VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ LIDAROVÝCH, KAMEROVÝCH A VEKTOROVÝCH DAT Z ŽELEZNIČNÍHO MOBILNÍHO MAPOVACÍHO SYSTÉMU

VISUALIZATION OF LIDAR, CAMERA, AND VECTOR DATA FROM A RAILWAY MOBILE MAPPING SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ZUZANA MIŠKAŇOVÁ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ONDŘEJ KLÍMA, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

164356

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Studentka:	**Miškaňová Zuzana**
Program:	Informační technologie
Název:	**Zobrazení lidarových, kamerových a vektorových dat z železničního mobilního mapovacího systému**
Kategorie:	Uživatelská rozhraní
Akademický rok:	2024/25

Zadání:

1. Seznamte se možnostmi vizualizace obrazových a geometrických dat z mobilních mapovacích systémů.
2. Navrhněte uživatelský systém pro vizualizaci agregovaných dat ze senzorů umístěných na čele vlaku včetně vektorových mapových dat.
3. Navržený systém implementujte v podobě uživatelské aplikace v prostředí Python s využitím existujících nástrojů a knihoven.
4. Proveďte experimenty s dodanými daty a vyhodnoťte vlastnosti a uživatelskou přívětivost aplikace.
5. Prezentujte dosažené výsledky.

Literatura:
Dle doporučení vedoucího.

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2 a částečně 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Klíma Ondřej, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2024
Termín pro odevzdání:	14.5.2025
Datum schválení:	12.11.2024

Abstrakt

Cieľom tejto práce je vytvoriť systém pre vizualizáciu dát z železničného mobilného mapovacieho systému, ktorý dáta zobrazí praktickým, prehľadným a prispôsobiteľným spôsobom. Tento systém bol implementovaný ako webová aplikácia s využitím programovacieho jazyka Python a frameworku Dash. Pre optimalizáciu výkonu prebieha vykresľovanie mračna bodov na strane klienta pomocou frameworku deck.gl. Vyvinutá aplikácia poskytuje dostatočnú funkcionalitu a výkonnosť. Avšak stále by boli možné ďalšie zlepšenia, najmä čo sa týka prispôsobiteľnosti. Táto nová webová aplikácia má potenciál byť využitá v oblasti vizualizácie dát z mobilných mapovacích systémov. Znalosti získané počas tvorby práce by tiež mohli byť užitočné pre ďalších vývojárov, ktorí vytvárajú systémy pre vizualizáciu dát.

Abstract

The aim of this thesis is to create a system visualizing data from a railway mobile mapping system, which will display data in a practical, clear and customizable way. The system was implemented as a web application using programming language Python and framework Dash. For performance optimization, the rendering of the point cloud and vector data is done on the client's side using framework deck.gl. The created application offers sufficient functionality and performance. Hovewer, there is still room for further improvement, mainly in customizability. This new web application has a potential to be exploited within the field of mobile mapping system data visualizations. The knowledge acquired throughout the work could also useful to other developers creating visualization systems.

Kľúčové slová

vizualizácia dát, mobilný mapovací systém, lidar, mračno bodov, deck.gl, webové aplikácie, Dash

Keywords

data visualization, mobile mapping system, lidar, point cloud, deck.gl, web aplications, Dash

Citácia

MIŠKAŇOVÁ, Zuzana. *Zobrazení lidarových, kamerových a vektorových dat z železničního mobilního mapovacího systému.* Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Ondřej Klíma, Ph.D.

Zobrazení lidarových, kamerových a vektorových dat z železničního mobilního mapovacího systému

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána Ing. Ondřeja Klímu, Ph.D. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Zuzana Miškaňová
26. apríla 2025

Poďakovanie

Ďakujem môjmu vedúcemu Ing. Ondřejovi Klímovi, Ph.D. za poskytnutú pomoc a cenné rady, ktoré mi v priebehu tvorby práce výrazne pomohli.

Obsah

1 Úvod **2**

2 Vizualizácia dát z mobilných mapovacích systémov **3**
2.1 Dierkový model kamery 3
2.2 Skreslenie kamery 6
2.3 Vykresľovací reťazec 6
2.4 Framework deck.gl a jeho nadstavba Pydeck 6
2.5 Nastavenie polohy kamery vo frameworku deck.gl 7
2.6 Frameworky pre tvorbu webových aplikácií 10

3 Návrh aplikácie **13**
3.1 Návrh používateľského rozhrania 14
3.2 Problém nahrávania dát do aplikácie 14

4 Implementácia navrhnutej aplikácie **17**
4.1 Prvá etapa implementácie 17
4.2 Optimalizácia vykresľovania mračna bodov 19
4.3 Výber spôsobu nastavenia polohy kamery 21
4.4 Rozdelenie dát do vrstiev 22
4.5 Implementácia animácie pohybu vlaku 23
4.6 Zobrazenie prejazdného profilu vlaku 24
4.7 Implementácia skreslenia 24

5 Testovanie **25**
5.1 Zobrazenie dodaných dát 25
5.2 Výkonnosť 25
5.3 Vyhodnotenie používateľskej prívetivosti 25

6 Záver **26**

Literatúra **27**

Kapitola 1

Úvod

Kapitola 2

Vizualizácia dát z mobilných mapovacích systémov

Výstupom mobilného mapovacieho systému je množstvo dát z rôznych senzorov. Dáta, ktoré boli poskytnuté k tvorbe tejto práce, obsahovali v prvom rade mračná bodov získané z lidaru. Aby bolo možné s týmito mračnami bodov pracovať a vhodne ich zobrazovať, je potrebné, aby mobilný mapovací systém zaznamenával údaje o svojej polohe. V tomto prípade boli k dispozícii údaje o transláciách a rotáciách kamery s časovými razítkami. Ďalej býva súčasťou mobilného mapovacieho systému klasická kamera. Pre zobrazenie mračna bodov takým spôsobom, aby sa zobrazenie čo najviac zhodovalo s kamerovým záznamom, je nutné poznať parametre kamery – kalibračnú maticu a prípadne aj parametre skreslenia.

Pri vývoji aplikácie, ktorá má umožniť vizualizáciu týchto dát a navyše aj ďalších vektorových dát, je nutné poznať základné princípy zobrazovania 3D dát v počítačovej grafike. Ďalej je potrebné vybrať si a naštudovať technológie, pomocou ktorých bude aplikácia vytvorená, a to konkrétne niektorý z frameworkov pre tvorbu aplikácií v jazyku Python a vhodný framework pre zobrazenie grafických dát. Práve to je predmetom tejto kapitoly.

2.1 Dierkový model kamery

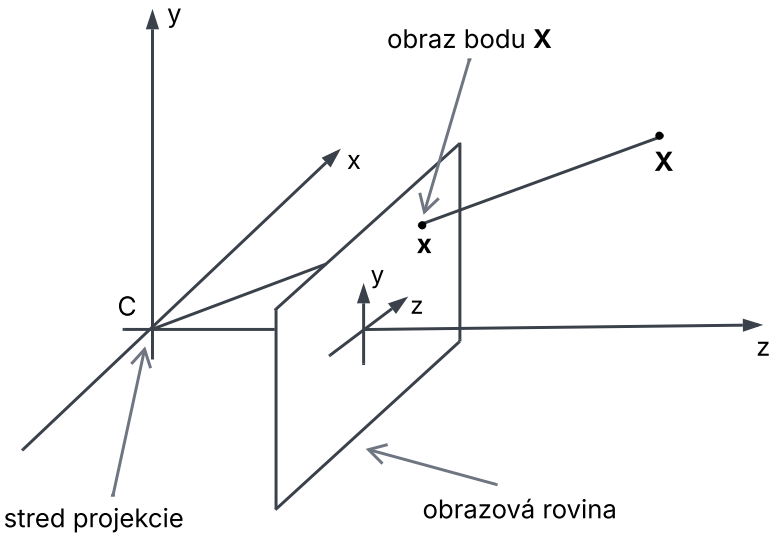
Kamera je v počítačovej grafike pojem, ktorý označuje projekciu bodov z trojrozmerného priestoru do roviny. Túto projekciu je možné vyjadriť pomocou matíc a je väčšinou bodová (*central projection*).

Existuje niekoľko rôznych modelov kamery, z ktorých najjednoduchší je dierkový model kamery (*pinhole camera model*). U tohto modelu je stred projekcie $\mathbf{C}$ (*camera centre*) v počiatku Euklidovského súradnicového systému a body sa premietajú do roviny $z = f$, ktorá sa označuje ako obrazová rovina (*image plane*).

