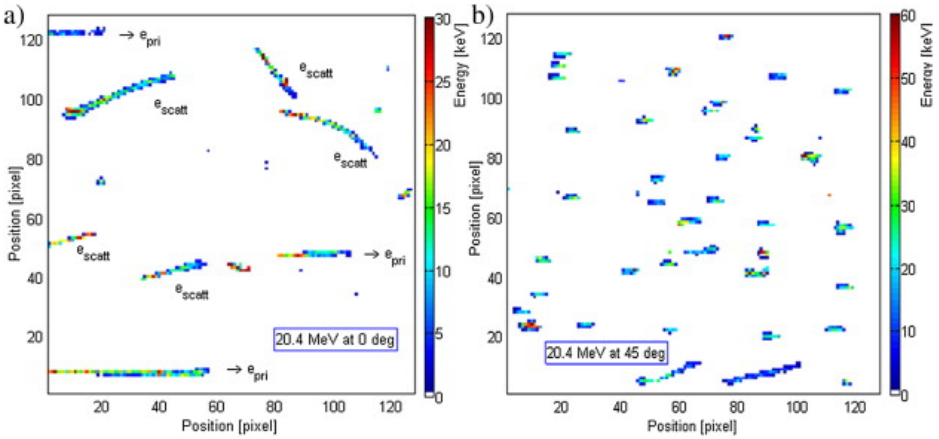
barvy vizualizovaného prvku. Čím větší má hodnotu, tím větší energii reprezentuje. Maximální možná hodnota též závisí na konkrétním detektoru. Poslední důležitou informací je časová známka udávající, v jakém čase byla data naměřena. Nazývá se *Time of Arrival* a udává se v různých časových jednotkách, například v mikro, nano nebo pikosekundách. Množina pixelů s naměřenou energií, které jsou sdružené podle nějaké vlastnosti, se nazývá *energy bin* nebo zkráceně *bin*. V rámci této práce budou sdružovány pixely podle času. Bin tedy bude obsahovat pixely se stejnou časovou známkou. Způsob uložení dat je pak závislý na konkrétním formátu. Dále budou ukázány příklady některých dat a jejich popis.



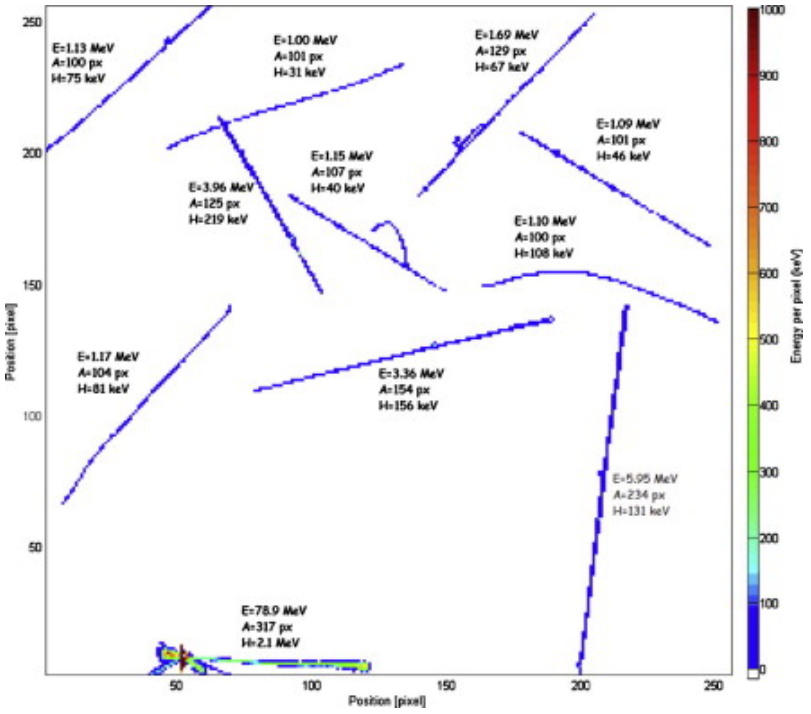
Obrázek 2.2: Příklad jednoduché vizualizace dat [12]

Na obrázku 2.2 se nachází obecný příklad vizualizace dat. Černé pixely znázorňují 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 naměřen náboj. U některých částic lze určit i jejich směr vůči 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 obarveny dle dané barevné škály a jednotlivé barvy reprezentují energii. U obou obrázků je v modrém obdélníku poznamenána 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: Příklad vizualizace dat se znázorněnou energií [13]

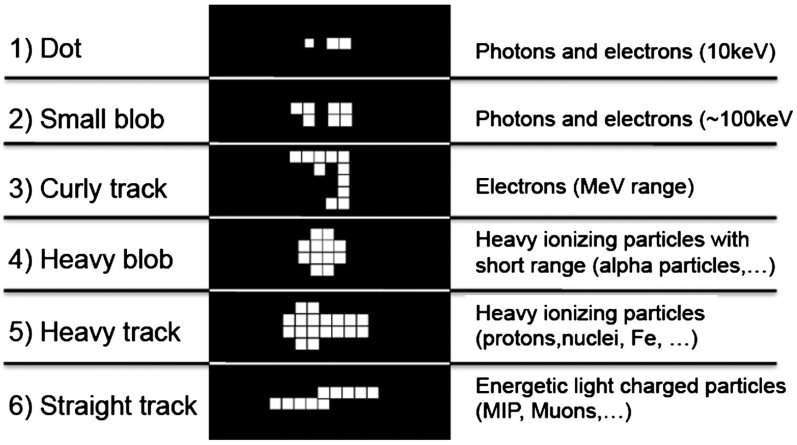


Obrázek 2.4: Příklad vizualizace dat s popisem zaznamenaných částic [12]

Na obrázku 2.4 jsou zobrazeny další příklady vizualizace dat. Energie je opět znázorněna barvou dle dané barevné škály, v obrázku jsou sloučena data z několika snímků do jednoho. U zaznamenané částice je vždy uvedena energie, jakou byla částice prohnána detektorem (E), počet příslušných pixelů (A) a nejvyšší naměřená energie ve shluku pixelů (H). U většiny zaznamenaných částic byla pozice detektoru téměř rovnoběžná se směrem částic, díky

tomu zaznamenaná data tvoří dlouhé čáry.

Každý druh částic má své specifické chování a zanechává specifické obrazce, podle kterých lze rozeznat, která částice byla zaznamenána. V tabulce na obrázku se nachází příklady takových obrazců a jakou částici představují. Analýzu dat lze provádět manuálně pomocí některé 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].



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

2.3 Formáty výstupních dat

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

2.3.1 Formát Time Frame

Data pro tento formát jsou získávána časovým měřením, kdy v určitý moment detektor provede snímek a zaznamená všechna data, které v onen moment detekoval. Data snímku jsou znázorněna jako 2D rastrový obrázek, kde každý pixel popisuje, zda a jaká na něm 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 souřadnici. Barva pixelu určuje hodnotu *Time over Threshold*.

2.3.2 Formát Data Driven

Data pro tento formát jsou získávána takzvaným událostním měřením. Tím je myšleno, že detektor čeká na událost a když ji naměří, zaznamená ji. Událostí je myšleno průchod částice detektorem. Data v tomto formátu jsou uložena v textové podobě po řádcích. Jedna řádka reprezentuje jeden pixel, ve kterém byla událost naměřena. Obsahuje informace o souřadnicích daného pixelu, času, kdy k události došlo a naměřené energii. Formát zapsání řádky se může lišit v závislosti na detektoru, například některé hodnoty mohou být zadány přímo nebo je potřeba je z uvedených dat dopočítat, nebo může být použit jiný oddělovač hodnot. Pro tuto práci je použit formát pro čip *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 dopočítat. Reprezentuje index v maticovém modelu, kde jsou řádky poskládány za sebou do jednorozměrného pole. Výpočet se provede následovně:

   - Souřadnice X: hodnota je celočíselně vydělena šířkou matice:

   $$X = \frac{coordinate}{matrix\_width} \tag{2.1}$$

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

   $$Y = coordinate \pmod{matrix\_width} \tag{2.2}$$

2. **ToA** – Time of Arrival, hrubá časová známka - hodnota, která následně slouží k výpočtu celkové časové známky (**TToA**).

3. **FToA** – Fine Time of Arrival, jemná časová známka - hodnota, která následně slouží k výpočtu celkové časové známky (**TToA**).

4. **ToT** – Time over Threshold, zaznamenaná energie v daném pixelu, hodnota reprezentuje jeho barvu.

5. **TAB** – Tabulátor, oddělovač 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 časová známka, která udává, kdy k naměření události došlo. Nazývá se **Total Time of Arrival**(zkráceně *TToA*) a spočítá se podle vzorce:

$$TToA = ToA \times CONST\_1 - FToA \times CONST\_2 \tag{2.3}$$

CONST_1 a *CONST_2* jsou konstanty udávané 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 časového rozsahu jsou naměřená data shromážděny do jednoho binu. Udává se v pikosekundách. [8]

3 Metody zobrazování dat z částicových detektorů

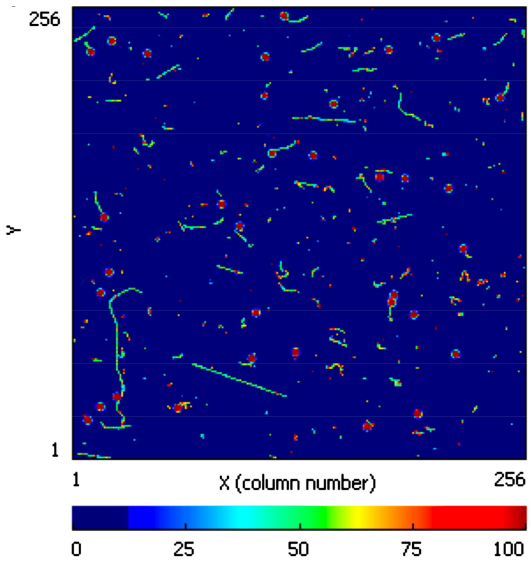
Tato kapitola je zaměřena na popis zobrazovacích metod dat z částicových detektorů. Nejprve jsou popsány příklady současných vizualizačních metody a následně jsou vybrány dvě vhodné zobrazovací metody, které budou implementovány v rámci této bakalářské práce.

3.1 Současné vizualizační metody

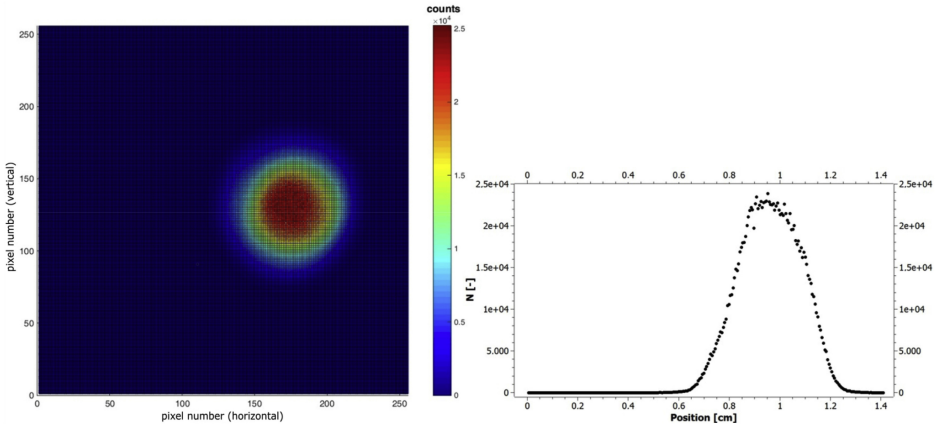
V této části jsou rozebrány současné vizualizační metody pro data z částicových detektorů a je zde uvedeno několik příkladů z vědeckých článků. Vizualizace dat již byla lehce načrtnuta v kapitole 2, tam byl ale kladen důraz primárně na popis významu dat. Metody jsou rozděleny do tří kategorií: vizualizace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

3.1.1 Vizualizace 2D dat

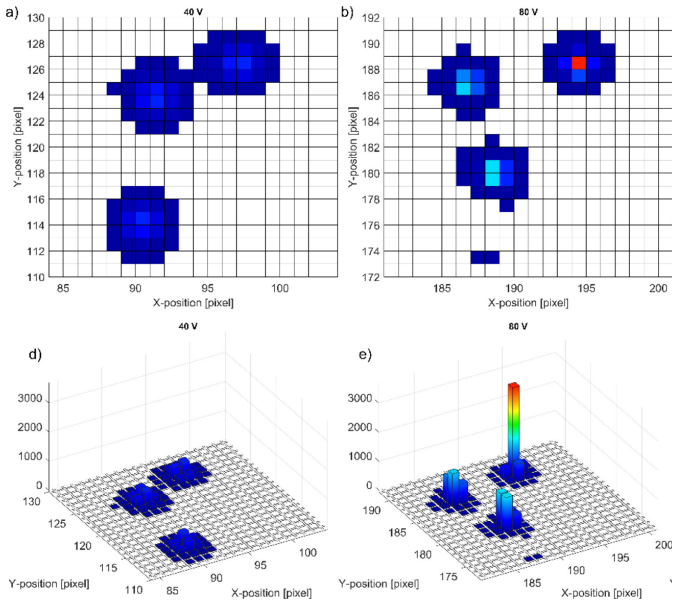
V této části jsou popsány možné vizualizační metody 2D dat. Není podstatné, zda data v obrázku jsou z jednoho či několika binů či zda byla nějak upravována po výstupu z detektoru, ale vstupem pro vizualizaci musí být 2D rastrový obrázek. Nejdříve je uveden příklad obecné vizualizace podobný těm uvedeným v kapitole 2 na obrázku 3.1. Jedná se o nejjednodušší formu vizualizace, kdy jsou jednotlivé pixely obarveny podle barevné škály. Ta je hlavním prvkem sloužícím k tomu, jak rozpoznat jevy v datech. Existují metody, jak data dále zvýraznit. Typicky jsou zaměřené na zvýraznění energie v jednotlivých pixelech. Jednou z možností je udělat řez v konkrétním bodě v obrázku a zobrazit energie v řezu. Znázorněno je to na obrázku 3.2. Na něm je vidět jak původní vizualizace 2D, tak zobrazení řezu v konkrétní řádce vedle sebe. V tomto příkladu je pro zobrazení řezu použit 2D graf, kdy na ose X je pozice pixelu v řádce a na ose Y energie naměřená v daném bodě. Další možností podobnou k použití řezu 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 původní 2D podobě a jak ve vizualizované 3D podobě.



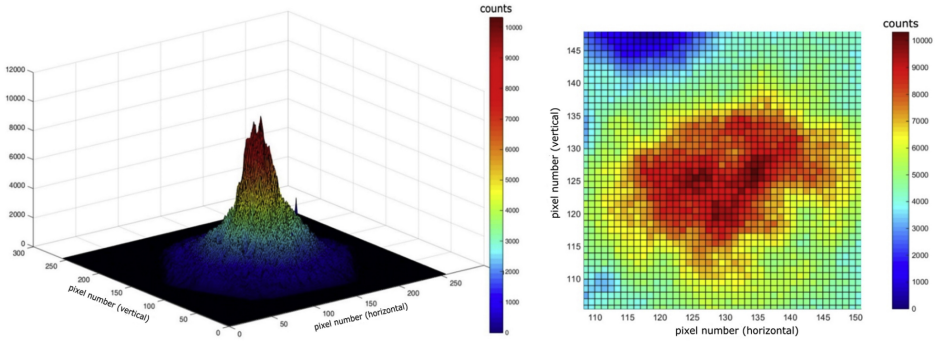
Obrázek 3.1: Příklad jednoduché 2D vizualizace. [15]



Obrázek 3.2: Příklad vizualizace řezem. [29]



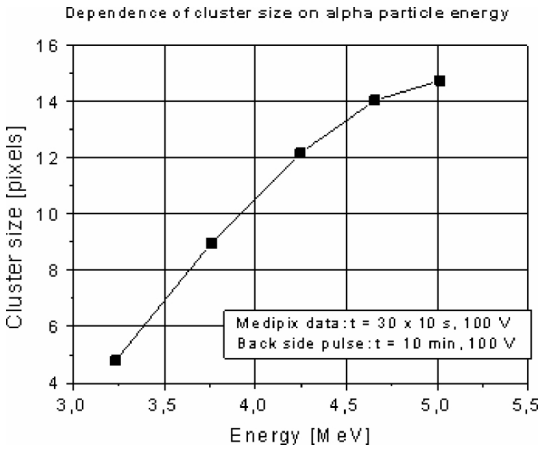
Obrázek 3.3: Příklad 3D vizualizace, na osách X a Y jsou pozice jednotlivých pixelů a osa Z reprezentuje energii pro jednotlivé pixely. [34]



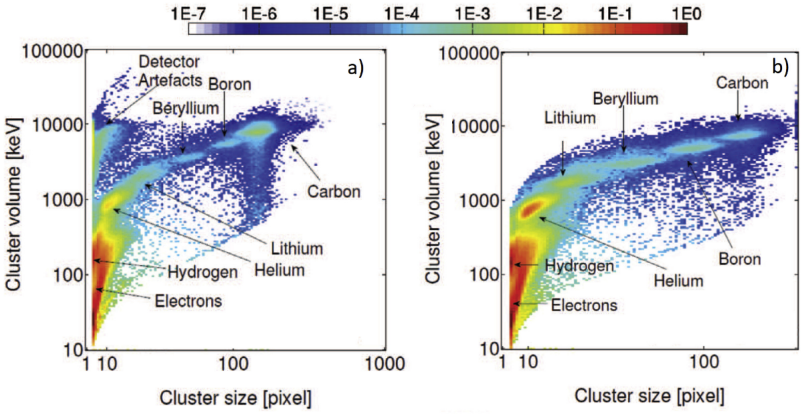
Obrázek 3.4: Příklad 3D vizualizace, na osách X a Y jsou pozice jednotlivých pixelů a na ose Z je vynesena energie pro jednotlivé pixely. [29]

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 naměřeny 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 článku, kde bylo zpozorováno, že velikost clusteru pixelů (shluku sousedních pixelů; velikost clusteru udává počet pixelů v něm) je závislá na energii zaznamenané částice. [6] Čím byla energie částice vyšší, tím více pixelů se ve shluku nacházelo. Výsledky z měření jsou zaznamenány v grafu na tomto obrázku, na ose X je vynesena energie a na ose Y počet pixelů ve shluku. Obrázek 3.6 ukazuje zajímavou vizualizaci, která se také věnuje závislosti velikosti clusteru na energii. Jedná se o 2D graf, na ose X je velikost clusteru (udaná jako počet pixelů pro daný cluster) a na ose Y „obsah“ clusteru (suma energií všech pixelů pro daný cluster). Barevně jsou oddělena data z jednotlivých binů podle barevné škály. Popisky v grafu popisují jaké prvky jsou v jednotlivých clusterech zaznamenány. Tím, že se jedná o vizualizaci z několika samostatných binů, může se jednat i o možnost vizualizace dat z několika časových známek.



