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

VIZUALIZACE PRŮJEZDU ŽELEZNIČNÍ TRATĚ S ŘEŠENÍM VIDITELNOSTI

RAILWAY TRACK VISUALIZATION WITH FOCUS ON VISIBILITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JANA BRANDEJSOVÁ**
AUTHOR

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

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

164916

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Studentka:	**Brandejsová Jana**
Program:	Informační technologie
Název:	**Vizualizace průjezdu železniční tratě s řešením viditelnosti**
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. Pro efektivní zobrazení se zaměřte i na řešení viditelnosti objektů v mračně bodů.
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

Cílem této bakalářské práce je navrhnout intuitivní uživatelský systém v podobě mobilní aplikace pro vizualizaci agregovaných dat ze senzorů umístěných na čele vlaku. Aplikace umožňuje zpracování velkého množství dat v reálném čase a jejich zobrazení na přenosných zařízení běžného výkonu. Řešení se zaměřuje na viditelnost objektů v mračně bodů a jejich vykreslení. To zahrnuje transformaci každého bodu ze 3D prostoru do 2D roviny pomocí perspektivní projekce a úpravu výsledného obrazu eliminací zkreslení. Výsledkem práce je systém implementující uživatelské rozhraní, jehož vstupem je mračno bodů, matice pro výpočty transformace a snímky z průjezdu vlaku tvořící video reálné scény. Výstupem je animace promítající body z mračna bodů na pozadí videa s objekty reálného světa.

Abstract

The goal of this bachelor's thesis is to design an intuitive user system for visualizing aggregated data from sensors placed on the front of a train. The application enables real-time processing of large dataset and its visualization on standard-performance portable devices. The solution focuses on the visibility of objects in the point cloud and their visualization. This includes transforming each point from 3D space into a 2D plane, using perspective projection and refining the result image by eliminating distortion. The result of the thesis is a system implementing a user interface, where the input consists of a point cloud, transformation matrices, and a recording with real-world imagery. The output is an animation of points from the point cloud projected on the background of the recording with real-world objects.

Klíčová slova

Mračno bodů, dírková komora, viditelnost bodů, Android, Python, Kivy, vícevláknová aplikace.

Keywords

Point cloud, pinhole camera model, points visibility, Android, Python, Kivy, multithreaded application.

Citace

BRANDEJSOVÁ, Jana. *Vizualizace průjezdu železniční tratě s řešením viditelnosti.* Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Ondřej Klíma, Ph.D.

Vizualizace průjezdu železniční tratě s řešením viditelnosti

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Ondřeje Klímy, Ph.D. Zpracovávaná data mi poskytlo AŽD Praha s.r.o. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....

Jana Brandejsová

7. května 2025

Obsah

1 Úvod **4**

2 Postup zpracování mračna bodů a vhodné nástroje **5**
2.1 Vizualizace dat z mobilních mapovacích systémů 5
2.2 Princip modelu dírkové komory 7
2.3 Viditelnost bodu v mračně bodů 9
2.4 Python knihovny pro práci s mračny bodů 10
2.5 Multiplatformní Python knihovny pro tvorbu uživatelského rozhraní 12

3 Návrh uživatelského systému **17**
3.1 Požadované funkce systému 17
3.2 Předpokládané nároky uživatelů 19

4 Implementace **21**
4.1 Projekce bodů z prostoru do roviny 22
4.2 Viditelnost bodů v rovině po projekci 23
4.3 Vizualizace bodů 25
4.4 Implementace pozadí ze snímků kamery 28
4.5 Synchronizace bodů a obrazu na pozadí 29
4.6 Načítání souborů z paměti zařízení 31
4.7 Funkce implementované pro zvýšení komfortu uživatele 40

5 Testování **43**
5.1 Funkční testování 43
5.2 Uživatelské testování 44

6 Navržená rozšíření pro budoucí vývoj **47**
6.1 Optimalizace rychlosti a využití GPU 47
6.2 Online streamování dat 48
6.3 Rozšíření uživatelského rozhraní 49
6.4 Vizualizace vektorových dat 50

7 Závěr **51**

Literatura **52**

A Vysvětlení zkratek **54**

Seznam obrázků

2.1 Možnosti využití dat získaných mobilním mapováním [12]. 6
2.2 Vůz EDITA - rozmístění kapes pro snímací zařízení [1]. 7
2.3 Vůz EDITA na trati Dolní Bousov – Kopidlno [1]. 7
2.4 Projekce bodu X na obrazovou rovinu. Vlevo je zobrazeno uspořádání se středem kamery C, hlavní osou, a obrazovou rovinou. Vpravo je vysvětlení výpočtu pro souřadnice bodu v obrazové rovině [6]. 7
2.5 Sjednocení souřadnicových systémů za pomoci rotace R a translace t [6]. . . 8
2.6 Projekce bodů ze dvou objektů, které se nacházejí v prostoru za sebou [17]. 9
2.7 Set bodů $\mathcal{N}(p)$ velikosti 6 [17]. 9
2.8 Závislost hloubky při výběru nejbližších bodů [17]. 9
2.9 Mračno bodů vykresleno knihovnou Open3D. 11
2.10 Nástroj Napari [13]. 14
2.11 Aplikace Travel tips [23]. 14
2.12 Hra Deflectouch [5]. 15
2.13 Aplikace MathPath Console [10]. 15

3.1 Zobrazení dat přes celou obrazovku se skrytými nástroji. 18
3.2 Návrh rozložení ovládacích prvků aplikace. 20

4.1 Čas strávený na projekci snímků včetně výpočtu viditelnosti přesnější metodou odhadem viditelnosti. 24
4.2 Čas strávený na projekci snímků včetně výpočtu viditelnosti odstraněním duplicitních bodů. 25
4.3 Aplikace při Buffering vykreslování se stavovým ukazatelem. 28
4.4 Zobrazení odpovídajících si snímků bodů a pozadí díky synchronizaci 31
4.5 Žádost o oprávnění za běhu aplikace při prvním pokusu o přístup k úložišti. 33
4.6 Otevření správce souborů MDFileManager v kořenovém adresáři. 34
4.7 Nastavení, které umožňuje udělit oprávnění pouze k mediím. 34
4.8 Otevření systémového správce souborů prostřednictvím SAF. 35
4.9 Aplikace po prvním spuštění, čekající na nahrátí souborů. 36
4.10 Úspěšné zobrazení prvního snímku nahraného videa. 38
4.11 Vykreslení prvního snímku z nahraných souborů. 39
4.12 Dialogové okno pro nastavení rychlosti animace. 40
4.13 Dialogové okno pro výběr položek zobrazených v části obrazovky Stav animace. 41
4.14 Zobrazení okna s uživatelskou nápovědou. 42

5.1 Čas potřebný k projekci i vykreslení jednotlivých snímků. 44
5.2 Úkol 1. Načtení konfiguračních dat. 45
5.3 Úkol 2. Změna barvy bodů. 45

5.4 Úkol 3. Nastavení informačních položek. 46
5.5 Úkol 4. Spuštění a restartování animace. 46
5.6 Úkol 5. Přetočení animace. 46

6.1 Návrh uživatelského rozhraní s parametry pro ovládání projekce. 49
6.2 Snímek z projekce s detekovanými kolejemi. 50

Kapitola 1

Úvod

Technologie se v dnešní době obzvlášť rychle zdokonalují a o autonomní dopravní prostředky je stále větší zájem. O to více se musí dbát na jejich bezpečnost a rychlé řešení problémů v krizových situacích, při kterých je důležité včas rozpoznat objekty tvořící překážky. Jedním z mnoha řešení může být použití mobilního mapování, kde se data sbírají pomocí speciálních kamer, ukládají například do mračen bodů a mohou být následně analyzována.

Cílem této práce je vytvořit uživatelský systém pro vizualizaci těchto nasnímaných dat. Systém, neboli aplikaci vyvíjenou pro přenosná zařízení, jako je tablet, případně mobil. Aplikace umožní uživateli prohlížet získaná data na zařízení, které je skladné, příruční a nevyžaduje stálé kabelové připojení do sítě nebo k periferiím. Zajišťuje tedy uživateli, že její použití bude možné z jakéhokoli požadovaného místa, a tím zvyšuje komfort. Aplikace poskytuje prohlížení mračna bodů zachycující průjezd úsekem železniční tratě v podobě posloupnosti jednotlivě generovaných snímků z úhlů pohledů nasnímaných kamerami. Uživatel si může spustit dané snímky jako video, nebo snímky prohlížet jednotlivě. Užitečnou funkcí je promítnutí snímků na fotografie reálného prostředí jako pozadí a mít možnost porovnat s objekty tvořenými body.

Uvedené kapitoly postupně provádějí celým vývojem aplikace. Nejdříve je potřeba vysvětlit používané termíny, popsat důležité výpočetní procesy zpracování dat a udělat přehled nástrojů, ze kterých je možné vybírat při implementaci. Následuje kapitola popisující návrh aplikace, jaké nástroje bude vhodné použít, uvádí možné funkce, které by mohla aplikace nabízet, a představuje prvotní návrh grafického uživatelského rozhraní. Kapitola implementace je stěžejní částí celé práce a popisuje zvolené řešení navržených funkcí, jakým problémům se čelilo a výsledná odvedená práce. Následuje testování, které ověřuje aplikaci z funkčního i uživatelského hlediska. Ke konci jsou uvedeny návrhy pro rozšíření aplikace při budoucím vývoji. Celá práce je ukončena závěrem, kde je zhodnocen průběh vývoje a vyzdviženy dosažené cíle.

Kapitola 2

Postup zpracování mračna bodů a vhodné nástroje

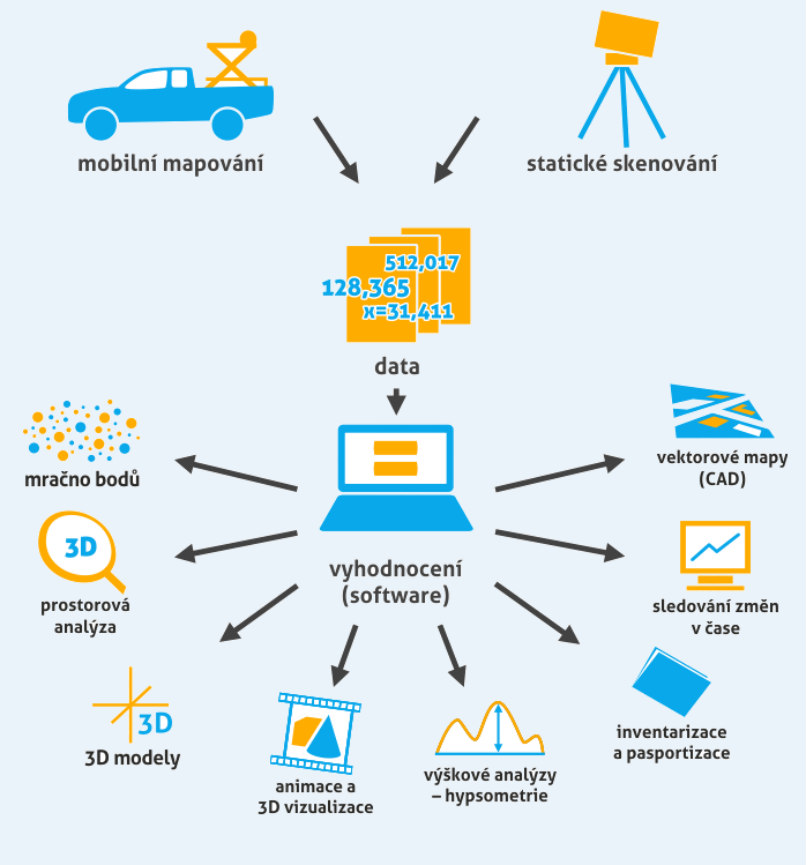
Zde uvedená kapitola podrobněji vysvětluje význam hlavních používaných termínů, kterým je potřeba rozumět. Další důležitou částí jsou popsané principy pro práci s mračnem bodů. Nejdříve je nutné říct, co to vlastně mračno bodů je, jaké má vlastnosti a proces zpracování a převedení dat, která uchovává. Ke konci se nachází část zaměřená na vytvoření přehledu o užitečných Python nástrojích a knihovnách. Představuje konkrétní nástroje, o kterých je dobré vědět, jelikož svými funkcemi umožňují značné zjednodušení při implementaci. Popisuji tam knihovny pro práci s mračnem bodů, jejich klíčové vlastnosti a některé ukázky nad konkrétními daty. V neposlední řadě podkapitola uvádí a porovnává možnosti různých nástrojů pro tvorbu grafického uživatelského rozhraní na základě podporovaných platforem.

2.1 Vizualizace dat z mobilních mapovacích systémů

Mobilní mapování je proces sběru geoprostorových dat z mobilních prostředků, jako jsou auta, drony, lodě a vlaky [24]. Mapovacími systémy se rozumí zařízení určená k bezkontaktnímu podrobnému měření. Zařízení obvykle sestává z vlastní mobilní platformy, mobilních laserových skenerů, digitálních kamer (video, černobílých, barevných, infračervených, multispektrálních, termálních), odometrů, přijímačů GNSS, měřicí jednotky a výpočetních prostředků, které slouží k synchronizaci činností všech složek systému a ke správě zaznamenaných dat [4]. Získaná data jsou následně zpracována do různých výstupů prostřednictvím vyhodnocovacích softwarů do formy, aby bylo možné s nimi dále co nejefektivněji pracovat, jak je vidět na obrázku 2.1. V případě této práce se budu zabývat daty pouze ve formě mračna bodů.

Mračno bodů

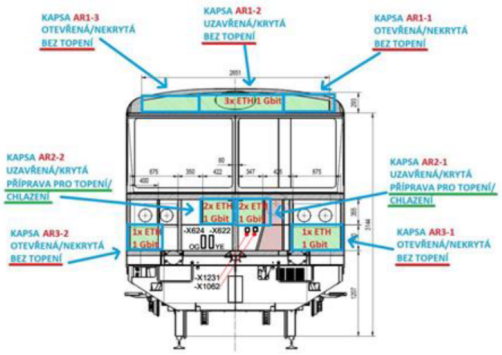
Mračno bodů je základním výstupem laserového skenování nebo fotogrammetrie. Jedná se o shluk tisíců až několika milionů bodů ve třírozměrném prostoru, v němž má každý bod mračna vlastní souřadnice XYZ. Jednotlivé body mračna mohou dále obsahovat informace o barvě (buď v odstínech šedi, nebo v celé škále RGB) [21].



Obrázek 2.1: Možnosti využití dat získaných mobilním mapováním [12].

AŽD – vůz EDITA

Veškerá data použitá v této práci poskytla společnost AŽD, která se věnuje výzkumu a vývoji nejmodernějších technologií zabezpečovací techniky a efektivity dopravy na železnici. Pro zlepšení optimalizace ekonomiky železniční dopravy, zvýšení bezpečnosti a řešení problémů s nedostatkem strojvedoucích se společnost AŽD rozhodla zapojit i do vývoje autonomní železnice. První měřící vůz měl nainstalovanou senzorickou konzoli pro detekci objektů, ale po čase se ukázala jistá omezení. Proto došlo k rozhodnutí vytvořit speciální vozidlo pro výzkum, a tak vznikl vůz EDITA – Experimentální Drážní vozidlo pro Inovativní Technologie AŽD. Obrázek 2.3 zachycuje vůz na trati Dolní Bousov – Kopidlno, což je vlastní trať společnosti AŽD určená pro experimentální účely. Na čelní části vozu je konstrukční řešení pro senzorické prvky určené pro autonomní řízení vlaku. To zahrnuje oddělené kapsy pro různé senzorické prvky (kamery s různou ohniskovou vzdáleností, Lidary pro měření vzdálenosti, termokamera...). Umístění kapes je znázorněno na obrázku 2.2. Vozidlo EDITA bude primárně využíváno pro testování a vývoj nových produktů AŽD v oblasti autonomní železnice. Patří sem systém rozpoznávání objektů a překážek, autonomní vedení vlaku bez nutné účasti člověka a další [1].



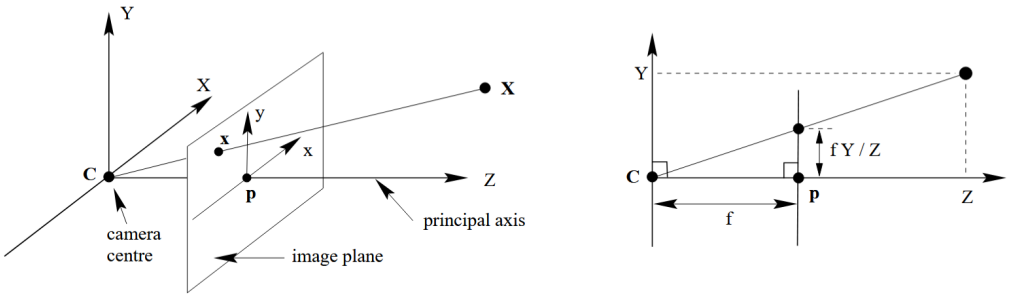
Obrázek 2.2: Vůz EDITA - rozmístění kapes pro snímací zařízení [1].

Obrázek 2.3: Vůz EDITA na trati Dolní Bousov – Kopidlno [1].

2.2 Princip modelu dírkové komory

Model dírkové komory je velmi jednoduchý matematický vztah popisující závislost mezi projekcí bodů v trojrozměrném prostoru a dvourozměrnou obrazovou rovinou. Každý bod v prostoru vyjádřený vektorem o třech složkách $[x, y, z]$, lze přepočítat na bod se souřadnicemi $[x, y]$, a tudíž je možné ho jednoduše promítnout do roviny.