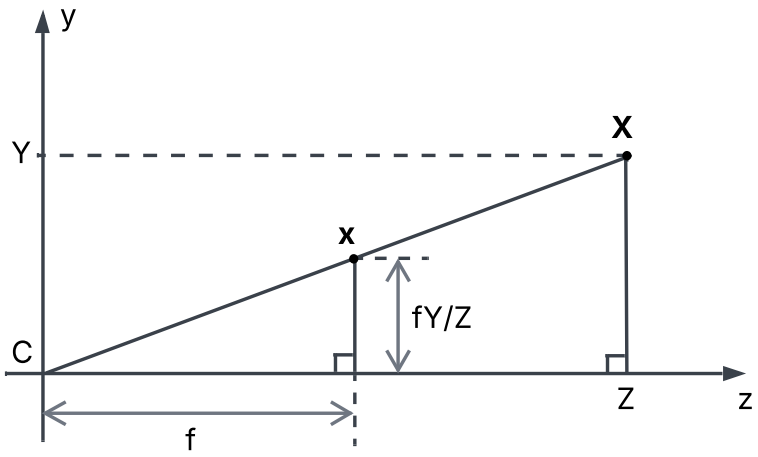
Princíp zobrazenia je nasledovný: obrazom bodu $\mathbf{X} = (X, Y, Z)^T$ je bod $\mathbf{x}$, kde priamka vedúca bodom $\mathbf{X}$ a stredom projekcie $\mathbf{C}$ pretína obrazovú rovinu (obrázok 2.1). Z podobnosti trojuholníkov je možné odvodiť, že bod $\mathbf{x}$ má súradnice $(fX/Z, fY/Z, f)^T$, postup je naznačený na obrázku 2.2.

Keďže všetky obrazy bodov ležia v obrazovej rovine $z = f$, je možné poslednú súradnicu zanedbať a zapisovať súradnice bodu $\mathbf{x}$ v súradnicovom systéme obrazovej roviny ako $(fX/Z, fY/Z)^T$.

Túto projekciu môžeme v homogénnych súradniciach zapísať pomocou násobenia matíc nasledovným spôsobom:



Obr. 2.1: Zakladný princíp dierkového modelu kamery.



Obr. 2.2: Náčrt odvodenia súradníc bodu $\mathbf{x}$.

$$\mathbf{x} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \mathbf{X} = \mathrm{P}\mathbf{X}$$

Maticu P nazývame projekčnou maticou kamery (*camera projection matrix*).

Rozšírenia základného dierkového modelu kamery

V praxi väčšinou chceme vyjadriť body na obrazovej rovine v súradnicovom systéme, ktorý nemá stred v bode $(0, 0, f)^T$, ale v ľubovoľnom bode $(-p_x, -p_y, f)^T$ (obrázok 2.3). Obrazom bodu $\mathbf{X} = (X, Y, Z)^T$ (v súradnicovom systéme kamery) potom bude bod $\mathbf{x} = (fX/Z +$

$p_x, fY/Z + p_y, f)^T$ (v súradnicovom systéme obrazovej roviny). To je možné zahrnúť do projekčnej matice nasledovným spôsobom:

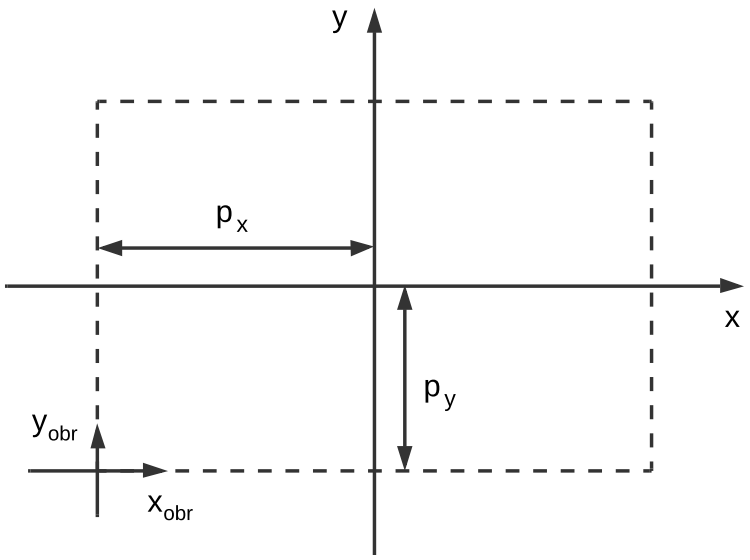
$$\mathrm{P} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix}$$

$$\mathbf{x} = \begin{pmatrix} fX/Z + p_x \\ fY/Z + p_y \\ 1 \end{pmatrix} = \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Maticu

$$\mathrm{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix}$$

nazývame kalibračnou maticou kamery (*camera calibration matrix*) a medzi ňou a projekčnou maticou platí vzťah $\mathrm{P} = \big[\mathrm{K}|0\big]$, kde 0 predstavuje nulový stĺpcový vektor.



Obr. 2.3: Súradnicový systém so stredom v bode $(-p_x, -p_y, f)^T$ obrazovej roviny.

Parametre f, p_x a p_y označujeme ako **vnútorné parametre kamery**.

Až doteraz sme predpokladali, že kamera má stred v počiatku súradnicovej sústavy a je „otočená“ v smere osi z, t. j. že obrazová rovina je rovnobežná s rovinou xy. V praxi to tak však nebýva a kamera má v súradnicovom systéme, v ktorom sú definované zobrazované body, istú rotáciu a transláciu. V takom prípade rozlišujeme všeobecný súradnicový systém a súradnicový systém kamery (*world coordinate frame* a *camera coordinate frame*).

Ak $\widetilde{\mathbf{X}}$ je vektor súradníc bodu $\mathbf{X}$ vo všeobecnom súradnicovom systéme a vektor $\widetilde{\mathbf{X}}_{\mathrm{cam}}$ reprezentuje ten istý bod v súradnicovom systéme kamery, tak platí vzťah

$$\widetilde{\mathbf{X}}_{\mathrm{cam}} = \mathrm{R}(\widetilde{\mathbf{X}} - \widetilde{\mathbf{C}}),$$

kde $\widetilde{\mathbf{C}}$ sú súradnice stredu kamery vo všeobecnom súradnicovom systéme a R je rotačná matica 3×3 reprezentujúca orientáciu súradnicového systému kamery. To vedie k novému vyjadreniu projekčnej matice ako $\mathrm{P} = \mathrm{KR}\big[\mathrm{I}|-\widetilde{\mathbf{C}}\big]$, kde I je jednotková matica 3×3.

Parametre R a $\widetilde{\mathbf{C}}$ označujeme ako **vonkajšie parametre kamery**.

Zdrojom všetkých informácií, ktoré boli uvedené v tejto sekcii, je kniha *Multiple View Geometry in Computer Vision* od autorov R. Hartley a A. Zisserman [3]. Odtiaľ sú prevzaté aj obrázky.

2.2 Skreslenie kamery

[parametre skreslenia, výpočet]

2.3 Vykresľovací reťazec

[world space, view space, clip space, screen space, view matrix, projection matrix, viewport transform, vzťah k dierkovému modelu kamery]

2.4 Framework deck.gl a jeho nadstavba Pydeck

Pri vývoji webovej aplikácie pre vizualizáciu väčšieho množstva dát, u ktorej má vykresľovanie prebiehať na strane klienta, teda vo webovom prehliadači, sa hodí priamo či nepriamo použiť niektorý z frameworkov pre zobrazovanie dát v jazyku JavaScript.

Takým vhodným frameworkom je napríklad deck.gl, ktorý je určený na zobrazovanie veľkých sád dát. Je zameraný najmä na zobrazovanie geografických dát mapových podkladoch, ale hodí sa aj na iné typy dát. Vyznačuje sa vysokou presnosťou a výkonnosťou. Pre akceleráciu využíva rozhrania WebGPU a WebGL2 [4].

Vizualizácia dát v deck.gl sa skladá z dvoch základných častí:

- Vrstvy (`Layers`). Do vrstiev sa ukladajú zobrazované dáta. Framework deck.gl ponúka vyše 30 preddefinovaných typov vrstiev, ktoré zodpovedajú rôznym často sa vyskytujúcim typom dát. Pre túto prácu je významná najmä vrstva `PointCloudLayer`, ktorá je určená na zobrazenie mračna bodov, a vrstva `PathLayer`, ktorá je určená na zobrazenie trás.

- Pohľad (`View`). Definuje vlastnosti kamery, napríklad zorné pole a prednú a zadnú orezávaciu rovinu (*near plane* a *far plane*). Je možné buď určiť všetky tieto vlastnosti osobitne, alebo rovno zadať vypočítanú projekčnú maticu. Časť `ViewState` určuje polohu a orientáciu kamery, pričom u orientácie je možné určiť iba dva uhly (*bearing* a *pitch*). Typ pohľadu definuje spôsob interakcie vizualizácie s používateľom, napríklad pre zobrazenie trate z pohľadu strojvedúceho je ideálny typ `FirstPersonView`.