Obrázek 3.5: Graf znázorňující závislost velikosti clusteru na energii zaznamenané částice. [6]

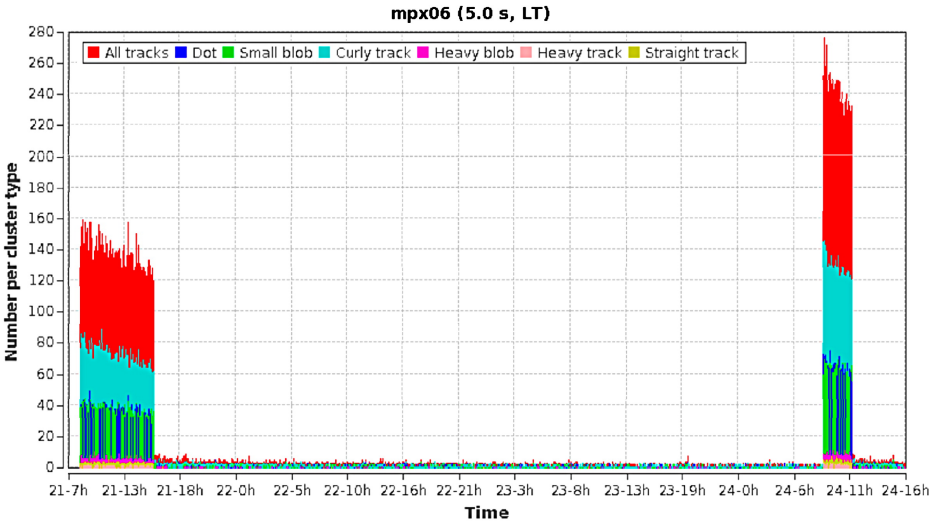


Obrázek 3.6: Graf znázorňující závislost velikosti clusteru na obsah clusteru (suma energií všech pixelů pro daný cluster). [32]

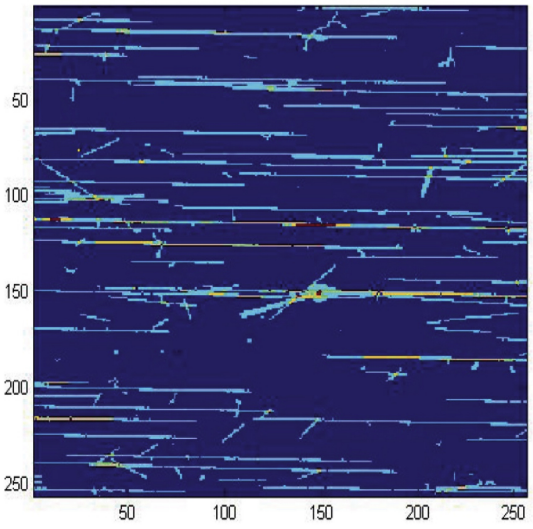
3.1.3 Zobrazení dat v čase

Tato část popisuje možnosti vizualizace s ohledem na zobrazení dat v čase. Cílem je zobrazit data z několika časových známek tak, aby bylo možné pozorovat jejich vývoj v čase. Jedna z možností je popsána v článku *Measuring radiation environment in LHC or anywhere else, on your computer screen with Medipix* [15], z něhož byl převzat obrázek 3.7. V tomto článku je popsáno měření, kdy byly analyzovány biny z několika časových 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, kdy se z naměřených dat prozkoumají pozorované jevy a vynesou se na číselnou osu. Průzkumem dat na časové ose lze určit jevy na které se zaměřit a konkrétní biny, které je obsahují, pak hlouběji prozkoumat. Pro průzkum jednotlivých binů může být použita některá z výše popsaných vizualizačních metod.

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



Obrázek 3.7: Graf ukazující typy clusterů v binech pro konkrétní časové známky. Typy clusterů jsou odděleny barevně podle legenda.[15]



Obrázek 3.8: 2D vizualizace, kdy jsou do jednoho obrázku sloučena data z několika časových známek. U pixelů se stejnými souřadnicemi je jejich hodnota sečtena. Pixely z pouze jedné časové známky jsou světle modré, z několika žluté až červené. [14]

3.2 Zvolení vhodných zobrazovacích metod pro komponenty

V této části 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 čase. Jako druhá byla tím pádem zvolena 2D metoda za účelem porovnání, jak velký rozdíl bude v přínosu informace mezi dvěma dimenzemi. Vzhledem k experimentální podstatě práce nelze dopředu určit, která metoda bude ideální. Až na výsledcích práce bude určeno, zda je zvolená metoda pro zobrazování daných informací vhodná a zda poskytuje potřebné informace. Obě zvolené metody budou mezi sebou porovnány, zda obě poskytují stejné množství informací.

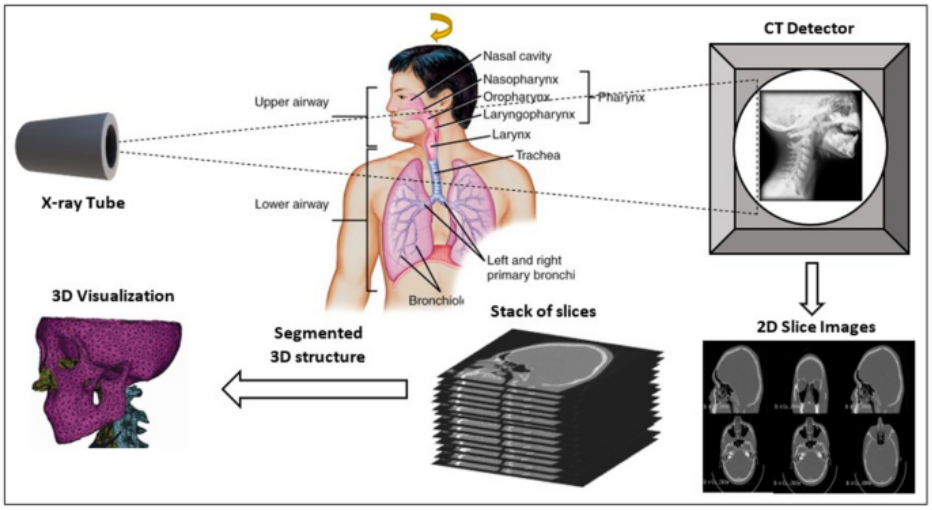
3.2.1 2D metoda

V této části je rozebrána volba vhodné zobrazovací 2D metody. Metoda musí umět řešit stejné problémy jako 3D metoda, ale jiným způsobem. Ta je zaměřena hlavně na převod 2D dat do 3D podoby a jejich vynesení na číselnou osu. Vzhledem k chybějící časové ose se 2D metoda zaměří spíše na to, jak vizualizovat data z několika časových snímků ve 2D. Pro zobrazení v čase je zvolena již existující podoba popsaná výše v části 3.1.2, kdy data z několika snímků budou promítnuta do jednoho snímku. Přidanou hodnotou bude přidání funkcionality, aby při sloučení nebyla ztracena informace o původních binech. Dosaženo toho bude pomocí zakomponování interaktivity, kdy bude možné kliknout na každý pixel a zobrazit informace o něm, jako je jeho energie, souřadnice a časové známky, ze kterých se skládá. V místech, kde budou kolidovat pixely s naměřenou energií z několika časových známek budou, bude barva určena jinak. Na výběr je několik řešení, například: pixely v kolizi obarvit nějakou specifickou barvou; sečíst barvy všech pixelů, které jsou v kolizi; zvolit barvu jako počet časů, které jsou v kolizi. Zvolena byla metoda sečíst barvy všech pixelů, aby bylo na pohled zřetelné, že je v daném místě vyšší energie a že mohlo dojít ke sloučení energie z několika časů. Konkrétně bude sečtena hodnota ToT jednotlivých pixelů a na základě té bude přiřazena barva, jako pro jakýkoliv jiný pixel. Vzhledem k možnému většímu množství zdrojových dat bude zobrazována jen volitelná podmnožina dat. Vizualizace by byla jinak pravděpodobně nepřehledná.

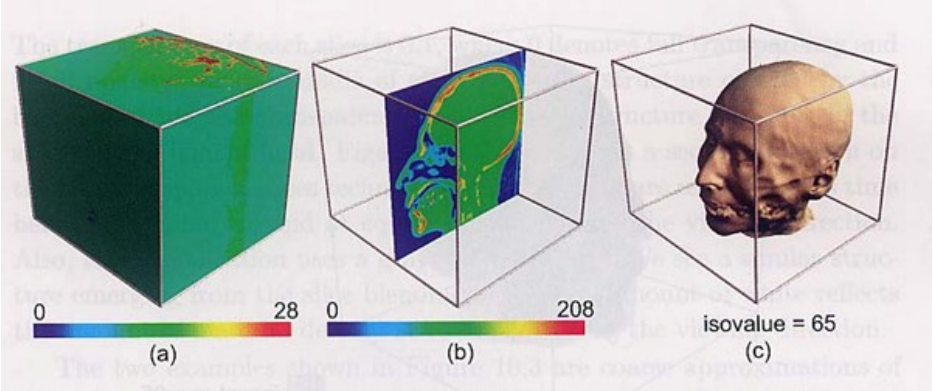
3.2.2 3D metoda

V této části je rozebrána volba vhodné zobrazovací 3D metody. Metoda musí umět řešit dva základní problémy: jak vhodně převést zdrojové 2D snímky na 3D a jak snímky vynést chronologicky na časovou osu. Pro její zvolení se lze inspirovat v podobných úlohách zaměřených na zobrazování objemových dat. Blízkým takovým kandidátem je technologie CT (computed tomography)[30], která se zaměřuje na zobrazení 3D vizualizací na základě naměřených 2D snímků. Často se používá v lékařství pro vizualizaci struktury částí lidského těla nebo i pro vizualizaci a analýzu struktur materiálů. CT sken nasnímá cílový objekt z několika úhlů a z každého skenovaného úhlu vytvoří 2D snímek. Množina 2D snímků popisuje objekt z několika úhlů a lze z nich tedy vytvořit 3D vizualizaci. Celý proces je znázorněn na obrázku 3.9. Na obrázcích 3.10 a 3.11 je poté vyobrazen proces, kdy z bloku 2D snímků je vytvořena 3D vizualizace. Blok je zanalyzován vhodným algoritmem a jsou vybrána data, které je potřeba transformovat do 3D. Tyto jednotlivé prvky jsou následně transformovány do tzv. voxelů, což jsou 3D ekvivalent pixelů a jsou reprezentovány jako krychle. Z toho je i odvozen jejich název, pixel je složen z anglického slovního spojení „picture element“ (česky obrazový prvek), voxel ze slov „volume element“ (česky objemový prvek). Na obrázku 3.12 jsou pro porovnání znázorněny rozdíly mezi rastrovou a vektorovou grafikou ve 2D a 3D. Tento způsob, tedy použití voxelové grafiky, lze použít i pro 3D vizualizaci dat z částicových detektorů. Vstupem je také množina 2D snímků, které je potřeba vizualizovat ve 3D. S tím rozdílem, že metoda pro CT slouží k zobrazení v prostoru, data z detektorů budou zobrazována v čase. Voxelová grafika je obecně častou technikou pro zobrazování reality, protože umožňuje přímý převod z 2D rastrové grafiky do 3D, z jednotlivých voxelů lze sestavit prostor podle požadovaných vlastností a každému voxelu lze přiřadit specifické vlastnosti. V současnosti jsou modelovány na softwarové vrstvě pomocí polygonů, v historii byly pokusy optimalizovat hardware na vykreslování voxelů, ale v současnosti je optimalizován na vykreslování polygonů.

3D vizualizace tedy bude vytvořena za pomoci voxelové grafiky. Jednotlivé biny/snímky budou převedeny do 3D a umístěny na osu Z, která poslouží jako časová osa. Jednotlivé snímky budou vytvořeny tak, že se vymodelují voxely odpovídající jednotlivým pixelům v obrázku. Jako souřadnice lze použít jednotlivé hodnoty *ToA* každého snímku. Chronologicky na ni budou vyneseny jednotlivé snímky. Člověk, který pak bude data analyzovat, si bude moci v čase prohlédnout vývoj jednotlivých snímků. Vzhledem

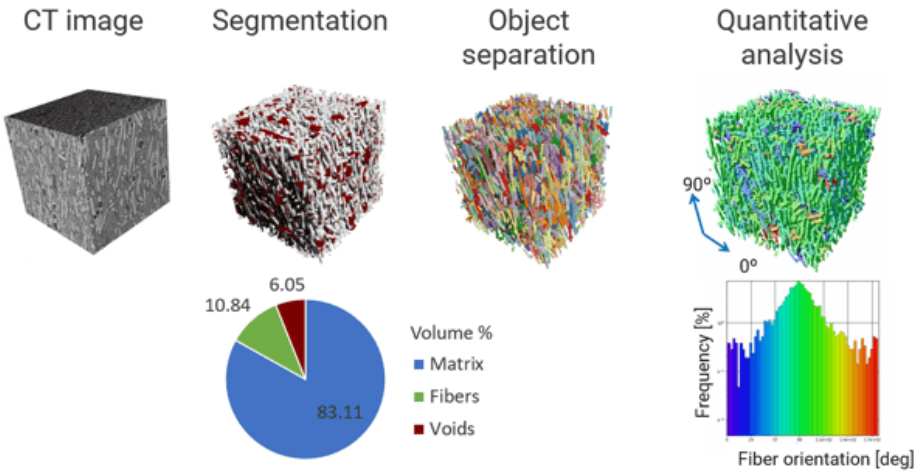


Obrázek 3.9: Proces vytvoření 3D obrazce při použití technologie CT (computed tomography). [26]

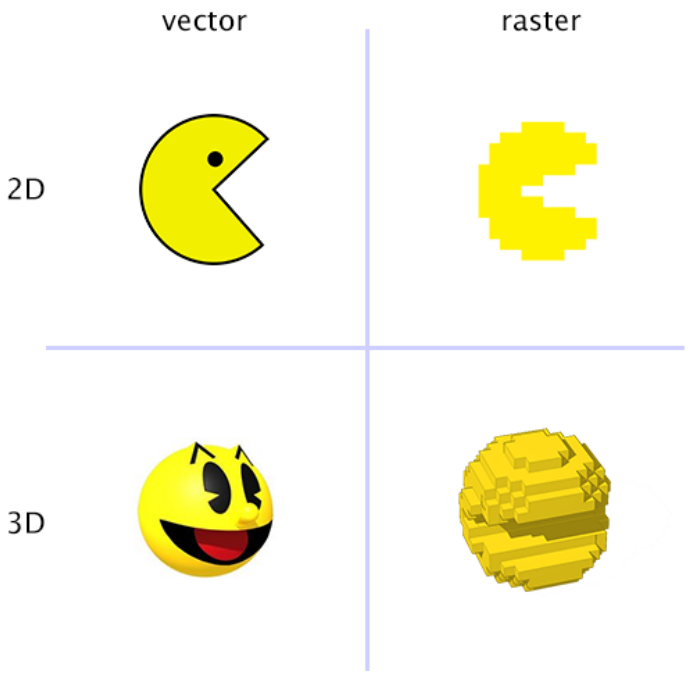


Obrázek 3.10: Znázornění konstrukce 3D obrazce ze zdrojových dat. a) množina naměřených dat; b) řez naměřenými daty, 2D snímek; c) konstrukce 3D obrazce ze zdrojových dat [33]

k možnému většímu množství zdrojových dat bude zobrazována jen volitelná podmnožina dat. Teoreticky by 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 vzdálenějších souřadnicích osy Z by bylo zbytečné.



Obrázek 3.11: Znázornění 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: Znázornění rozdílů mezi vektorovou a rastrovou grafikou ve 2D a 3D. [16]

4 Návrh komponenty

Tato kapitola je zaměřena na návrh sestrojení samotné komponenty, která je předmětem této práce. Budou vyhotoveny dvě komponenty (pro 2D a 3D metodu), jejich návrh je v mnoha oblastech společný 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 odvíjeny další části návrhu. Obecně při návrhu platí, že co nejvíce společných vlastností pro obě komponenty bude sloučeno 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 funkcionalitu.
- Technologie komponenty musí být kompatibilní s programovacím jazykem Java, minimálně verze 8.
- Komponenta bude vyhotovena v technologii podporující GUI (Grafical User Interface) a vykreslování grafických objektů. Vizuálně bude tvořena pouze grafickými objekty, ale půjde 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 půjdou získat, aby mohli být umístěny do okna aplikace. Ideálně v nějaké dynamické struktuře, která se sama bude starat o aktualizaci informací, aby se aplikace samotná nemusela ptát na nové hodnoty.
- Komponenta bude reprezentována hlavní třídou, umístěním instance této třídy do aplikace lze volat její funkcionalitu.
- Obě komponenty budou vizualizovat pouze část načtených dat, komponenta bude implementovat grafický ukazatel aktuální pozice zobrazeného okna v celkových datech. Ten půjde z komponenty získat a umístit do okna aplikace.

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

4.2 Základní pojmy

V této části jsou popsány pojmy, které jsou použity pro implementaci komponenty a jsou důležité k pochopení dalšího textu. Jedná se o fyzikální pojmy, do jisté míry už zmíněny v kapitole 2 a v zde jsou zopakovány. Částicový detektor udává naměřená data ve formě rastrového obrázku, barvy jednotlivých pixelů udávají, zda jak velká byla na tomto místě zaznamenaná energie. Obrázky jsou často čtvercové, ale můžou 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, zaznamená data do souboru jakmile naměří událost. Tato událost odpovídá právě jednomu pixelu s naměřenou energií v rastrovém obrázku. Nadále budou nazývány jako *událost*, případně anglickým ekvivalentem *event*. Shluku události pro jeden konkrétní čas se říká *bin*. Zároveň se z něj 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 umožňuje tvořit GUI (Grafical User Interface) a vykreslování grafických komponent (ideálně 2D i 3D prvků). Lze použít dvě různé technologie, kdy by každá implementovala jednu z těchto vlastností a následně by byly spolu propojeny, ale pro jednoduchost bude vybrána taková, který umí obojí. Grafické komponenty budou použity k vykreslení vybrané vizualizační metody, přes prvky GUI bude bude vizualizace ovládána. Ty sice nebudou součástí samotné komponenty, tu bude tvořit pouze vizualizační část, ale bude výhodné, aby je podporovala. a Omezením při výběru je zadání, aby kořenovou technologií pro komponentu byl programovací jazyk Java. Zvolená technologie tedy musí být s tímto jazykem kompatibilní. Nejznámějšími možnostmi jsou technolo-

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

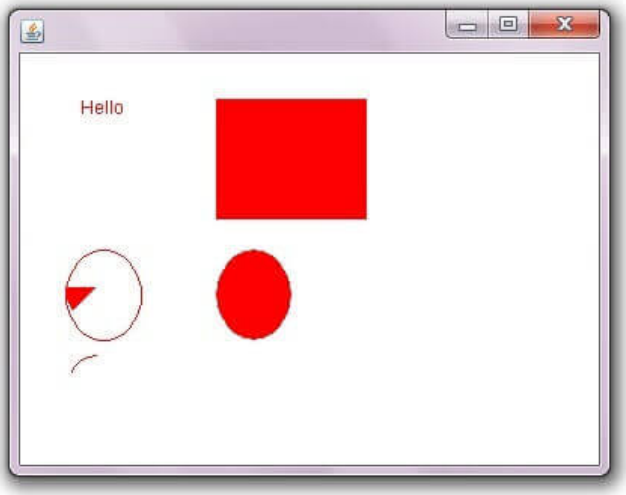
AWT a Swing jsou si velmi podobné, protože Swing vychází z AWT, takže byly sloučeny do společné části.