Hlavní princip promítnutí bodu z prostoru do roviny vychází z centrální projekce bodů, kde středem projekce je počátek euklidovského souřadnicového systému znázorněného na obrázku 2.4. Střed projekce je také nazýván jako střed kamery nebo optický střed a je značen jako C. Rovina, na kterou jsou body z prostoru promítány, se nazývá obrazová rovina a je kolmá na osu procházející optickým středem, nachází se ve vzdálenosti f od středu a je umístěna před středem kamery. Bod, kde se obrazová rovina protíná s hlavní osou, se nazývá hlavní bod p.



Obrázek 2.4: Projekce bodu X na obrazovou rovinu. Vlevo je zobrazeno uspořádání se středem kamery C, hlavní osou, a obrazovou rovinou. Vpravo je vysvětlení výpočtu pro souřadnice bodu v obrazové rovině [6].

Převedení trojrozměrného bodu X se souřadnicemi $[x, y, z]$ do obrazové roviny spočívá ve spojení středu kamery s bodem X čárou, která protne obrazovou rovinu. V tomto místě vzniká bod x s novými dvourozměrnými souřadnicemi $[x, y]$ pro obrazovou rovinu. Přesný výpočet je zapsán vzorcem 2.1.

$$(X, Y, Z)^T \rightarrow \left(\frac{fX}{Z}, \frac{fY}{Z} \right)^T \tag{2.1}$$

Centrální projekce pomocí homogenních souřadnic

Pokud jsou body z prostoru reálného světa prezentovány homogenními vektory, projekce na body do roviny bude probíhat následovně.

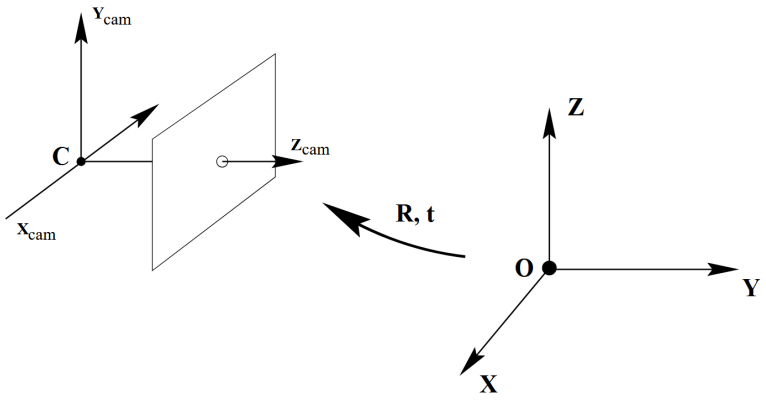
Zavádím notaci X pro bod prostoru reprezentovaný homogenním vektorem $(X, Y, Z, 1)^T$, dále x pro bod roviny reprezentovaný homogenním vektorem $(fx, fy, z)^T$. Projekci potom lze vyjádřit jako

$$x = KX \tag{2.2}$$

kde K je kalibrační matice kamery. Složky f_x a f_y jsou vzdálenosti kamery a určují, jak se bod prostoru mapuje na obrazovou rovinu. Složky c_x a c_y jsou souřadnice hlavního bodu obrazové roviny.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Dosud uvedené vztahy předpokládají, že kamera je umístěna v počátku Euklidovského souřadnicového systému s pohledem souběžným s hlavní osou systému. To však nemusí vždy platit, a proto je nutné souřadnicový systém kamery sjednotit s Euklidovským pomocí rotace R a translace t, jako je znázorněno na obrázku 2.5



Obrázek 2.5: Sjednocení souřadnicových systémů za pomoci rotace R a translace t [6].

Výsledný obecný vztah 2.4 popisuje projekci prostorového bodu do roviny a jednotlivé složky rozepisuje vztah 2.5 uvedený níže.

$$x = K[R \mid t]X \tag{2.4}$$

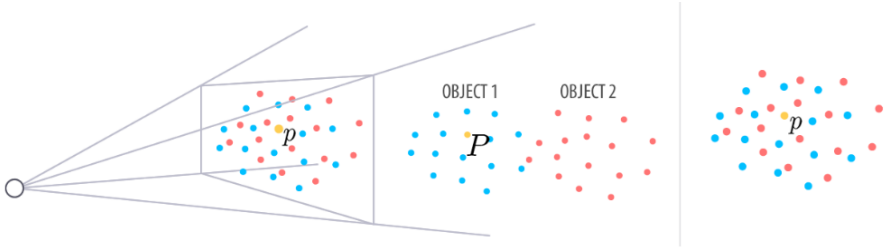
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.5}$$

2.3 Viditelnost bodu v mračně bodů

Při mobilním mapování se využívá dopravních prostředků, ke kterým je připevněna kamera, nebo jiné zaznamenávací zařízení, snímající okolí v pohybu. S každým posunem kamery dopředu se před ní zobrazují stále nové objekty, které byly do té doby celé, nebo alespoň částečně skryté, a zároveň se některé objekty dostávají mimo zorné pole kamery, tedy vypadávají ze záběru. Při generování mračna bodů z těchto nasnímaných dat se všechny body sloučí do jednoho, nebo více souborů, kde se předpokládá, že každý soubor obsahuje nějakou část úseku z trasy kamery. Z toho vyplývá, že z velké většiny míst v mračně bodů bude vidět mnohem víc bodů, než bylo vidět v daný okamžik z místa kamery, jelikož se v souboru nachází body z více míst. Tyto body se různě překrývají, promítá se jich víc do jednoho místa a zobrazují objekty, které by správně vidět být neměly. Z tohoto důvodu se provádí korekce a existují principy, které se snaží zobrazení těchto bodů eliminovat.

Metoda odhadu viditelnosti

Zde uvedená metoda pochází od francouzských autorů [17], kteří se zaměřili na problematiku viditelnosti v mračnech bodů s proměnnou hustotou neorganizovaných bodů. Tedy mračna bodů získaná právě mobilním mapováním městských prostředí a obecně venkovních prostor.

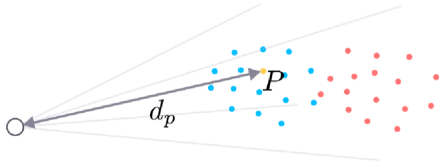


Obrázek 2.6: Projekce bodů ze dvou objektů, které se nacházejí v prostoru za sebou [17].

Při projekci bodů do 2D roviny z určitého úhlu pohledu se body z různých objektů mohou promítnout těsně vedle sebe, i když ve 3D modelu jsou hodně vzdálené, ale nacházejí se na stejné přímce za sebou. Tento jev znázorňuje obrázek 2.6, kde modré body tvoří první objekt a červené body objekt druhý, ale po projekci se body různě prolínají a objekty už nejsou zřetelně rozpoznatelné.

V tomto případě se pro výpočet pravděpodobnosti viditelnosti bodu využívá algoritmu k-nejbližších sousedů, anglicky označováno jako KNN vyhledávání (k-nearest neighbour search).

Obrázek 2.7: Set bodů $\mathcal{N}(p)$ velikosti 6 [17].



Obrázek 2.8: Závislost hloubky při výběru nejbližších bodů [17].

Na začátku je potřeba rozhodnout o počtu sousedních bodů z promítnuté roviny, které budou brány v potaz při výpočtu. Pro menší datasety jsou vhodná nízká čísla v řádu jednotek a pro velké datasety je vhodnější použít vyšší čísla, která zajistí vyhlazení a odstranění nechtěného šumu. Aplikováním KNN vyhledáváním na konkrétní bod z roviny vznikne set bodů $\mathcal{N}(p)$ o dříve definovaném počtu. Znázorněno na obrázku 2.7. K výběru bodů se využívá hloubky aktuálně porovnávaného bodu, která odpovídá euklidovské vzdálenosti mezi daným bodem a středem pohledu jako na obrázku 2.8. Tím je zajištěno, že jsou zvoleny body na stejné úrovni a nikoli ty, které se nacházejí v pozadí.

Konečně samotný výpočet pravděpodobnosti lze popsat vzorcem

$$\alpha_p = e^{-\frac{\left(d_p - d_p^{\min}\right)^2}{\left(d_p^{\max} - d_p^{\min}\right)^2}} \tag{2.6}$$

kde $d_p^{\min} = \min d_q \wedge q \in \mathcal{N}(p)$ odpovídá vzdálenosti nejbližšího bodu vzhledem ke středu pohledu a $d_p^{\max} = \max d_q \wedge q \in \mathcal{N}(p)$ naopak nejvzdálenějšímu bodu.

Pravděpodobnost tedy vyjadřuje $\alpha_p \in [0, 1]$. Pokud $\alpha_p = 0$, tak bod s největší pravděpodobností není viditelný a bod, pro který platí, že $\alpha_p = 1$, je určitě viditelný. Hranici v intervalu od 0 do 1, která bude určovat, jestli je konkrétní bod viditelný, či ne, je podle autorů této metody nejlepší stanovit jako průměrnou hodnotu viditelnosti všech bodů promítnutých v rovině. Pro jasné ohodnocení každého bodu je definována binarizace následovně

$$\hat{\alpha}_p = \begin{cases} 1 & \text{if } \alpha_p \geq \bar{\alpha}, \\ 0 & \text{jinak.} \end{cases} \tag{2.7}$$

s hodnotou $\bar{\alpha} = \frac{1}{\text{Card}(\Phi_p)} \sum_{p \in \Phi_p} \alpha_p$ označující průměrnou hodnotu viditelnosti všech bodů Φ_p promítnutých do roviny.

2.4 Python knihovny pro práci s mračny bodů

Python je populární, vysokoúrovňový, interpretovaný programovací jazyk. Jeho univerzálnost a široká škála využití zajišťují právě knihovny a rozsáhlá komunita uživatelů, díky které se neustále vyvíjí. Při práci s mračnem bodů je důležité vědět formát uložených dat, v jakém jsou stavu a jakým způsobem se bude s nimi pracovat. Každá knihovna implementuje řešení pro konkrétní problémy a je na uživateli, aby si ji vhodně vybral pro své účely. Mezi základní schopnosti knihoven pro práci s mračnem bodů je možné zařadit načítání a ukládání mračen do různých typů formátů a případně převody mezi nimi, vizualizace bodů ve 3D prostoru a možnost rotace nebo škálovatelnosti pro náhledy z různých směrů, filtrace dat a potlačování šumu, a v neposlední řadě analýza vlastností bodů, jako jsou výpočty hustoty nebo vzdálenosti. Pro obecný přehled uvádím krátký výčet knihoven, které jsou běžně využívané a zároveň by mohly být vhodným kandidátem pro řešení mého problému.

Open3D knihovna

Open3D je open-source knihovna, která umožňuje vývoj softwaru pro práci s 3D daty. Frontend poskytuje sadu datových struktur a algoritmů v jazycích C++ a Python. Backend je optimalizovaný a podporuje paralelní vykonávání vybraných operací, díky čemuž je možné zpracovávat i rozsáhlá mračna bodů relativně rychle na více jádrech CPU. Knihovna byla vyvinuta od úplných základů s minimálním množstvím závislostí na jiných nástrojích. Lze ji využívat na různých platformách [14].

Pro porovnání a ukázku uvádím část kódu zpracovávající nalezení souboru a načtení dat do vlastní struktury, ze které body z modelu umí zobrazit v novém okně spuštěném při vizualizaci.

```
import open3d

# Nacteni point cloudu.
pc = open3d.io.read_point_cloud("my_point_cloud.pcd")

# Zobrazeni okna s point cloudem.
o3d.visualization.draw_geometries([pcd])

# Program ceka na zavreni okna.
```

Výpis 2.1: Načtení mračna bodů pomocí Open3D knihovny.

Uvedená část kódu vytvoří a otevře nové okno s modelem, umožňující prohlížení bodů v prostoru pomocí rotací a škálování vůči výchozímu bodu. Následující obrázek 2.9 zachycuje pohled na model průjezdu trati z boku v základním nastavení funkce pro vykreslení bez žádných zvýraznění ani modifikací.

Obrázek 2.9: Mračno bodů vykresleno knihovnou Open3D.

Point cloud library

Knihovna Point cloud library (PCL) je rozsáhlý open-source projekt pod licencí BSD pro zpracování 2D/3D snímků a mračen bodů. Obsahuje mnoho algoritmů včetně filtrování, rekonstrukce povrchů, modelování a spojování více mračen bodů do jednoho. Tyto algoritmy lze využít při odstraňování odlehlých bodů z šumem narušených dat, extrakci klíčových bodů, rozpoznávání objektů dle jejich geometrických tvarů, nebo při vytváření a vizualizaci povrchů z mračna bodů. PCL je multiplatformní knihovna a byla úspěšně zkompilována a nasazena na systémech Linux, macOS, Windows i Android. Pro zjednodušení vývoje je rozdělena do menších dílčích knihoven, které lze kompilovat samostatně. Tato modularita je důležitá zejména při distribuci PCL na platformách se sníženým výpočetním výkonem či omezenou velikostí úložiště [20].

Pro porovnání implementace načítající mračno bodů, je zde ukázka kódu, která nalezne a vykreslí mračno bodů.

```
import pcl

# Nacteni point cloudu.
pc = pcl.load("my_point_cloud.pcd")

# Zobrazeni okna s point cloudem.
visual = pcl.pcl_visualization.CloudViewing()
visual.ShowMonochromeCloud(pc)

# Ukonceni az po zavreni okna.
while not visual.WasStopped():
    pass
```

Výpis 2.2: Načtení mračna bodů pomocí PCL knihovny.

PyVista

Projekt PyVista je kolekce open-source* softwaru s volnou licencí FLOSS zaměřující se na 3D vizualizaci a analýzu, případně zpracování mesh dat v Pythonu. Knihovna poskytuje základní nástroje pro 3D vykreslování a typy mesh struktur. I když PyVista je python knihovna, její závislé nástroje, na kterých je postavena, jsou implementovány v jazyce `C++`, jako například Visualisation Toolkit (VTK). To přináší i výhody, jako například to, že je knihovna kompatibilní s jakýmkoli softwarem založeným na VTK. Balíček PyVista pro Python nabízí stručné a dobře zdokumentované rozhraní využívající výkonný vizualizační backend VTK [22]. Tato knihovna sama o sobě přímo nepodporuje soubory typu *.pcd*, bylo by nutné nejdříve extrahovat jednotlivé body ze souboru s využitím některé z výše uvedených knihoven a provést konverzi. Z tohoto důvodu nástroj PyVista není nejvhodnější pro řešení daného problému.

2.5 Multiplatformní Python knihovny pro tvorbu uživatelského rozhraní

Tato část se zabývá přehledem multiplatformních knihoven, které umožňují tvorbu grafického uživatelského rozhraní přímo v Pythonu. Nabízí nástroje pro standardní grafické prvky, díky nimž je možné rychle vytvořit prototypy, jednodušší aplikace pro vstup dat či jejich vizualizaci, ale i propracované komplexnější aplikace dle požadavků zákazníka. Tyto knihovny jsou velmi výhodné pro vývojáře pracující převážně v Pythonu, předurčují snadné použití díky kompatibilitě kódu a využívání stejných struktur, není tedy potřeba učit se celý nový jazyk pro vytvoření jednoduchého grafického uživatelského rozhraní.

Pro zvolení správného nástroje je potřeba dopředu vědět požadavky na grafické rozhraní, zjistit si možnosti jednotlivých nástrojů a porovnat, které nejlépe splňují stanovené podmínky. Velkou roli hraje platforma, na kterou má být aplikace nasazena. Pro desktopové aplikace uvádím nástroj PyQt 2.5, který nabízí rozmanitý výběr funkcí a je vhodný pro větší projekty s náročnějšími požadavky na grafické rozhraní. Mezi nástroje využívané k vývoji mobilních aplikací jsem uvedla konkurenční nástroje BeeWare 2.5 a Kivy 2.5. Oba se zaměřují na grafická rozhraní pro zařízení Android a iOS, přičemž jeden nástroj podporuje přirozený vzhled aplikací pro jednotlivé platformy a druhý více využívá možnosti dotykových zařízení. O každém z nástrojů je možné se více dočíst v odkazovaných sekcích.

Pro obecný přehled a porovnání vlastností mezi uvedenými nástroji doporučuji sekci 2.5, kde je shrnutí znázorněno tabulkou.

PyQt

Prvním způsobem, kterým lze vytvářet multiplatformní GUI aplikace v jazyce Python, je za pomocí Qt frameworku. Jelikož je Qt psané v jazyce `C++`, PyQt vytváří vazby mezi Pythonem a `C++` knihovnami, díky nimž je možné psát logiku aplikace v Pythonu a zároveň využívat pokročilé funkce a komponenty poskytované frameworkem Qt.

Při vývoji aplikace je velmi jednoduché využívat předdefinované základní prvky grafického rozhraní, zvané widgety, které je možné si následně upravit do vlastního požadovaného vzhledu a chování. Mezi takové prvky patří tlačítka, textová pole, grafická zadávání vstupů jako jsou posuvníky, nebo výběry z možností, dále tabulky a složitější grafické prvky 3D vykreslování, zobrazování grafů, multimédia a mnoho dalších. Právě díky PyQt, které překládá rozhraní těchto `C++` knihoven do podoby, která je snadno použitelná z pythonu, je možné všechny tyto Qt moduly a třídy využívat. K dispozici je také nástroj Qt Designer, který usnadňuje vizuální návrh uživatelského rozhraní.