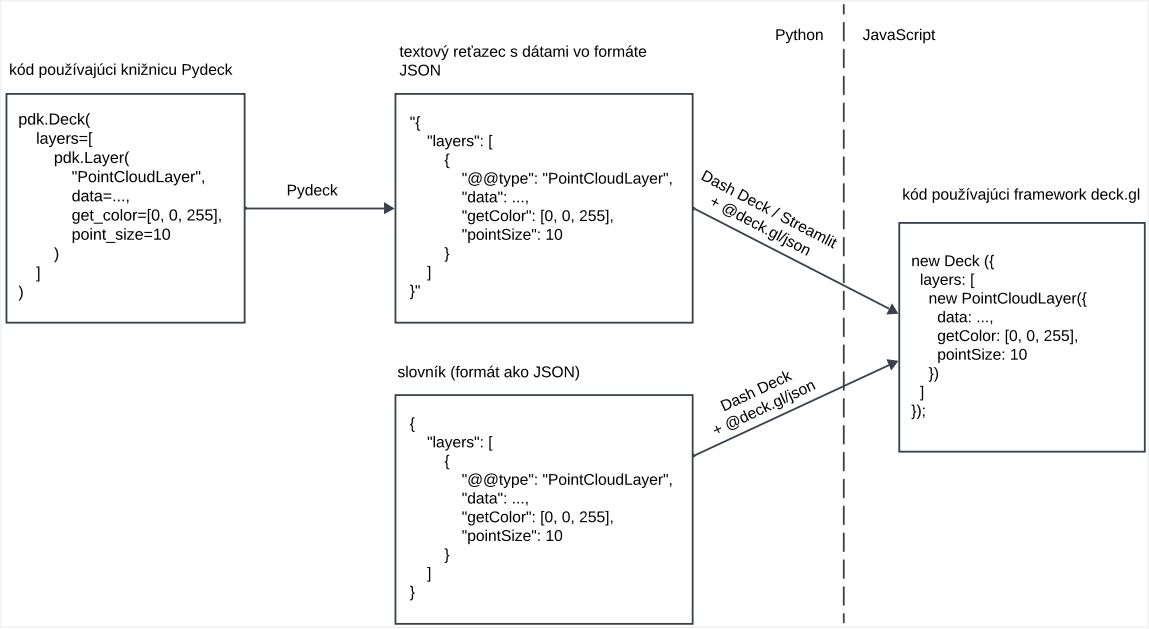
To, že sa deck.gl na vizualizáciu dát z mobilného mapovacieho systému naozaj hodí, dokazujú ukážky na jeho webových stránkach. Použiteľnosť vrstvy `PointCloudLayer` demonštruje plynulá animácia mračna vyše 800 000 bodov.

Na webových stránkach deck.gl sa nachádza aj galéria projektov vytvorených pomocou tohto frameworku. Medzi nimi je Autonomous Visualization System, toolkit pre vývoj webových aplikácií pre vizualizáciu dát z autonómnych vozidiel [9]. Na webových stránkach

tohto projektu je ukážka takej aplikácie, ktorá zobrazuje vozidlo, dáta z lidaru, vektorové dáta, ako napríklad trasu vozidla, a záznam z kamery. To je veľmi podobné aplikácii vyvíjanej v rámci tejto práce, ale je tu niekoľko rozdielov. Poprvé, mračno bodov nie je agregované, a teda sa v každom kroku animácie zobrazuje iba v tom momente nasnímaná časť, a podruhé, kamerový záznam sa zobrazuje oddelene, a nie na pozadí mračna bodov a vektorových dát.

Hoci je framework deck.gl primárne určený pre použitie v Javascripte, je možné ho použiť aj v jazyku Python, a to pomocou knižnice **Pydeck**. Tá je pomerne jednoduchá a podstatou jej činnosti je, že prevedie kód napísaný v jazyku Python do formátu JSON. Framework deck.gl má totiž modul @deck.gl/json, ktorý prijíma reprezentáciu vizualizácie vo formáte JSON a transformuje ju do javascriptového kódu (na definície funkcií a deck.gl objektov)[1].

Knižnica Pydeck je dobrým prostriedkom na vytvorenie jednoduchých vizualizácií, s ktorými môže používateľ interagovať pohybmi myši. Jej možnosti sú však oproti pôvodnému frameworku deck.gl veľmi obmedzené. Nie je vhodná na vytváranie zložitejších animácií s veľkým množstvom dát, pretože sa aj po tej najmenšej zmene musia dáta a definícia vizualizácie nanovo prevádzať do formátu JSON a následne na javascriptový kód (obrázok 2.4), čo je veľmi časovo náročné.



Obr. 2.4: Schéma vzťahov medzi technológiami Pydeck, Dash Deck a deck.gl a transformácií, ktorými prechádza definícia zobrazenia.

2.5 Nastavenie polohy kamery vo frameworku deck.gl

Súčasťou dodaných dát z mobilného mapovacieho systému boli rotácie a translácie kamery. Pre nastavenie polohy vo frameworku deck.gl existujú dve možnosti:

[1] Ukážka rozhrania modulu @deck.gl/json je na https://deck.gl/playground.

1. Aplikovať transformáciu na dáta a nechať kameru v počiatku súradnicového systému, prípadne v nejakom inom pevnom bode.

2. Nechať dáta v pôvodnom stave a aplikovať všetky transformácie iba na kameru.

Je zrejmé, že pre dosiahnutie rovnakého výsledku musí byť transformácia použitá v druhom prípade inverzná k tej, ktorá je použitá v tom prvom.

Bližším popisom a zhodnotením výhod a nevýhod týchto dvoch metód sa zaoberajú nasledujúce dve podsekcie.

Aplikovanie transformácií na dáta

Ako bolo zmienené v sekcii 2.4, dáta sa vo frameworku deck.gl členia do vrstiev. Každej vrstve je potom potrebné priradiť pole s dátami a definovať pre ňu takzvané prístupové funkcie (*data accessors*), ktoré určujú, akým spôsobom sa z poľa s dátami získa poloha a farba prvku. Napríklad u vrstvy `PointCloudLayer` je potrebné definovať funkciu `getPosition` pre získanie polohy bodu a `getColor` pre výpočet farby bodu [4].

Práve vo funkcii `getPosition` je možné aplikovať na polohu bodu ľubovoľnú transformáciu.

Napríklad v prípade, že je potrebné posunúť kameru v mračne bodov o 10 jednotiek v kladnom smere po osi x, by bolo možné vykonať túto transformáciu pomocou funkcie `getPosition` tak, že by sa posunul každý bod po osi x o 10 jednotiek v zápornom smere. V prípade, že by bol v poli dát každý bod vo formáte `[x, y, z, intenzita]`, by potom funkcia `getPosition` vyzerala takto:

```
function getPosition(d) {
  return [d[0] - 10, d[1], d[2]];
}
```

Výhodou tohto prístupu je, že je možné vykonať ľubovoľnú transformáciu, pretože do funkcie možno napísať akýkoľvek kód.

Veľkou nevýhodou je časová náročnosť, pri zmene polohy kamery sa totiž musí prepočítať poloha každého bodu. Tieto výpočty sa vykonávajú na procesore a výsledky sa následne nahrávajú na grafickú kartu. To konštatuje aj samotná dokumentácia frameworku deck.gl, kde sa píše, že kľúčom k tvorbe výkonných aplikácií je minimalizácia aktualizácií vrstiev, pri ktorých dochádza k prepočítavaniu dát a ich opätovnému nahrávaniu na grafickú kartu. Taktiež je tam uvedené, že prístupové funkcie by mali byť čo najtriviálnejšie, pretože sa počítajú pre každú položku v dátach, t. j. pre každý bod v mračne bodov, a teda sa každé pridanie operácií naviac výrazne prejaví na výkonnosti. 99% procesorového času venovaného aktualizácii dát vrstvy sa strávi práve volaním prístupových funkcií [5].

Aplikovanie transformácií na kameru

Meniť polohu a orientáciu kamery je v deck.gl možné pomocou troch parametrov: `position`, `bearing` a `pitch` [4]. To sú rotácie iba podľa dvoch osí, a teda nie je možné dosiahnuť všeobecnú rotáciu – chýba možnosť nastaviť rotáciu okolo tretej osi, takzvaný *roll* uhol.

V dokumentácii deck.gl je popísaná aj trieda `Viewport`, u ktorej je možné namiesto parametrov `position`, `bearing` a `pitch` nastaviť maticu pohľadu `viewMatrix`, čo znamená možnosť použiť akúkoľvek rotáciu. Problém je však v tom, že sa v dokumentácii už nikde nepíše, ako túto triedu použiť. Ide teda o nezrovnalosť, pravdepodobne pozostatok

z predchádzajúcich verzií deck.gl, kde táto možnosť bola, ale medzičasom bola zrušená. V aktuálnej verzii je možné túto maticu už iba prečítať, ale nie nahradiť vlastnou.

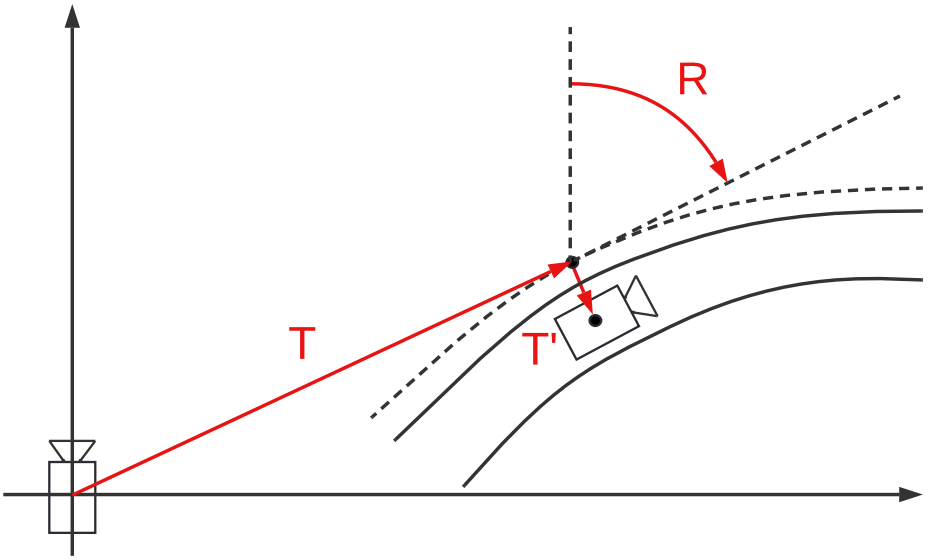
Nevýhodou tohto spôsobu nastavenia polohy kamery teda je, že sa nedá nastaviť *roll* uhol. Tento problém je však možné vyriešiť trikom, a to otočením HTML elementu `canvas`, do ktorého deck.gl vykresľuje výsledné zobrazenie, pomocou CSS vlastnosti `transform`.