AWT je z uvedených nejstarší technologií, představena byla v roce 1995 spolu s první verzí Java. V současnosti už se ale téměř nepoužívá, protože už není dále vyvíjena. Tím pádem je zastaralá a neobsahuje pokročilé komponenty, které moderní GUI frameworky využívají. V současnosti se hodí pouze na menší a jednodušší aplikace. Díky svému odkazu se v Javě stále vyskytuje a některé z novějších frameworků používají některé funkce AWT. AWT umožňuje kreslení vlastní grafiky pomocí třídy `java.awt.Graphics` nebo `java.awt.Graphics2D`. Umožňuje pouze kreslení jednoduchých 2D obrazců, jakou jsou obélníky, 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 převáděla do 2D, kterou by pak následně vykreslovala třída `Graphics`.

Swing vychází z frameworku AWT, představen byl v roce 1996. Jedná se o následníka AWT, funkcionalita je vystavěna 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 novějších frameworků. Přesto je stále používán v mnoha projektech, zejména z historických důvodů v těch, které ho používají z minulosti a je poměrně složité přejít na jiný framework. Kreslení obrazců se zde provádí též přes třídu *java.awt.Graphics*, právě proto že Swing je vystavěn 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

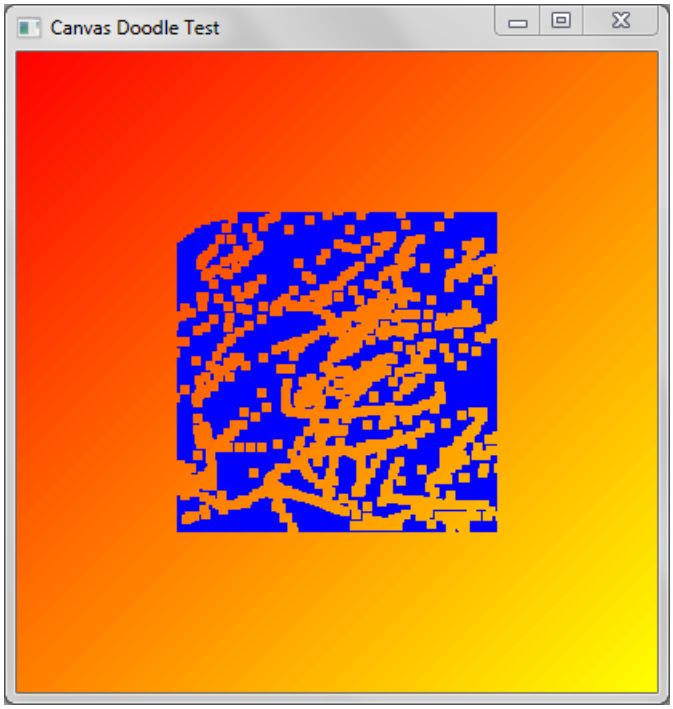
JavaFX patří k nejnovějším a nejlepším Java GUI frameworkům, představena byla v roce 2008 jako součást JDK. Od JDK verze 11 a dále není JavaFX jeho součástí, ale je vydávána jako samostatná knihovna, kterou je potřeba do projektů importovat. JavaFX je stále vyvíjena a pravidelně



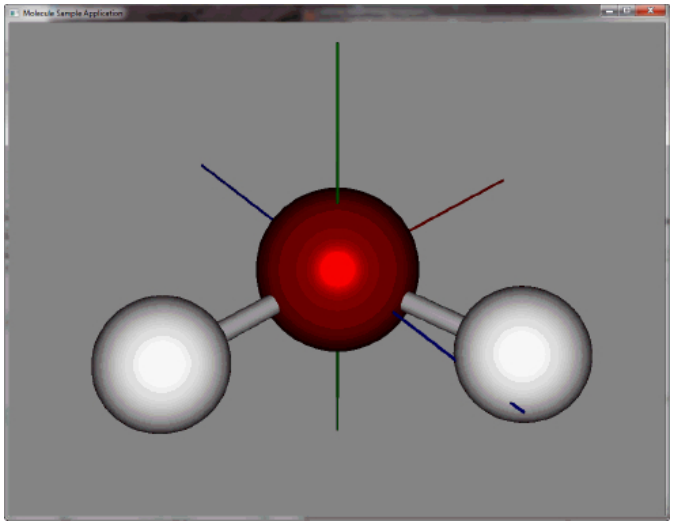
Obrázek 4.1: Příklad kreslení 2D objektů pomocí frameworku Swing [36]

jsou vydávány nové verze, což usnadňuje rozvoj a údržbu aplikací. Většina nových Java GUI aplikací vyvíjených v poslední dekádě používají JavaFX. Tím že je novější a neustále vyvíjena, poskytuje modernější a bohatší vzhled GUI než třeba AWT nebo Swing. Umožňuje snadnou práci s technologiemi jako *CSS* nebo *FXML*. JavaFX umožňuje vykreslování 2D i 3D grafických komponent. 2D obrazce lze vykreslit několika způsoby. Zaprvé lze vytvářet samostatné 2D objekty, které jsou umisťovány přímo do scény (lze použít předdefinované jako obdélník či kruh apod., tak i lze vytvořit vlastní pomocí třídy `javafx.scene.shape`). Zadruhé lze použít třídu `javafx.scene.canvas.Canvas`, která vytvoří „plátno“, do kterého lze tvořit libovolné obrazce. Příklad použití metody `Canvas` je na obrázku 4.2. Zatřetí lze ještě využít komponenty pro vykreslení přímo předaných obrázků (vytvořit uvnitř aplikace jako obrázek v nějakém standardizovaném formátu (jpg, png,...)), například `javafx.scene.image.ImageView`. Tyto metody lze použít pro 2D vizualizaci, například pro sloučení více snímků do jednoho a následně ho vykreslit. Lze ho vykreslit do plátna jako nový snímek, nebo ho vytvořit uvnitř aplikace jako obrázek v nějakém standardizovaném formátu (jpg, png,...) a následně ho vykreslit pomocí knihovní funkce. Pro vykreslování 3D obrazců lze použít podobný způsob jako pro 2D, kdy jsou obrazce vytvořeny jako samostatné objekty (krychle, kvádr, koule,...) a následně jsou umístěny do scény. Často se vykreslují do samostatné scény vložené do původní scény (javafx.scene.SubScene), protože 3D scéna bude mít jiné vlastnosti než scéna s 2D komponentami. Příklad vykreslení 3D objektů se nachází na obrázku 4.3. Případně by bylo samozřejmě možné také využít

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



Obrázek 4.2: Příklad kreslení pomocí třídy `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 pokračujícímu vývoji

této technologie. Bude tedy potřeba i řešit import knihovny JavaFX napříč různými 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 řešit až tvůrce tohoto projektu. Bude to ale potřeba vyřešit u demonstrační aplikace, která je součástí této práce.

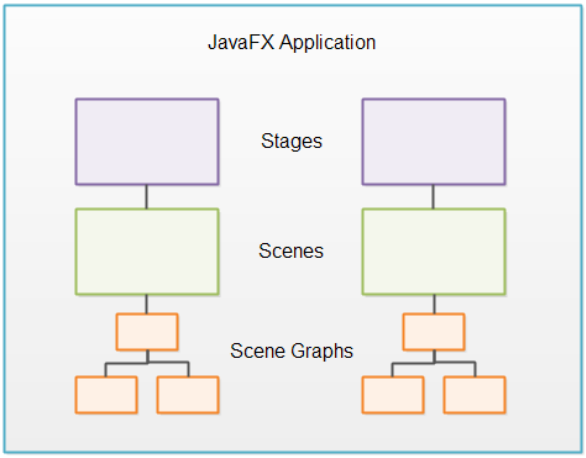
4.4 Architektura komponenty

Tato část je zaměřena na navržení architektury a struktury komponenty. Při návrhu je potřeba vzít v potaz, že jde o implementaci komponenty a ne samostatně spustitelného programu. Komponenta bude program, který půjde 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 zapouzdřena a byla dostupná pouze přes toto *API*. Proto bude vyčleněna jedna třída komponenty jako hlavní třída, její veřejné metody ho budou reprezentovat. Komponenta bude navržena jako vlastní grafický prvek JavaFX (jako jsou tlačítka, okna pro výpis, apod.), který lze přímo umístit do scény jako *Node*. Snadné řešení jak toho dosáhnout je, aby komponenta dědila od nějaké existující grafické komponenty JavaFX. Seznam všech komponent lze nalézt pod zdrojem [23].

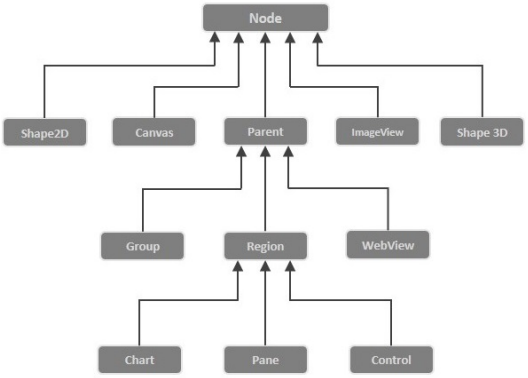
4.4.1 Popis základních struktur JavaFX

Pro lepší pochopení, jaké přesně místo bude v hierarchii JavaFX komponenta zaujímat, jsou v následujících odstavcích popsány základy architektury JavaFX. Předmětem této části není poskytnout čtenáři návod 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 vnější 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 prostředí, které slouží k zobrazování objektů samotného GUI. Všechny prvky, které je potřeba zobrazit, jsou umístěny do scény. *Stage* může najednou 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 dědí od třídy *javafx.scene.Node*. Z toho vyplývá, že aby navržena vizualizační komponenta fungovala správně jako komponenta JavaFX, musí od této třídy také dědit. Toho lze dosáhnout tak, že bude komponenta dědit už od nějaké existující JavaFX komponenty a umístěna 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 či skryje

text s nápisem „Hello World!“. Na tomto krátkém příkladu je znázorněno využití prvků JavaFX popsaných výše. Nejprne 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 framewrork JavaFX. Následně se inicializují všechny kontrolní prvky (*controls component*), tlačítko (*Button*) a text (*Label*). Poté je inicializován prvek umístění (*layout component*) *VBox* root (Vertical Box; prvky jsou umisťovány 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 veřejné metody pro ovládání komponenty a bude dědit od existující JavaFX komponenty typu *layout*, aby bylo možné do ní umístit další prvky. Dědění od komponenty typu *controls* se nehodí, protože ty slouží k poskytování konkrétní funkcionality.

```java
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

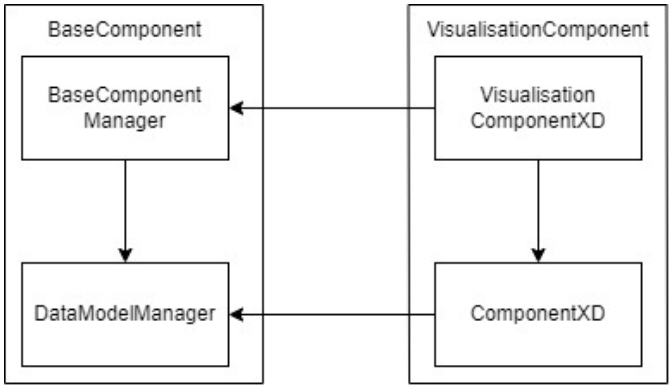
4.4.2 Navržená architektura

V této části je rozebrána navržená architektura komponenty. Jak bylo zmíněno v předchozí části, 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 části: *Model* - obsahuje data a logiku aplikace, *View* - zobrazení uživateli, tato část bude provádět samotnou vizualizaci a *Controller* - ovladače pro uživatele, přímo manipuluje s modelem, zde bude reprezentován hlavní třídou komponenty. Tato hlavní třída bude řídí operace mezi datovým modelem a vizualizací. Například datovému modelu zadá požadavek na načtení nových dat a až tuto operaci dokončí, předá hlavní třída 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 části *Controller*.

Část komponenty musí být navržena tak, aby byla nezávislá na zvolené vizualizační metodě. To je výrazné usnadnění pro vývoj, protože nebude potřeba společný 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 společná část bude nazývána jako **BaseComponent**, tedy základní část komponenty. Společné budou primárně operace s daty, proto do této společné části bude zařazen datový model a z hlavní třídy operace pracující s daty. Balíček **VisualisationModel** pak bude kompletně vyměnitelný podle vizualizační metody. Při rozdělení komponenty na dvě části bude potřeba i rozdělit hlavní třídu 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 dědit od původní hlavní třídy (v té zůstanou operace *BaseComponent*) a bude obsahovat funkcionalitu pro vizualizaci. Tato sekundární třída bude tou třídou, jejíž instanci bude uživatel přidávat do programu. Vizualizační část komponenty se bude nazývat **VisualisationComponent2D** pro 2D metodu a **VisualisationComponent3D** pro 3D metodu. Vizualizační operace a sekundární hlavní třída spadají do části **VisualisationComponent2D** nebo **Visuali-**

sationComponent3D. Tato část je závislá na zvolené vizualizační metodě.

Diagram architektury komponenty se nachází na obrázku 4.6. Diagramy datového modelu a vizualizací se nachází níže v příslušných částech. Třída `BaseComponentManager` je hlavní třídou části komponenty *BaseComponent*, třída `DataModelManager` je hlavní třídou datového modelu.
`VisualisationComponentXD` (za x se doplní 2 nebo 3 podle toho, zda se jedná konkrétně o 2D nebo 3D metodu) bude hlavní třídou vizualizační části a zároveň hlavní třídou celé komponenty. `ComponentXD` pak bude hlavní třídou samotné vizualizace, bude poskytovat a řidit veškeré vizualizační operace.



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

4.5 Komunikace mezi komponentou a aplikací, která ji využívá

V této části je rozebrán způsob, jakým budou data importována či exportována z komponenty a jak budou upravovány její parametry. Jedná se o vstupní data pro vizualizaci, parametry detektoru, jehož data budou vizualizována (rozměry vstupních dat, konstanty pro výpočet časových známek v datech) a parametry vizualizace. K určení způsobů importu a exportu je potřeba vzít v potaz navrženou architekturu v části 4.4.2. Podle navržené architektury bude komponenta navržena rozdělena na dvě části, *BaseComponent* a *VisualisationComponentxD*. Z tohoto rozdělení lze určit i způsoby nahrávání dat a informací. Je potřeba vzít i v potaz, jaký způsob importu a exportu by byl uživateli příjemný. Například bude vhodné, aby úpravy parametrů vizualizace byly viditelné ihned.

Část *BaseComponent* bude stejná pro jakoukoliv vizualizační metodu. Způsob nastavení dat a parametrů by neměl být nijak závislý 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 či zapnutí logování. Vstupní data jsou zadána v textovém souboru, který bude předán do komponenty z vnějšku přes aplikaci, využívající komponentu. Ta ho předá obslužným operacím v datovém modelu, které spustí načítání. 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 z vnějšku, protože jsou klíčové pro fungování komponenty a navíc vždy musí obsahovat nějakou hodnotu. Lze je udržovat separátně v nějakém konfiguračním souboru a načíst vždy když o ně bude vyžádáno, ale to by bylo uživatelsky nepřívětivé. Takže budou udržovány přímo v komponentě v jedné třídě určené k držení těchto hodnot. Aby byla aktualizace univerzální pro obě komponenty, bude prováděna přes samostatné zobrazitelné okno, které uživatel zobrazí a použije v případě potřeby aktualizace těchto hodnot. V rámci tohoto okna bude rovnou zajištěna validace předaných hodnot.

U parametrů části *VisualisationComponentxD* je potřeba zvolit jiný způsob, protože vizualizační metoda bude vyměnitelná a navíc je potřeba měnit její parametry tak, aby se ve vizualizaci projevily ihned a uživatel na první pohled poznal rozdíl. Možnost přes samostatně okno je také možná, ale nemusí být pro uživatele tak pohodlná. Místo toho lze prvky pro aktualizaci vystrčit mimo komponentu, aby je obstarávala aplikace a hlavní třída pouze poskytne metody pro nahrání nových hodnot. Nové hodnoty půjdou v okně aplikace 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 propsali 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 ně lze umístit *listener*, který bude detekovat změny hodnot v `Property`, případně lze na ně nasadit *binding*. Jednotlivé hodnoty budou

odděleny specifickým oddělovačem a identifikovány specifickými kódy a aplikace sama je převede do podoby čitelnější pro uživatele.

4.6 Datový model

V této části je popsán návrh datového modelu, způsobu uložení dat a datových operací.

4.6.1 Uložení dat

V této části je rozebrán návrh způsobu uložení dat z částicových detektorů určených pro vizualizaci. Jako vstupní formát dat bude uvažován pouze *data driven*, protože data budou vynášena v 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 časová známka a třetí naměřená energie v pixelu. Časová známka může být stejná pro více pixelů, protože v jednom čase může být naměřen 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 konkretní časovou známku - reprezentovat jako bin
- Biny budou vzestupně seřazeny vzestupně podle časových známek
- K binům bude potřeba přistupovat podle indexu (pro předaný index bude vrácen bin, který je na daném indexu umístěn) a podle časové známky přidělené 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 odpovídali struktuře rastrového obrázku. To ale bude zbytečně paměťově náročné, protože bude potřeba alokovat paměť i pro prvky, pro které nebyla zaznamenána energie. Lepší bude zvolit strukturu, která uloží pouze hodnoty se zaznamenanou energií. Pro event lze vytvořit třídu určenou jako nositel jeho hodnot, množinu eventů pro bin lze uložit jako seznam těchto tříd. Nepředpokládá se úprava tohoto seznamu po načtení 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 paměťově náročné datové typy. Z toho důvodu jsou dále popsány jednotlivé informace, jaké je potřeba ukládat a jaké pro ně byly zvoleny datové typy. Pro určení vhodného datového typu byla použita tabulka 4.1, která popisuje velikost, minimální a maximální hodnoty celočíselných datových typů v Javě.

1. **Souřadnice pixelu v obrázku** – ve formátu data driven jsou souřadnice uloženy v jedné hodnotě a hodnoty souřadnic X a Y je potřeba dopočítat. Kvůli přehlednosti a snazšímu používání budou v datovém modelu uloženy souřadnice X a Y samostatně jako dvě proměnné. Jedná se o celočíselné proměnné, maximální hodnoty jsou závislé na delkách 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 nevejde, ale do typu short ano a zbyde ještě hodně prostoru pro případně 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 paměťově náročné, ale rozdělení do dvou proměnných umožní následně snadnější práci.

2. **Time over Threshold (ToT)** – naměřené energie v eventu, poslouží k určení barvy pixelu nebo krychle při vizualizaci. Hodnota bude převzata rovnou ze vstupních dat a nebude se dále nijak přepočítávat. Až následně vizualizační a barevný model na základě této hodnoty určí barvu, kterou prvku přidělí. Hodnota může nabývat i desítek tisíc, u typu short může hrozit přetečení, u typu int je dostatečný buffer pro rozšíření. Byl tedy zvolen datový typ int.

3. **Time of Arrival (ToA)** – časová známka, očekávány 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 bytů	Min: -9,223,372,036,854,775,808 Max: 9,223,372,036,854,775,807

Tabulka 4.1: Celočíselné datové typy v Javě, 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 částicové detektory typu Medipix.[3]

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

Bin lze reprezentovat třídou, jejími parametry bude *ToA* a pole s eventy daného binu. K binům 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 nepředpokládá, že po načtení budou biny přidávány či odebírány, 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 souřadnicích a hodnotě *ToT*. Pro bin bude třída pojmenována jako BinData a uchovává pole událostí binu a dvě časové známky. Dvě jsou

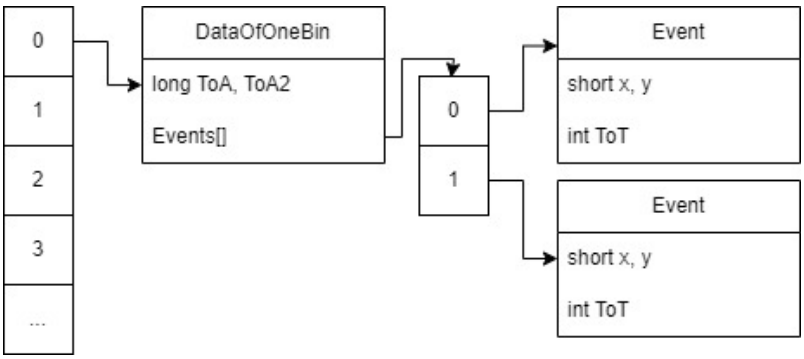
```
// class representing event
class Event{
    private short x, y;
    private int ToT;
}