PyQt je multiplatformní, podporuje hlavní desktopové operační systémy jako Windows, macOS i Linux a je dostupný pod licencí GPL v3 a pod komerční placenou licencí. Co se týče mobilních platforem Android a iOS, je možné i pro ně tímto způsobem vytvářet aplikace, ale s pomocí dalších nástrojů, které umožní jejich nasazení. PyQt je velmi mocný nástroj, ale také má své limity a není dobré se ho snažit použít vždy a na všechno. Z tohoto důvodu se dále zaměřím na nástroje, které jsou více určené pro vývoj aplikací na mobilních platformách [18].

OpenCV

Open Source Computer Vision Library je knihovna s licencí Apache 2. Byla vyvinuta za účelem poskytnout společnou infrastrukturu pro aplikace využívající počítačové vidění a urychlit strojové vnímání v komerčních produktech. Knihovna obsahuje více než 2500 optimalizovaných algoritmů, které mají široké spektrum využití. Patří mezi ně například algoritmy pro detekci a rozpoznávání obličejů, identifikaci objektů, klasifikaci lidských činností na videu, sledování pohybu očí a pohybujících se objektů, extrakci 3D modelů, tvorbu 3D mračen, hledání podobných snímků v databázích, odstranění efektu červených očí při pořizování fotografie s bleskem a mnoho dalších. Knihovna má rozhraní pro `C++`, Python, Javu i MATLAB a podporuje Windows, Linux, Android a macOS [15].

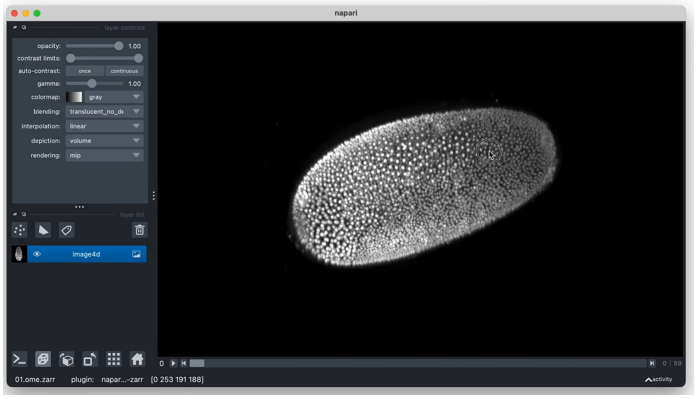
Přestože je OpenCV knihovna spíše zaměřená na zpracování dat, umožňuje i jejich jednoduché zobrazení v podobě uživatelského rozhraní. Nabízí omezené množství nástrojů pro práci s obrázky, jejich načtení, zobrazení a uložení, je možné zachycovat videa z kamer a následně je přehrávat. Najdou se tam i nástroje pro kreslení jednoduchých geometrických tvarů, nebo lze využít pohyby s myší jako s štětcem. Pro ovládání vstupních parametrů je možné využít posuvníků [16]. Určitě se nedá mluvit o těchto nástrojích jako o rozsáhlých pro tvorbu uživatelských rozhraní a komplexních aplikacích, ale spíše o nástrojích určených k rychlému testování nebo ladění algoritmů. Pro vytvoření aplikací na vyšší úrovni je vhodnější OpenCV integrovat do některé z GUI knihoven nebo pro to určených frameworků.

BeeWare

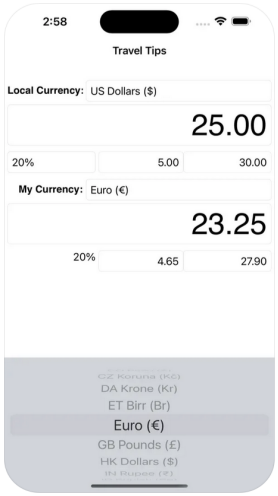
BeeWare je sada nástrojů a knihoven vyvinutých za účelem umožnit vývojářům vytvářet nativní uživatelské rozhraní aplikací v Pythonu a nasazovat je na různých platformách, jako jsou Windows, Linux, macOS, iOS i Android. Mezi hlavní cíle BeeWare patří jednoduchost a univerzálnost, zajistit, aby nástroje byly natolik jednoduché, aby je lehce zvládl i začátečník, ale dostatečně mocné, aby zkušený vývojář mohl vytvářet propracované a komplexní aplikace. Jednotlivé nástroje jsou na sobě nezávislé a je možné je v jednotlivých projektech samostatně využívat. Zakladatelé se při jejich vytváření zaměřili na to, aby každý nástroj měl jednu hlavní funkcionalitu, kterou zvládá velmi dobře. Tento přístup zajišťuje, že při náročnějších požadavcích lze potřebné nástroje zřetězit a vzniknou tak kompozice se stále velmi schopnou funkcionalitou [2].

Velkou výhodou nástrojů BeeWare je snaha si zachovat přirozený vzhled pro každou vyvinutou aplikaci, nehledě na to, na jakou platformu bude nasazena. Všechny komponenty grafického rozhraní mají jednotný vzhled a chovají se tak, jak by uživatel daného operačního systému předpokládal na základě naučených principů.

Příkladem použití nástroje BeeWare jsou níže zobrazené ukázky. Python knihovna Napari 2.10 využívaná k vizualizaci více rozměrných struktur a jejich úpravě [3]. Dále aplikace Travel tips 2.11 dostupná ke stažení v iOS App Store na výpočet spropitného a pro převod měn.



Obrázek 2.10: Nástroj Napari [13].



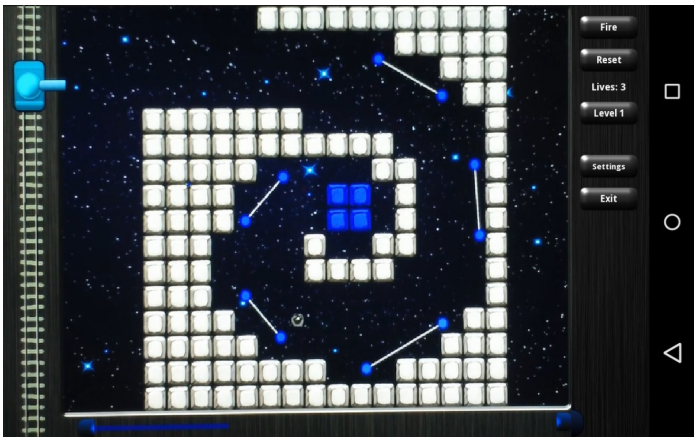
Obrázek 2.11: Aplikace Travel tips [23].

Kivy

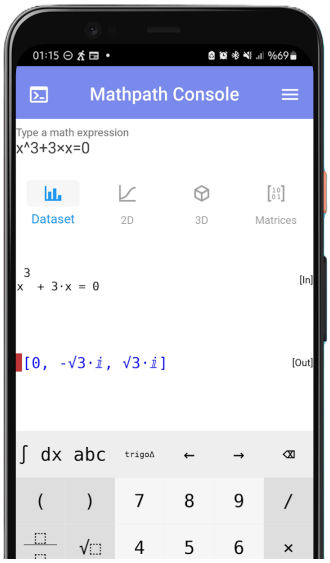
Kivy je open-source framework pro vývoj multiplatformních aplikací s grafickým rozhraním v jazyce Python. Umožňuje vytvářet aplikace pro Windows, macOS, Linux, Android a iOS. Menší nevýhodou tohoto frameworku je, že nezajišťuje přirozenost uživatelského rozhraní napříč platformami. Využívané nástroje mají vlastní vzhled, který je samozřejmě možné libovolně upravit. Zkušený vývojář bude schopný vytvořit přirozeně vypadající i chovající se aplikaci, ale pouze pro danou platformu, pro kterou se rozhodl aplikaci vyvíjet, a při nasazení na zařízení, které nespadá do stejné kategorie, už aplikace nebude tolik intuitivní.

Vývojáři Kivy se o přirozenost ani tolik nesnažili, místo toho se zaměřili na způsob ovládání aplikací. Zejména dotyková zařízení mají velkou výhodu oproti ostatním, jelikož Kivy podporuje ovládání gesty a více dotyky, tak zvaný multi-touch, je tedy ideální pro tvorbu her a interaktivních aplikací pro mobilní zařízení a tablety. Dalším aspektem je rychlost, na kterou si Kivy potrpí. Ve všech případech, kdy je to možné, se využívá síly grafického procesoru, který je v některých případech (určité typy úloh a algoritmů, jako je vykreslování) rychlejší a výkonnější oproti obecnému centrálnímu procesoru. Dále bych zmínila flexibilitu, nejen že podporuje všechny hlavní operační systémy, snaží se být kompatibilní i s třetími stranami, externími poskytovateli různých nástrojů. Pro Windows je to podpora WM_TOUCH technologie. Pokud má zařízení instalované ovladače pro dotykové ovládání pro dotykový displej nebo pero, Kivy je schopné s těmito zařízeními pracovat. Podobně to platí i pro MacOS zařízení, jako je magická myš, která umí rozpoznat různá gesta, nebo trackpad [7].

Pro ukázku uvádím dvě aplikace, které byly vyvinuty pomocí frameworku Kivy. Dotyková hra Deflectouch na obrázku 2.12 vyhrála první místo v Kivy soutěži. Cílem hry je sestřelit modré kostky vypálenou střelou z děla a pomocí dvou prstů se v prostoru natahují odrazné plochy pro střelu, aby se dostala k cíli. Hra má interaktivní prvky i pestré zvukové efekty. Na obrázku 2.13 je aplikace MathPath pro studenty, která umožňuje vkládat matematické rovnice a zobrazí jejich řešení v podobě výsledku, postupu řešení krok za krokem, nebo 2D a 3D grafu.



Obrázek 2.12: Hra Deflectouch [5].



Obrázek 2.13: Aplikace MathPath Console [10].

Srovnání multiplatformních Python knihoven

Jak bylo zmíněno v úvodu této kapitoly, každá aplikace je specifická a potřebuje individuální přístup. Občas se vývojář musí přizpůsobit potřebám každého projektu a zvolit ten nejvhodnější nástroj, který umožní danou funkčnost správně implementovat. Proto zde formou tabulky 2.1 uvádím srovnání představených knihoven, tak aby bylo přehledně hned na první pohled vidět, která knihovna je vhodná pro jakou činnost a některé jejich výhody

a nevýhody, které je potřeba při výběru brát v potaz. Pro co nejadekvátnější porovnání bylo využito těchto zdrojů [19, 9, 8], společně s oficiálními stránkami každé knihovny.

	 **PyQt**	 **OpenCV**	 **BeeWare**	 **Kivy**
Platformy	Windows, macOS, Linux	Windows, macOS, Linux, Android, iOS	Windows, macOS, Linux, Android, iOS	Windows, macOS, Linux, Android, iOS
Licence	GPL nebo komerční	Apache License 2.0	BSD	MIT
Využití	Desktopové GUI aplikací	Počítačové vidění, zpracování obrazu, jednoduché GUI	Mobilní i desktopové GUI aplikací	Mobilní i desktopové GUI aplikací
Výhody	Profesionální, přirozený vzhled, bohaté funkce a nástroje, Qt designer	Bohaté funkce pro analýzu obrazu, vícejádrové zpracování, kompatibilní s Python, C/C++, Java	Přirozený vzhled na všech platformách, rozšíření specializovanými nástroji, rozsáhlá komunita	Vícedotykové ovládání, zrychlení pomocí GPU, mnoho nástrojů
Nevýhody	Komplexní a rozsáhlé pro začátečníky, chudá Python dokumentace	Složitost, velmi omezené funkce pro tvorbu GUI	Ve vývoji, bugy, změny API, méně funkcí	Nepřirozený vzhled, malá komunita, chudší dokumentace

Tabulka 2.1: Porovnání nástrojů.

Kapitola 3

Návrh uživatelského systému

Jelikož cílem této bakalářské práce je navrhnout a implementovat uživatelský systém pro vestavěná zařízení, která budou za předpokladu převážně zastoupena tablety, bude jeho implementace řazena do vývoje mobilní aplikace a takto se na něj dále odkazovat.

Při vývoji mobilní aplikace je potřeba si uvědomit možnosti takového vývoje. Jaké jsou jeho přednosti, zaměřit se na maximální využití nabízených prostředků a kompenzovat tak případné nevýhody oproti aplikacím desktopovým nebo webovým. Mezi tyto přednosti zcela jasně patří interaktivní prvky ovládané dotykem, gesty a pohyby v případě přítomnosti akcelerometru, dále především přenositelnost zařízení. Naopak nevýhody jsou způsobeny omezenými zdroji, ať už je to kratší výdrž baterie nebo menší výkon zařízení, je potřeba se zaměřit na důkladnou optimalizaci aplikací a zajistit jejich ladné fungování a šetrnost ke zdrojům.

Tato kapitola představuje funkce, které by výsledná aplikace měla mít implementované pro splnění požadavků. Popisuje, jaké je očekávané chování od jednotlivých funkcí a co všechno by měly zahrnovat. Další důležitou částí je návrh grafického uživatelského prostředí takovým způsobem, aby bylo zajištěno uživatelského komfortu při ovládání aplikace. Jednotlivé očekávané vlastnosti jsou vysvětlené a navrhují, jak je vhodné je vyřešit ke spokojenosti uživatele.

3.1 Požadované funkce systému

Hlavním cílem aplikace je dokázat, že i méně výkonná zařízení, jako je tablet, zvládají zpracovat velké množství dat získaných z mračen bodů. Nejdůležitější je v tomto případě správná vizualizace bodů a měla by být středem pozornosti. Pro uživatele je to podstatná část, proto se zobrazí hned při spuštění aplikace na hlavní stránce a rozprostírá se přes celou obrazovku, využívá maximální dostupný prostor na zobrazení dat. Návrh obrazovky ilustruje obrázek 3.1, kde ovládací prvky se nacházejí po stranách obrazovky a jsou skryté, tak aby byl stále zajištěn co největší možný podíl obrazovky na zobrazování dat.

První skupina funkcí slouží k ovládání animace, neboli projekci bodů na pozadí videa reálné scény z kamery. Uživateli musí být umožněno animaci spustit, tedy zahájit posloupnost projekcí. Každá projekce vykreslí snímek tvořený body a zobrazí ho na pozadí odpovídající aktuální scéně. Body musí co nejpřesněji kopírovat reálné tvary a jednoznačně vykreslovat viditelné objekty reálného světa. Dále musí uživatel být schopen animaci pozastavit, přerušit projekci a nechat zobrazený právě prohlížený snímek. Následně má uživatel opět možnost pokračovat v přehrávání animace od snímku, na kterém se pozastavil. V případě

Obrázek 3.1: Zobrazení dat přes celou obrazovku se skrytými nástroji.

rozhodnutí o ukončení animace je možné využít funkci restartování, která animaci přeruší a navíc ji uvede do výchozí podoby na počáteční snímek, připravenou pro opětovné spuštění. Pokud bylo dosaženo posledního snímku projekce, došlo k přehrání celého videa, animace se ukončí a setrvává na posledním snímku, dokud není uživatelem spuštěna od začátku.

Dalším ovládacím prvkem animace je posuvník, který umožňuje zcela volné přetáčení videa dopředu i zpět na libovolný snímek, od kterého po spuštění animace pokračuje. Zároveň představuje časovou osu videa s ukazatelem reálného umístění na ose dle aktuálního zpracovávaného snímku. Z časové osy lze vyčíst celkovou dobu trvání animace a čas, který už uplynul.

K ovládání výsledného vzhledu animace patří funkce pro přepínání viditelnosti jednotlivých částí. Celkový obraz je tvořen dvěma částmi. První se nachází v popředí a tvoří ji body získané projekcí z mračna bodů vykreslené na průhledném pozadí. Tato část je použita jako textura, která se mapuje na snímky reálné snímky z kamery tvořící druhou část obrazu. Uživateli je animace zobrazena jako spojení těchto částí do celku, ale má možnost si zvolit, pokud některá z částí má být skryta. Pak se v animaci přehrávají pouze snímky projekce na základním neutrálním pozadí, nebo je spuštěno video z kamery. Pro větší uživatelskou přívětivost je uživateli umožněno změnit barvu vykreslovaných bodů a změna se projeví hned po zvolení nové barvy. Výchozí nastavení barvy je zvoleno tak, aby se svým kontrastem co nejlépe odlišovala od barev na pozadí.

Uživatel má možnost stanovit si rychlost přehrávání animace. Zvolit si zpomalení nebo zrychlení přehrávání videa podle potřeby, za předpokladu, že to výpočetní rychlost zařízení umožní. I v tomto případě musí být zaručeno sjednocení bodů se snímky z kamery tak, aby jako celek představovaly odpovídající scénu.

Aplikace je schopna informovat o aktuálním stavu animace, pokud si to uživatel vyžádá. Umožní nastavení konkrétních parametrů, které mají být pravidelně aktualizovány, pokud dojde k jejich změně. Mezi tyto parametry patří pořadí právě zpracovávaných snímků bodů a videa, počet zobrazovaných bodů každého snímku, maximální počty snímků a případně upravení rychlosti přehrávání.

Nezbytnou součástí aplikace je import dat, tak, aby bylo možné s nimi dále pracovat. Uživatel má možnost přes rozhraní aplikace přistoupit k úložišti na svém zařízení a zvolit soubor, který má být zpracován. Musí být dodrženy určité podmínky, se kterými je uživatel obeznámen, aby byl import možný, jako je dodržení typů souborů, případně jejich názvy. V případě porušení podmínek se jedná o nevalidní data a uživatel je náležitě upozorněn.