Za poznámku tiež stojí, že ak potrebujeme na kameru aplikovať nejakú zložitejšiu transformáciu než iba jednu rotáciu a posunutie, tak je potrebné odvodiť vzorce, ktorými sa táto zložitejšia transformácia prevedie na parametre `position`, `bearing` a `pitch`. Konrétnym prípadom, ktorý bolo potrebné vyriešiť v tejto práci, sa zaoberá nasledujúca kapitola.

Veľkou výhodou naopak je, že na rozdiel od predchádzajúceho spôsobu nedochádza k žianym aktualizáciám vrstiev, teda nie je potrebné prepočítavať polohy bodov na procesore ani ich znova nahrávať na grafickú kartu. Tým sa výrazne zvýši výkonnosť.

Upresnenie polohy kamery

Experimentmi s dodanými dátami sa zistilo, že ak je mračno bodov zobrazené presne podľa dodaných údajov o polohe a parametroch kamery, výsledné zobrazenie nesedí presne na záber z kamery. Aby naozaj sedelo, je potrebné pridať isté posunutie kamery. Taká situácia je zjednodušene znázornená na obrázku 2.5 – je zadaná translácia T a rotácia R, ale kameru je ešte potrebné posunúť doprava transláciou T'.



Obr. 2.5: Zjednodušený príklad situácie, kedy je potrebné kombinovať viacero transformácií kamery. Je k dispozícii translácia T a rotácia R, ale kameru je ešte potrebné posunúť doprava transláciou T', aby bola v strede koľajníc.

Ak by bolo pri implementácii v deck.gl možné použiť aplikovanie transformácií na dáta, mal by tento problém jednoduché riešenie: z translácie T a rotácie R sa zloží transformačná matica, ktorá sa použije v prístupovej funkcii, a translácia T' sa priamo použije ako poloha kamery.

Ak však tento prístup nie je vhodný, napríklad z dôvodu horšieho výkonu, je nutné nájsť transláciu T_V a rotáciu R_V tak, aby platil vzťah

$$TRT' = T_V R_V.$$

Transformácie T_V a R_V sa potom už totiž dajú priamo previesť na parametre `position`, `bearing` a `pitch` (na prevod rotácie na `bearing` a `pitch` sa dá použiť napríklad trieda `Rotation` z knižnice `scipy`).

Nech

$$T = \begin{pmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T' = \begin{pmatrix} 1 & 0 & 0 & t'_1 \\ 0 & 1 & 0 & t'_2 \\ 0 & 0 & 1 & t'_3 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$T_V = \begin{pmatrix} 1 & 0 & 0 & t_{v1} \\ 0 & 1 & 0 & t_{v2} \\ 0 & 0 & 1 & t_{v3} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{a} \quad R_V = \begin{pmatrix} r_{v11} & r_{v12} & r_{v13} & 0 \\ r_{v21} & r_{v22} & r_{v23} & 0 \\ r_{v31} & r_{v32} & r_{v33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Potom

$$TRT' = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t'_1 r_{11} + t'_2 r_{12} + t'_3 r_{13} + t_1 \\ r_{21} & r_{22} & r_{23} & t'_1 r_{21} + t'_2 r_{22} + t'_3 r_{23} + t_2 \\ r_{31} & r_{32} & r_{33} & t'_1 r_{31} + t'_2 r_{32} + t'_3 r_{33} + t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

a

$$T_V R_V = \begin{pmatrix} r_{v11} & r_{v12} & r_{v13} & t_{v1} \\ r_{v21} & r_{v22} & r_{v23} & t_{v2} \\ r_{v31} & r_{v32} & r_{v33} & t_{v3} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Z toho je vyplýva, že musí platiť $R_V = R$ a

$$\begin{pmatrix} t_{v1} \\ t_{v2} \\ t_{v3} \end{pmatrix} = \begin{pmatrix} t'_1 r_{11} + t'_2 r_{12} + t'_3 r_{13} + t_1 \\ t'_1 r_{21} + t'_2 r_{22} + t'_3 r_{23} + t_2 \\ t'_1 r_{31} + t'_2 r_{32} + t'_3 r_{33} + t_3 \end{pmatrix} = \begin{pmatrix} r_{v11} & r_{v12} & r_{v13} \\ r_{v21} & r_{v22} & r_{v23} \\ r_{v31} & r_{v32} & r_{v33} \end{pmatrix} \begin{pmatrix} t'_1 \\ t'_2 \\ t'_3 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}.$$

Tým sme našli hľadanú transláciu T_V a rotáciu R_V.

2.6 Frameworky pre tvorbu webových aplikácií

Keďže hlavným cieľom práce bolo vytvoriť aplikáciu v jazyku Python, a to ideálne webovú, bolo potrebné preskúmať existujúce technológie, ktoré to umožňujú.

Porovnanie frameworkov Streamlit a Dash

Streamlit a Dash sú frameworky, ktoré majú rovnaké zameranie: oba slúžia na tvorbu webových aplikácií pre prácu s dátami (*data apps*) v jazyku Python. Dash je oproti Streamlitu na nižšej úrovni abstrakcie, pretože sám o sebe nemá žiaden vizuálny štýl a mnohé jeho komponenty sa priamo mapujú na HTML elementy, napríklad `dash.html.Div` a `dash.html.H1` [6, 7].

Oba frameworky majú podporu pre Pydeck, u Streamlitu je priamo k dispozícii element `st.pydeck_chart` a Dash má na tento účel vytvorenú prídavnú knižnicu **Dash Deck**.

Ukázalo sa však, že `st.pydeck_chart` podporuje iba pohľad `MapView`, ktorý je určený na zobrazenie dát na mape a nedá sa použiť na perspektívne zobrazenie bodov v trojrozmernom priestore. Preto je pre účely tejto práce element `st.pydeck_chart` prakticky nepoužiteľný.

Dash Deck má navyše tú výhodu, že umožňuje vynechať Pydeck a definovať zobrazenie iba pomocou slovníkov so štruktúrou zodpovedajúcou tej, ktorú vyžaduje modul @deck.gl/json, čo je tiež znázornené na obrázku 2.4. To trochu zefektívni vykonávanie zmien vo vizualizácii, keďže taká reprezentácia umožní jednoduchšie vykonávanie úprav.

Z týchto dôvodov bol pre implementáciu zvolený framework Dash, ktorý je podrobnejšie popísaný v nasledujúcej podsekcii.

Popis frameworku Dash

Dash je framework, ktorý umožňuje tvorbu webových aplikácií v jazyku Python. Vytvorený kód aplikácie sa spúšťa v rámci HTTP servera a generuje HTML dokument a skripty v jazyku JavaScript, ktoré server odošle klientovi. Tieto skripty následne komunikujú sa serverom, čím sa zabezpečuje funkcionalita aplikácie. Server je bezstavový a neuchováva si žiadne informácie o klientoch.

Kód webovej aplikácie napísanej s využitím frameworku Dash sa skladá z dvoch základných častí: komponentov a callbackov.

Komponenty sú prvky používateľského rozhrania, ktoré sa skladajú do stromovej štruktúry. Je možné použiť komponenty priamo zodpovedajúce HTML elementom (Dash HTML Components), komponenty z knižnice Dash Core Components (napríklad `Graph`, `Input`, `Tabs`, `Upload`) a ďalšie špeciálne komponenty [6].

Pre túto prácu je významný komponent `Store` z knižnice Dash Core Components, ktorý umožňuje uložiť dáta na strane klienta. Hodí sa na uloženie dát, ktoré sa následne zobrazujú pomocou frameworku deck.gl.

Ďalej je možné použiť knižnicu **Dash Bootstrap Components** (DBC), ktorá poskytuje ďalšie komponenty, ikony, lepšie možnosti pre rozloženie stránky, a v neposlednom rade vizuálne štýly [2].

Tam, kde nestačia štýly a možnosti rozloženia stránky z DBC, je možné u komponentov dodefinovať vlastné pravidlá v jazyku CSS.

Callbacky vytvárajú funkcionalitu aplikácie. Každý callback je funkcia, ktorá má vstupy a výstupy. Vstupy a výstupy sú vždy atribúty konkrétnych komponentov. Callback sa zavolá na začiatku behu aplikácie (táto vlastnosť sa dá zrušiť) a potom vždy, keď sa zmení niektorý z jeho vstupov. Callback môže mať aj špeciálne vstupy typu `State`, ktorých zmena callback nespustí. Platí obmedzenie, že každý atribút komponentu môže byť výstupom maximálne jedného callbacku [6]. U vstupov žiadne obmedzenia nie sú.

Existujú dva typy callbackov.

- Obyčajné callbacky. Sú napísané v jazyku Python. Klient má iba informáciu o ich vstupoch a výstupoch. Keď sa zmení niektorý zo vstupov, klient pošle na server požiadavok obsahujúci hodnoty všetkých vstupov. Server spustí kód callbacku a pošle klientovi odpoveď, ktorá obsahuje hodnoty všetkých výstupov.