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

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

Ukázka kódu 3: Struktur pro uložení dat určených k vizualizaci

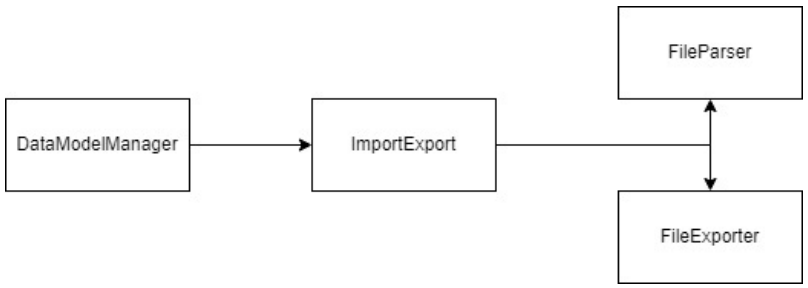
z důvodu operace *grouping*, popsané v části 4.6.3. Při ní dochází ke sloučení několika binů do jednoho, je pak uchovávána časová známka nejnižšího a nejvyššího binu a jsou z nich počítány souřadnice pro osu Z ve 3D metodě. Pokud operace grouping není na datech provedena, hodnoty obou časových známek jsou shodné. Znázornění se nachází na ukázce kódu 3 a na obrázku 4.7. Při použití takové struktury pro uložení dat lze každý event jednoznačně 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 části je popsán návrh architektury datového modelu. Datové operace, které bude potřeba obsloužit jsou: uchovávání a poskytování vstupních dat, uchovávání dat potřebných k chodu komponenty, import vstupních dat, export uložených dat. Datový model bude mít hlavní třídu zvanou `DataModelManager`, jejím úkolem je řídit a distribuovat datové operace. Zároveň v ní budou uchovávána načtená data a vizualizace bude volat veřejné metody této třídy pro jejich získání. Datové operace budou distribuovány do více tříd v rámci balíčku datového modelu. Dále bude existovat třída pro řízení a provedení veškerých operací importu a exportu, bude nazvána `ImportExport`. Načítání ze souboru bude provedeno přes třídu `FileParser` (popsáno je v části 4.6.3), export do souboru přes třídu `FileExporter`.



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

4.6.3 Načítání ze souboru a operace grouping

V této části je popsán návrh implementace načítání dat ze souboru a operace *grouping*. *Grouping* je operace, kdy jsou data z daného počtu binů sloučeny do jednoho binu. Ve vizualizaci to umožní uživateli prohlédnout větší množinu dat najednou. Počet binů, které mají být sloučeny budou volitelné uživatelem. Při sloučení budou k výslednému binu přiřazeny dvě časové známky, nejnižší a nejvyšší z originálních binů a to z toho důvodu, že na základě časových známek budou totiž binům přiřazovány souřadnice osy Z. Kdyby byla přidělena pouze jedna hodnota (nejnižší, průměr, apod.), při operaci *grouping* by se ztratily detailní informace o vzdálenostech mezi biny a souřadnice osy Z by pak mohly být nepřesné. Tím, že se použijí nejnižší a nejvyšší hodnota, stále půjde určit vzdálenosti mezi biny stejně dobře jako když operace provedena nebude. *Eventy*, které jsou ve sloučeném binu v kolizi (mají shodné souřadnice), budou sloučeny do jednoho, jehož hodnota ToT bude sečtena z jednotlivých eventů v kolizi. Tato operace bude řešena

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

Pro implementaci načítání ze souboru je jednoznačně definováno, jak má vypadat struktura, ve které mají být data z *parseru* předána. Pro snadné nahrazení metody pro načítání bude použito rozhraní, které bude definovat hlavičku metody, která bude pro načítání volána. Vstupním parametrem bude soubor, který má být načten a vracet bude strukturu s načtenými daty. Požadavky na datovou strukturu jsou podobné jako na samotný datový model. Hlavním požadavkem je, aby obsahovala každou časovou známku obsaženou v souboru a pro ni seznam událostí. Bylo by možné tedy zvolit stejnou strukturu jako pro data v datovém modelu, ale zaměření je na strukturu snadno použitelnou i pro samotné načítání. Datový model používá statická pole, ale ty jsou pro načítání ze souboru nevhodná, protože dopředu není známo, kolik binů či 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 některá z implementací struktury mapa (`Map<K,V>`) [19]. Jako klíč je použita časová známka a jako hodnota třída reprezentující bin, pro kterou seznam událostí bude implementován dynamickou strukturou, aby bylo možné prvky přidávat. Na výběr je z 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 ní lze seřadit klíče vzestupně či sestupně. V mapě budou klíče seřazeny vzestupně, aby část kódu, která bude provádět 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 javě 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 měnit jejich pořádí. Linked je naopak lepší, když se data budou modifikovat a měnit jejich pořádí. [17] Při načítání budou data do struktury pouze přidávána, nebude se měnit jejich pořádí. Úprava může následovat při operaci *grouping*, která bude provedena až po skončení

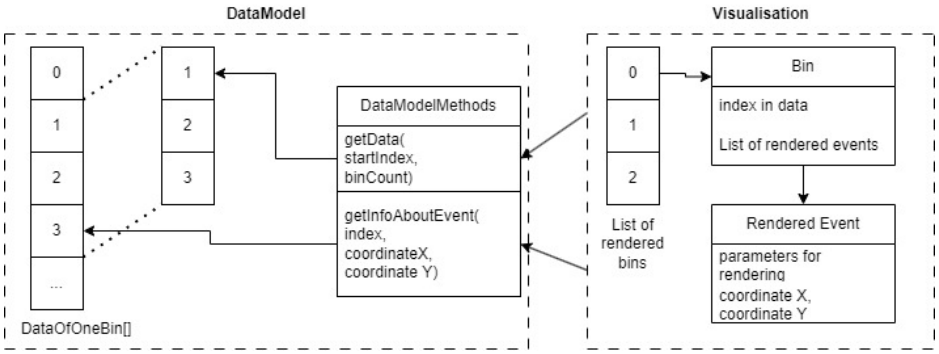
načítání, kdy budou data kompletní. Při ní budou seznamy eventů sloučeny do sebe a následně z nich budou odstraněny všechny eventy v kolizi a budou nahrazeny jedním sloučeným. Právě kvůli operaci *grouping* bylo rozhodnuto, že bude použit `LinkedList<E>`, při slučování bude potřeba méně režie než pro slučování polí `ArrayList<E>` a dojde k odstraňování prvků ze seznamu, u `ArrayList<E>` by pro to bylo potřeba 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 při potřebě vizualizovat nová data si je vyžádá od datového modelu přes definované metody. Je potřeba vyřešit dva hlavní problémy, zaprvé v jakém formátu a podle jakých parametrů budou data předávána vizualizaci a zadruhé jak ve vizualizaci data označit tak, aby mohla být zpětně identifikovatelna (například při kliknutí myší na vizualizovaný event bude potřeba zobrazit všechny dostupné informace o něm, takže se bude potřeba zeptat datového modelu). Pro snazší pochopení je finální propojení znázorněno na obrázku 4.9.

Pro určení formátu předávání dat vizualizaci je potřeba vzít v potaz, jakým způsobem vizualizace bude data potřebovat a jakým způsobem budou data uložena v datovém modelu. V sekci 4.6.1 bylo stanoveno, že data budou uložena do pole objektů reprezentující biny seřazeného vzestupně podle časových známek. K jednotlivým binům se bude přistupovat primárně podle indexů. Vizualizace (2D i 3D) bude vždy vyžadovat konkrétní část z celkových dat, které následně zobrazí. Bude znát tedy hranice tohoto okna, které chce zobrazit. Může 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 binů, které má vrátit. Aby byla poskytnutá data použitelná pro jakoukoliv vizualizační metodu, budou vracena ve stejném formátu, v jakém jsou uložena a vizualizace vezme pouze ta data, která bude potřebovat. Vraceno tedy bude pod-pole celkových dat o velikosti žádaného počtu binů počínající binem, který byl v originálním poli na daném indexu. Vrácena bude plná kopie původních dat, aby vizualizační model nemohl přepisovat data v datovém modelu. Dvě vizualizační metody zvolené pro tuto práci budou vyžadovat: souřadnice jednotlivých eventů (poslouží k určení souřadnic jednotlivých pixelů či voxelů) a indexy jednotlivých binů.

Tímto způsobem si vizualizace získá data, která chce aktuálně vykreslit. Je ještě potřeba určit způsob provázání v opačném směru, kdy bude chtít vizualizace zjistit informace o konkrétním pixelu či voxelu. Lze to vyřešit 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 paměťové náročnosti a hlavně k redundanci dat. Je tedy potřeba určit systém, jakým označit vykreslený event tak, aby ho datový model dokázal jednoznačně identifikovat. V 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 může v sobě mít může, protože budou použity k určení jeho polohy, můžou ale být přepočítány podle požadavků dané vizualizace. Originální souřadnice si tedy každý event ponechá, budou uloženy v datovém typu short, jako v původních datech. Při žádosti o více informací o konkrétním eventu tedy vizualizace předá datovému modelu index binu, ve kterém se Event nachází a jeho původní souřadnice.

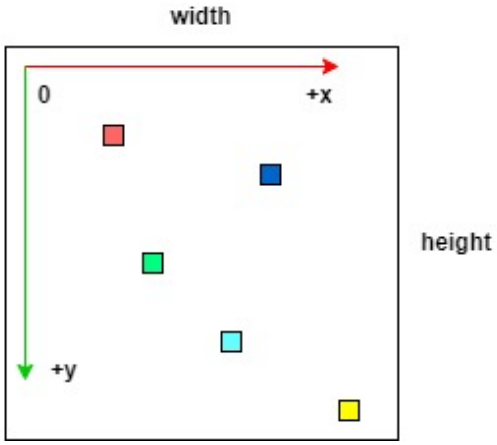


Obrázek 4.9: Navržené propojení datového modelu a vizualizace. Obrázek byl vytvořen v programu draw.io. [10]

4.7 2D metoda

V této části je rozepsán návrh implementace 2D metody. Typ 2D metody byl zvolen v části 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 ztratí informace o tom, z jakých časových známek data pocházejí. Proto bude potřeba

doimplementovat funkcionalitu, která tyto informace obstará jiným způsobem. Ta bude zahrnuta do interaktivity vizualizace pomocí myši, kdy na každý pixel půjde kliknout a zobrazit si o něm informace, včetně časových známek, ze kterých byl pixel složen. Tato interaktivita bude implementována odchytáváním událostí myši **MouseEvent** a detekci, zda uživatel kliknul na konkrétní pixel, informace budou následně získány z datového modelu metodou popsanou v části 4.6.4. Znázornění navržené 2D vizualizace je znázorněn na obrázku 4.10.

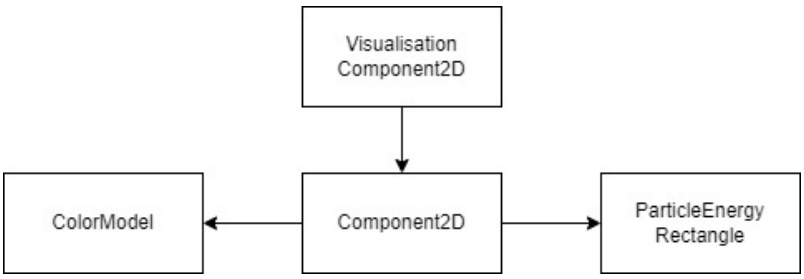


Obrázek 4.10: Navržené 2D vizualizace. Obrázek byl vytvořen v programu draw.io. [10]

Pro implementaci lze zvolit ze dvou základních možností, zmíněných v sekci 4.3.2, zda bude vykreslování provedeno přes třídu `Canvas` nebo pomocí 2D objektů JavaFX. *Canvas* dává programátorovi větší volnost s vykreslením, ale zároveň si musí více funkcí implementovat sám, jako interaktivitu pomocí myši. Při zachycení **MouseEvent** při stisku tlačítka myši je potřeba získat její souřadnice a spočítat, zda a na který čtverec reprezentující událost bylo kliknuto a následně ho zvýraznit. Ještě lze zvolit metodu s použitím tzv. image mapy, tedy neviditelné pole s interaktivními prvky nad pixely s energií. Při kliknutí na příslušné pole se spustí pole pro zvýraznění příslušeného pixelu. Vykreslení pomocí 2D objektů JavaFX (jako`Line, Rectangle` dědících od třídy `javafx.scene.shape.Shape`) je snazší na implementaci, včetně implementace interaktivity. Pro 2D objekty JavaFX lze přímo nastavit detekci kliknutí a následné akce. Ještě je možné použít komponentu `javafx.scene.image.ImageView`, která slouží k přímému zobrazování rastrových obrázků, k vykreslení se jí předá přímo instance `javafx.scene.image.Image`. Do okna se pak umístí jako běžná GUI

komponenta. Implementováno bude řešení s vykreslením pomocí JavaFX 2D objektů a to hlavně z důvodu jednodušší implementace interaktivity. U něj je potřeba zvolit, jak přesně mají být vykreslovány. Každý jednotlivý event bude reprezentován třídou `Rectangle`, kterému lze přiřadit barvu a detekci interaktivity. Tato implementace může mít nevýhodu náročnějšího výpočetního výkonu při větším množství vykreslených objektů, pokud bude eventů v jednom binu větší množství.

Na obrázku 4.11 se nachází návrh architektury 2D vizualizace. Na vrchu se nachází hlavní třída vizualizační části, `VisualisationComponent2D`, která v sobě drží instanci na třídu `Component2D`. To je hlavní třída vizualizace samotné, která provádí veškerou vizualizaci. Vykresluje jednotlivé eventy jako `ParticleEnergyRectangle`, který dědí od třídy JavaFX 2D `Rectangle` a navíc obsahuje parametry potřebné pro zpětnou identifikaci eventu. Třída `ColorModel` slouží k přidělování barev jednotlivým eventům na základě jejich hodnoty ToT.



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

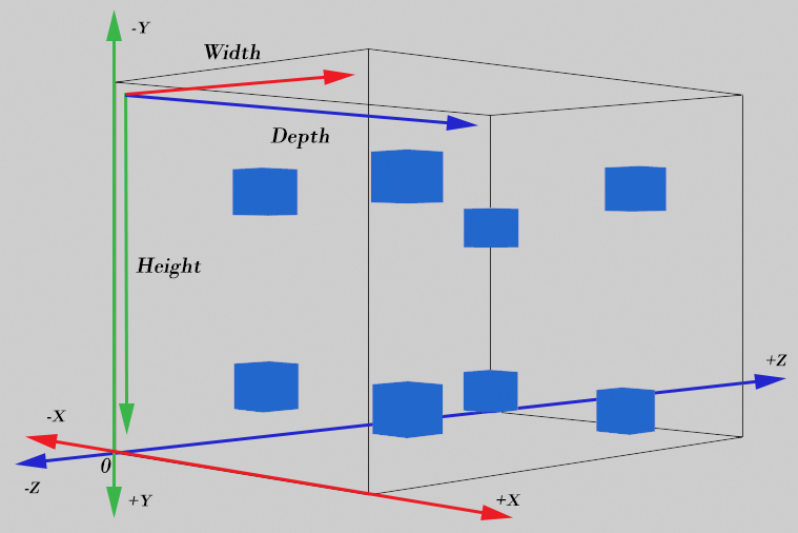
4.8 3D metoda

V této části je rozepsán návrh implementace 3D metody. V části 3.2.2 byla zvolena jako metoda, kdy bude použita tzv. voxelová grafika. Osa Z se použije jako časová osa, biny budou za sebe poskládány chronologicky a jednotlivé eventy budou transformovány na voxely. Na výběr jsou dvě základní možnosti implementace. První je využít předdefinovaných 3D objektů JavaFX jako je `Box` a umisťovat je přímo do scény na souřadnice scény. Veškeré transformace a vykreslení pak provede sám framework JavaFX. Druhá, složitější, je implementovat vlastní řešení. Souřadnice, objekty a transformace bude vypočítávat vlastní část a budou zobrazovány do scény například pomocí vykreslení do `Canvas`. První řešení umožňuje i jednodušší implementaci

interaktivity, protože lze nastavit detekci kliknutí přímo jednotlivým 3D objektům. Pro druhou variantu je to složitější, 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 vhodně zvoleném řešení může být lépe optimalizováno oproti vykreslování předefinovaných 3D objektů JavaFX. To je zaměřeno spíše na vykreslování objektů s podporou GUI. Zvolena bude první metoda, tedy použít definované 3D objekty JavaFX, hlavně z důvodu jednodušší implementace.

V JavaFX se vykreslení 3D objektů často 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 natočit. Pro separátní scénu se používá třída `SubScene`, kterou lze vložit do obyčejné `Scene`. 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 roustoucí směrem dolů. Stejný směr platí i pro souřadnice dat, počátek 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 dopočítat. 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 přidělena přímo `Group` reprezentující bin, takže se posunou i všechny krychle které `Group` obsahuje. Při vykreslování je potřeba vzít 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 převod 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 65 536 (256x256). Výpočetní výkon by pak ale mohl být zbytečně vytížen vykreslováním voxelů bez energie. Navíc by bylo potřeba vyřešit průhlednost jednotlivých voxelů, aby byly dobře viditelné i další biny. Dále je možnost vykreslovat pouze voxely se zaznamenanou energií. Takové řešení se zdá jako nejjednodušší a díky 3D prostoru možná i nejvýhodnější, protože nebude potřeba řešit problém s průhlednosti voxelů s nezaznamenanou energií. Zvoleno tedy bylo řešení vykreslovat pouze eventy se zaznamenanou energií.

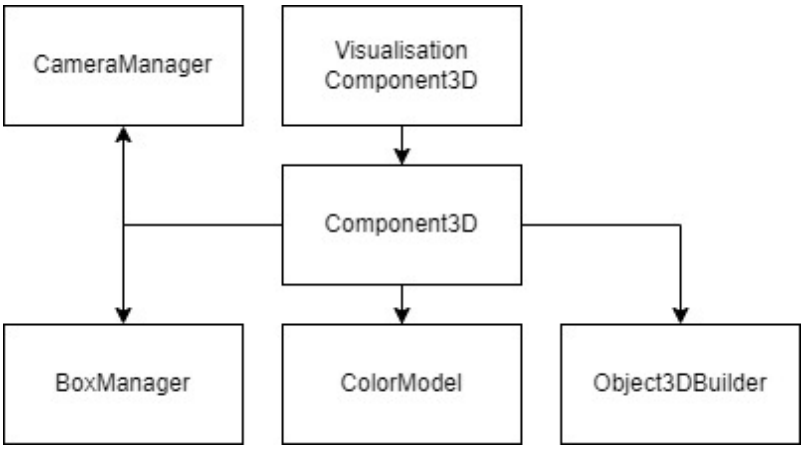


Obrázek 4.12: Navržené 3D vizualizace. Návrh byl vytvořen v programu blender. [4]

Architektura tříd 3D vizualizace se nachází na obrázku 4.13. Je rozdělena tak, aby každá třída obstarával nějakou z hlavních funkcionalit 3D vizualizace. Hlavní třída vizualizační části komponenty je `VisualisationComponent3D`, která v sobě drží instanci na `Component3D`. To je hlavní třída vizualizace samotné, bude v sobě držet data vizualizace a řídit a distribuovat vizualizační operace. Třída `BoxManager` se zaměří na převedení eventů do voxelů a jejich vykreslení, `ColorModel` bude reprezentovat barevný model přiřazující barvy jednotlivým voxelům a `Object3DBuilder` obstará vytvoření a manipulaci všech ostatních 3D objektů. `CameraManager` je manažer poskytující kameru a operace pro ni.