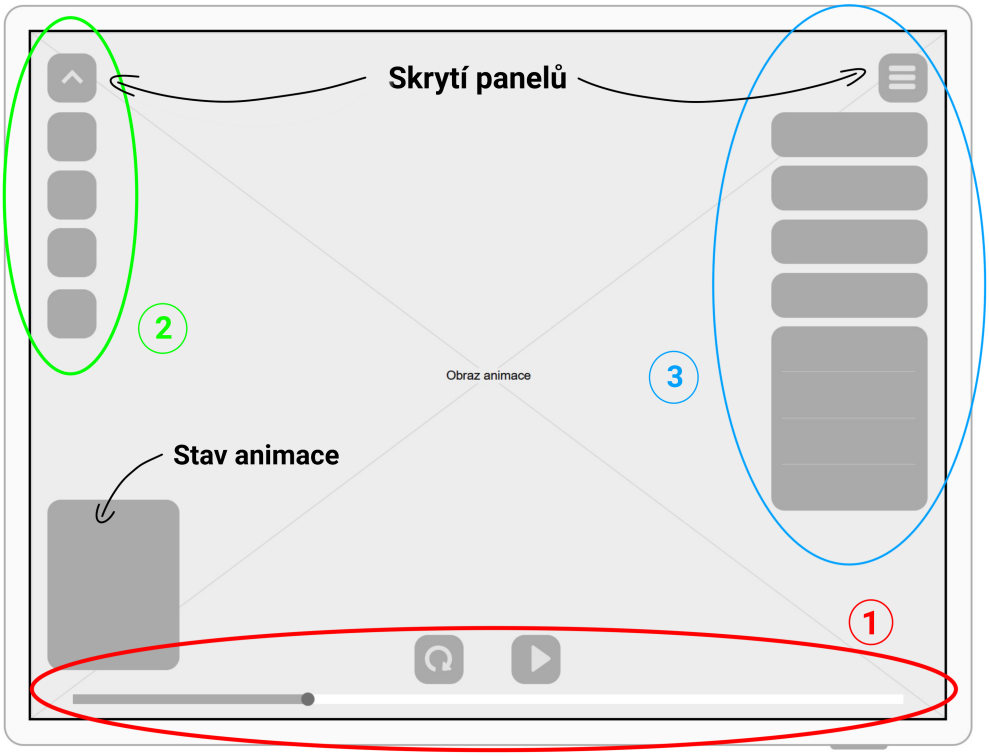
Důležitou vlastností aplikace je zachování stavu všech nastavených parametrů, jak si je uživatel nastavil. Při ukončení a zavření aplikace se uloží stav, v jakém ji uživatel opustil, a při dalším spuštění aplikace obnoví uložené nastavení společně s rozložením jednotlivých komponent, aby uživatel mohl bez nadbytečných úprav pokračovat tam, kde přestal.

3.2 Předpokládané nároky uživatelů

Velmi důležitým aspektem pro vývoj kvalitní aplikace je znát cílového uživatele a mít tak možnost zaměřit se na jeho konkrétní potřeby. V mém případě by mohl být potenciálním uživatelem dospělý člověk s technickým zaměřením, pohybujícím se v oblasti vlakové dopravy. Od aplikace očekává, že si může spustit záznam z vlaku při průjezdu železniční tratí, prohlížet jednotlivé záběry úseků, vracet se zpět k důležitým událostem a záznam přetáčet. A to díky importu dat, jako je samotné mračno bodů, sada matic provádějící projekci do 2D snímků, video zachycující průjezd daným úsekem. Uživateli by mělo být umožněno nastavení jednotlivých parametrů, které ovlivňují způsob zobrazení dat. Mezi tyto parametry může patřit kalibrace bodů vůči reálné scéně, volba barvy zobrazovaných bodů, případně omezení některých vizualizačních prvků.

Nicméně každý uživatel od aplikace bude očekávat intuitivní ovládání. Jednotlivé ikony a popisky by měly být sebevysvětlující, tak aby bylo jasné, co za funkce se za nimi nachází. Pokud by měla být aplikace složitější a obsahovala větší množství různých funkcionalit, bylo by vhodné využít úvodní tutoriál na seznámení, který uživatele provede a předvede, kde se co nachází a jak se ovládá. Přesto je dobré snažit se udržet přehlednost, mít funkce logicky seskupené a ty podobně zaměřené dát k sobě. To je znázorněno na obrázku 3.2, kde logické sekce jsou zvýrazněny a očíslovány od jedničky po trojku. Sekce jedna je vyhrazena pro ovládání videa. Obsahuje standardní ovládací prvky, jako je jednoduché přehrávací tlačítko pro spuštění nebo pozastavení a tlačítko označené šipkou pro restartování videa, kdy se video přetočí na úplný začátek. Volné přetáčení videa umožňuje posuvník s časovou osou pro lepší orientaci a znázornění aktuální pozice ve videu. Sekce dvě obsahuje rychlé akce na ovládání vizualizace dat a sekce tři je pro nastavení aplikace a import dat.

Minimalismus je také ceněná vlastnost. Přeplněné a tudíž nepřehledné rozhraní může být uživateli nepříjemné, bude se špatně orientovat a mnohdy chtěný výsledný obraz nebude dobře viditelný, upozaděný nebo tak malý, že na něm nic neuvidí. Toto řeší schopnost skrytí méně používaných panelů. Uživatel se může rozhodnout, kterou sadu nástrojů bude právě potřebovat, a ostatní, místo toho, aby ubírali místo na obrazovce, si zmenší pouze na jedno tlačítko, kterým si v případě potřeby zase panel zobrazí. Obě sekce dvě i tři na obrázku 3.2 umožňují skrytí prvků pouze na jedno horní tlačítko. Pro maximální využití displeje lze implementovat režim prohlížení, kdy se obraz zvětší na celou obrazovku a veškeré ovládací prvky se skryjí až na jedno tlačítko, které vrátí všechny skryté nástroje.



Obrázek 3.2: Návrh rozložení ovládacích prvků aplikace.

Absolutní spokojenost uživatele zajišťuje míra personalizace obsahu v rozhraní aplikace. I když je obsah zaměřený na konkrétní uživatele, každý přece jen s aplikací může pracovat trochu jinak. Proto dát uživateli možnost upravit si zobrazované prvky je ceněné. To umožňuje část Stav animace označená na obrázku 3.2, kde si uživatel může zvolit, o které parametry má zájem a chce, aby se tam zobrazovaly. Jedná se především o informace k právě probíhající animaci, které lze vybrat v nastavení aplikace. Případně lze celý panel skrýt funkcí ze sekce dvě z obrázku 3.2.

Pro pohodlnou práci s aplikací je nutná plynulost a rychlost. Uživatel nesmí mít pocit, že stráví více času čekáním než reálnou prací. Především u výpočtově a graficky náročnějších aplikací, kam bych zařadila i tu mou, je důležité použít vhodné nástroje a dobře rozvrhnout využití dostupných zdrojů. Nedílnou součástí bude profilování a následná optimalizace drahých operací.

Kapitola 4

Implementace

Cílem této kapitoly je podrobnější popis implementace navrženého uživatelského systému vyvinutého za účelem umožnit prohlížení objemných mračen bodů na přenosných zařízeních, jako je tablet, s možností zobrazit data na pozadí snímků reálné scény pořízených z kamery umístěné na čele vlaku. Aplikace je implementována tak, aby splňovala požadavky představené v kapitole 3 návrhu uživatelského systému, přičemž je kladen důraz na optimalizaci, aby byla zajištěna co největší plynulost pro méně výkonná, běžná zařízení.

Architekturu aplikace lze rozdělit do několika modulů, logických celků, z nichž každý má specifickou funkci. Správný průběh zajišťuje hlavní modul `Controller`, který má na starost zpracování příchozích požadavků od uživatele, jejich vyhodnocení a předání pomocným modulům, kde je provedena adekvátní akce a následné propsání výsledků zpět k uživateli. Hlavní funkce pomocných modulů jsou popsány v dílčích částech této kapitoly. Jedná se především o průběh zpracování dat ze vstupních souborů, jejich následná synchronizace a vizualizace pro uživatele. Mezi rozšiřující funkce lze zařadit import souborů s daty pro zpracování a některé další pro zvýšení komfortu uživatele.

K vývoji systému byl zvolen programovací jazyk Python a to především z důvodu množství dostupných rozšiřujících knihoven, které usnadňují vývoj a rozšiřují aplikaci. Tvorba grafického uživatelského prostředí byla uskutečněna s využitím open-source frameworku Kivy. Hlavním důvodem pro výběr tohoto nástroje je podpora zrychlení výpočtů s využitím grafického procesoru u některých typů úloh. Společně s Kivy byla využita i knihovna KivyMD, která poskytuje již stylisticky předdefinované komponenty, což usnadnilo a urychlilo celý vývoj. Dále stojí za to zmínit knihovnu OpenCV, která přispěla při řešení vizualizace snímků na pozadí ve formě videa i při projekci bodů a jejich zkreslení. Veškeré maticové operace spojené s projekcí bodů a práce s datovými poli jsou řešeny pomocí knihovny NumPy, která je optimalizovaná pro vysoký výkon.

Pro účely nasazení systému na zařízení a jeho testování se využívá nástroj Buildozer, který umožňuje kompilaci aplikací napsaných v Pythonu pro mobilní platformy, jako jsou Android nebo iOS. Je nutné k tomu vytvořit konfigurační soubor, kde se definují parametry, jako název aplikace, závislosti, typ platformy, verze a další specifikace. Po spuštění kompilace Buildozer stáhne potřebné závislosti, nastaví prostředí, zkompiluje kód a výstupem je instalační soubor typu APK, připravený k instalaci na zařízení. K samotné kompilaci se využívá nástroj p4a (python for android), jelikož Android přímo nepodporuje běh aplikací napsaných v Pythonu. Instalaci a ladění zprostředkovává nástroj ADB (Android Debug Bridge), který zajišťuje komunikaci s připojeným Android zařízením a umožňuje čtení výstupních ladících zpráv nezbytných pro detekci chyb.

4.1 Projekce bodů z prostoru do roviny

Implementace projekce vychází z teoretického postupu popsaného v části 2.2, kde je vysvětleno, jakým způsobem k tvorbě 2D snímků z bodů dochází. Jedná se především o maticové operace v konkrétním pořadí s konfiguračními daty. Při práci s daty se využívá převážně dvou typů souborů, kde mračno bodů je uloženo v datovém souboru .pcd a konfigurační soubory využívají .csv typ pro tabulková data, kam patří rotační a translační vektory, nebo projekční matice. Soubory obsahují data pro všechny jednotlivé projekce bodů z prostoru, které jsou tvořeny posloupností pozic kamery, a na každém řádku souboru se nachází data určená pouze pro konkrétní pozici vůči mračnu bodů. Konfigurační soubory zajišťují, aby se vykreslily vždy jen body ve viditelném poli kamery a na správných souřadnicích tak, že výsledná projekce je správně natočená a shoduje se s reálnou scénou.

Translační vektor určuje posunutí kamery z její výchozí polohy ve směru osy pohybu o úsek, který odpovídá uražené dráze za jednotku času. Hodnoty jednotlivých složek translačního vektoru se s časem zvyšují a posouvají kameru dále po dráze. Pro získání bodů, které jsou v dané pozici pro kameru viditelné, jinými slovy nacházejí se před kamerou a ne za ní, je potřeba před každou projekcí odečíst složky translačního vektoru od všech bodů z mračna. Jelikož dráha kamery je téměř paralelní s osou z, všechny body, které mají po odečtení tuto složku kladnou, se nacházejí před kamerou a pro danou pozici jsou viditelné.

Po získání viditelných bodů je možné aplikovat rotační vektor pro nastavení úhlu pohledu v mračně bodů. K tomu je nejprve potřeba přenést body do systému homogenních souřadnic, které umožňují reprezentaci bodů v prostoru pomocí čtyř souřadnic místo tří, jako je to u kartézských souřadnic. K původním třem složkám x, y, z přibývá ještě jedna, často označovaná jako w s hodnotou 1. Tímto je zajištěno, že lze všechny transformace, jako je posunutí i otočení, vyjádřit jednou matematickou operací s maticemi a tou je násobení. V opačném případě, kdy by body byly vyjádřeny pouze třemi složkami, muselo by se provádět zvlášť sčítání a násobení matic. Takto se operace sjednotí a práce s nimi bude jednodušší, jelikož je možné aplikování transformací řetězit. Další výhodou je následná perspektivní projekce bodů do roviny. Po aplikování projekční matice jsou body stále v homogenní podobě a pro převod do 2D je potřeba získat pouze složky x, y aby se bod dal zobrazit v rovině. Proto jsou body převedeny zpět do systému kartézských souřadnic, čímž ubývá jedna složka, a následně je nutné provést normalizaci bodů tímto způsobem $[x, y] = [x/z, y/z]$. Je však potřeba pohlídat, aby při normalizaci nedošlo k dělení nulovou složkou, z tohoto důvodu je potřeba nejdříve takové body odstranit. Tím je zajištěno, že se správně projeví efekt perspektivy. Po těchto úpravách se dále pracuje pouze s viditelnými body nacházejícími se před kamerou i se správným rozmístěním v rovině. Výsledný obraz projekce je ještě potřeba otočit o 180 stupňů, aby snímek odpovídal reálné scéně.

Přesto tento proces není ještě zcela hotový, jelikož při projekci dochází ke zkreslení reálného obrazu a jednotlivé body by se jevily jako posunuté, tak zvaná distorze obrazu. K deformaci obrazu může docházet z několika důvodů. Mezi ty nejběžnější důvody patří nedokonalost reálných čoček, které se vyskytují v zařízeních používaných pro sběr dat. Povrch čoček může být nerovnoměrný, což způsobuje, že paprsky neprocházejí vždy skrz přesně definované ohnisko. V případě složení více čoček do jedné soustavy mohou být vůči sobě mírně posunuté a tím opět vznikají nepřesnosti. Aby se zkreslení co nejvíce eliminovalo, je nutné provést korekci. K tomu je využita funkce knihovny OpenCV *undistortPoints*. Z konfiguračních souborů je potřeba získat koeficienty distorze, které jsou dané kalibrací kamery. Koeficienty se v tomto konkrétním případě skládají z pěti členů představujících radiální a tangenciální zkreslení. Dále funkce vyžaduje normalizované souřadnice bodů a projekční

matici. Po výpočtu jsou body ve finálním tvaru pro zobrazení na 2D snímku a je možné pokračovat ve zpracování, kde v dalším kroku je řešena viditelnost bodů pro daný úhel pohledu. O tom je více vysvětleno v sekci 4.2 Viditelnost bodů.

Počátky samotné projekce bodů byly velmi komplikované. Především kvůli transformaci bodů mračna, kdy nedocházelo ke správné rotaci a body se tudíž zobrazovaly úplně mimo výsledný snímek. Při vykreslení se často stávalo, že se nezobrazily žádné body, nebo došlo k vykreslení bodů na hraně snímku a obraz tedy nebyl vidět celý. Ve všech případech byl obraz špatně natočený, neodpovídal směru pohledu, který byl očekáván, aby se shodoval s reálnou scénou. Všechny problémy spočívaly ve špatném pořadí prováděných transformací. Je velice důležité dodržet jednotlivé kroky projekce, nejprve provést posunutí, aby se získala výchozí pozice kamery a teprve poté díky detailnější rotaci získat správný úhel pohledu do mračna bodů.

4.2 Viditelnost bodů v rovině po projekci

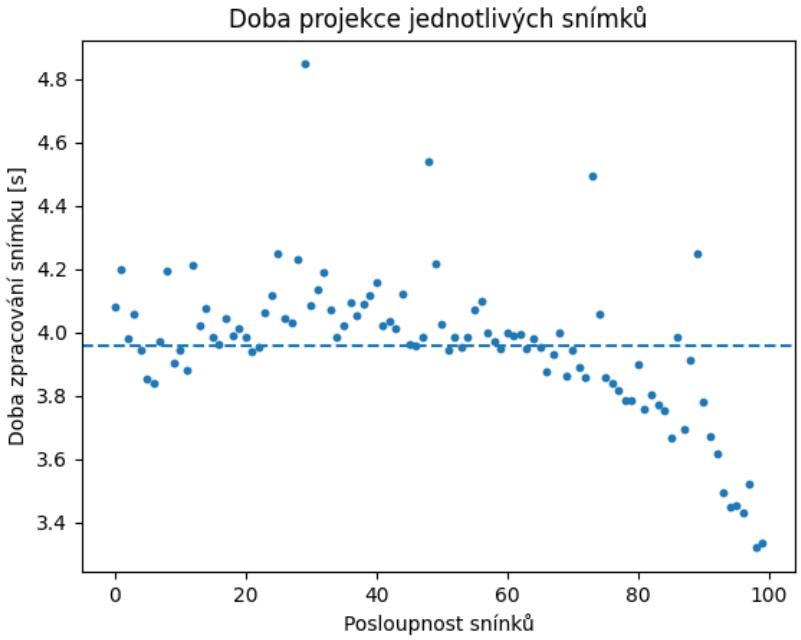
Po dokončení projekce se ve výsledném 2D snímku nachází všechny body ležící před kamerou, ale i ty, tvořící objekty, které jsou pro daný úhel pohledu skryté za jinými, více v popředí. Tím dochází k prolínání objektů do sebe a plocha v těchto místech je přeplněna body a jednotlivé objekty již nejsou rozpoznatelné. Více je tato problematika popsána v sekci 2.3, kde je problém vysvětlen a navrženo řešení, jakým je možné tento následek omezit za využití metody odhadu viditelnosti.