- Klientske callbacky. Sú napísané v jazyku JavaScript. Klient má k dispozícii celý kód callbacku. Keď sa zmení niektorý zo vstupov, klient vykoná kód callbacku bez akejkoľvek komunikácie so serverom.

Je zrejmé, že klientske callbacky sú efektívnejšie než tie obyčajné, pretože ich nespomaľuje réžia komunikácie so serverom. Najvýraznejšie sa to prejaví vtedy, keď vstupy alebo výstupy obsahujú veľké množstvo dát.

K aplikácii vytvorenej pomocou frameworku Dash je možné pridať vlastné štýlové predpisy a skripty alebo moduly v jazyku JavaScript. Je potrebné uložiť ich do priečinka s názvom `assets` v koreňovom priečinku aplikácie. Rovnako je nutné postupovať aj pri vkladaní obrázkov a videí. Všetky štýlové predpisy, skripty a moduly z priečinka `assets` Dash automaticky načítava a pripája k aplikácii [6].

Kapitola 3

Návrh aplikácie

Cieľom tejto práce bolo vytvoriť používateľskú aplikáciu. Výsledná aplikácia mala byť webová, aby ju bolo možné spustiť jednoducho pomocou webového prehliadača. Čo sa týka funkcionality, mala by spĺňať nasledujúce body:

- Zobrazenie dát z mobilného mapovacieho systému. Tieto dáta sú tvorené mračnom bodov z lidaru, kamerovým záznamom, údajmi o pohybe vlaku a ďalšími vektorovými dátami a mali by byť zobrazené z pohľadu strojvedúceho vlaku.
- Umožniť používateľovi vybrať si konkrétnu pozíciu vlaku alebo prehrať si animáciu pohybu vlaku s nastaviteľnou rýchlosťou.
- Umožniť používateľovi nahrať súbory s dátami na zobrazenie: súbor s mračnom bodov, video z kamery na čele vlaku, súbor s vektorovými dátami, textové súbory s údajmi o pohybe vlaku – pole translácií, pole rotácií a pole zodpovedajúcich časových razítok.
- Zobrazovať dva typy dát z lidaru – buď malé, postupne nasnímané kusy mračna bodov s časovými razítkami, alebo iba jedno celistvé mračno bodov, ktoré vzniklo ich spojením (u oboch typov ale ide o agregované dáta – všetky body sú v jednej spoločnej súradnicovej sústave). Umožniť používateľovi prepínať medzi týmito dvoma typmi.
- Poskytnúť používateľovi možnosti prispôsobenia zobrazenia, ako napríklad zmeny viditeľnosti jednotlivých vrstiev (mračno bodov, vektorové dáta, kamerový záznam) a základné nastavenia zobrazenia mračna bodov (rôzne farebné škály podľa intenzity, veľkosť a priehľadnosť bodov, zobrazenie bodov len do určitej vzdialenosti) aj vektorových dát (farba a hrúbka čiar).
- Mať prepínač na zapnutie a vypnutie skreslenia mračna bodov a vektorových dát podľa parametrov skreslenia kamery.
- Zobrazovať prejazdný profil vlaku v predikovanej polohe v rôznych vzdialenostiach pred vlakom – 25, 50, 75 a 100m. Zobrazovať aj čiaru spájajúcu predikované polohy. (Súbory s predikovanými polohami sú súčasťou dát, ktoré aplikácia načítava.)

3.1 Návrh používateľského rozhrania

Pri návrhu používateľského rozhrania aplikácie bolo prioritou zobrazenie vizualizácie ako hlavnej časti aplikácie na čo najväčšej ploche a tiež tak, aby bola viditeľná vo všetkých stavoch.

Z ovládacích prvkov sú za najdôležitejšie považované prvky pre zmenu polohy vlaku, ktoré sú umiestnené v spodnom paneli. Všetky ostatné ovládacie prvky sú skryté v bočnom paneli, ktorý sa dá vysunúť tlačítkom v ľavom hornom rohu, a sú rozdelené do troch záložiek – nahrávanie dát, prispôsobenie zobrazenia a samostatná záložka pre prejazdný profil. Celkový návrh vzhľadu je na obrázku 3.1.

Hlavnou ideou návrhu je používanie čo najviac štandardných prvkov, aby sa v aplikácii používatelia ľahko zorientovali. Tomu napomáha aj použitie knižnice Bootstrap, ktorá je u webových aplikácií veľmi často používaná, a preto budú jej prvky pravdepodobne pre používateľov intuitívne.

Pre používateľskú prívetivosť by mala aplikácia spĺňať nasledujúce body:

1. všeobecné vlastnosti:
    - intuitívnosť,
    - responzívny design,
    - hladký beh animácií,
    - možnosť exportu a importu nastavení zobrazenia,
    - svetlý aj tmavý režim,
    - kompatibilita s rôznymi prehliadačmi, prípadnú nekompatibilitu používateľovi ohlásiť,
2. v záložke „Dáta“:
    - prehľadné zobrazenie, aké dáta boli nahrané, možnosť ich nahradenia inými dátami,
    - zobrazovanie indikátoru, že práve prebieha nahrávanie dát.

[obrázok: farebné škály pre mračno bodov]

3.2 Problém nahrávania dát do aplikácie

Dáta, ktoré má vyvíjaná webová aplikácia zobrazovať, sú rozdelené do väčšieho množstva súborov rôznych typov. Nahrávať ich do aplikácie klasickým spôsobom po jednom či po nejakých skupinách by bolo síce možné, ale pre používateľov časovo náročné a veľmi nepraktické.

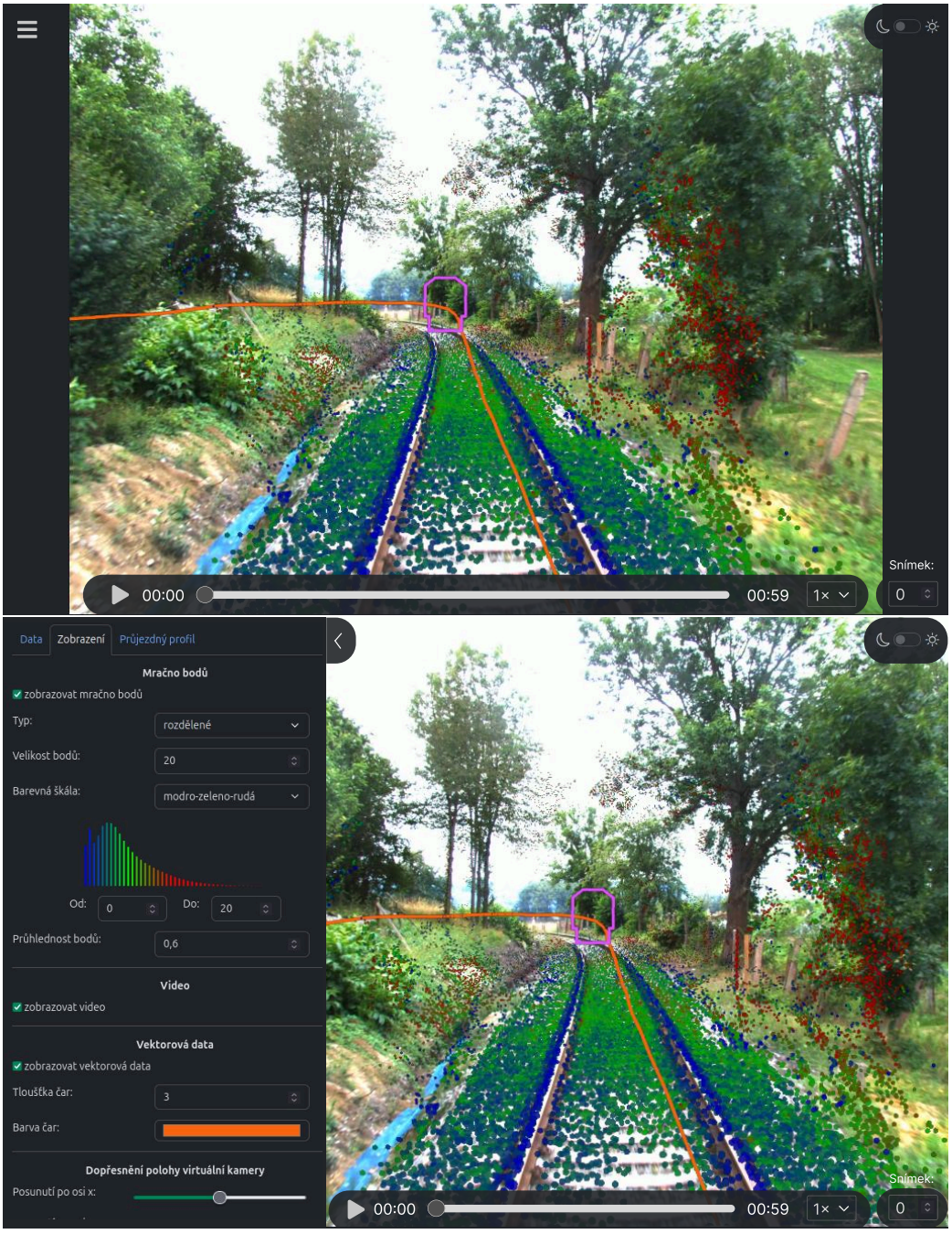
Keďže je aplikácia určená úzkej cieľovej skupine používateľov, môžeme si dovoliť predpokladať, že používateľ bude buď spúšťať server webovej aplikácie lokálne, alebo bude mať k serveru aspoň prístup. Za tohto predpokladu má problém s nahrávaním dát veľmi elegantné riešenie, a tým je **projektový súbor**, teda textový súbor, v ktorom sú špecifikované cesty k dátovým súborom uloženým na serveri. Používateľovi stačí nahrať tento súbor a aplikácia podľa neho automaticky nahrá všetky potrebné dátové súbory.