4.8.1 Výpočet souřadnic pro osu Z

Osa Z reprezentuje časovou osu, souřadnice se dají spočítat podle jednotlivých časů. Ke každému binu je přidělena časová známka, *ToA*. Na základě té lze vypočítat souřadnice. Nelze použít výchozí hodnoty *ToA* přímo jako souřadnice, protože hodnoty *ToA* jsou vysoké a často daleko od sebe. Dva vedle sebe ležící biny by pak mohly být v nedosažitelné vzdálenosti a uživatel by musel kameru dlouho posunovat. Navíc *ToA* prvního binu nebývá hodnota 0. Místo toho budou souřadnice přepočítány pomocí *ToA* jednotlivých binů a spočítané souřadnice budou přidělovány indexům těchto binů. Bylo by možné zvolit jednoduché řešení a všech binům přiřadit rovnou souřadnice 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 čase mezi jednotlivými biny a jak jsou od sebe vzdálené. Některé jevy mohou být rozprostřeny přes několik *ToA* a pozná se to právě tím, že tyto *ToA* budou soubě blízké. Souřadnice tedy musí zachovat vizuální vzdálenosti od sebe.

Souřadnice tedy budou spočítány podle následujícího algoritmu. Vezme se minimální rozdíl *ToA* mezi dvěma biny nalezený v aktuálně načtených 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 souřadnic osy Z. Pokud byla operace provedena, první *ToA* značí nejnižší čas ze sloučených binů druhá *ToA* nejvyšší. Vzdálenosti mezi dvěma biny jsou počítány jako rozdíly k sobě bližších souřadnic, 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 vydělena podle vzorce 4.1. Zároveň bude ještě zaveden parametr maximální povolené vzdálenosti mezi daty pro případ, kdyby i po přepočtu byla vzdálenost příliš vysoká a pro vizualizaci nepoužitelná. Pokud bude vypočítaná vzdálenost vyšší než definované maximum, bude použito právě ono maximum. Podle tohoto algoritmu budou spočítány vzdálenosti mezi jednotlivými biny. Následně je ještě potřeba přiřadit souřadnice konkrétním binům. Souřadnice

prvního binu bude stanovena jako 0. K následujícím binům se vždy přičte jejich vzdálenost od posledního binu.

$$final\_range = \frac{init\_range}{MIN\_range} \tag{4.1}$$

4.8.2 Kamera

Kamera se chová rozdílně pro 2D a 3D prvky GUI, takže pro **SubScene** s 3D vizualizací bude vytvořena nová samostatná kamera. Aplikace pak bude mít dvě kamery, jednu výchozí pro hlavní scénu obsahující komponentu a další 2D prvky a druhou vytvořenou čistě pro 3D prvky a umístěnou do *SubScene* komponenty. Kameru lze umístit do scény jako jakýkoliv jiný prvek JavaFX. Pro 3D kameru je potřeba nastavit několik parametrů rozdílných od 2D kamery, podrobný popis se nachází pod odkazem [22]. Na ukázce kódu níže jsou zobrazeny dva možné konstruktory pro kameru. Druhý konstruktor `PerspectiveCamera(boolean fixedEyeAtCameraZero)` byl představený v Java 8 a obsahuje navíc parametr `fixedEyeAtCameraZero`. Ten určuje, zda má kamera zůstat na stejné pozici, když dojde ke změně promítané oblasti nebo velikosti okna. Při nastavení na *true* bude čočka kamery umístěna na souřadnice (0, 0, 0) svého souřadnicového systému. Při výchozím nastavení *false* má kamera definovaný souřadnicový systém s definovaným počátkem v levém horním rohu scény jako pro 2D. Při změně velikosti okna dojde ke změně pozice kamery tak, aby zůstala v levém horním rohu okna. Takové nastavení se hodí pro 2D rozložení, ale ne pro 3D. U 3D naopak je potřeba, aby kamera zůstala na místě a zobrazovala prvky stále stejně. Pro 3D kameru je tedy potřeba nastavit tento parametr jako true.

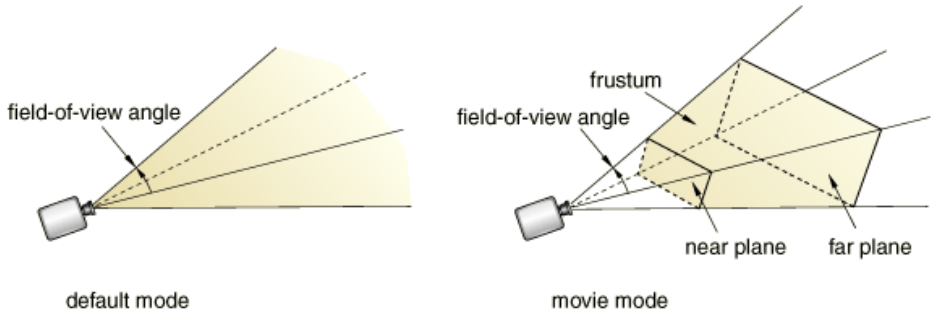
Dalším parametrem kamery je **Field of View** (česky zorné pole). Udává, jak velký úhel má kamera zabírat, úhel je udán ve stupních. Udává, jak velký úhel má kamera zabírat, úhel je udán ve stupních. Hodnota bude nastavena na 30°, která podle dokumentace třídy JavaFX *Camera* odpovídá módu *Telephoto*. Posledními důležitými parametry jsou **Near Clip** a **Far Clip**. Značí oříznutí pohledu tak, aby byl z něho zobrazen jen požadovaný objem. Znázornění těchto parametrů 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: Znázornění 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 věnuje 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 důležitých funkcionalit, v některých případech přímo tříd a zdrojového kódu. Popis funkcionalit je zaměřený na konkrétní popis algoritmů a fungování jednotlivých procesů, podrobný popis jednotlivých funkcí je pak obstarán prostřednictvím dokumentačních komentářů ve zdrojových souborech komponent a aplikací. Třídy jsou popsány slovně i znázorněny pomocí UML (Unified Modeling Language) diagramů. [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 dvěma způsoby, podle typu parametru. Parametry části *BaseComponent* lze upravovat přes samostatně zobrazitelné okno nazvané „Component Settings“. Jsou to parametry nezávislé na vizualizační metodě. Parametry vizualizační metody pak lze upravovat z vnějšku komponenty, přes prvky aplikace. Změny ve vizualizaci se projeví ihned.

- Data a parametry o načtených datech a vizualizaci jsou ukládány do přes JavaFX `StringProperty` v definovaném formátu. Těchto Properties je několik, každá pro jiný typ dat. Lze je z komponenty exportovat a přidat do příslušného GUI pole aplikace.

- Komponenta dědí od třídy JavaFX `Group`, lze umístit do scény jako běžnou JavaFX komponentu.

- Vizualizace v komponentách se ovládá pomocí myši a klávesnice (konkrétní klávesy jsou definovány v uživatelské příručce), lze posouvat vizualizované okno a v případě 3D komponenty lze klávesami ovládat i kameru a posouvat ji ve vizualizaci. Myší lze zvýraznit konkrétní vykreslený event a u 3D vizualizace lze ovládat i kameru.

- Při vývoji komponent a aplikací byl použit nástroj *Maven*.

5.1 Architektura

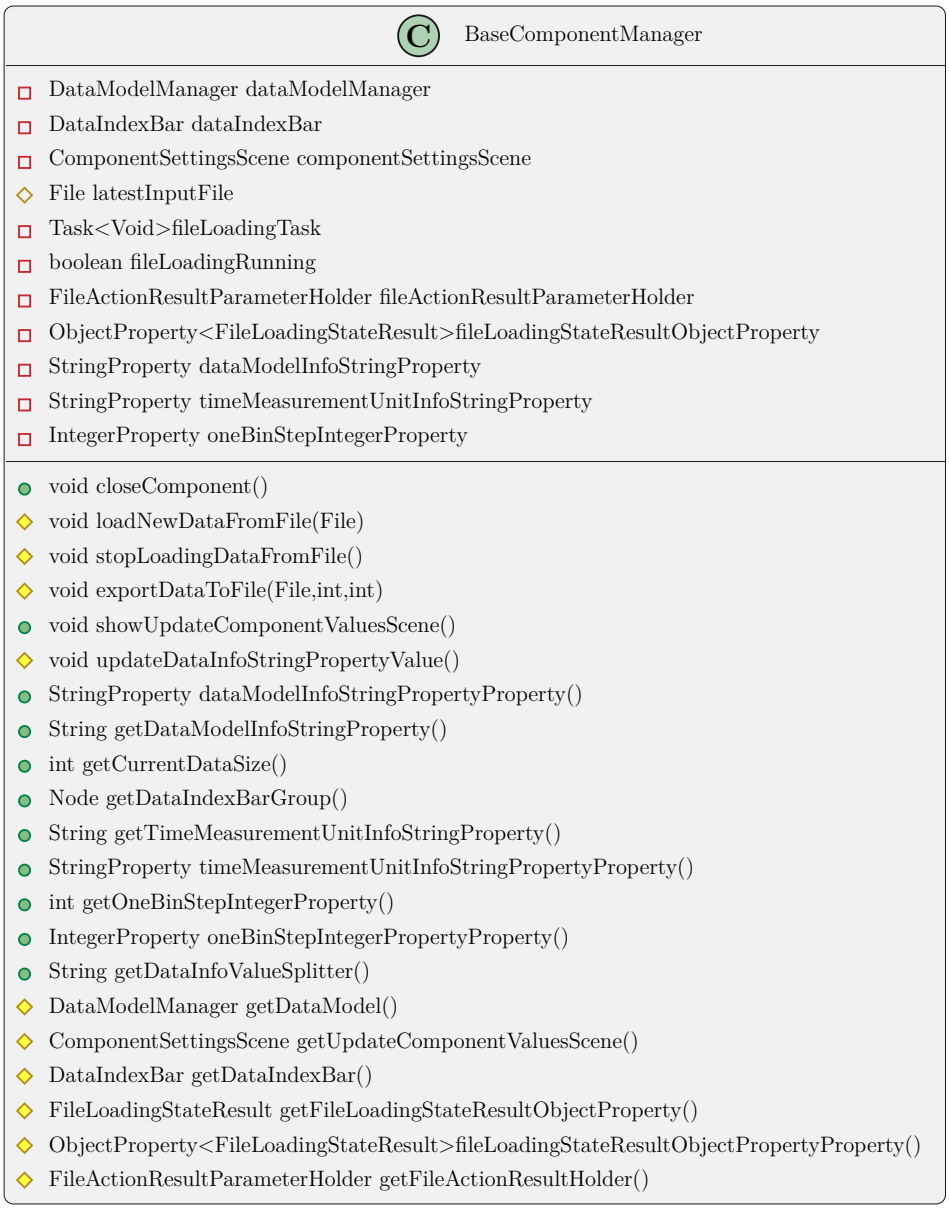
Tato část je zaměřena na popis architektury vyhotovené komponenty.

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 hlavně datové operace, a **VisualisationComponent**, poskytuje hlavně vizualizační operace. *BaseComponent* je společná pro obě komponenty a pokrývá veškerou funkcionalitu nezávislou na vizualizační metodě. 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 vizualizační metodu, konkrétně: uložení dat a datové operace jako načítání ze souboru (část 5.2), uchování dalších důležitých 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, proto je implementován v *BaseComponent*) a několik JavaFX `Property` udávající informace o načtených datech, zadaných parametrech komponenty a aktuální index počátku vizualizovaného indexu. Část *VisualisationComponent* je implementovaná specifiky pro každou metodu, jejich struktury jsou si podobné, aby byly jednodušší na porozumění. Obecně obě poskytují získání *Node* obsahující vizualizaci, veřejné 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 dědí od třídy JavaFX `Group`. Hlavní třída části **VisualisationComponent** má název pro 2D komponentu určen jako `VisualisationComponent2D` a pro 3D komponentu `VisualisationComponent3D`. Instanci třídy `VisualisationComponent2D` nebo `VisualisationComponent3D` umístí programátor jako grafický prvek *Node* do scény či nějaké *layout* komponenty JavaFX. V demonstrační aplikaci je umístěna 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 metod je zde zmíněn konstruktor `BaseComponentManager()` se třemi parametry: `boolean calculateZAxis` - indikátor, zda mají být v datovém modelu spočítány souřadnice pro osu Z. Nastavení je závislé na konkrétní vizualizační metodě, pro 3D je zapnuté, pro 2D vypnuté. Druhý parametr `boolean turnOnLogging` - indikátor, zda má být zapnuto logování do konzole pro všechny operace komponenty. Komponenta navíc loguje ve vlastním

formátu. Třetí parametr `boolean initializeDataIndexBar` - indikátor, 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 důležitou metodou této části je `void loadNewDataFromFile(File inputFile)`, který slouží ke spuštění načítání ze souboru. V této metodě se pro to vytvoří i samostatné vlákno, které je uloženo do parametru *Task<Void>fileLoadingTask*. Po dokončení načítání je výsledek uložen do příslušné struktury a hlavní třída vizualizace pak zahájí následné operace. Je povoleno pouze jedno aktivní vlákno, pokud uživatel zažádá o další načítání, je jeho požadavek odmítnut.

Hlavní třídy `VisualisationComponent2D` a `VisualisationComponent3D` obsahují obě dva konstruktory, jeden s přednastavenými parametry komponenty a druhý dává programátorovi možnost nastavit si všechny parametry sám (detailní popis se nachází v uživatelské příručce, která je součástí příloh práce). Dále obsahuje všechny metody obstarávající funkcionalitu komponenty: načtení dat, úprava parametrů a získání *DataIndexBaru* a `Properties` obsahující informace a parametry komponenty.

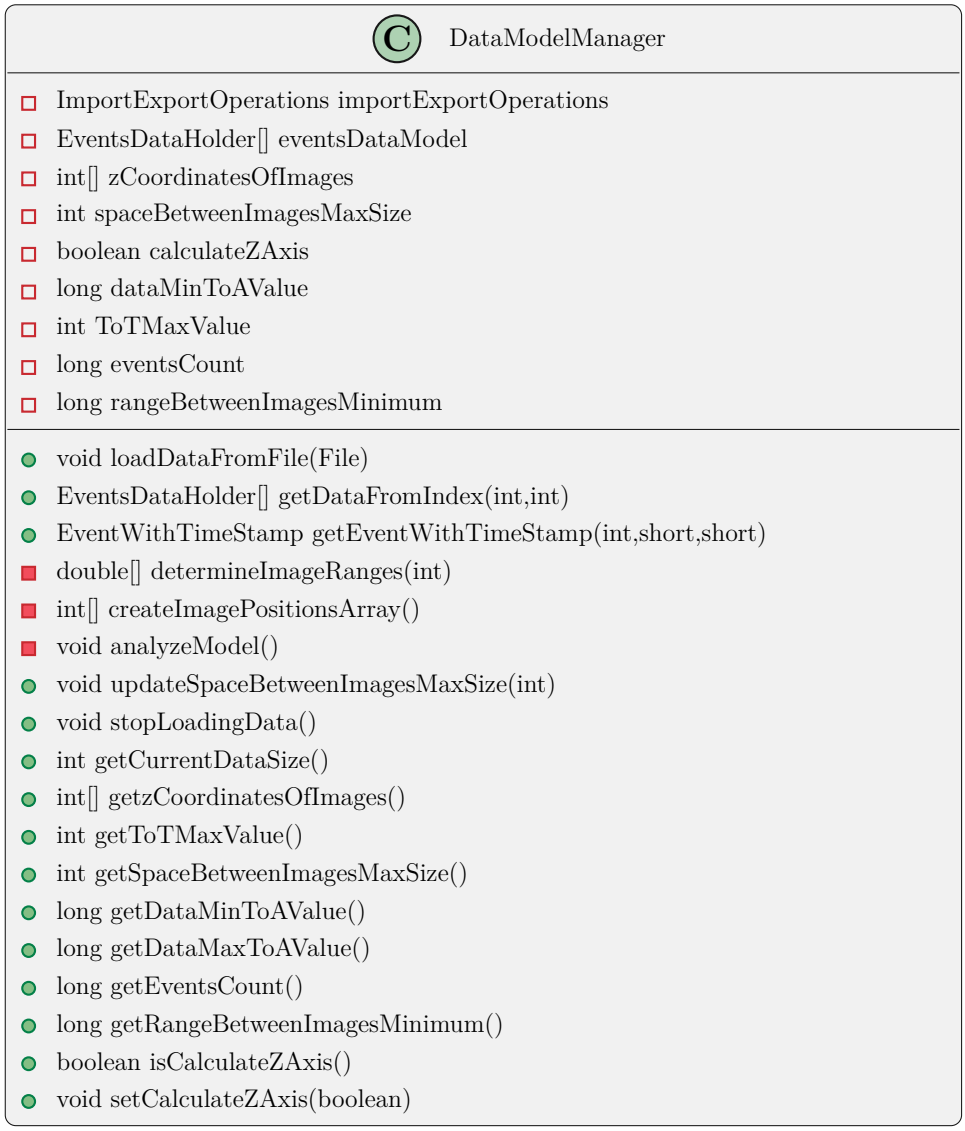
5.2 Datový model

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 třídou datového modelu je třída `DataModelManager`, znázorněná na obrázku 5.2. Tato třída v sobě uchovává uložená data pro vizualizaci a slouží jako zprostředkovatel datových operací. Operace buď sama provádí 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 o načtených datech, poskytnutí dat podle počátečního indexu a požadovaného počtu binů, poskytnutí konkrétní události podle indexu a souřadnic X a Y (tato trojice parametrů jsou minimální nutné parametry k jednoznačné identifikaci konkrétní události), určení indexu binu podle předané časové známky a metody pro aktualizaci některých parametrů datového modelu. Při získání dat podle indexu nejsou vráceny reference do struktur s uloženými daty, ale jsou vytvo-

řeny jejich kopie. Je tak učiněno z bezpečnostních důvodů, aby vizualizační metody nemohly modifikovat uložená data.