Tato metoda, dle tvůrců, kteří ji zveřejnili a popsali ve své práci [17] dosahuje velmi vysokých a pozitivních výsledků. Při testování na anotovaném datasetu dosáhli více než 90% přesnosti při rozpoznávání viditelných a skrytých bodů, což bylo nejvíce mezi všemi uvedenými a porovnávanými metodami.

Na základě těchto výsledků byla metoda implementována i pro řešení viditelnosti v tomto případě. K výpočtu se využívá jak 3D tvaru bodů před projekcí, tak 2D tvaru výsledných souřadnic pro zobrazení na snímek. A to z důvodu vytvoření hloubkové mapy o velikosti výsledného snímku s nejnižšími hodnotami hloubky pro každou pozici. Tím se eliminují duplikace bodů, které se vykreslují na stejné místo a ponechají se jen ty nejbližší. Následně se pro každý bod získá určitý počet jemu nejbližších bodů a porovnává se jejich hloubka, jak moc se liší. To zajišťuje KNN vyhledávání, ze kterého se pak spočítá pravděpodobnost viditelnosti bodu v rozmezí intervalu (0,1) a podle nastaveného prahu je bod určen viditelným, nebo skrytým.

Po nasazení systému s tímto řešením na zařízení došlo k výraznému až několikanásobnému zpomalení. Přestože se využívá optimalizovaných výpočetních metod knihovny NumPy a vektorových operací, velké množství bodů je stále velmi časově náročné ke zpracování. Z důvodu vzájemné závislosti mezi vstupními 2D souřadnicemi (potřebnými pro výpočet viditelnosti) a výsledkem samotné projekce je nezbytné provádět projekci a následný výpočet viditelnosti postupně. Nelze oba výpočty spustit paralelně a tím ušetřit alespoň část výpočetního času.

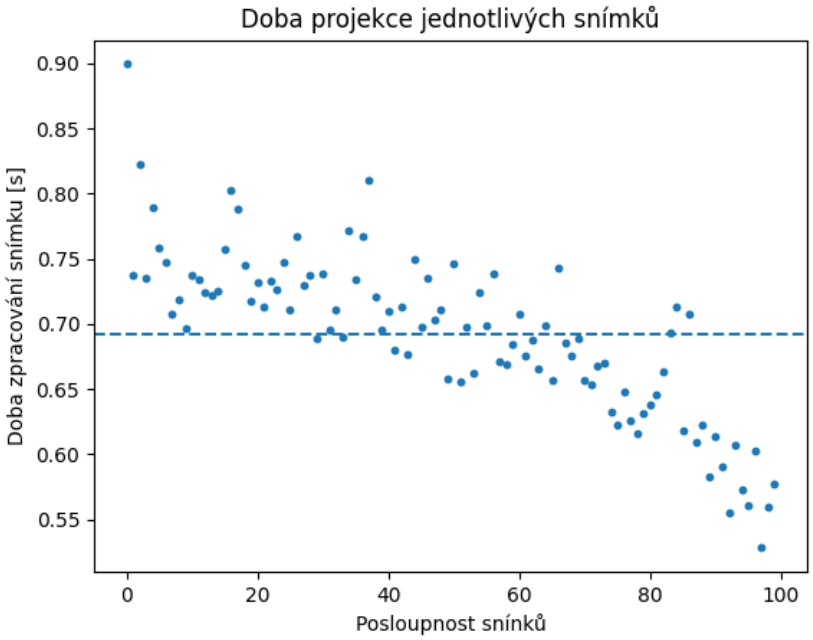
Jelikož rychlost je klíčová pro plynulou vizualizaci snímků, je nutné snížit nároky na přesnost zobrazovaných bodů a zaměřit se více na udržení akceptovatelné doby výpočtu jednoho snímku. Implementace této metody odhadu viditelnosti není pro tento konkrétní případ vizualizace vhodná. Proto bylo nalezeno a implementováno řešení, které sice není zdaleka tak přesné, ale je mnohem méně výpočetně náročné a umožní rychlejší vykreslování bodů.



Obrázek 4.1: Čas strávený na projekci snímků včetně výpočtu viditelnosti přesnější metodou odhadem viditelnosti.

Aktuálně implementované řešení zajišťuje viditelnost pouze těch bodů, které se pro danou pozici na snímku nacházejí nejblíž. Vychází se opět ze třetí souřadnice bodu v prostoru, která určuje hloubku, neboli vzdálenost každého bodu. Jednotlivým bodům je přiřazen klíč, závislý na jejich pozicích v rovině, dle kterého jsou následně všechny body seřazeny. Klíč je shodný pro všechny body nacházející se na stejné pozici, proto jsou zároveň řazeny na základě své hloubky, tak, že ve skupině shodných bodů se na první místo řadí ten, který je nejblíže. Tomuto bodu je přiřazena pravdivá hodnota a společně všechny body tvoří masku, kterou lze aplikovat a získat tak opravdu pouze viditelné body, které je potřeba vykreslit. Dále je omezena vykreslovací vzdálenost bodů, aby zbytečně nedocházelo ke zpracovávání velmi vzdálených bodů, které se ve snímku ani neprojeví. Tato vzdálenost je určena na základě konkrétního mračna bodů. Veškeré výpočty tvoří rychlejší vektorové operace a současně se sníží množství bodů, což umožní jejich rychlejší vykreslení a ušetření času.

Detailnější porovnání uvedených metod je vizualizováno pomocí přiložených grafů. Oba snímky zachycují průjezd stejným úsekem tvořeným prvními sto snímky. Úsek obsahuje dlouhou rovinnou část zakončenou zatáčkou, za kterou do poslední chvíle není vidět. Z tohoto důvodu na obou grafech dochází na konci ke zrychlení, protože s pohybem vpřed množství viditelných bodů ubývá, ale před zatáčkou moc nepřibývá. Průměrná doba ze všech sto snímků odpovídá čárkované čáře. Jak lze vidět na grafu ze snímku 4.1, který odpovídá výpočtu pomocí navrhované přesnější metody, doba výpočtu jednoho snímku se pohybuje kolem čtyř vteřin. Po implementaci velmi zjednodušené metody, však na úkor přesnosti, se doba výpočtu výrazně zkrátila a každý snímek bylo možné zpracovat do jedné vteřiny, jak lze vidět na průběhu grafu 4.2. Následující tabulka 4.1 porovnává obě metody vůči reálnému záběru kamery a uvádí hodnoty, kterých by se v ideálním případě mělo při výpočtech dosáhnout.



Obrázek 4.2: Čas strávený na projekci snímků včetně výpočtu viditelnosti odstraněním duplicitních bodů.

Metoda	Průměrný čas 1 snímku [s]	Zpomalení
Reálná kamera	0.1	–
Navrhovaná přesná metoda	3.96	40×
Implementovaná metoda	0.69	10×

Tabulka 4.1: Porovnání výpočetních metod s reálnou kamerou jakožto ideálním řešením.

Díky optimalizaci metody se dosáhlo mnohem lepších výsledků, které se více přibližují očekávaným hodnotám. Přesto by stále rychlost výpočtů měla být až téměř desetkrát rychlejší, aby odpovídala reálné rychlosti kamery. Z důvodu nižší výkonnosti používaného zařízení je problém rychlosti řešen za použití více vláken, tím se náročné výpočty rozloží na více částí, které je možné spouštět paralelně, a celková doba projekce se značně urychlí. Práce s vlákny je více popsána v kapitole 4.3 Vizualizace bodů.

4.3 Vizualizace bodů

Jak uvádí sekce 4.2, rychlost implementovaného řešení projekce není ideální, proto se při vizualizaci musí počítat se značným zpomalením. Z tohoto důvodu je vykreslování bodů prováděno dvěma lehce odlišnými způsoby a uživatel si při spuštění animace může zvolit, kterou metodu preferuje.

První způsob, označovaný jako Real time, zpracovává snímky postupně, bez ohledu na zpomalení. Takže výsledná animace běží rychlostí, jakou jsou snímky zpracovávané. Druhý způsob implementace, Buffering, se snaží zpomalení kompenzovat. K tomu využívá vlákna, která umožňují paralelní vykonávání částí programu. Real time a Buffering implementace se liší hlavně ve způsobu běhu jednotlivých částí zpracování bodů, ale samotné zpracování

využívá stejné metody pro vizualizaci. Některé společné metody zpracování jsou uvedeny níže.

Získání pixelového snímku jako trojrozměrné pole

Díky projekci jsou body v 2D podobě a je možné je vykreslit. Po celou dobu až do teď se s body pracuje jako s polem ve tvaru $(N, 2)$, kde N označuje počet bodů a každý bod má dvě souřadnice (x, y). V dalším kroku se body převádějí na obraz, který je uložen v podobě třírozměrného pole pixelů o požadované výšce a šířce jako výsledný snímek. Pole je ve tvaru (řádky × sloupce × barva), přičemž barva je zastoupena ve čtyřech kanálech. První tři kanály obsahují barevné složky RGB a čtvrtý, tak zvaný alfa kanál, slouží k určení průhlednosti pixelu. Barva všech bodů je shodná a je vyčtena ze vstupního parametru od uživatele. Naopak intenzita je součástí mračna bodů a je definována pro každý bod. Výsledný obraz je tvořen zápisem barvy a intenzity do výsledného snímku na pozici odpovídající souřadnicím z původního pole bodů.

Získání komponenty pro vykreslení na obrazovce

Pro vykreslení se využívá komponenta `Widget` z Kivy frameworku, která zobrazí pole pixelů jako texturu. Velikost této komponenty je nastavena na základě rozměrů vykreslovaného snímku. Prvně se vytvoří prázdná textura typu RGBA se čtyřmi kanály, aby odpovídala vygenerovanému snímku, který se do textury načte už jen jako jednorozměrné pole, které textura podporuje. Pro správné vykreslení na obrazovce se do připravené komponenty, která určuje velikost a pozici, vloží obdélník s namapovaným obrázkem jako texturou a vyvolá se metoda pro aktualizaci. Tím je zajištěno, že dojde k propsání všech posledních změn. Ukázka 4.1 znázorňuje implementaci popsané metody.

```
# Vytvoreni textury ospravne velikosti.
texture = Texture.create(size=(img_width, img_height), colorfmt='rgba')
# Namapovani obrazu s body na texturu.
texture.blit_buffer(image.flatten(), colorfmt='rgba', bufferfmt='ubyte')

widget = Widget(size=(img_width, img_height))
# Vlozeni textury do komponenty.
with widget.canvas:
    Rectangle(texture=texture, size=widget.size, pos=widget.pos)
    # Aktualizace obrazovky.
    widget.canvas.ask_update()
```

Výpis 4.1: Vykreslení textury na obrazovku pomocí komponenty Widget.

Real-time vykreslení

Tato metoda vychází ze sekvenčního zpracování. Jinými slovy, zpracovává jednotlivé snímky postupně za sebou, ale do dalšího zpracování se pustí až po dokončení a vykreslení předchozího snímku. Je tím zajištěna správnost pořadí snímků, ale dochází ke zpomalování generování snímků z důvodu čekání na ukončení předchozí generace.

Celý proces zpracování a vykreslení bodů na obrazovku využívá jedno hlavní vlákno, které všechno postupně zpracovává. Proces se opakovaně spouští v intervalu daném nastavenou rychlostí od uživatele, vyčítá zadané parametry nastavení, jako je barva, a podle interního čítače snímků vyvolá metodu projekce, které předává konfigurační matice pro daný snímek. Projekcí jsou získána validní data k zobrazení a mohou být převedena do finální podoby. K tomu se využívá obou společných metod na vykreslení. Nejdříve dojde k vytvoření pixelového snímku, který je následně namapován jako textura na komponentu na obrazovce.

Při optimalizaci došlo k přidání druhého vlákna, které má na starost pouze projekci, což je výpočetně nejnáročnější proces. Účelem je snaha co nejvíce urychlit zpracování, ale přitom zachovat sekvenční přístup. Vykreslení bodů je spuštěno automaticky hned jakmile je projekce na druhém vlákně dokončena a vykreslení dále probíhá výše popsaným postupem. Projekce dalšího snímku v pořadí je započata až po dokončení vykreslování.

U sekvenčního vykreslování dochází k problému se synchronizací s videem na pozadí. Jelikož rychlost generování jednotlivých snímků není konstantní, ale více proměnlivá, stávalo se, že se snímky rozcházely. Proto bylo nutné zavést opatření, která tomuto zamezila. Více o synchronizaci snímků a videa vysvětluje sekce 4.5 Synchronizace.

Buffering vykreslení

Implementace této metody se zaměřuje na udržení rychlosti zpracování podle reálné kamery a je toho dosaženo za pomocí více vláken a společné fronty vygenerovaných snímků.

Princip spočívá v započetí procesu zpracovávání jednotlivých snímků hned, jakmile se uživatel přepne na tuto metodu, aniž by byla spuštěna samotná animace. Generování snímků probíhá ve čtyřech samostatných vláknech. Každé vlákno při spuštění nového generování získá index snímku, který je právě na řadě ke zpracování. Provede projekci bodů pro danou konfiguraci a s využitím společné metody převede body do tvaru pixelového snímku. V této podobě data ukládá do fronty. Jelikož zpracování každého snímku může trvat různě dlouho, je velmi pravděpodobné, že se projekce ve vláknech nedokončí ve stejném pořadí, v jakém začala. Proto s každým vložením nového prvku do fronty je potřeba snímky seřadit tak, aby bylo stále udržováno jejich správné pořadí. Z tohoto důvodu se do fronty pro každý snímek ukládá i jeho index, podle kterého se řadí. Jakmile je snímek zpracovaný a zařazen ve frontě, vlákno započne stejný postup pro další snímek na řadě. Takto probíhá plnění fronty a dokud není spuštěna animace, snímky stále jen přibývají.

Jakmile uživatel spustí animaci, v hlavním vlákně začne docházet k vyjímání snímků z fronty od nejnižšího indexu. Předtím, než dojde k vyjmutí, se ověřuje, že je ve frontě snímek s očekávaným indexem podle pořadí aktuálně zobrazovaných snímků. Pokud ne, je možné, že ještě nebyl zpracován a do fronty bude teprve zařazen, proto se o jeho vyjmutí pokusí znovu při dalším výběru z fronty. Po úspěšném získání snímku v pixelové podobě se opět využije společné metody, která z něj vytvoří texturu a namapuje ji na komponentu připravenou na zobrazení uživateli. Samotné vložení komponenty už musí probíhat z hlavního vlákna, jelikož Kivy framework nezajišťuje synchronizaci přístupu z různých vláken a změny by mohly způsobit nežádoucí chování. Proto veškerou interakci s grafickým uživatelským rozhraním zprostředkovává hlavní vlákno.

Aby se předešlo problémům s vyjímáním snímků z fronty, předtím než do ní byly vložené, je kontrolována minimální kapacita fronty. Pokud kapacita fronty klesne pod tuto nastavenou hodnotu, animace se zastaví, aby dále nedocházelo k vyjímání z prázdné fronty, a nechá se vlákny opět naplnit. Pro informaci uživatele se využívá stavového ukazatele

fronty, který oznamuje, kolik procent fronty je aktuálně naplněno. Ukazatel je možné vidět na obrázku 4.3 v pravém dolním rohu.

Obrázek 4.3: Aplikace při Buffering vykreslování se stavovým ukazatelem.

Jelikož se fronta začne hned průběžně plnit bez ohledu na aktuální snímek, změna barvy bodů v průběhu není možná. Barva by se aplikovala pouze na snímky, které se od té chvíle začnou vkládat do fronty, ale pokud je fronta o sto snímků napřed, změna by se projevila až po dlouhé době. To by na uživatele mohlo působit jako špatná responzivita prostředí a z tohoto důvodu změna barvy v průběhu této metody není možná.

4.4 Implementace pozadí ze snímků kamery

Pozadí tvoří důležitou část animace, při které se prolínají body a záběry z reálné scény. Dostupná data jsou v tomto případě jednotlivé snímky pořízené kamerou umístěnou na čele vlaku a časové razítko pro každý snímek. První možností, jak pozadí vykreslovat, je zobrazovat snímky po jednom a postupně v daném pořadí, tak jak byly pořízeny kamerou. To by ale znamenalo pro každý snímek explicitně přistupovat do umístění v úložišti pro jeho načtení. Z důvodu potřeby načítat velké množství obrázků za velmi krátkou dobu není toto řešení nejvhodnější, jelikož přístup do úložiště je považován za obecně náročnější operaci. Proto implementované řešení využívá časových razítek ke zjištění počtu snímků za vteřinu. Výsledná frekvence videa odpovídá 25 fps (frames per second – snímků za sekundu) a za pomocí nástroje FFmpeg lze ze snímků vytvořit video požadovaného typu. Pro účely vytvoření videa na pozadí byly použity tyto přepínače.

```
ffmpeg -framerate 25
       -i "Images/img_cam/%d.jpg"
       -c:v mjpeg
       -q:v 5
       -pix_fmt yuv420p
       output.avi
```

Kde `framerate 25` určuje frekvenci videa, přepínač `-i` označuje umístění vstupních souborů, přepínač `-c:v` definuje kódovací formát pro video, `-q:v` přepínač znamená kvalitu pro video, přičemž čím nižší číslo, tím lepší kvalita a `-pix_fmt` určuje formát pixelů. `Outpup.avi` je název výstupního videa.