Presný formát projektového súboru bolo potrebné navrhnúť. Kompletný návrh je uvedený v prílohe [DOPLNIŤ]. Bol zvolený formát TOML, ktorý je jednoduchý, praktický a

dobre čitateľný [8]. U dátových súborov, ktorých môže byť rôzny počet, napríklad súbory s dátami z lidaru, sa do projektového súboru uvedie, v akom priečinku sa nachádzajú, ako sú pomenované a koľko ich je. Predpokladá sa, že sú tieto súbory jednotne pomenované a očíslované (napríklad `pcd_0.pcd`, `pcd_1.pcd`, ...).

Keby však bol projektový súbor jediným možným spôsobom nahrávania dát do aplikácie, malo by to isté nevýhody – napríklad pre vyskúšanie zmeny v jednom dátovom súbore by bolo potrebné meniť projektový súbor a znova ho nahrávať.

Preto by mala výsledná aplikácia kombinovať oba prístupy, projektový súbor aj klasické nahrávanie dát. Pre jednoduchosť a prístupnosť by mala umožniť nahranie základných súborov klasickým spôsobom. Naopak pre pokročilejšie použitie by mala mať možnosť nahrania projektového súboru, podľa ktorého by mala nahrať všetky špecifikované dátové súbory. Potom by mal používateľ prehľadne vidieť všetky nahrané súbory a mať možnosť tieto súbory jednotlivo ľubovoľne nahradzovať inými súbormi.



Obr. 3.1: Návrh používateľského rozhrania aplikácie vytvorený pomocou nástroja Figma.

Kapitola 4

Implementácia navrhnutej aplikácie

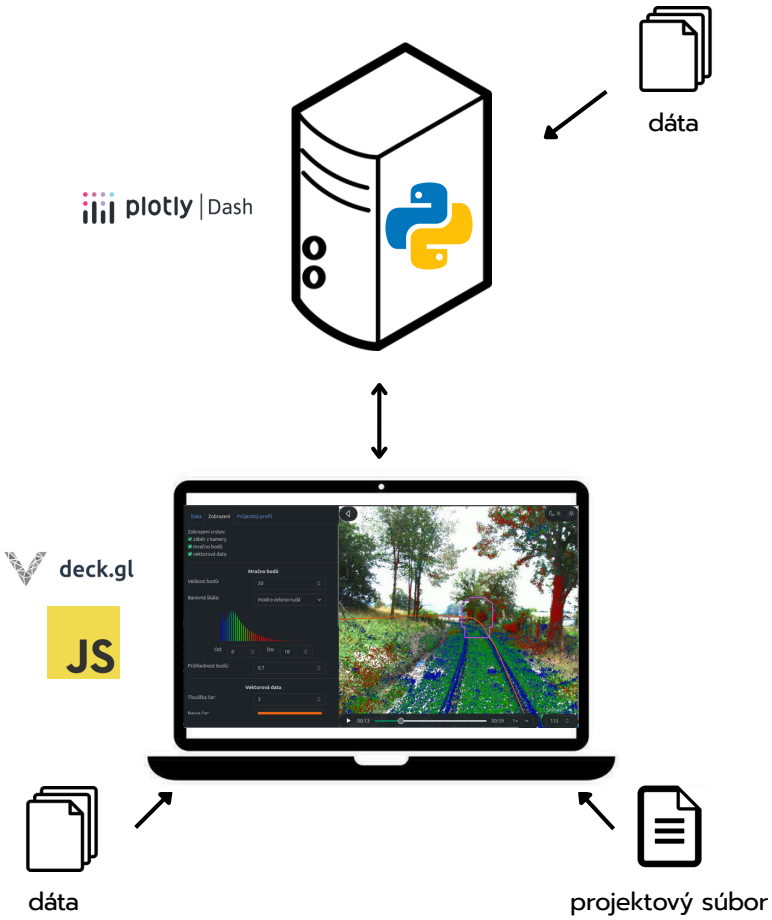
Navrhnutá webová aplikácia je implementovaná v jazyku Python s využitím frameworku Dash. Pre optimalizáciu obsahuje aplikácia aj skript napísaný v jazyku Javascript, ktorý používa pre vykresľovanie mračna bodov a vektorových dát framework deck.gl a na rozdiel od zvyšku kódu aplikácie sa vykonáva priamo na klientovi. Táto štruktúra je schematicky znázornená na obrázku 4.1.

Zdrojový kód je rozdelený do nasledujúcich súborov:

- `app.py` – Hlavný súbor, inicializuje a spustí aplikáciu. Definuje základné rozloženie aplikácie – vizualizácia na celú výšku okna, spodný panel a bočný panel so záložkami.
- `params.py` – Počiatočné parametre.
- `loading_functions.py` – Pomocné funkcie pre načítanie dát zo súborov.
- `general_functions.py` – Pomocné funkcie pre výpočet projekčnej matice, transformačných matíc a operácií s rotáciami.
- Súbory s komponentmi a callbackmi rozdelené podľa štruktúry používateľského rozhrania:
  - `animation_control_components.py`, `animation_control_callbacks.py` – komponenty v spodnom paneli,
  - `data_tab_components.py`, `data_tab_callbacks.py`, `visualization_tab_components.py`, `visualization_tab_callbacks.py`, `profile_tab_components.py`, `profile_tab_callbacks.py` – komponenty v záložkách v bočnom paneli.
- `visualization.js` – Vizualizácia mračna bodov a vektorových dát pomocou frameworku deck.gl, riadenie animácie phybu vlaku.

4.1 Prvá etapa implementácie

Prvým krokom implementácie bolo napísanie skriptu v jazyku Python, ktorý vykresľoval mračno bodov bez použitia špeciálnych knižníc či frameworkov. K dispozícii pritom bola

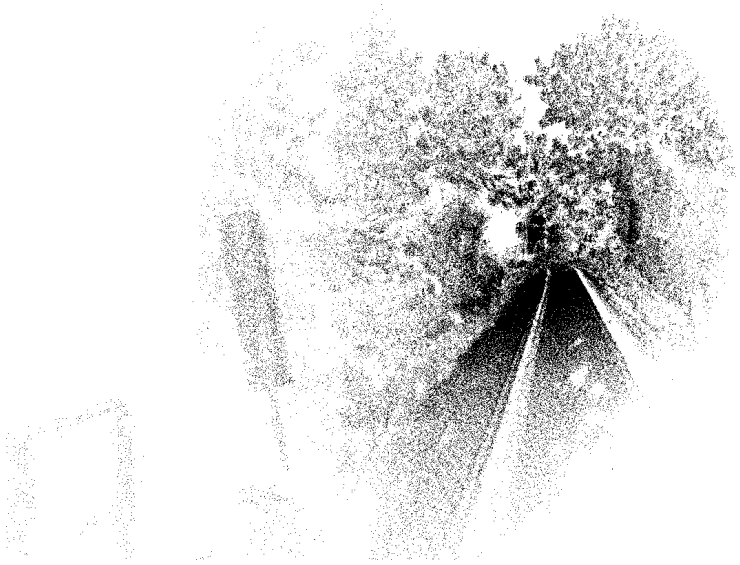


Obr. 4.1: Schéma finálnej implementácie.

jedna sada ukážkových dát, ktorá sa skladala z mračna bodov, kalibračnej matice a postupnosti translácií a rotácií, ktoré predstavovali pohyb vlaku po trati. Mračno bodov obsahovalo 201 880 bodov s intenzitou od 0 do 42 a bolo definovaných 500 polôh vlaku. K tomu boli dodané obrázky, ktoré ukazovali, ako by mal vyzerať výsledok zobrazenia (obrázok 4.2).

Cieľom tejto prvotnej fázy bolo najmä oboznámenie sa s dátami a základnými princípmi vykresľovania bodov. Výsledky, ktoré sa nakoniec podarilo dosiahnuť, boli podobné referenčným obrázkom, aj keď nie úplne identické. Ukázalo sa, že údaje o pohybe kamery majú iný súradnicový systém ako mračno bodov (líšilo sa poradie ôs) a že je potrebné pridať isté posunutie a rotáciu, aby sa výsledky podobali referenčným obrázkom (posunutie kamery do stredu vlaku, podľa dodaných dát bola naľavo od stredu).

Ďalej už nasledovala práca s existujúcimi knižnicami a frameworkami v jazyku Python. Prvotným plánom bolo použiť knižnicu Pydeck (pre vizualizáciu dát) s frameworkom Streamlit (pre tvorbu GUI). Pydeck bol zvolený preto, že je nadstavbou nad javascriptovým frameworkom deck.gl, ktorý pri vykresľovaní dát využíva hardwarovú akceleráciu (GPU) a má dobrú výkonnosť. Ukázalo sa však, že Streamlit nepodporuje Pydeck v plnej miere, a teda sa nedá využiť. Preto bol namiesto Streamlitu použitý framework Dash, ktorý už mal podporu pre Pydeck dostatočnú.