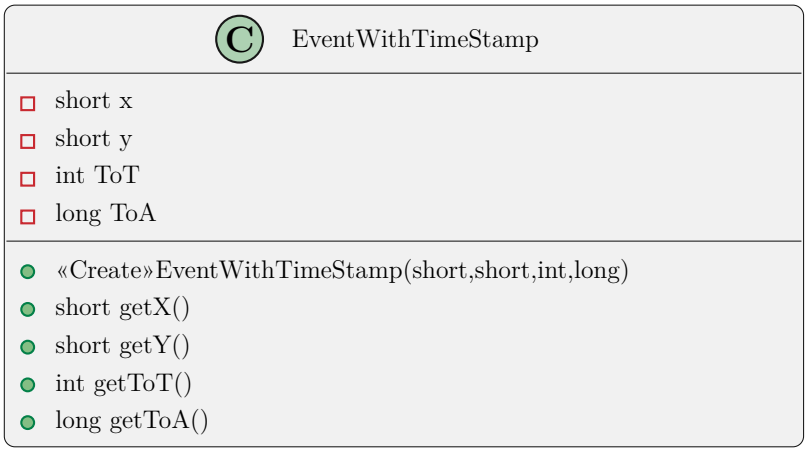


Obrázek 5.2: UML diagram třídy DataModelManager

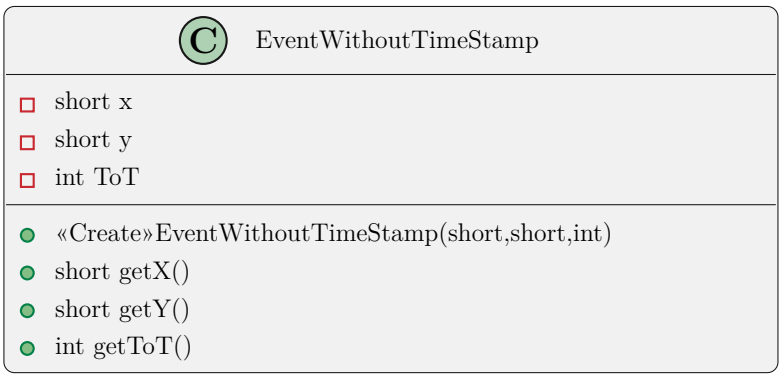
5.2.2 Uložení dat

V této části je popsán způsob uložení dat určených k vizualizaci, které jsou uchovávány ve třídě `DataModelManager`. K jejich uložení se používají třídy, které jsou uložené v balíčku `BaseComponent.DataOperations.EventRepresentations`:

`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 přiřadit ke konkrétnímu eventu i časovou známku, *ToA*. `BinData` slouží jako reprezentant jednoho binu, celkové uložení dat je reprezentováno jako pole tříd `BinData` seřazených vzestupně podle *ToA* jednotlivých binů.



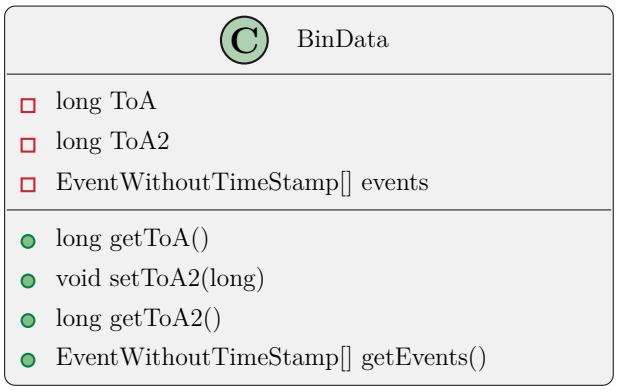
Obrázek 5.3: UML diagram třídy EventWithTimeStamp



Obrázek 5.4: UML diagram třídy EventWithoutTimeStamp

5.2.3 Načítání dat ze souboru

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

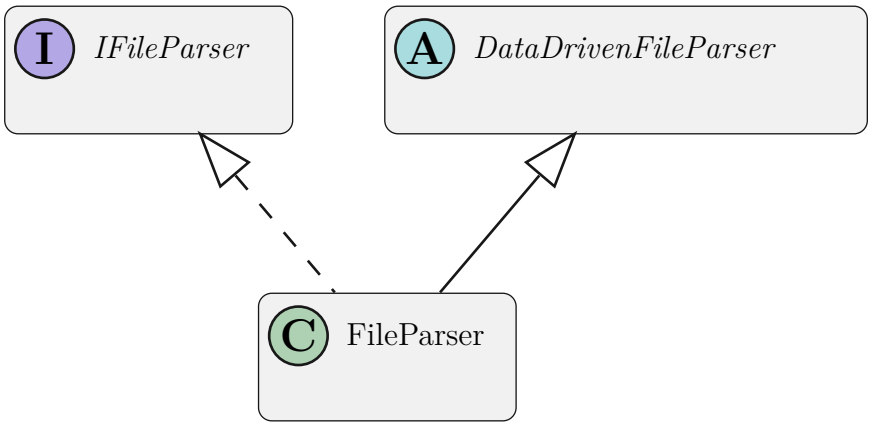


Obrázek 5.5: UML diagram třídy BinData

`DataOperations`. Třída `FileParser` implementuje rozhraní `IFileParser` a dědí od abstraktní třídy `DataDrivenFileParser`.
Instance rozhraní `IFileParser` je umístěna jako parametr třídy`ImportExport`. Rozhraní je zavedeno z toho důvodu, aby byly různé metody načítání dat snadno zaměnitelné. Rozhraní definuje metodu pro načítání dat
`loadDataDrivenDataFromFile(File)`, která vždy musí vracet načtená 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 onen bin. Dále rozhraní ještě definuje *getter* a *setter* parametru, zda má být přerušeno načítání dat (při požadavku o přerušení od uživatele je zavolán právě *setter*, který nastaví hodnotu na *true*).
`DataDrivenFileParser` obsahuje pouze jednu metodu určenou ke konverzi řádky s eventem ze vstupního souboru do třídy `EventWithTimeStamp`. Tato třída je abstraktní, protože její funkce je použitelná pro jakoukoliv implementovanou metodu načítání ze souboru. Po skončení načítání 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 několika binů do jednoho. Operaci je implementována ve třídě `GroupingTool`, je volitelná a uživatel si může zvolit, z kolika snímků budou data slučována. Pokud zvolí hodnotu *1*, *grouping* neproběhne. Finálnímu binu jsou přiděleny dvě časové známky, nejnižší a nejvyšší ze sloučených binů. Jsou použity dvě z důvodu následného určování souřadnice Z ve 3D vizualizaci. Při konfliktu eventů (eventy ze slučovaných binů mají shodné



Obrázek 5.6: UML diagram tříd file parseru

souřadnice) dojde k jejich sloučení do jednoho eventu, výsledná hodnota ToT je součtem ToT všech jednotlivých událostí. Dochází k ní okamžitě po dokončení načítání dat ze souboru, aby bylo zajištěno, že jsou všechna načtená 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 `originalMap`. 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 či zda bude vytvořen nový. Po dokončení této iterace jsou v nové mapě uloženy nové biny, ještě je potřeba provést sloučení událostí v konfliktu. Opět je provedena iterace nad daty a všechny eventy v binech 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 originalMap
        // 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: Znázornění algoritmu operace *grouping*

5.2.5 Důležité hodnoty komponenty

V balíčku `DataOperations` se nachází třída `ComponentParameters` obsahující důležité hodnoty komponenty. Jedná se o hodnoty nezávislé na vizualizační metodě, například parametry detektoru, ze kterého byla data získána nebo z kolik snímků má být slučováno při operaci *grouping*. Tyto hodnoty jsou uloženy jako veřejné statické proměnné a je dovoleno je měnit pouze skrze okno vytvořené samotnou komponentou. Jedná se o okno definované ve třídě `ComponentSettingsScene`, popsané v části 5.3. Konkrétně se jedná o sířku a výšku vstupních dat (imageWidth, imageHeight), konstanty sloužící k výpočtu *ToA* při načítání (ToAConstant1, ToAConstant2), řetězec značící řádkový komentář ve vstupním souboru (lineCommentMark), počet udávající kolik binů má být sloučeno při operaci *grouping* (binCount) a oddělovač hodnot ve `StringProperties` pro uložení informací, které si může 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 číselných proměnných musí být kladná, řetězec lineCommentMark nesmí být prázdný. Úpravy lze provést kdykoliv, nicméně většina funkcí pracujících s těmito hodnotami je vytvořena tak, aby se změny neprojevily hned, ale třeba až pro další datovou sadu. Proto je ideální hodnoty měnit 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 = "|";
}
```

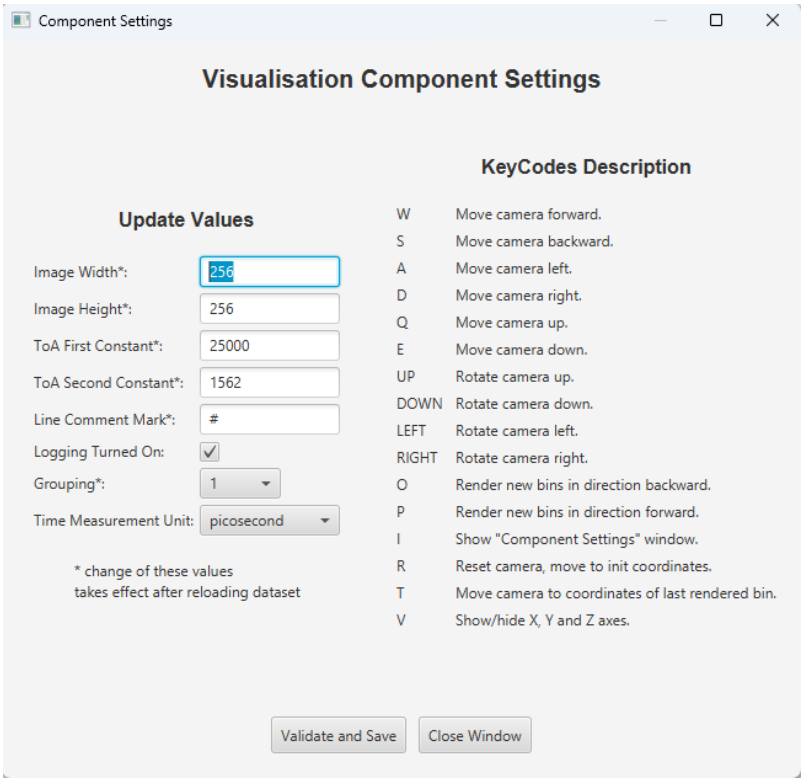
Ukázka kódu 6: Uložení parametrů komponenty

5.3 Okno Settings

V této části je popsána implementace okna pro aktualizaci parametrů části *BaseComponent*. Konkrétně se jedná o tzv. důležité hodnoty komponenty (popsané v části 5.2.5), zapnutí logování a nastavení jednotky času 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 balíčku *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 zablokované. Layout komponenty je dle třídy `BorderPane`. Okno je ukázáno na obrázku 5.7.

5.4 2D Vizualizace

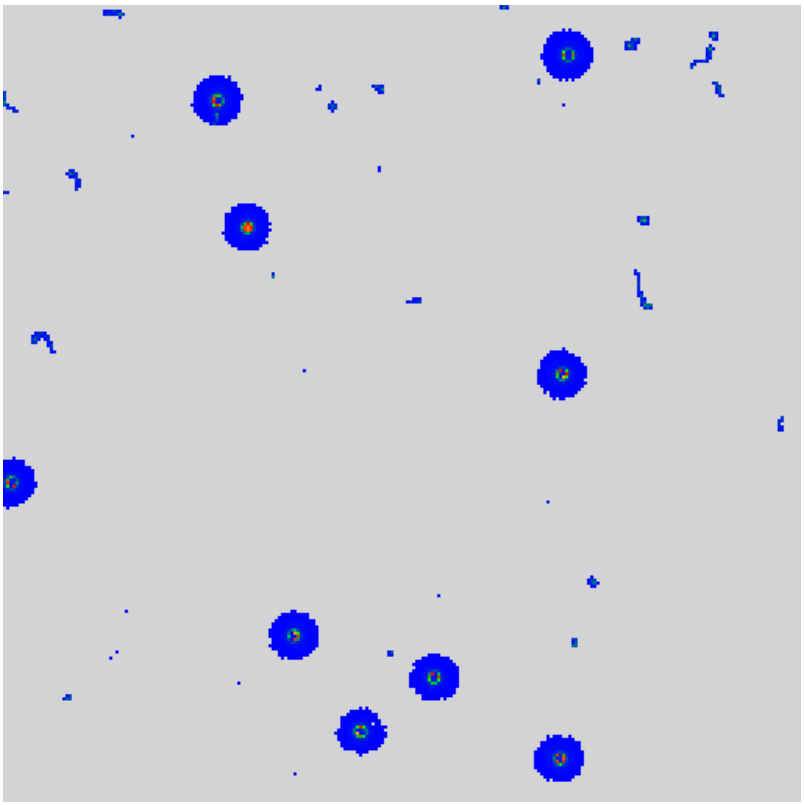
V této části je popsána implementace 2D vizualizační metody. Podle návrhu jsou slučovány biny z rozpětí daného uživatelem do jednoho binu a tento výsledný je vykreslen jako 2D obrázek. Implementace je vytvořena v části komponenty `VisualisationComponent` a ukázka vizualizace na obrázku 5.8. Nejdříve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



Obrázek 5.7: Okno s nastavením parametrů komponenty

Vizualizace je založena na tom, že data z jednotlivých binů jsou sloučena do jednoho a ten je vykreslen jako 2D obrázek. Rozpětí, kolik binů bude zobrazeno, určuje uživatel. Klávesami O a P může posouvat s vizualizovaným oknem dopředu a dozadu. Rychlost vykreslování dalších binů lze nastavit a lze vykreslit data na konkrétním indexu nebo časové známce. Jednotlivé eventy jsou vykresleny jako čtverce, barva je jim přidělena barevným modelem, vychází z hodnoty ToT příslušného eventu. Na každý čtverec lze kliknout a ten se následně zvýrazní (jeho barva je změněna na růžovou). Informace o vybraném eventu jsou uloženy do příslušné `StringProperty` pomocí které aplikace do příslušného `TextArea` tyto informace vypisuje.

Hlavní třídou 2D vizualizační části je třída `VisualisationComponent2D`, přes tuto třídu jsou prováděny všechny akce k vizualizaci. Samotná vizualizace je pak prováděna přes jednu třídu, `Component2D`. Jednotlivé čtverce reprezentující události jsou vymodelovány pomocí třídy `ParticleEnergyRectangle`, která dědí od JavaFX 2D třídy `Rectangle` a obsahuje navíc původní X a Y souřadnice eventu.



Obrázek 5.8: Znázornění 2D vizualizace

Nejprve je vykreslen obyčejný `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 biny sloučit a sečíst hodnoty ToT eventů se shodnými souřadnicemi. Toho je dosaženo tak, že získané eventy jsou postupně umísťovány do matice (odpovídá rastrovému obrázku), souřadnice matice jsou shodné se souřadnicemi eventů. Pokud už na nějaké pozici nějaký event umístěný je, je k němu přičtena 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 `ParticleEnergyRectangle`. 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

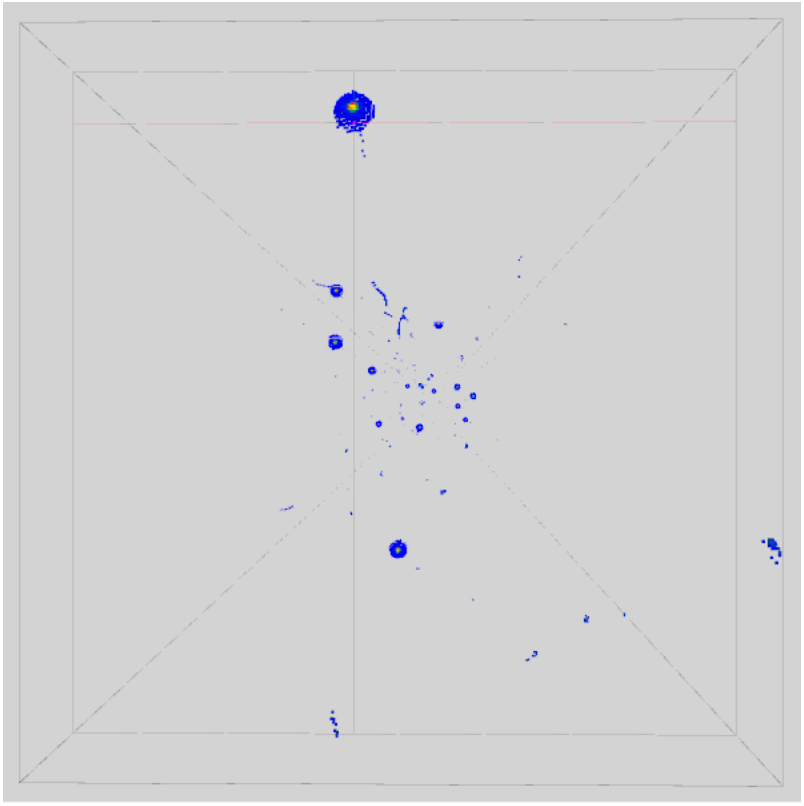
V této části je popsána implementace 3D vizualizační metody. Podle návrhu byla použita tzv. voxelová grafika, kdy eventy v datech jsou vymode-

lovány jako krychle(voxely). Implementace je provedena v části komponenty *VisualisationComponent* a ukázka vizualizace na obrázku 5.9. Nejdříve je popsána vizualizace z funkcionálního hlediska a následně z programového.

Vizualizace je založena na tom, že data z jednotlivých binů jsou poskládána chronologicky za sebou. Hranice snímků jsou vyznačeny obalovým kvádrem, jeho šířka a výška odpovídá šířce a výšce vstupních dat. Data jsou vykreslována od počátku souřadnicového systému. Vykreslena je pouze část celkových dat, velikost vykreslené části je definována uživatelem a lze s oknem hýbat dopředu/dozadu klávesami. Hloubka obalového kvádru je nastavena na souřadnici posledního vykresleného binu. Jednotlivé eventy jsou vykresleny jako voxely/krychle pomocí třídy JavaFX `Box`, jejich souřadnice odpovídají souřadnicím eventu. Krychle mají přidělenou barvu barevným modelem, vychází z hodnoty ToT příslušného eventu. Na každou krychli lze kliknout, ta se následně zvýrazní (její barva je změněna na růžovou) a v její pozici je zobrazen osový kříž, který zvýrazní její pozici. Informace o vybrané krychli jsou uloženy do příslušné `StringProperty` pomocí které aplikace do příslušného `TextArea` tyto informace vypisuje. Přes definované klávesy lze posouvat a otáčet kameru ve vizualizaci. Vizualizovat lze ve dvou módech, které se liší v tom, jak se bude model vizualizace chovat při posunu zobrazeného okna dopředu a dozadu.

- **STATIC** – zobrazené okno zůstane „na místě“. Souřadnice Z prvního binu bude vždy 0, poslední souřadnice se aktualizuje podle souřadnice posledního binu.
- **MOVABLE** – při posunu souřadnice Z současných binů zůstanou stejné, nové biny se přidají na konec a ze začátku se uberou. Při posunu dat je pak pak potřeba i posunout kameru ve směru tvoření dat.

Jednotlivé eventy jsou zobrazeny ve formě voxelů/krychlí. Jsou zobrazeny pouze eventy se zaznamenanou energií. Krychle je vytvarována pomocí JavaFX 3D objektu `Box`. Krychle v sobě nesou informaci o originální X a Y souřadnici, aby mohla být zpětně identifikovatelná v datovém modelu. Z toho důvodu je definován objekt `ParticleEnergyBox`, který dědí od třídy `Box` a navíc obsahuje jako dva parametry tyto souřadnice. Bin je reprezentován třídou `Group`, také kvůli zpětné identifikace musí v sobě nést číslo indexu v datech, takže je definován objekt `BinGroup`, který dědí od třídy `Group` a má jako parametr právě číslo indexu. Parametry v `ParticleEnergyBox` a `ImageGroup` jsou *final*. Do seznamu prvků `Children` `BinGroup` je povoleno



Obrázek 5.9: Znázornění 3D vizualizace

umisťovat pouze objekty `ParticleEnergyBox`. Souřadnice Z jsou přiřazeny pouze `BinGroup`, takže se na danou souřadnici přesunou všechny souřadnice v binu. Na ukázce kódu 7 se nachází ukázka metody `buildParticleEnergyBox(EventWithoutTimeStamp eventToBuild)` k sestavení jedné krychle `ParticleEnergyBox` podle předaného eventu. Na tomto příkladu je i dobře znázorněno, jak pracovat s objekty a souřadnicemi. Lokální proměnné `x`, `y` a `pixelDefaultColor` slouží k uložení souřadnic a hodnoty *ToT*. Souřadnice X a hodnota *ToT* se ponechá beze změny, souřadnice Y je potřeba dopočítat. Jak bylo znázorněno na obrázku 4.12, osa Y v JavaFX 3D scéně směřuje dolů. Data jsou vykreslována do horní části, ale osa Y ve zdrojových datech též směřuje dolů. Je tedy potřeba k hodnotě Y přičíst výšku vstupních dat. Tak se boxy v počátku souřadnic objeví v horním okraji obalového rámu, z hlediska dat stále budou mít souřadnici 0. Následně je vytovřena instance `ParticleEnergyBox`, jako parametry musí být zadány originální souřadnice X a Y a velikost hrany krychle. Velikost hrany je stanovena na hodnotu 1 pro všechny krychle, hodnota je neměnitelná.