Načtení a ovládání videa zajišťuje knihovna OpenCV. Konkrétně třída `VideoCapture()`, která poskytuje rozhraní pro čtení video souborů a po předání cesty k validnímu souboru jej načte do objektu pro další zpracování. Velkou výhodou tohoto přístupu je načítání videa jako jednotlivých snímků za pomoci metody `read()`, která vrátí snímek jako matici. Jelikož standardní barevný formát používaný v OpenCV je BGR, pro správné zobrazení je nutné ho změnit na formát RGB podporovaný Kivy komponentami. Samotné zobrazení snímku funguje podobným způsobem jako u vykreslování bodů. Snímek videa je ve tvaru matice s RGB kanály, tedy v pixelové podobě, a musí být převeden pomocí `frame.tobytes()` do kompatibilní podoby s Kivy texturami. V posledním kroku je nová textura namapována na stávající komponentu určenou pro zobrazování videa.

Funkce OpenCV knihovny umožňují velmi jednoduché ovládání videa. V případě, že dojde k restartování videa uživatelem, nebo pouze k přetočení videa posuvníkem do jeho jiné části, funkce jako

```
video.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)
```

zajistí nastavení požadovaného snímku přes objekt, do kterého bylo video načteno. Při spuštění videa dochází k cyklickému vytváření nové textury pro daný snímek, který se automaticky inkrementuje pro zobrazení dalšího v pořadí.

4.5 Synchronizace bodů a obrazu na pozadí

Implementace jednotlivých modulů pro vizualizaci bodů a pozadí popisuje zvlášť sekce 4.3 Vizualizace bodů a 4.4 Implementaci pozadí ze snímků. K vytvoření plynulé animace je nutné zajistit správné načasování obou částí. Pokud by synchronizace nebyla řešena, docházelo by k rozcházení obrazu, body z projekce by neseděly na snímky pozadí a při nesprávné rychlosti by jedna část stále vykreslovala další scény, i když druhá by byla dokončena a k dalšímu vykreslování by už nedocházelo.

Menší komplikací je neshoda vykreslovacích frekvencí mezi snímky projekcí bodů a snímky reálné scény z videa. Při udržování původní rychlosti, tak aby odpovídala pohybu vlaku, frekvence videa odpovídá dvaceti pěti snímkům za vteřinu, kdežto snímky získané projekcí bodů udávají pouze deset snímků za vteřinu. Obě hodnoty byly získány z konfiguračních souborů kamery. Z toho vyplývá, že nelze obě části vykreslovat zároveň ve stejný okamžik a po jednom snímku, obrazy by se určitě rozešly a neodpovídaly si navzájem.

Synchronizací je nutné zajistit dodržení frekvencí pro obě části. Při požadavku na spuštění animace dojde k volání společné metody, která nastaví všechny potřebné parametry

a vytvoří dvě časované události, každou pro jednu část animace. Tyto události jsou nastavené na pravidelné opakování v určitém intervalu. K tomu se využívá nástroje `Clock`, který je součástí Kivy frameworku.

```
self.points_event = Clock.schedule_interval(self.update_points, interval)
```

Takto se naplánuje opakované spouštění funkce `update_points()` s daným časovým intervalem. Obě části animace, vykreslování bodů a snímků z kamery, mají každá svou metodu, kde dochází ke zpracování požadavku na vykreslení. Právě tyto metody jsou vyvolávány událostmi, kde každá má nastavený vlastní interval opakování. Tím je zajištěno nezávislé vykreslování bodů i snímků se správnou frekvencí. Při nastavování rychlosti animace uživatelem dochází pouze k úpravě intervalu událostí tak, aby byl stále zachován stejný poměr frekvencí a došlo k ekvivalentnímu zpomalení nebo zrychlení u obou částí.

Jakmile má dojít k pozastavení animace, je nutné zrušit opakované spouštění události, aby se snímky přestaly vykreslovat. To zajistí také nástroj `Clock`.

```
Clock.unschedule(self.points_event)
```

Při úplném restartování snímků, kdy dojde k jejich zastavení a změně aktuálního snímku, je nutné zrušit všechny procesy spuštěné na vedlejších vláknech. Tak jak je uvedeno v kapitole 4.3 Vizualizace bodů, k vykreslování se využívají další vlákna pro urychlení. Především u typu vykreslování real-time se na vlákně spouští projekce bodů, která hned po dokončení automaticky spustí vykreslení snímku na obrazovku. Pokud došlo k přerušení animace předtím, než vlákno projekci dokončilo, znamená to, že se body vykreslí i po zastavení animace. Toto je nutné ošetřit. Nejvhodnějším řešením v tomto případě, jelikož se pracuje s více vlákny, je za pomocí synchronizačního prostředku zvaného mutex. Mutex, neboli zámek, je zamykatelný objekt, který zaručuje, že pouze jedno vlákno může přistoupit k chráněnému zdroji. Jakmile tedy dojde k zastavení animace, hlavní vlákno přistoupí pomocí mutexu k proměnné povolující vykreslení a zakáže jej. Při pozdějším dokončení procesů na některých vláknech se nejdříve ověří, že je vykreslení povolené, jinak k vykreslení nedojde. Povolení se opět nastaví až při dalším spuštění animace.

Obrázek 4.4 potvrzuje účinnost synchronizace real-time vykreslování. Animace byla spuštěna a po určité době zastavena. Díky synchronizaci a včasnému ukončení procesů ve vláknech, oba snímky bodů i pozadí si navzájem stále odpovídají.

V případě Buffering způsobu vykreslování dochází k podobnému principu, s tím rozdílem, že se mutex využívá k přistupování ke frontě vygenerovaných snímků. Vkládání i vybírání z fronty řídí mutex, který zajišťuje, že frontu mění vždy pouze jedno vlákno a že nedochází k nekonzistentním stavům fronty. Při restartování animace dojde k vyprázdnění celé fronty, aby se mohly začít vkládat nové snímky generované v pořadí od aktuálně zobrazovaného. Opět je nutné ukončit procesy všech vláken, tím se vlákna uvolní a začnou zpracovávat snímky od začátku.

Synchronizaci je nutné udržet i při změně rychlosti animace. U obou typů vykreslování to vyžaduje určité kroky navíc. Real-time vykreslování se vyznačuje postupným zobrazováním snímků, které je dané rychlostí jejich zpracování. Ta se bohužel nerovná rychlosti reálného pohybu vlaku, ale je daleko pomalejší. Z tohoto důvodu dochází k přeskakování snímků při nastavení vyšší rychlosti, než které stačí implementace. Při spuštění prvního snímku začne jeho zpracování a na obrazovku se vykreslí hned, jakmile je dokončeno. Pokud je rychlost tak vysoká, že mezitím přišel požadavek na zpracování dalšího snímku, nemůže být požadavku vyhověno a daný snímek se přeskočí. Snímky jsou přeskakovány tak dlouho, dokud nedojde k vykreslení prvního snímku. Až po té lze spustit zpracování dalšího

Obrázek 4.4: Zobrazení odpovídajících si snímků bodů a pozadí díky synchronizaci

snímku aktuálně v pořadí. Tento přístup synchronizaci lehce porušuje, jelikož vykreslování je zpožděné a snímky se neshodují. V animaci to způsobuje posunutí bodů vůči pozadí a trhané vykreslování.

U buffering typu vykreslování k tomuto nedochází, ale plynulost obrazu je závislá na množství vygenerovaných snímků ve frontě. Pokud se stihne vygenerovat všechny snímky ještě před spuštěním animace, je zaručen hladký průběh animace až do konce. V případě, že dojde ke spuštění animace příliš brzy, kdy není vygenerováno dostatečné množství snímků a rychlost je natolik vysoká, dojde při animaci k úplnému vyprázdnění fronty. S prázdnou frontou není možné udržet nastavenou rychlost vykreslování a aby nedocházelo k rozladění bodů a pozadí, celá animace je pozastavena. Mezitím vlákna s generováním snímků stále běží a postupně naplňují frontu. Uživatel má možnost animaci opět kdykoli pustit.

4.6 Načítání souborů z paměti zařízení

Rozhodnutí o implementaci importu souborů podpořilo více důvodů. Především to rozšíří využití aplikace a zvýší uživatelskou přívětivost. Uživatel nebude omezen pouze na předem definovaná data, ale bude mít větší volnost při práci s vlastním datovým setem. Díky tomu se aplikace stává více univerzální a přizpůsobitelná konkrétním potřebám jednotlivých uživatelů. Další výhodou je výsledná velikost instalačního balíčku. Tím, že není nutné přikládat všechny datové soubory přímo do balíčku, dojde k výraznému snížení celkové velikosti aplikace. Tyto soubory, především videa z kamery, jsou paměťově náročnější a jejich vynecháním se zrychlí samotné stahování i instalace aplikace. Uživatel si importuje veškerá potřebná data po instalaci aplikace.

Při zvažování, jaké varianty importu dat by přicházely v úvahu, se porovnávaly dvě hlavní možnosti. První varianta spočívá v importu jediného souboru, který by obsahoval

seznam cest v úložišti zařízení, kde se jednotlivé soubory nacházejí. Pravděpodobně by se jednalo o soubor JSON nebo XML, který by si každý uživatel musel vytvořit na základě rozmístění dat v jeho zařízení. Tento soubor by pomocí správce souborů byl předán aplikaci, která by ho přečetla a zpracovala. Cesty k datům uložila do vlastních datových struktur, a jakmile by bylo dat potřeba, tak znovu přistoupila do paměti a soubory si přečetla. Tento přístup je náročnější a případně i odbornější, jelikož vyžaduje od uživatele větší přípravu importovaných dat a znalost daného typu souboru. Z tohoto důvodu by nemusel být zcela vhodný.

Druhá varianta využívá komprimovaného souboru, což je považováno za běžnější způsob importu dat, se kterým se uživatelé spíše už někdy setkali. Veškeré soubory, které by byly potřeba nahrát do aplikace, jsou vloženy do jednoho adresáře a komprimovány do souboru typu ZIP. Aplikaci je pomocí správce souborů předán také jediný soubor, který si následně exportuje a jeho obsah uloží na předem stanovené místo, kam může přistupovat kdykoli při běhu programu a data má stále k dispozici. Z důvodu větší uživatelské přívětivosti a preference pro rozšíření uživatelských možností byla v aplikaci implementována tato metoda, která je dále rozepsána.

Primárním cílem je umožnit uživateli nahrávat větší množství různých typů souborů. Z tohoto důvodu musí aplikace uživateli umožnit prohlížení složek na jeho zařízení. S využitím frameworku Kivy a rozšíření KivyMD byla zvolena komponenta MDFileManager pro správu souborů a navigaci ve složkách [11].

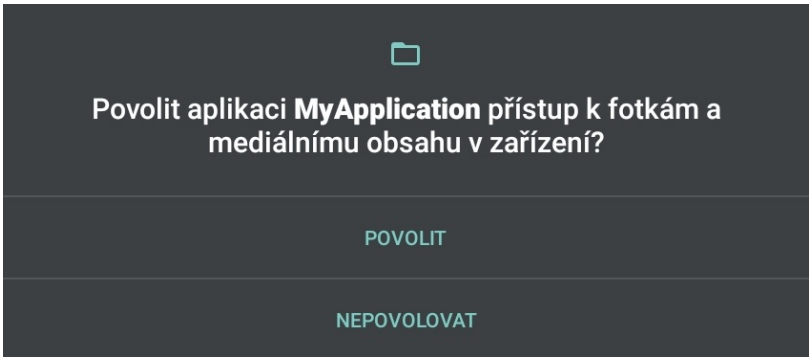
MDFileManager není systémový správce souborů, který má každé zařízení, ale jedná o KivyMD komponentu zobrazující vlastní grafické rozhraní. K tomu využívá mnoho dílčích komponent, jako je *ScrollView, OneLineListItem, MDList* a jiné. Výhodou tohoto přístupu je možnost snadné přizpůsobitelnosti vzhledu. Pro navigaci v souborovém systému se využívá reálných (absolutních) cest, které přímo odpovídají fyzickému umístění souborů a složek na interním úložišti zařízení. Při inicializaci správce je specifikována výchozí cesta k adresáři, jehož obsah je následně zobrazen v uživatelském rozhraní. Po výběru souboru uživatelem správce vrací plnou cestu k danému souboru ve formě řetězce znaků, se kterým může aplikace dále pracovat.

```
# ziskani cesty k primarnimu externimu ulozisti.
path = primary_external_storage_path()
file_manager = MDFileManager(
    # funkce zpracovavajici zavreni prohlizece bez vyberu souboru.
    exit_manager=self.exit_manager,
    # funkce zpracovavajici vybrany soubor.
    select_path=self.select_path,
)
# zobrazeni spravce souboru s vychozi cestou.
file_manager.show(path)
```

Výpis 4.2: Použití nástroje MDFileManager k procházení souborů.

Jelikož *MDFileManager* není výchozí správce souborů, nemá automaticky přidělený přístup k úložišti daného zařízení a je potřeba si zažádat o práva pro čtení a zápis do externí paměti. S vývojem Androidu se tato oprávnění a způsob o jejich žádání v jednotlivých verzích lehce liší převážně z důvodu bezpečnosti. U prvních verzí androidů se oprávnění

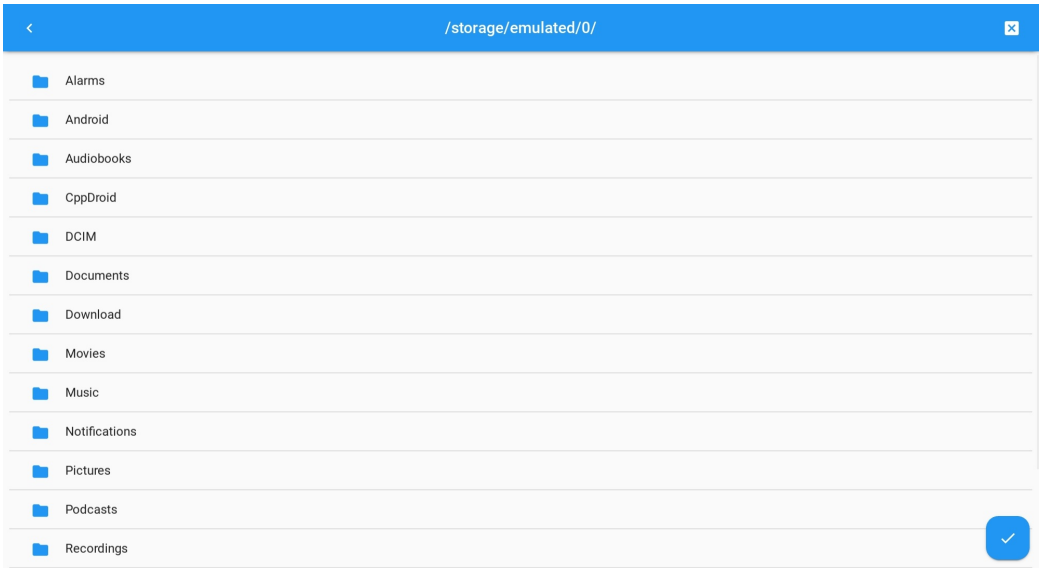
udělovala při instalaci, tedy bylo potřeba uvést do manifestu aplikace žádost o práva čtení a zápisu. Po udělení měla aplikace plný přístup k veřejnému úložišti, ale hrozilo riziko zneužití dat. Později přišel Android se zásadní změnou, oprávnění za běhu aplikace. I když jsou všechna oprávnění uvedena v manifestu, musí se po spuštění aplikace při prvním pokusu o přístup do úložiště znovu požádat uživatele o souhlas, který jej může potvrdit, nebo zamítnout na zobrazeném dialogu. Tím byla zvýšena kontrola uživatele nad přístupem k jeho datům. U novějších verzí Androidu byla zavedena scoped storage, neboli vyhrazená část paměti, ke které má aplikace přístup. Je tam jmenovitě vytvořená složka pro každou aplikaci, kam je možné libovolně zapisovat i číst. Jsou tím chráněna data aplikací i data uživatele před aplikacemi. Dále je možné využívat sdílené paměti, kam mohou přistupovat všechny aplikace s uděleným oprávněním. Jedná se o umístění `Downloads/`, `Documents/`, `DCIM/` a podobné.



Obrázek 4.5: Žádost o oprávnění za běhu aplikace při prvním pokusu o přístup k úložišti.

Po nastavení všech nutných oprávnění a správného výchozího adresáře se aplikace vždy při prvním spuštění správně dotázala na potvrzení přístupu k mediálním souborům, jak je znázorněno na obrázku 4.5 a při požadavku na import dat se otevřela stránka správce se soubory k výběru. Grafické rozhraní správce, tak jak zachycuje obrázek 4.6, se vyznačuje minimálním designem s předpokladem, že vzhled bude upraven pro konkrétní potřeby uživatele.