Obr. 4.2: Referenčný obrázok, ktorý bol dodaný na začiatku práce spolu s mračnom bodov a údajmi o pohybe kamery v ňom. Obrázok je tu pre lepšiu viditeľnosť s invertovanými farbami a zväčšeným kontrastom.

Pomocou tohto frameworku bola vytvorená jednoduchá webová aplikácia zobrazujúca ukážkové dáta a umožňujúca základné nastavenia vzhľadu. Do tejto aplikácie bolo zahrnuté aj zobrazovanie kamerového záznamu, pre tento účel bolo provizórne použité video z prejazdu tej istej trate natočené v roku 2012. Pri tom sa zistilo, že vykresľovanie dát pomocou kombinácie knižníc Pydeck a Dash Deck tak, ako je to vo vzorových príkladoch v dokumentácii knižnice Dash Deck, je značne neefektívne. Jedno vykreslenie mračna bodov zaberalo zhruba 3 sekundy, čo bolo spôsobené transformáciami medzi rôznymi formátmi, ako to bolo popísané v sekcii 2.4.

Tento čas sa podarilo významne zredukovať použitím knižnice Dash Deck bez knižnice Pydeck. Výsledok však stále nebol dosť dobrý na to, aby mohla hladko bežať animácia pohybu vlaku, a ani napriek rôznym pokusom sa ho už nepodarilo v rámci tejto kombinácie technológií vo významnejšej miere vylepšiť. Čas vykreslenia mračna bodov bol pritom úmerný počtu bodov a príčina neefektivity bola v spôsobe implementácie knižnice Dash Deck. Bolo navyše potrebné rátať s tým, že aplikácia bude zaťažená aj zobrazovaním videa. Preto bolo rozhodnuté optimalizovať vykresľovanie dát vynechaním knižnice Dash Deck a použitím frameworku deck.gl priamo z jazyka Javascript.

4.2 Optimalizácia vykresľovania mračna bodov

Pythonový framework Dash interne používa na vytváranie používateľského rozhrania JavaScript, konkrétne framework React [6]. Jeho tvorcovia rátali s tým, že vývojári, ktorí ho budú používať, budú niekedy chcieť priamo použiť javascriptový kód. Preto to tento framework umožňuje viacerými spôsobmi:

- Klientske callbacky. Podobajú sa tým štandardným, ale na rozdiel od nich bežia iba na klientovi a píšu sa v jazyku JavaScript. Štandardné callbacky bežia na serveri a píšu sa v Pythone.

- Skripty v jazyku JavaScript, ktoré je možné pripojiť k generovanej HTML stránke. Spustia sa automaticky pri otvorení stránky v prehliadači.
- Vytváranie vlastných komponentov používateľského rozhrania pomocou frameworku React.

V tejto práci bola pre jednoduchosť využitá prvá a druhá možnosť. Bol napísaný skript `visualisation.js`, v ktorom sú definované funkcie pre zobrazenie a zmeny zobrazenia dát pomocou deck.gl. Odkazy na tieto funkcie sú potom priradené do objektu `window`, aby bolo možné volať ich z klientskych callbackov. Tento objekt teda tvorí rozhranie skriptu `visualisation.js`, ktoré je využívané zvyškom kódu aplikácie.

Pre správne pripojenie zdrojových kódov frameworku deck.gl k javascriptovému kódu tak, aby mohol fungovať v rámci aplikácie napísanej vo frameworku Dash, je nutné použiť bundler. V rámci tejto práce bol použitý bundler Webpack. Výstupnému skriptu je potrebné nastaviť príponu `.mjs`, aby ho Dash spustil ako modul.

Meranie výkonnosti vykresľovania mračna bodov pred a po optimalizácii je popísané v tabuľkách 4.1 a 4.2.

Nastavenie parametra `interval` [ms]	Čas, za ktorý prebehla celá animácia (100 snímok) [s]	Priemerný čas vykreslenia jedného snímku [ms]
1000	101	1010
800	80	800
750	76	760
700	72	720
650	71	710
600	70	700

Tabuľka 4.1: Výkonnosť vykresľovania mračna bodov obsahujúceho 201 880 bodov **pred optimalizáciou**, teda iba pomocou kódu v jazyku Python. Použité technológie: Python, Dash, Dash Deck. Z nameraných údajov vyplýva, že minimálny čas potrebný vykreslenie jednej snímky je asi 750 ms, čo znamená, že rýchlosť nedosahuje ani 2 snímky za sekundu.

Nastavenie parametra `interval` [ms]	Čas, za ktorý prebehla celá animácia (500 snímok) [s]	Priemerný čas vykreslenia jedného snímku [ms]
250	126	252
200	101	202
150	76	152
125	68	136
100	63	126
75	62	124

Tabuľka 4.2: Výkonnosť vykresľovania mračna bodov obsahujúceho 201 880 bodov **po optimalizácii**, teda s vykresľovaním dát v jazyku Javascript. Použité technológie: Python, Dash, Javascript, deck.gl. Z nameraných údajov vyplýva, že minimálny čas potrebný vykreslenie jednej snímky je asi 150 ms, čo zodpovedá rýchlosti asi 6 snímok za sekundu.

Riadenie animácie pomocou komponentu `Interval` frameworku Dash je pomerne ťažkopádne, pretože funguje na základe komunikácie medzi klientom a serverom a tým animáciu spomaľuje. Preto bolo vhodné ho niečím nahradiť, napríklad riadením animácie v Javascripte iba na klientskej strane.

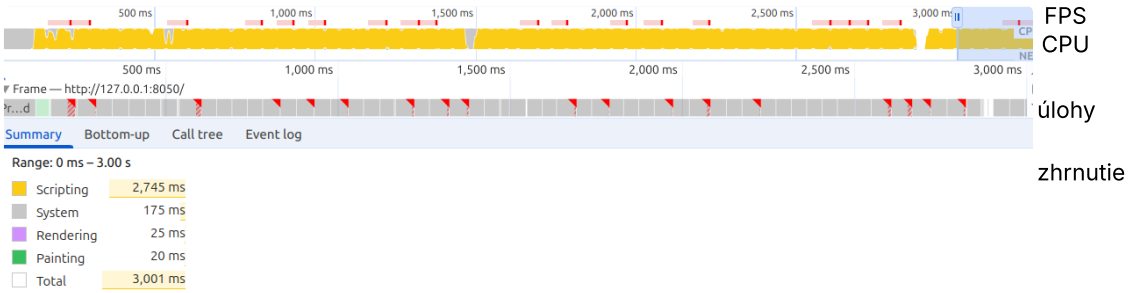
[odmerať a popísať, ako sa to vylepšilo]

4.3 Výber spôsobu nastavenia polohy kamery

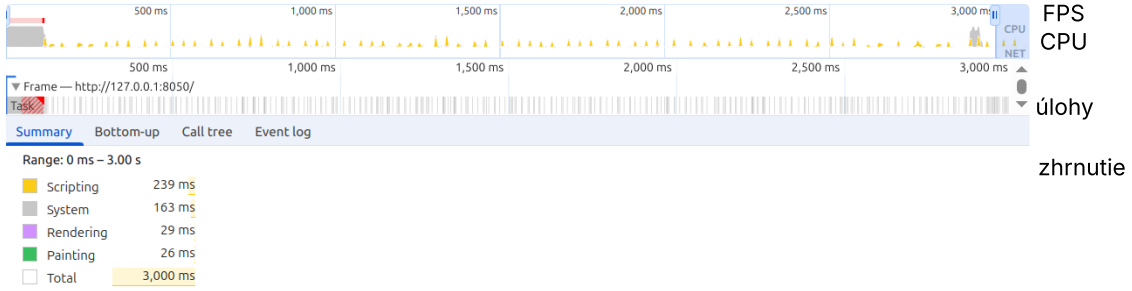
V sekcii 2.5 boli popísané dva rôzne spôsoby nastavenia polohy kamery vo frameworku deck.gl. V implementácii aplikácie bol použitý najprv prvý spôsob, teda aplikovanie transformácií na dáta, pretože bol jednoduchší a priamo vyplynul z prvej fázy implementácie (vykreslenie mračna bodov v jazyku Python bez použitia frameworku). Ten bol následne zmenený na druhý spôsob, teda aplikovanie transformácií na kameru, pretože sa ukázalo, že tak dôjde k významnej optimalizácii.

Nasledujúce merania prebiehali na notebooku s operačným systémom Ubuntu 24.04.1, procesorom Intel Core i5-9300H × 8, grafickou kartou Intel UHD Graphics 630 (CFL GT2) a veľkosťou operačnej pamäte 16 GiB, v prehliadači Google Chrome 133.0.6943.126 s rozlíšením obrazovky 1480 × 832. Bol použitý profiler vstavaný v tomto prehliadači.

V aplikácii sa prehrávala animácia pohybu kamery spojená s videom. Vizualizované dáta tvorila jedna vrstva `PointCloudLayer`, ktorá obsahovala 201 880 bodov.



Obr. 4.3: Profiling animácie pohybu kamery pred optimalizáciou. Transformácie sa aplikujú na dáta a počítajú sa na CPU.