```
ParticleEnergyBox buildParticleEnergyBox(
    EventWithoutTimeStamp eventToBuild){

    final int center = 0;
    int pixelDefaultColor;
    double x, y;

    pixelDefaultColor = eventToBuild.getToT();
    x = eventToBuild.getX();
    y = eventToBuild.getY()
        - ImportantComponentValues.imageHeight * boxSize;

    ParticleEnergyBox particleEnergyBox =
        new ParticleEnergyBox(boxSize,
        eventToBuild.getX(), eventToBuild.getY());
    particleEnergyBox.setMaterial(new PhongMaterial(
        colorModel.determineColorByPixelValue(
            pixelDefaultColor)));

    particleEnergyBox.setTranslateX(center + x * boxSize);
    particleEnergyBox.setTranslateY(center + y * boxSize);
    particleEnergyBox.setTranslateZ(center);

    return particleEnergyBox;
}
```

Ukázka kódu 7: Znázornění 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, mění 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 třída `ColorModel` reprezentující barevný model přiřazující barvy eventům ve vizualizaci. Je shodný pro 2D a 3D metodu. Veřejná metoda třídy `determineColorByPixelValue(int pixelValue)` slouží

k přidělení barvy na základě předané hodnoty ToT eventu. Přiřazení funguje na principu procentuálního rozdělení všech možných barev pro načtená data. Třída uchovává pole objektů `Color` s předgenerovanými hodnotami dlouhé 101 prvků (0 - 100 včetně), minimální a maximální hodnotu *ToT*. Minimální je vždy 0, maximální hodnota je aktualizována při načtení nových dat a je získána právě z dat. Hodnota ke vrácení je určena tak, že je spočítán procentuální podíl předané *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 části 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 objektů *Rectangle*. První v červené barvě znázorňuje celý soubor a druhý v modré barvě je umístěn přes červený a znázorňuje pozici zobrazeného okna. Délka červeného obdélníku je volitelná. Levý okraj je index 0, pravý je poslední index. K aktualizaci dochází tak, že třída udržuje referenci na *IntegerProperty* s hodnotou aktuální pozice zobrazeného okna, jeho hodnotu aktualizuje vizualizace. *ColorModel* na tuto referenci umístil *listener* a při jeho aktualizaci je určena nová pozice modrého obdélíku. Pozice je určena tak, že je porovnána nová hodnota oproti maximu v datech s délkou červeného obdélníku. Procentuální podíl je použit jako finální pozice.

Obrázek 5.10: Ukazetel pozice aktuálně zobrazeného okna *DataIndexBar*

5.8 Předávání výsledků akcí v komponentě

Pro předávání výsledků a stavů o akcích komponenta využívá vlastní definované výjimky jazyka Java. Jsou použity primárně když nastane nestandardní situace, jako jsou nevalidní předané vstupy, chyby při vizualizaci, apod. Jsou navrženy tak, aby předávali informace o nastalé akci samotnému programu a ten s nimi mohl následně pracovat a zároveň, aby poskytovali informace užitečné pro uživatele. Jsou rozděleny na dvě základní skupiny.

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

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

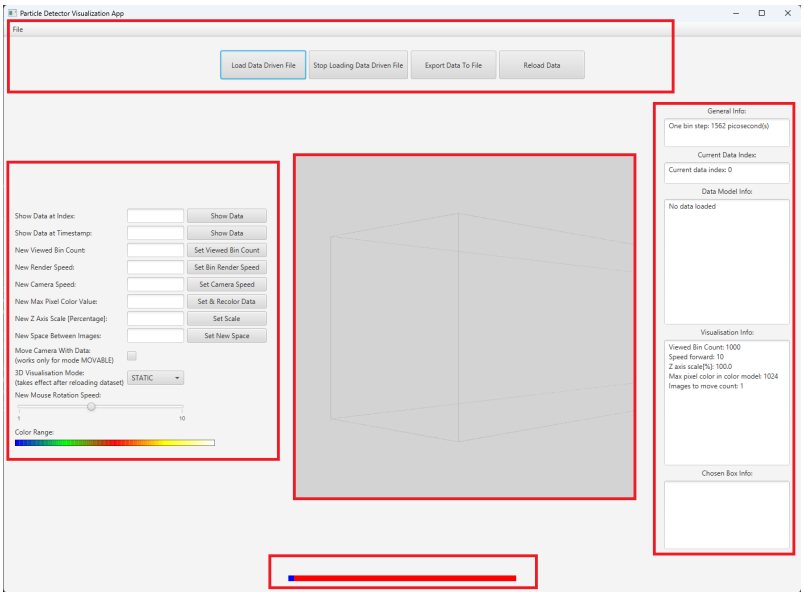
5.9 Demonstrační aplikace

V této části je popsán popis implementace demonstrační aplikace, která využívá komponentu a předvádí její funkcionalitu. 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 výchozí `Stage primaryStage`, rozložení prvků v okně je dáno layout komponentou `BorderPane`. Znázornění aplikace včetně 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 prostřední části (`MiddlePane`) se nachází komponenta samotná. Pro obě varianty je reprezentována šedým čtvercem, do kterého je vizualizace vykreslována. V levé části (`LeftPane`) se nachází prvky pro úpravu vizualizace, většina je složena z `Labelu` označující konkrétní prvek, `TextFieldu` pro zadání hodnoty a tlačítka pro nahrání 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 funkcionalitu se stará komponenta samotná.

TextArea pro zobrazení hodnot komponenty jsou vyplněna 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 čitelného pro uživatele a zobrazí do příslušných `TextArea`. Každé tlačítko v aplikaci po jeho stisknutí zavolá obslužnou metodu, která převezme parametry pro akci a zavolá příslušnou metodu v API komponenty. Parametry si metoda buď vytáhne z příslušných prvků (z `TextField` pro aktualizaci vizualizace), nebo jejich získání sama poskytne (např. při zadání načítání ze souboru metoda pro uživatele otevře okno pro vybrání souboru (přes třídu JavaFX `FileChooser`).



Obrázek 5.11: Ukázka rozložení prvků v demonstrační aplikaci

6 Testování

V této kapitole je rozebráno testování komponent. Dělí 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 obě demonstrační 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 přívětivosti a závěrem byly obě zvolené vizualizační metody spolu porovnány.

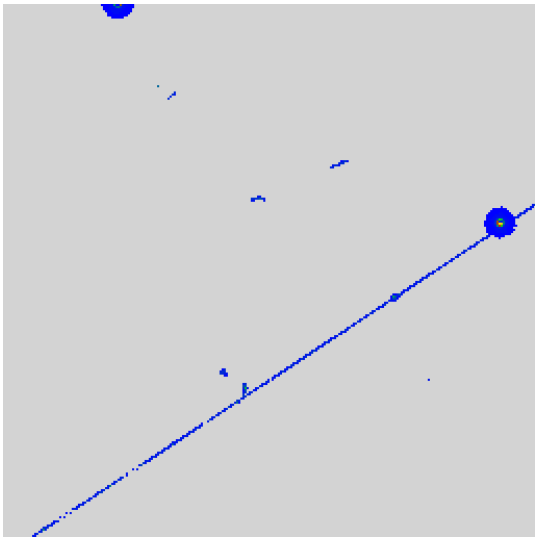
6.1 Programové testování

Programové testování je zaměřeno na testování výpočetních operací, pokryto je pomocí Java Unit testů. Výpočetní operace jsou načítání ze souboru poskytované třídou `DataModelManager` a 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 načtena a uložena správně. Na to byl vygenerován soubor s daty se specifickými vlastnostmi. Obsahuje tři biny, eventy každého tvoří jiný obrazec. Po načtení jsou data zkontrolována, zda jsou ve správných binech správně uloženy patřičné obrazce. Následně je otestována i operace *grouping*, zda jsou obrazce správně sloučeny. Na toto byl vytvořen soubor se specifickým počtem časových známek, aby bylo možné výsledek snadněji ověřit. Následně byly ověřeny všechny chybné stavy při načítání podchycované výjimkami, pro některé z těchto testů byly vygenerovány testovací soubory se specifickou chybou.

6.2 Uživatelské testování

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

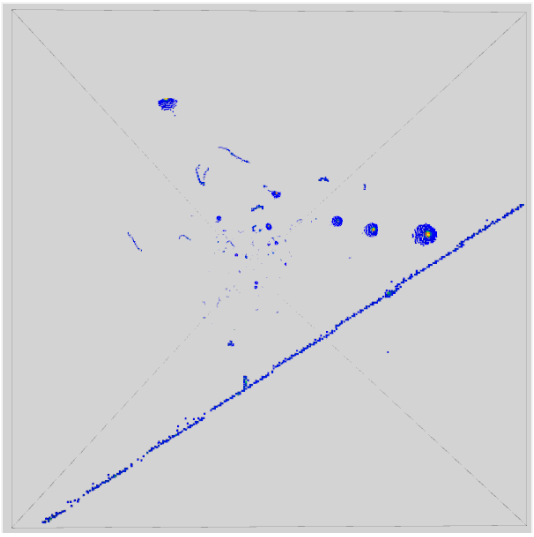
data. Testování bylo provedeno na několika datových sadách, například z observatoře nacházející se 2 300 m nad mořem nebo z atomového krytu. Výsledky jsou pro obě vizualizační metody pozitivní, v obou se podařilo najít požadované jevy. Hlavním cílem bylo najít v datech tzv. miony, jedná se o kosmické záření a projevují se jako dlouhé rovné čáry. Nalezeny byly ve všech poskytnutých datových sadách. Příklad se nachází na obrázcích. Zajímavé bylo i pozorování, že se v datové sadě z observatoře miony vyskytují častěji než v sadě z atomového krytu. Toto pozorování i potvrzuje správnost vizualizace, protože taková hustota výskytu těchto jevů odpovídá umístění detektorů.



Obrázek 6.1: Zobrazení mionu ve 2D komponentě.

Dále se zadavatel zaměřil na otestování uživatelské přívětivosti aplikací. Vyhodnotil, že ovládání přes klávesnici je intuitivní a je s ním spokojen. Ovládání vizualizaci myší je také v pořádku, ale měl požadavek, zda by rychlost otáčení modelu v 3D vizualizaci mohla být nastavitelná. Do komponenty byla následně úprava rychlosti přidány, v aplikaci se upravuje pomocí přidaného šoupátka. Kritičtěji se vyjádřil k intuitivnosti některých vizuálních prvků aplikace, protože nebylo podle původního pojmenování úplně jasné, jaké parametry přesně upravují a případně, k čemu přesně parametry slouží. Podle jeho poznatků byly tyto prvky ještě upraveny.

K 3D vizualizační metodě samotné poznamenal, zda by šlo upravit nasvícení scény. Při prohlížení voxelů pod některými úhly jsou některé voxely stínované a vizuálně by pro něj bylo příjemnější, aby stínování bylo vypnuto.



Obrázek 6.2: Zobrazení mionu ve 3D komponentě.

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

Při testování ještě zachytil chybu 3D komponenty, kdy se při znovu načtení nějaké datové sady při prvním kliknutí na voxel vyskočilo okno *Alert* 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 časových 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 časové známky jednotlivých eventů. Celkově 3D vizualizaci vyhodnotil jako povedenou a lze s ní nadále pracovat a případně rozšiřovat. 2D vizualizaci vyhodnotil tak, že oproti 3D není příliš užitečná není a díky možnosti používat 3D metodu není potřeba 2D používat. Maximálně by šla použít jako doplněk k 3D komponentě, kdyby ve 2D vizualizaci šla například exportovat vybraná množina dat do 2D obrázku.

7 Závěr

Cílem této práce bylo vytvořit prototyp komponenty pro vizualizaci dat z částicových detektorů. V rámci práce byly zvoleny dvě zobrazovací metody vhodné pro zobrazování dat z částicových detektorů v čase 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 rešerši částicových 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 těchto dat. Nejdříve byly zanalyzovány současné metody vizualizace, následně byly zanalyzovány možnosti vizualizace objemových dat a na základě těchto analýz 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 čase. Jako druhá byla zvolena 2D metoda, kdy jsou data z několika časových známek sloučena 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 přidělování 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 užitečnější, efektivnější a do budoucna použitelná.

Práci by 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 by bylo zobrazení několika komponent najednou, kdy by každá zobrazovala jinou datovou sadu. Všechny vizualizace by šlo procházet najed-

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

Literatura

[1] *What is an API?* [online]. IBM, 2023. [cit. 11. března 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. března 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 Equipment*. 2013, 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] Prokeš, 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

Seznam tabulek

Seznam ukázek kódu

I Příloha A - Uživatelská příručka

Uživatelská příručka popisuje používání vizualizační komponenty a demonstrační aplikace. Uživatelská příručka je rozdělena na dvě hlavní části: Vizualizační komponenta - zaměření na přidání vizualizační komponenty do projektu z programátorské ho hlediska, Demonstrační aplikace - popis jak spustit a ovládat aplikaci. Komponenta a aplikace jsou vytvořeny v technologii Java, GUI a grafické komponenty ve frameworku JavaFX. Vyvinuty byly ve verzích Java a JavaFX 8, použitelné jsou ve všech verzích od 8 výše. K datu vyhotovení byla nejaktuálnější verze Java i JavaFX 19. Při použití verze Java 11 a vyšších je nutné framework JavaFX připojit ke komponentě či aplikaci externě.

Komponenta a aplikace se nachází v adresáři Aplikace_a_knihovny, který obsahuje celkem šest adresářů.

- base_component - Základní část komponenty poskytující operace s daty. Použijte při implementaci vlastní vizualizační metody.
- component_2D – komponenta s 2D vizualizační metodou.
- component_3D – komponenta s 3D vizualizační metodou.
- application_2D – aplikace využívající 2D komponentu
- application_3D – aplikace využívající 3D komponentu
- tests – jednotkové testy použity k testování výpočetních operací komponenty.

I.1 Vizualizační komponenta

V této části se nachází popis práce s vizualizačními komponentami, jedná o balíčky component_2D a component_3D.

I.1.1 Přidání komponenty do projektu

Postup přidání komponenty do projektu je stejný pro obě varianty komponenty, 2D a 3D, rozdíl je až v používaní 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 několik, buď lze přidat .jar do classpath projektu (IDE jako je například Eclipse nebo Intellij IDEA to běžně umožňují) 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 všem 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 JavaFX `Group`. Přidejte tedy instanci této třídy do scény jako běžnou JavaFX komponentu. K dipozici 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 veřejných 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 počítat souřadnice binů pro osu Z. Tento parametr je závislý na použité vizualizační metodě. Pro 2D komponentu je potřeba zadat false (komponenta by validně fungovala i s hodnotou true, ale vypočítané 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 či vypnout.
- `int initSubSceneHeight` – počáteční výška SubScene, do které bude 3D vizualizace vykreslena.
- `int initSubSceneWidth` – počáteční šířka SubScene, do které bude 3D vizualizace vykreslena.
- `int initTimeWindow` – počáteční velikost časového okna, tedy počet, kolik binů má být najednou vykresleno.

- `int initImageViewHeight` – počáteční výška okna, do které bude 2D vizualizace vykreslena.
- `int initImageViewWidth` – počáteční 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*, bližší popis níže v části 3D vizualizace. Jako parametr metody zadejte řetězcový kód určující, jaký mód má být použit. Kód „STATIC“ pro mód *Static*, kód „MOVABLE“ pro kód *Movable*. Pokud je předaný kód nevalidní, automaticky je nastaven mód *Static*.

Obrázek I.1: Ukázka vzhledu DataIndexBar

Seznam metod, které poskytují obě 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 špatně zvoleného vstupního souboru. Po zavolání 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 metodě, která zajistí, že komponenta událost správně zpracuje.