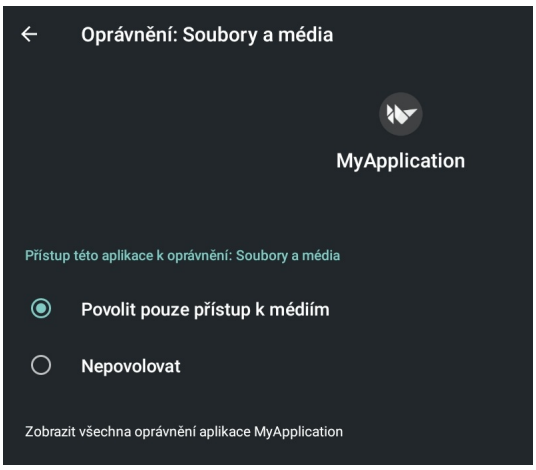
Základní funkce pro navigaci, přecházení mezi adresáři pomocí tlačítka zpět, nefungovalo zcela správně. Bylo možné se dostat do vyšších adresářů, než byl nastavený kořenový adresář, a pak docházelo ke ztrátě kontextu mezi mnoha systémovými složkami. Daleko větší problém však byl, že se ve složkách nezobrazovaly všechny soubory, které tam při prohlížení úložiště měly být. Přestože při inicializaci správce bylo povoleno všech typů přípon souborů, v aplikaci byly zobrazeny pouze `.jpg` a `.png` soubory. Analýza tohoto chování odhalila, že u novějších verzí androidu je potřeba kromě oprávnění `READ_EXTERNAL_STORAGE` a `WRITE_EXTERNAL_STORAGE` přidat i oprávnění `MANAGE_EXTERNAL_STORAGE`, jelikož bez něj má aplikace přístup pouze k mediálním souborům na sdíleném úložišti. Ale ani po zahrnutí tohoto oprávnění nebylo dosaženo požadovaného efektu a přístup k dalším typům souborů zůstával stále omezený. Jako ověření, že opravdu jde o problém s přiděleným oprávněním aplikaci, nikoli o chybu v kódě při inicializaci správce, byla provedena inspekce nastavení aplikace přímo na nainstalovaném zařízení. Na stejném místě, kde některé aplikace měly povolenou správu všech souborů, měla aplikace povolen pouze přístup k médiím a žádnou možnost to změnit, tak jako znázorňuje obrázek 4.7. Poslední experiment měl za úkol



Obrázek 4.6: Otevření správce souborů MDFileManager v kořenovém adresáři.

zjistit, podle čeho se rozpoznávají mediální soubory od ostatních. Testovací data představuje soubor, který byl vytvořen jako textový s příponou `.txt` a uložen na zařízení do sdíleného úložiště, tak aby k němu aplikace měla zaručený přístup. Následně došlo pomocí přejmenování ke změně jeho přípony na mediální typ `.jpg`. Při ověření přes aplikaci a implementovaný správce souborů se testovací soubor začal zobrazovat v seznamu a přestože se dle filtru jedná o mediální typ, aplikace byla schopna daný soubor otevřít a vyčíst obsah jako běžný textový soubor. To dokazuje, že při rozdělování souborů na média a soubory se bere v potaz pouze přípona, nikoli reálný typ souborů.

Tato zjištění vedla k závěru, že bude vhodnější přejít na alternativní způsob správy souborů. Najít řešení, které umožní konzistentní načítání a zpracování libovolných typů datových souborů.



Obrázek 4.7: Nastavení, které umožňuje udělit oprávnění pouze k mediím.

Po předchozím naražení na výše popsaný problém bylo nalezeno řešení, při kterém aplikace vyvolá výchozího systémového správce souborů, který je součástí operačního systému

Android, a to prostřednictvím *Storage Access Frameworku* (SAF). K tomu se používá Java Native Interface (JNI) přes knihovnu PyJNIus. Tímto je podpořeno přirozenější prostředí pro uživatele, který svého správce určitě zná a využívá, tudíž se v něm již orientuje a nemusí si zvykat na jeho novou interpretaci. Toto řešení je pro daný účel nejvhodnější, splňuje hlavní požadavky, které jsou popsány níže, a z tohoto důvodu je právě takto v aplikaci implementováno. Ukázka systémového správce souborů je na obrázku 4.8, kde je otevřený kořenový adresář.

Obrázek 4.8: Otevření systémového správce souborů prostřednictvím SAF.

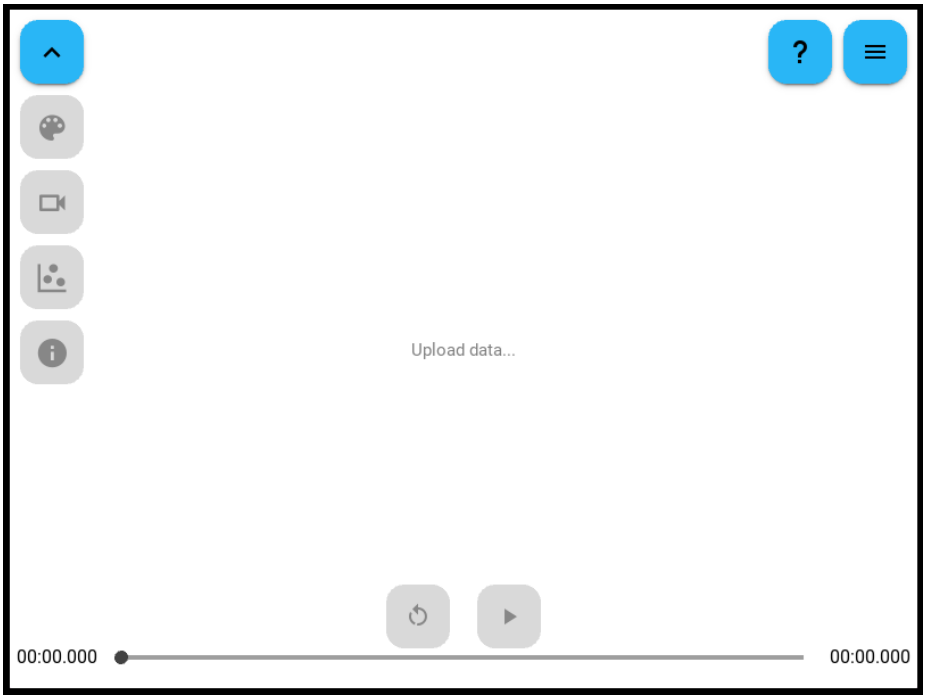
SAF je mechanizmus, který uživateli umožňuje bezpečným a standardizovaným způsobem vybírat soubory nebo složky z různých typů úložišť – od interní paměti, přes SD kartu až po cloudové služby typu Google Drive. Cílem SAF je zajistit řízený přístup uživatelem k souborům a při tom plně respektovat zásady ochrany soukromí a omezení, vyplývající z konceptu scoped storage pro jednotlivé verze Androidu. Pro interakci se systémovým správcem se využívá Java třídy *Intent*, představující jednu ze základních komponent Android API, které slouží k zasílání asynchronních požadavků. V případě SAF se jedná například o `Intent.ACTION_OPEN_DOCUMENT`, který spustí systémový výběr souboru, dále `Intent.ACTION_CREATE_DOCUMENT` pro uložení nového souboru do paměti, nebo `Intent.ACTION_OPEN_DOCUMENT_TREE` pro výběr celé složky.

V prostředí Python je přístup k těmto Java třídám a metodám zajištěn prostřednictvím *Java Native Interface* (JNI). V Pythonu se konkrétně používá knihovna PyJNIus, která umožňuje dynamicky načítat Java třídy pomocí *autoclass* a volat jejich metody, jako by šlo o běžné Python objekty. Díky tomu lze inicializovat *Intent*, nastavit jeho typ, definovat filtry a následně jej spustit pomocí metody *startActivityForResult*, přičemž výsledek je ošetřen ve funkci navázané na danou aktivitu.

Větším rozdílem u tohoto způsobu je typ návratové hodnoty předávané ze systémového správce. Nejedná se totiž o absolutní cestu, která udává fyzické umístění na úložišti, ale o tak zvaný *Uniform Resource Identifier* (URI), který jednoznačně označuje určitý zdroj. Představuje abstraktní reprezentaci souborů a obsahu, zjednodušeně řečeno, URI je odkaz na objekt, nikoli jeho fyzická cesta. Dále je URI spojováno i s oprávněním, které je automaticky přiděleno Androidem, ale pouze pro zvolený soubor. Aplikace tak získá právo

obsah číst, případně i zapisovat, aniž by musela žádat o globální oprávnění. To zajišťuje daleko větší míru bezpečnosti, jelikož aplikace nemůže přistupovat mimo tento konkrétní obsah. Co se týče formátu, tak řetězec nejčastěji začíná schématem `content://`, což značí, že se jedná o data poskytovaná z Android komponenty *Content Provider*. Ten spravuje data, poskytuje řízený přístup k datům určitého typu a rozhoduje, kdo a jak k nim může přistupovat. Pro komunikaci s *Content providerem* se využívá třída *Content Resolver*, která aplikaci slouží k získání dat reprezentovaných pomocí URI.

Velkou výhodou tohoto přístupu je, že není potřeba žádat o jakákoli oprávnění. Při výběru souboru nebo složky se využívá bezpečného systémového dialogu. Android sám zajišťuje řízení přístupu k vybranému zdroji. Uživatel tím, že vědomě vybere konkrétní soubor nebo složku, automaticky uděluje aplikaci dočasné oprávnění k tomuto objektu.



Obrázek 4.9: Aplikace po prvním spuštění, čekající na nahrátí souborů.

Nahrání videa do aplikace

Jedním z hlavních účelů správce souboru je importování videa do aplikace, tak aby bylo možné jej zobrazovat na pozadí animace. Samotná implementace videa a jeho ovládání je popsána v sekci 4.4, zde je uvedeno, jakým způsobem a kam je video uloženo.

Předpokladem pro úspěšné nahrání je, že má uživatel na svém zařízení video ve formátu AVI. Jiné typy video souborů aplikace nepodporuje a uživatel je na to náležitě upozorněn. Jelikož video není součástí instalačního balíčku aplikace, pro spuštění videa na pozadí projekce v aplikaci je nutné ho nejdříve vybrat ze souborů a nahrát. To je především z praktického důvodu, kdy video soubory jsou často objemné a tedy paměťově náročné. V případě přiložení videa rovnou k instalačnímu balíčku by nejen zvětšilo jeho velikost, ale také dobu stahování a následné instalace. Dále proto, že se zaměřuji na univerzálnost aplikace, tedy umožňuji uživateli použít jeho vlastní data, která má možnost do aplikace nahrát, není tedy nezbytně nutné přikládat výchozí data.

Při nahrávání videa se využívá systémového správce souborů, popsaného výše v úvodu této sekce 4.6, za pomoci SAF přístupu do paměti. Požadovaný soubor se může nacházet ve sdílené části interního úložiště zařízení, nebo i na externím úložišti, jako je SD karta nebo Google Drive. Výběrem daného souboru uživatel opravňuje aplikaci k jeho čtení, případně zápisu. Aby aplikace mohla video soubor využívat dle potřeby, dojde k vytvoření jeho kopie, která se uloží ve vyhrazené části úložiště přístupné pouze konkrétní aplikaci. Při výběru souboru se vrací odkaz URI jako výsledná hodnota, proto pro jeho čtení je zapotřebí *Content Resolver*. Konkrétně je využita jeho metoda *OpenInputStream* k otevření binárního streamu pro čtení dat, na které URI ukazuje. Následně se otevře výstupní binární stream pro zápis do souboru v požadovaném adresáři a kopírování probíhá ve smyčce s metodami *read()* a *write()* pomocí předalokované vyrovnávací paměti o velikosti 4KB. Popsaný způsob je znázorněný na ukázce kódu níže. Z důvodu ilustrace je ukázka zkrácena pouze na nejdůležitější části, validace proměnných a blok `try-except`, ve kterém je kód uzavřený, byly vynechány.

```
dest_file = os.path.join(file_dir, file_name)
input_stream = self.content_resolver.openInputStream(uri)
with open(dest_file, "wb") as f_out:
    buffer = bytearray(4096)
    while True:
        read_len = input_stream.read(buffer)
        if read_len <= 0:
            break
        f_out.write(buffer[:read_len])
input_stream.close()
```

Výpis 4.3: Kopírování souboru pomocí OpenInputStream metody.

Po spuštění kopírování videa se uživatel dostává zpět na hlavní stránku aplikace, kde je informován o nahrávání videa a jeho zpracování. Jakmile je video nastavené, zobrazí se na pozadí, notifikace zmizí a jedná-li se o první nahrání souboru s videem, zpřístupní se uživateli veškeré možné akce pro práci s videem, které byly dosud zakázané.

Obrázek 4.10 zachycuje aplikaci ve stavu, kdy uživatel nahrál první soubor, právě video, které teď zobrazuje první snímek. Aplikace umožňuje uživateli použít nástroje pro přehrávání videa ve spodní části obrazovky a zpřístupnila některé funkce po levé straně, které jsou také vyhrazené k ovládání videa.

Obrázek 4.10: Úspěšné zobrazení prvního snímku nahraného videa.

Nahrání dat z komprimovaných souborů

Nahráváním dat je myšlen import samotného mračna bodů a konfiguračních souborů obsahujících matice a vektory pro výpočet projekce bodů do roviny. Obecně by se dalo říct, že je v tomto případě použit velmi podobný princip, jako při nahrávání videa, popsaný v části výše, jen s rozdílem, že je potřeba provést několik kroků navíc, jelikož se jedná o komprimovaný soubor. Uživatel se nejdříve musí ujistit, že se na jeho zařízení nachází soubor typu .zip s dodrženou předem definovanou strukturou adresáře. Tedy, že veškeré soubory se nacházejí v jedné hlavní složce a musí být řádně pojmenované, jinak není aplikace schopna od sebe soubory rozeznat a nemůže data správně zpracovat. Pokud by taková situace nastala, uživatel je upozorněn notifikací, že data nebyla kompletně načtena z důvodu jmenné konvence a nahrávání je potřeba opakovat.

Celý proces nahrávání probíhá následovně. Nejprve se aplikace přesvědčí, že uživatel opravdu vybral komprimovaný soubor typu ZIP. Aby bylo možné s daty pohodlně pracovat, je soubor překopírován do aplikační části úložiště stejným způsobem, jaký popisuje sekce 4.6 nahrávání videa a znázorňuje ukázka kódu 4.3. Po úspěšném vytvoření kopie dochází k extrakci všech souborů za využití Python modulu *zipfile* a jeho metody *extractall()*, které je předána cesta uložení komprimovaného souboru. Jakmile je extrakce dokončena, dojde k odstranění kopie komprimovaného souboru a přechází se na zpracování dat. Pokud předtím již byla spuštěna animace s jinými daty, vše je restartováno do výchozí podoby, včetně videa, pokud bylo nahráno. Začne výpočet prvního snímku bodů, textury jsou nově vykresleny s aktuálními daty a jakmile je vše dokončeno, aplikace přestane zobrazovat notifikaci nahrávání souborů a je uživateli plně k dispozici. Pokud video zatím nebylo nahráno, aplikace se nachází ve stavu, jaký uvádí obrázek 4.11 níže. Došlo k vykreslení prvního snímku na základní pozadí a uživateli jsou zpřístupněny ovládací prvky pro spuštění animace a další nástroje, které ovlivňují zobrazení bodů.

Obrázek 4.11: Vykreslení prvního snímku z nahraných souborů.

Dále je potřeba zmínit, že díky využití systému SAF je veškerá práce se soubory bezpečně řízena v souladu s pravidly Androidu. Soubory, které jsou prostřednictvím SAF zkopírovány a uloženy do části interního úložiště vyhrazené pro danou aplikaci, jsou po odinstalaci aplikace automaticky odstraněny. To znamená, že veškerý obsah uložený v tomto prostoru – ať už se jedná o dočasné soubory, importovaná data nebo výstupy aplikace – je nenávratně smazán spolu s aplikací. Tímto mechanismem je zajištěno, že aplikace po sobě nezanechává žádné zbytkové soubory ani data, čímž se eliminuje riziko zahlcení úložiště zařízení nebo potenciálního úniku citlivých informací.

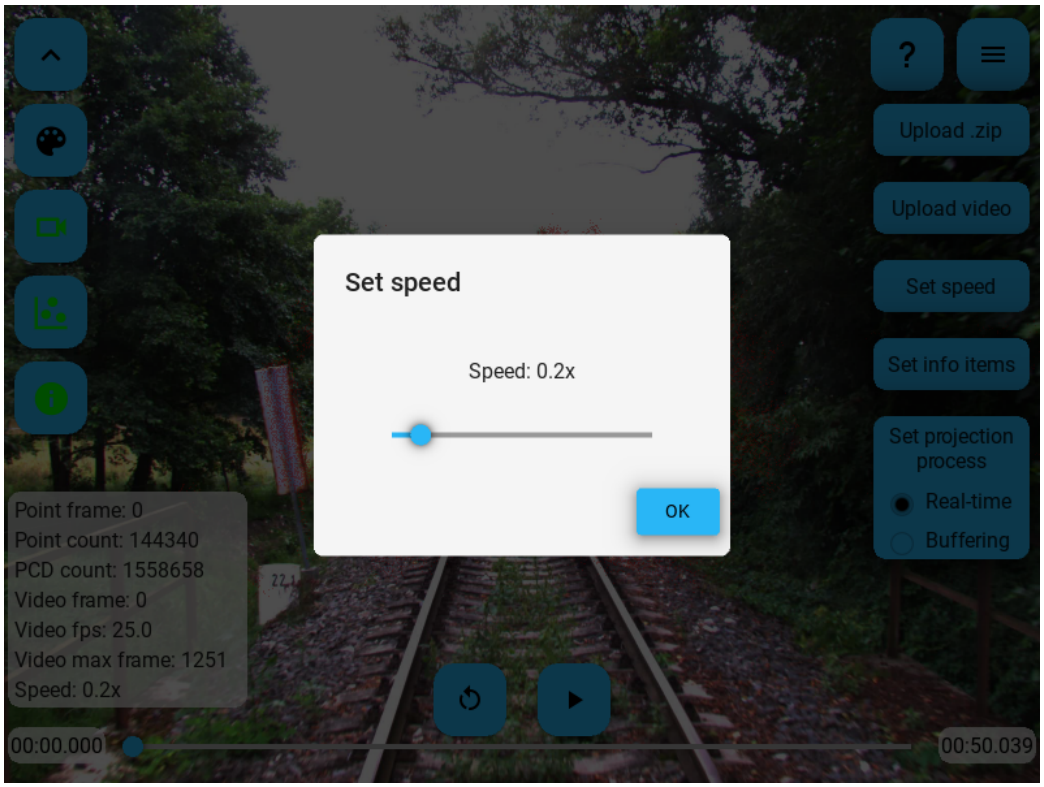
Jedinou nevýhodou tohoto přístupu s komprimovaným souborem je nutné dodržení definovaného pojmenování jeho obsahu. To je z důvodu, že se po extrakci bude nacházet v adresáři několik souborů stejného typu a aplikace není schopna rozpoznat, o jaký soubor se jedná, jinak než podle názvů. Možným řešením uvedeného problému by bylo přidat do všech souborů hlavičku obsahující metadata, díky kterým by soubory byly rozeznatelné. Funkci hlavičky by mohl zajišťovat první řádek souboru, kde by byl každý soubor jednoznačně označen kódem, který zastupuje název souboru. Aplikace by pak mohla provést extrakci souborů bez dalších validací. Pouze při vyčítání dat by bylo nutné nejdříve soubor otevřít, podle jednoznačného kódu identifikovat a přiřadit ke správné datové struktuře, aby dále byla data použita správným způsobem a nedošlo k záměně. Tímto by opadla nutnost striktního pojmenovávání, ale přibylo vyplňování hlaviček souborů. Aktuálně aplikace vyžaduje přesné názvy souborů, ale v případě dalšího rozšiřování by byl implementován způsob kontroly hlaviček.