Obr. 4.4: Profiling animácie pohybu kamery po optimalizácii. Transformácie sa aplikujú iba na kameru.

Z nameraných údajov vyplýva, že táto optimalizácia veľmi výrazne znížila zaťaženie procesora. Zatiaľ čo pred optimalizáciou (obr. 4.3) sa počas trojsekudového intervalu vykonával javascriptový kód 2,745 sekúnd, po optimalizácii (obr. 4.4) to bolo iba 239 milisekúnd, čo predstavuje viac než desaťnásobné zníženie.

Dôležitou metrikou pri posudzovaní výkonnosti animácií je snímková frekvencia meraná v FPS (*frames per second*, počet vykreslených snímkov za sekundu). V ideálnom prípade by sa snímková frekvencia mala blížiť 60 FPS, pretože vtedy sa animácia ľudskému oku zdá hladká [1].

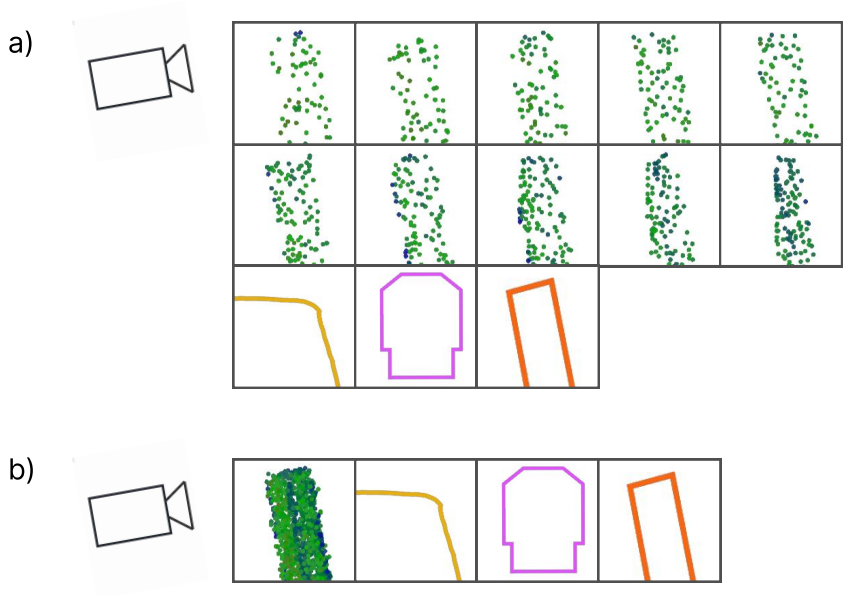
Podľa vstavaného ukazovateľa snímkovej frekvencie v prehliadači Google Chrome dosahovala skúmaná animácia pred optimalizáciu zhruba 28 až 34 FPS, zatiaľ čo po nej 51 až 55 FPS. Aj v tomto ohľade teda nastalo významné zlepšenie.

4.4 Rozdelenie dát do vrstiev

Pri vizualizácii dát pomocou frameworku deck.gl sú použité dva typy vrstiev:

- `PointCloudLayer` pre mračno bodov,
- `PathLayer` pre vektorové dáta.

Rozdelenie vizualizácie na vrstvy je znázornené na obrázku 4.5.



Obr. 4.5: Schéma rozdelenia dát do vrstiev. Počet vrstiev s mračnom bodov závisí od typu zobrazovaného mračna bodov.

Na začiatku tvorby práce bola na zobrazenie vektorových dát použitá vrstva `LineLayer`. Ukázalo sa však, že vstva `LineLayer` je prispôsobená iba pre zobrazovanie čiar na mapách a pri použití nemapového pohľadu ako `FirstPersonView` nefunguje dobre – niektoré čiary sa otáčajú kolmo ku kamere a tým pádom sa ich šírka zmenšuje, niekedy až tak, že úplne prestanú byť viditeľné. Toto správanie u vrstvy `LineLayer` nie je možné zmeniť. Naopak u vrstvy `PathLayer` je možné nastaviť vlastnosť `billboard`, ktorá znamená otočenie čiar smerom ku kamere, preto bola nakoniec táto vrstva použitá namiesto vrstvy `LineLayer`. `PathLayer` sa od `LineLayer` líši ešte tým, že čiary majú pevnú šírku v pixeloch a nezužujú sa podľa toho, ako ďaleko sú od kamery.

Vizualizácia obsahuje nasledujúce vrstvy:

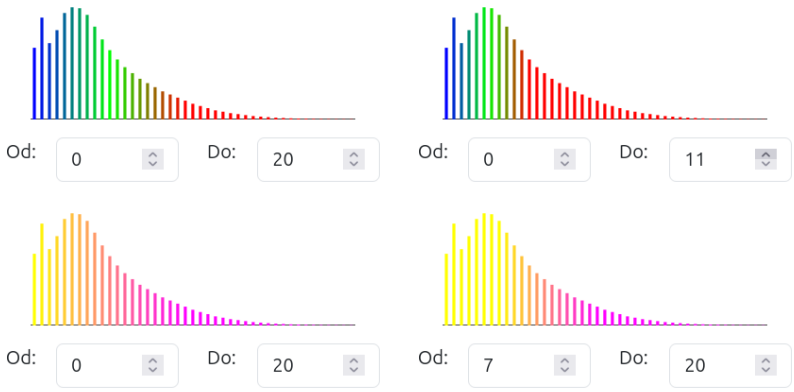
- Mračno bodov. Ak aplikácia zobrazuje *rozdelené* mračno bodov, teda malé, postupne nasnímané časti s časovými razítkami, tak je vrstiev desať a zobrazujú desať z týchto častí – jednu, ktorá zodpovedá danej pozícii, a deväť predchádzajúcich (obrázok 4.5, časť a)). Ak aplikácia zobrazuje *celistvé* mračno bodov, tak je celé v jednej vrstve (obrázok 4.5, časť b)).
- Čiara spájajúca predikované polohy prejazdného profilu vlaku.
- Prejazdný profil vlaku v predikovanej polohe. Od ostatných vektorových dát sa líši tým, že sa jeho poloha mení pri zmene polohy kamery. To je v implementácii dosiahnuté pomocou špeciálnej prístupovej funkcie `profileGetPath` (prístupové funkcie boli popísané v sekcii 2.5, podsekcia Aplikovanie transformácií na dáta).
- Ďalšie vektorové dáta.

Farebné škály

Okrem polohy má každý bod v mračne bodov v dodanej sade dát určenú ešte jednu vlastnosť – intenzitu v rozmedzí od 0 do 42, ktorú je vhodné farebne rozlíšiť.

Boli implementované dve farebné škály, klasická modro-zeleno-červená a žlto-fialová, pričom u oboch si môže používateľ zvoliť, od akej intenzity začína a pri akej končí. Ak je napríklad nastavená žlto-fialová farebná škála od 0 do 20, tak to znamená, že body s intenzitou 0 budú mať žltú farbu, body s intenzitou 20 a vyššou fialovú farbu a body s intenzitou od 1 do 19 kombinovanú farbu.

Pre prehľadnosť bol tiež implementovaný graf, ktorý používateľovi ukazuje relatívne počty bodov s jednotlivými intenzitami a aké majú nastavené farby. Ukážky možných nastavení sú na obrázku 4.6.



Obr. 4.6: Nastaviteľná farebná škála pre mračno bodov. Vodorovná os predstavuje intenzitu, vertikálna os predstavuje počet bodov.

4.5 Implementácia animácie pohybu vlaku

[obrázok prepisovania vrstiev]

4.6 Zobrazenie prejazdného profilu vlaku

4.7 Implementácia skreslenia

Kapitola 5

Testovanie

5.1 Zobrazenie dodaných dát

5.2 Výkonnosť

5.3 Vyhodnotenie používateľskej prívetivosti

Kapitola 6

Záver

Literatúra

[1] Basques, K. Analyze runtime performance. *Chrome for developers* online. Dostupné z: https://developer.chrome.com/. [cit. 2025-04-14]. Path: Docs; DevTools; Performance.

[2] Faculty Science Ltd. *Dash Bootstrap Components* online. Dostupné z: https://dash-bootstrap-components.opensource.faculty.ai/. [cit. 2025-04-24].

[3] Hartley, R. a Zisserman, A. *Multiple View Geometry in Computer Vision*. 2. vyd. Cambridge: Cambridge University Press, 2003. 152-156 s. ISBN 978-0-521-54051-3.

[4] OpenJS Foundation. *Deck.gl* online. 2024. Dostupné z: https://deck.gl/. [cit. 2025-01-14].

[5] OpenJS Foundation. Performance Optimization. *Deck.gl* online. 2025. Dostupné z: https://deck.gl/. [cit. 2025-04-12]. Path: Docs; Developer Guide; Using the API.

[6] Plotly. *Dash Python User Guide* online. 2025. Dostupné z: https://dash.plotly.com/. [cit. 2025-01-17].

[7] Snowflake Inc. *Streamlit* online. 2024. Dostupné z: https://streamlit.io/. [cit. 2025-01-17].

[8] Preston-Werner, T.; Gedam, P. et al. *TOML v1.0.0* online. 1. novembra 2021. Dostupné z: https://toml.io/en/v1.0.0. [cit. 2025-04-11].

[9] Uber ATG. *Autonomous Visualization System* online. Dostupné z: https://avs.auto/. [cit. 2025-04-11].