4. `void showDataAtNewIndexAction(int newIndex)` - Zobrazení dat počínající binem s předaným indexem. Předchozí zobrazené okno bude smazáno a budou vykreslena nová data na předaném indexu. Hodnota indexu musí být v rozsahu nula až počet_binů_v_datech).

5. `void showDataAtTimestampAction(long timeStamp)` - Zobrazení dat počínající binem s předanou časovou známkou (ToA). Pokud se v datech nenachází bin s předanou ToA, je použit bin s nejbližší hodnotou ToA k předané. Předchozí zobrazené okno bude smazáno a budou vykreslena nová. Předaná časová známka může mít jakoukoliv hodnotu, vždy bude v datech nalezena nejbližší.

6. `void updateTimeWindowAction(int newTimeWindow)` - Nastavení nové hodnoty parametru TimeWindow, tedy počtu, kolik binů má být najednou vykresleno. Nová hodnota musí být kladné číslo.

7. `void updateImagesToMoveCountAction(int newImagesToMoveCount)` - Nastavení nové hodnoty udávající, o kolik binů se má okno posunout při jednom zavolání akce posunu. Tedy, kolik binů má být při posunu přidáno 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 vypočítání barev v barevném modelu. Při snížování hodnoty dojde ke k přidělení výraznějších barev jednotlivým krychlím či čtvercům. Nová hodnota musí být kladné číslo.

9. `IntegerProperty oneBinStepIntegerPropertyProperty()` - Getter pro získání IntegerProperty s aktuální hodnotou parametru krok jednoho binu.

10. `StringProperty timeMeasurementUnitInfoStringPropertyProperty()` - Getter pro získání StringProperty s aktuální hodnotou nastavené jednotky času pro aktuální data.

11. `StringProperty dataModelInfoStringPropertyProperty()` - Getter pro získání StringPropery obsahující informace o parametrů datového modelu a katuálně načtených datech.

12. `Node getDataIndexBarGroup()` - Getter pro získání Group obsahující DataIndexBar.

13. `IntegerProperty dataIndexIntegerPropertyProperty()` - Getter pro získání IntegerProperty obsahující informace o aktuálním počátečním indexu, od kterého jsou vykresleny data.

Seznam metod, které poskytuje pouze 2D komponenta:

1. `StringProperty chosenSquareInfoStringPropertyProperty()` - Getter pro získání StringPropery obsahující informace o aktuálně zvoleném čtverci.

2. `StringProperty visualisation2DInfoStringPropertyProperty()` - Getter pro získání StringPropery obsahující informace o aktuálním nastavení 2D vizualizace.

Seznam metod, které poskytuje pouze 3D komponenta:

1. `void updateZAxisScaleAction(int scaleValuePercentage)` - Nastavení nové hodnoty měřítka osy Z. Hodnota musí být zadána v procententech v intervalu (0 - 100>. Defaultně je hodnota nastavena na 100%.

2. `void updateMaxSpaceBetweenImagesAction(int newSpaceBetweenImages)` - Nastavení nové hodnoty maximální možné mezery mezi dvěma snímky. Tato hodnota se používá pro výpočet souřadnic osy Z. Vzdálenost mezi dvěma biny nesmí být větší než tato hodnota. Hodnota musí být kladná.

3. `void updateSpeedForwardAction(int newSpeed)` - Nastavení nové hodnoty rychlosti jakou se kamera pohybuje ve směru dopředu či 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 při zobrazení nových dat.

5. `void updateSubSceneSize(int subSceneWidth, int subSceneHeight)` - Aktualizace šířky a výšky SubScene. Změna se projeví okamžitě. Hodnoty musí být kladné.

6. `void moveCameraWithDataAction(boolean moveCamera)` - Nastavení hodnoty, zda se má při použití akce showDataAtTimestampAction(long timeStamp) nebo showDataAtTimestampAction(long timeStamp) na pozici nově vykreslených dat přesunout i kamera. Použitelné pouze pro mód Movable, při módu Static kamera zůstává na místě.

7. `StringProperty chosenBoxInfoStringPropertyProperty()` - Getter pro získání StringPropery obsahující informace o aktuálně zvolené krychli.

8. `StringProperty visualisation3DInfoStringPropertyProperty()` - Getter pro získání StringPropery obsahující informace o aktuálním nastavení 3D vizualizace.

I.2 Demonstrační aplikace

I.2.1 Překlad 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 překladu 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 definuje 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 vyšší, ve kterých již není zakomponována JavaFX defaultně v Java, můžete ji přidat jako *dependency* do pom.xml. Pro překlad otevřete v kořenovém adresáři příkazovou řádku/terminál a zadejte příkaz „mvn clean install“, čímž spustíte překlad 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 spouští spustitelným .jar souborem `Particle_Detector_Visualization_App-jar-with-dependencies.jar` nacházejícím se po vygenerování v adresáři `/target`. Spouští buď dvojklikem myší na konkrétní archiv (nemusí fungovat vždy, může být potřeba ještě povolit v operačním systému) nebo z příkazové řádky (funguje vždy). 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, liší se pouze v prvcích pro úpravu vizualizace a v komponentě umís-

těné uprostřed. Podobu okna můžete vidět na obrázku I.2 a na obrázku I.3 jeho rozložení vršek, spodek, prostředek a levou a pravou část. V horní části se nachází tři tlačítka sloužící k importu a exportu dat, jejich podrobnější 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 ukazatel aktuální pozice zobrazovaných dat v celku. Na levé části se nacházejí ovládací prvky pro úpravu vzhledu nebo chování vizualizace. Vždy je uveden popis prvku, textfield k zadání nové hodnoty a tlačítko, po jejímž stisknutí je hodnota z textfield převedena na číslo a předána 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 počáteční index vizualizovaných dat, ve třetí statistické informace o aktuálně načtených datech, ve čtvrté informace o aktuální vizualizaci a v páté informace o aktuálně myší vybrané krychli/čtverci.

Popis tlačítek v horší 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 předán komponentě, ta provede jeho validaci a pokud je v pořádku, spustí načítání. To běžně trvá jednotky vteřin, u velkých souborů i desítky. Po úspěšném dokončení načítání se zobrazí *Alert* okno informující o úspěšném nahrání dat do komponenty, po kliknutí na tlačítko *OK* se okno zavře a do komponenty budou vykreslena počáteční data. Pokud během k načítání došlo k nějaké chybě, je ukončeno a zobrazí se *Alert* okno s popisem chyby.

2. **Stop Loading Data Driven File** – slouží k předčasnému ukončení načítání dat ze souboru. Po kliknutí bude načítání přerušeno a zobrazí se *Alert* okno informující o úspěšném přerušení.

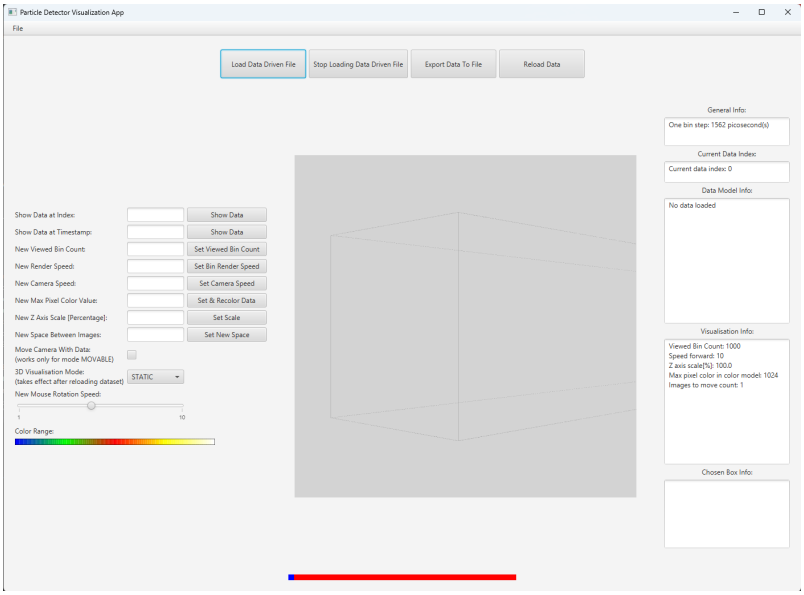
3. **Export Data To File** – slouží k exportu aktuálně zobrazeného okna do textového souboru. Po kliknutí se otevře souborové okno operačního systému, vyberte adresář, do kterého má být export proveden. Po vybrání bude adresář předán komponentě, která vezme aktuálně načtená data a exportuje je do nově vytvořeného textového souboru,

jehož jméno je složenino ze jména datové sady a počátečního a koncového indexu exportovaného okna. Po skončení se zobrazí okno *Alert* 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`.



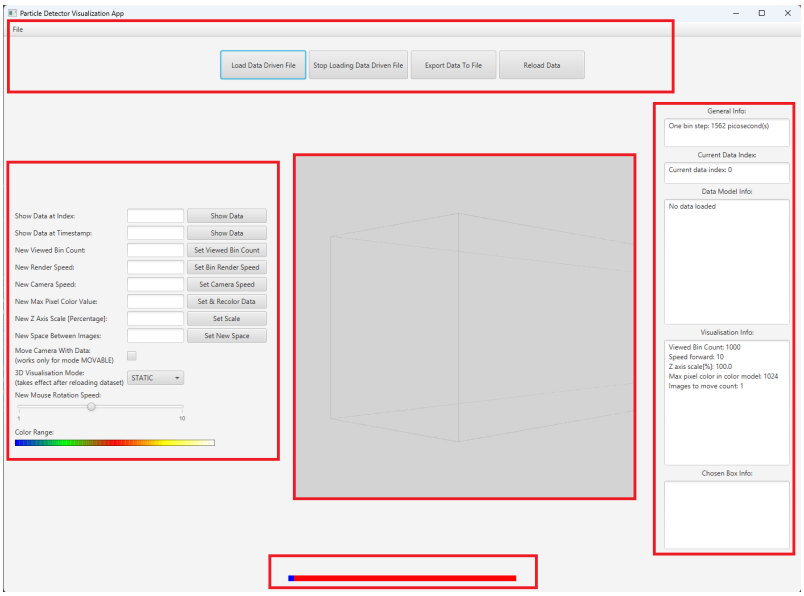
Obrázek I.2: Vzhled demonstrační aplikace..

I.3 Ovládání

V této části je popsáno ovládání komponenty. Komponenta se ovládá buď přes GUI prvky ve scéně (jejich význam popsán výše) nebo hlavně klávesnicí a myší, které slouží primárně pro ovládání vizualizací. Následuje popis ovládání přes klávesnici a myš.

Seznam kláves:

1. **Šipka doleva** - rotace kamery doleva, pouze 3D komponenta



Obrázek I.3: Rozložení prvků v demonstrační aplikaci.

2. **Šipka doprava** - rotace kamery doprava, pouze 3D komponenta
3. **Šipka nahoru** - rotace kamery nahoru, pouze 3D komponenta
4. **Šipka dolů** - rotace kamery dolů, pouze 3D komponenta
5. **W** - posun kamery dopředu, 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 dolů, pouze 3D komponenta
11. **O** - vykreslení dalších binů ve směru dopředu
12. **P** - vykreslení dalších binů ve směru dozadu
13. **T** - posun kamery na souřadnici Z posledního vykresleného binu, pouze 3D komponenta
14. **R** - posun kamery na její prvotní 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 myší:

1. **Kliknutí levým tlačítkem myši** - 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 myši** - podržením levého tlačítka myši a následným táhnutím myši můžete otáčet zobrazeným modele. Pouze 3D vizualizace.

3. **Posunutí pravým tlačítkem myši** - podržením pravého tlačítka myši a následným táhnutím myši 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 - Rychlost můžete upravit šoupátkem 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 naměřenou 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 binů seřazených chronologicky. Vždy je vykreslena jen část z uložených dat, vykreslení lze šoupat danými klávesami ve směru dopředu či dozadu. Pozice vykreslení lze zadat i přímo přes index či časovou známku počátečního binu.

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

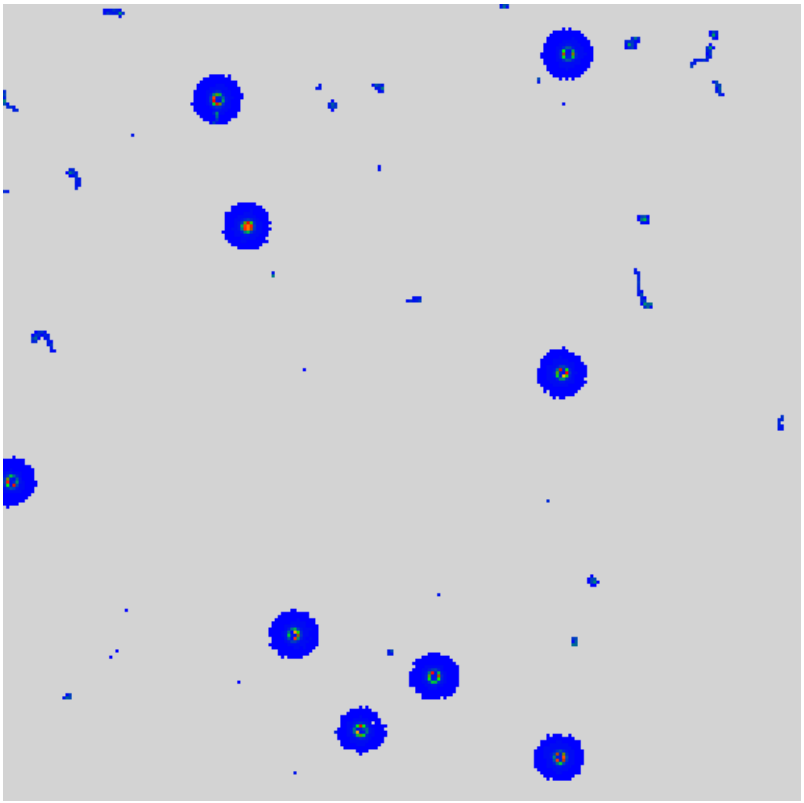
1. **Time Window** – hodnota udávající, kolik binů má být najednou vykresleno. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná.

2. **Space Between Images** – maximální možná vzdálenost mezi dvěma biny. Při určování souřadnic Z je použito toto maximum, vzdálenost mezi dvěma biny nesmí být větší než tato hodnota. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná.

3. **Render Speed** – kolik binů má být vykresleno při pohybu okna dopředu či dozadu. Platí pro oba vizualizační módy. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná.

4. **Max Pixel Color** – udávající maximální požnou hodnotu ToT v datech, pomocí této hodnoty jsou spočítány barvy pro krychle. Lze upravit a zadat jinou hodnotu než je v datech, čímž model přebarvit.

5. **Color Range** – znázornění barevné škály používané k obarvení čtverců. 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 přizpůsobené 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 binů po ose Z odpovídá jejich *ToA*. Vykreslená data jsou obalena kvádrem značícím jejich hranice. Na libovolnou krychli lze kliknout, zvýrazní se a napravo od komponenty se v příslušné oblasti vypíší informace o události, kterou krychle reprezentuje. Kliknutím mimo krychli se zvýraznění krychle smaže. Data jsou uložena jako seznam binů seřazených chronologicky. Vždy je vykreslena jen část z uložených dat, vykreslení lze šoupat danými klávesami ve směru dopředu či dozadu. Pozice vykreslení lze zadat i přímo přes index či časovou známku 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 binů vizualizované okno zůstane na místě a mění se souřadnice Z jednotlivých binů v požadovaném směru. Kamera zůstane na místě. MOVABLE znamená, že při vykreslování nových binů zůstanou souřadnice Z existujících binů stejné, ale posune se vizualizované okno v požadovaném směru. Je pak potřeba ještě posunout v tomto směru i kameru. Pro projevení změny módu je potřeba znovu načíst vstupní data.

2. **Time Window** – hodnota udávající, kolik binů má být najednou vykresleno. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná.

3. **Space Between Images** – maximální možná vzdálenost mezi dvěma biny. Při určování souřadnic Z je použito toto maximum, vzdálenost mezi dvěma biny nesmí být větší než tato hodnota. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná.

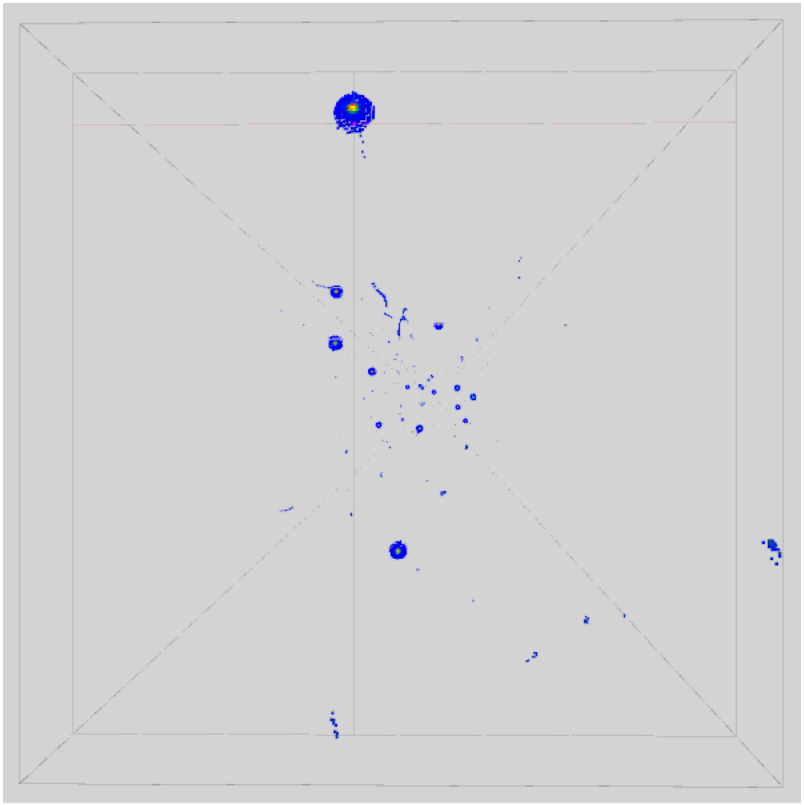
4. **Camera Speed** – rychlost posunu kamery ve směru dopředu či dozadu. Po aktualizaci je vizualizace ihned překreslena. Hodnota musí být kladná. V ostatních směr je rychlost pohybu kamery konstantní.

5. **Render Speed** – kolik binů má být vykresleno při pohybu okna dopředu či dozadu. Platí pro oba vizualizační módy. Po aktualizaci je vizualizace ihned překreslena. 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 překreslena. Při přesunu na konkrétní index nebo časovou známku je kamera přesunuta nezávisle na této hodnotě.

7. **Max Pixel Color** – udávající maximální požnou hodnotu ToT v datech, pomocí této hodnoty jsou spočítány barvy pro krychle. Lze upravit a zadat jinou hodnotu než je v datech, čímž model přebarvit.

8. **Color Range** – znázornění barevné škály používané k obarvení čtverců. Zleva stoupající.



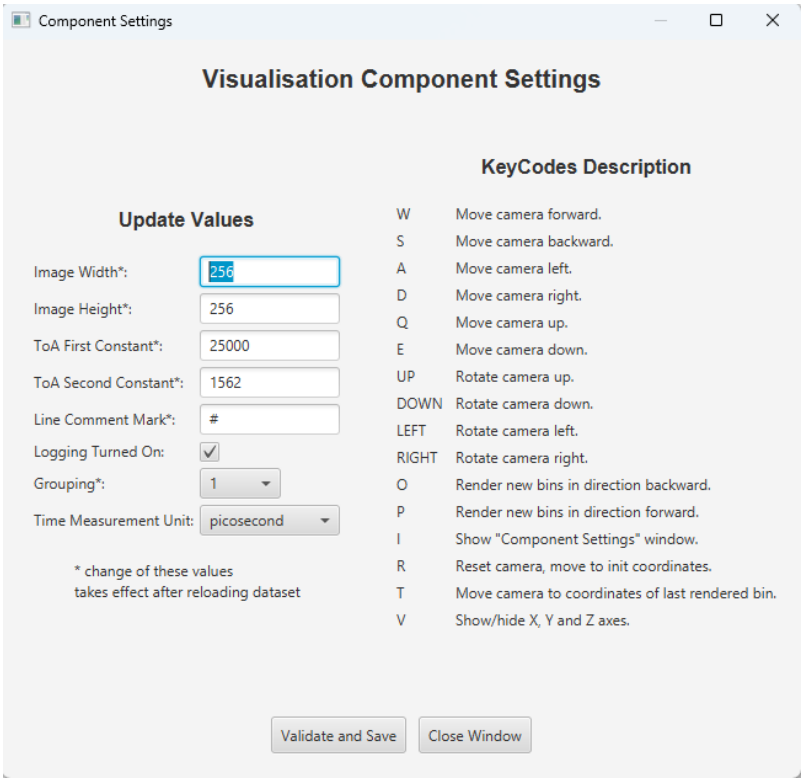
Obrázek I.5: 3D vizualizace

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. Zaprvé přes menu v levém horním rohu scény aplikace *File - Component Settings*, zadruhé stisknutím tlačítka *I*. 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 zadává přes textfieldy a aktualizují 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 značící řádkový komentář ve vstupních datech. Hodnota může být jakkýkoliv řětězec, pouze hodnota nesmí být prázdná.
6. **Logging Turned On** – zda má být zapnuto logování v komponentě, běžně loguje do konzole
7. **Grouping** – kolik binů má být slučováno při operaci grouping, označované též jako rebinování. Pokud je hodnota „1“, ke groupingu nedochází.
8. **Time Measurement Unit** – časová jednotka, ve které byla naměřena vstupní data. Není použita k dalším výpočtům, pouze k zobrazení aktuální hodnoty kroku jednoho binu v okně aplikace.



Obrázek I.6: Okno s nastavením komponenty