4.7 Funkce implementované pro zvýšení komfortu uživatele

Jak vychází z návrhu aplikace uvedené v kapitole 3, uživatelské rozhraní musí být natolik intuitivní, aby základní akce zvládl uživatel provést již při prvním použití. Některé zde popsané funkce vycházejí čistě z požadavků na systém, tedy jejich implementace byla nutná pro splnění zadání. Jedná se o ovládání rychlosti animace a výběr položek zobrazovaných ve stavu animace. Druhá část funkcí byla implementována z důvodu zjednodušení práce s aplikací. Uživateli je poskytnuta nápověda k ovládání některých funkcí a ukládá se stav aplikace, ve kterém byla ukončena.

Ovládání rychlosti animace

Součástí parametrů, které má uživatel možnost si nastavit, je i samotná rychlost animace, která určuje, kolik snímků se stihne vykreslit do jedné vteřiny. Problémy spojené s rychlostí animace a jejich řešení více popisuje část 4.5 synchronizace. Hodnota parametru udává násobek původní rychlosti, tedy nastavení hodnoty na 1 odpovídá originální, nijak upravené rychlosti, jakou se kamera pohybovala při pořizování snímků. Naopak hodnota 0.1 představuje zpomalení desetkrát vůči původní rychlosti.

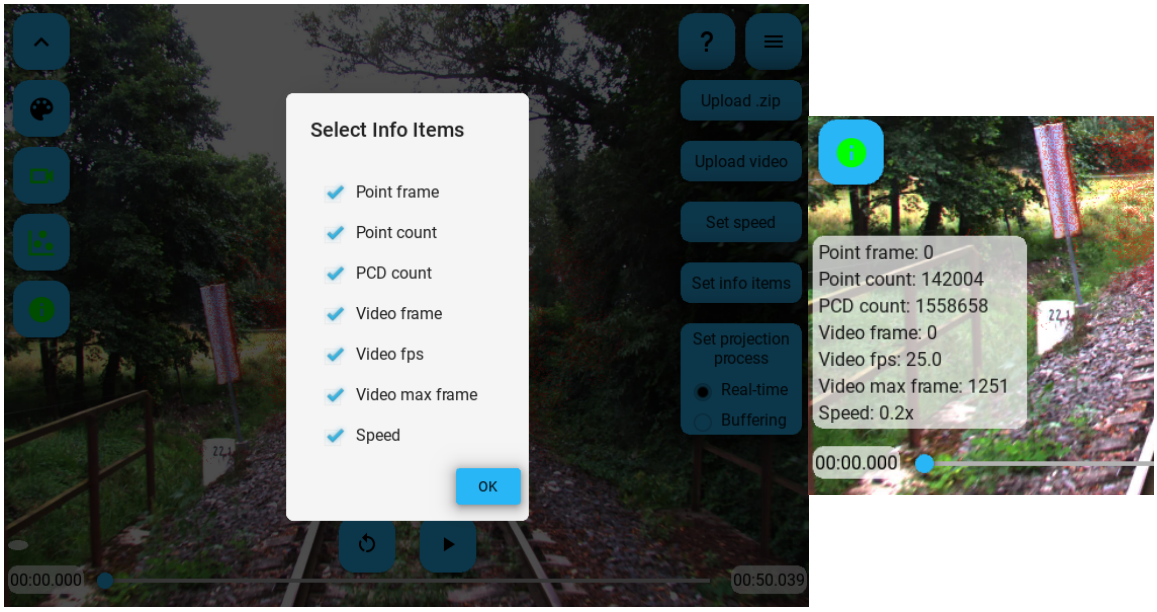


Obrázek 4.12: Dialogové okno pro nastavení rychlosti animace.

Uživatel nastavuje rychlost pomocí posuvníku na obrázku 4.12, který má určený rozsah adekvátně k možnostem aplikace. Při změně rychlosti dojde k restartování celé animace, jelikož je potřeba přerušit události, které definují interval generování snímků, a znovu je nastavit a spustit s ohledem na nově nastavenou rychlost.

Výběr zobrazovaných položek ve stavu animace

Na hlavní obrazovce aplikace se nachází sekce stav animace, která pravidelně aktualizuje hodnoty vybraných parametrů. Jaké parametry se mají zobrazovat, si uživatel může zvolit z daného seznamu zcela dle své libosti. Zobrazení seznamu při požadavku na změnu je implementováno jako dialogové okno, ve kterém se zobrazí jednotlivé položky se zaškrtávacími políčky, jak je představeno na obrázku 4.13 uvedeném níže.

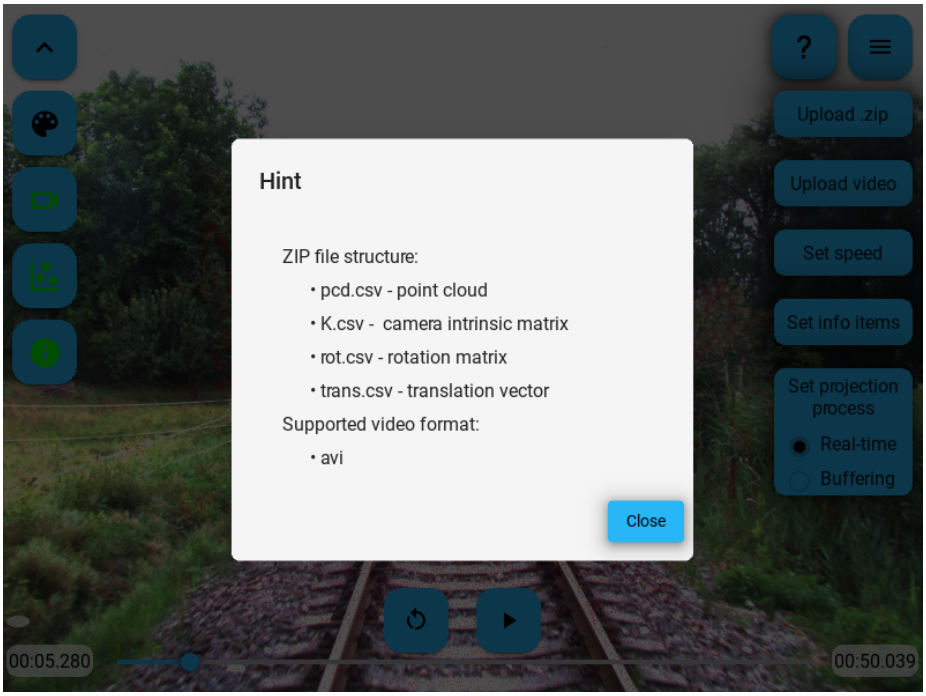


Obrázek 4.13: Dialogové okno pro výběr položek zobrazených v části obrazovky Stav animace.

Jakmile je uživatel spokojen se svým výběrem, zavře okno potvrzovacím tlačítkem a změny se hned propíšou. Všechny zvolené parametry se zobrazují v levém dolním rohu obrazovky na zesvětlené části pro lepší viditelnost. Tato část je dynamicky proměnná a její velikost vždy odpovídá aktuálně zobrazovanému obsahu tak, aby zabírala co nejméně místa na obrazovce. Pokud chce uživatel dočasně informace skrýt, může k tomu využít přepínač označený symbolem malého měkkého *i* a nemusí měnit výběr položek v nastavení.

Uživatelská nápověda

Uživatelská nápověda slouží k informování uživatele o faktech aplikace, které nemusí být na první pohled známé. V tomto případě nápověda obsahuje informace k nahrávání souborů. Definuje, jaká je validní struktura ZIP adresáře tak, aby aplikace byla schopná data zpracovat. A dále určuje podporované formáty videa. Implementace nápovědy je znázorněna na obrázku 4.14.



Obrázek 4.14: Zobrazení okna s uživatelskou nápovědou.

Ukládání stavu aplikace

Usnadnění používání aplikace zajišťuje funkce, která ukládá stav aplikace. Cílem této funkce je obnovit nastavení aplikace do stavu, ve kterém byla naposledy použita. Při práci s aplikací uživatel může měnit zobrazovací prvky, určuje data, která mají být viditelná, a tímto si definuje vlastní preference aplikace. První spuštění aplikace vytvoří soubor typu JSON s výchozími hodnotami pro všechny prvky, u kterých se provádí záloha jejich stavu. Jedná se především o parametry, které ovládají zobrazení animace, jako je viditelnost bodů a pozadí, barva bodů, rychlost animace, jestli je zapnutá funkce na aktualizaci stavů animace a které parametry mají být zobrazeny. Dále se zde ukládá příznak importovaných souborů. Soubor je uložen v úložišti zařízení v části vyhrazené pouze pro danou aplikaci. To zajišťuje bezpečný přístup k souboru a při odinstalaci aplikace dojde k jeho automatickému smazání.

S každou změnou uživatelského rozhraní, pokud jde o prvek, který je určen k záloze, dojde k aktualizování jeho stavu v souboru. Aktualizace jsou prováděny hned po změně parametru, aby se zajistila pravidelná záloha a vyvarovalo se ztrátě dat v případě nevyžádaného ukončení aplikace. Do souboru je ukládán i příznak importovaných souborů.

Při dalším spuštění aplikace nejdříve probíhá pokus o obnovu dat ze souboru. Zkontroluje se jeho existence na očekávaném místě a po úspěšném nalezení započne nastavování parametrů podle jejich uložených stavů. Jestliže byl nastaven příznak importu na pravdivý, znamená to, že uživatel už importoval data, se kterými aplikace má pracovat. Proto se aplikace pokusí o jejich načtení. Díky uložení souborů na konkrétní místa ve svém aplikačním úložišti není problém s jejich čtením hned po spuštění aplikace. Pokud by soubor nebyl nalezen, nebo došlo k jeho poškození a nebylo by možné jeho čtení, aplikace použije výchozí hodnoty jako při prvním spuštění a starý soubor nahradí nově vytvořeným s validními hodnotami.

Kapitola 5

Testování

Testování je nesmírně důležitou částí každého vývoje. Jeho hlavním cílem je zajistit, aby systém splňoval předem stanovené požadavky a fungoval dle očekávání. Testování umožní odhalit potenciální chyby systému včas, před tím, než dojde k jeho zveřejnění. Díky testování vzniká kvalitnější a spolehlivější aplikace pro uživatele. Pro vyvíjecí i testovací účely bylo zvoleno Android zařízení. Samotný proces testování probíhá opakovaně a lze jej rozdělit do dvou hlavních fází, kterým se věnuje tato kapitola. Jedná se především o funkční testování aplikace, které zajišťuje, že při běhu aplikace nedochází k chybám, a uživatelské testování soustředící se na interakci s uživatelem.

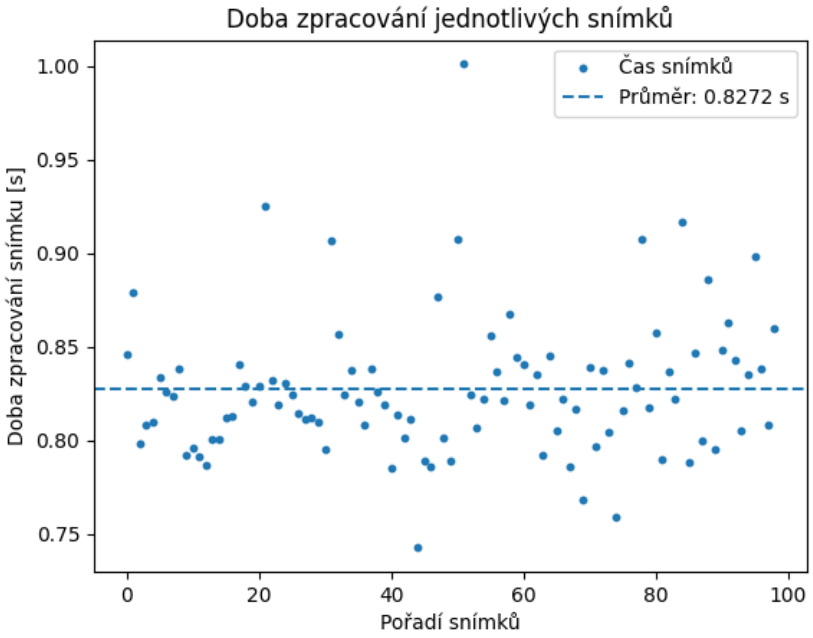
5.1 Funkční testování

U funkčního testování se ověřuje, že vyvíjená aplikace splňuje definované specifikace a že všechny její části fungují správně podle očekávání. Testováním jsou podrobeny všechny moduly a jejich funkce. Musí být zajištěno, aby dané funkcionality poskytovaly požadované výsledky a vyvolávaly odpovídající reakce. Důkladným testováním se zamezí výskytu chyb a předejde se neočekávanému chování, které by mohlo aplikaci uvést do nedefinovaných stavů.

K testování docházelo průběžně po dobu celého vývoje, především po implementaci nové funkcionality bylo ověřeno její očekávané chování a že svou implementací chybně neovlivnila stávající funkce. Testování probíhalo manuálně podle předem stanovených testovacích scénářů. Využívalo se k tomu nástroje ADB, který umožňuje komunikaci počítače s připojeným zařízením skrz USB kabel. Datový výstup z aplikace byl přesměrován do terminálu pro pohodlnější čtení ladících zpráv, díky kterým byla zajištěna detekce chyb. Tímto způsobem docházelo i k opravě všech nalezených chybových stavů, přičemž některé jsou uvedeny níže.

Sjednocení formátu vybrané barvy – Při implementaci funkce pro volbu barvy vykreslovaných bodů se využívá Kivy komponenty *MDColorPicker*, která otevře dialogové okno se třemi možnostmi, jak zvolit barvu, a to HEX, RGB, RGBA. Občas nastával problém, že v některých případech, kdy byla barva změněna, nedošlo k propsání změny až k bodům. Ladícími výpisy se přišlo na špatný formát návratové hodnoty u HEX a RGB typů, jelikož jsou tvořeny pouze třemi barevnými kanály, ale pro vykreslení bodů je potřeba čtyř kanálů, tak jako je u formátu RGBA. Z tohoto důvodu musela být přidána konverze, která zajišťuje jeden pevný formát pro správnou aplikaci barvy na body.

Testování rychlosti projekce – Při velkých časových ztrátách u výpočtu projekce bylo potřeba zjistit reálnou dobu trvání výpočtu jednoho snímku. A to z důvodu, aby se této rychlosti dala adekvátně přizpůsobit výchozí frekvence animace. K měření bylo využito nástroje `perf_counter()`, který je součástí Python modulu *time*. Zvolený byl díky své vysoké přesnosti při měření kratších časových intervalů. Měření vycházelo z nastavení původní rychlosti, tedy bez zpomalení projekce, a ověřovala se úspěšnost vykreslení snímků. Pokud docházelo k častému přeskakování snímků z důvodu vypršení času na projekci, nastavilo se větší zpomalení. Minimálního vypadávání snímků se dokázalo u zpomalení 0,1 krát, kdy se snímky stihly zpracovat do stanoveného intervalu. Výsledné měření je znázorněno bodovým grafem na obrázku 5.1, kde je doba projekce i vykreslení pro prvních sto snímků. Měření vedlo k nastavení výchozího zpomalení na hodnotu 0,2 krát, které zajistí největší možnou rychlost s nejmenším množstvím přeskakovaných snímků.



Obrázek 5.1: Čas potřebný k projekci i vykreslení jednotlivých snímků.

Testování synchronizace - Synchronizace je důležitou součástí animace, proto zajištění správného fungování je klíčové. Testováním se ověřilo, že doba přehrávání videa a vykreslování bodů si navzájem odpovídá. Při spuštění animace a ponechání volného průběhu musí obě části animace skončit ve stejný okamžik. Důkladnější testování spočívá v narušování běhu videa s využitím ovládacích prvků, jako je pozastavení nebo přetočení pomocí posuvníku. Po doběhnutí animace na konec je potřeba ověřit, že ani zásahy do přehrávání nenarušily synchronizaci a ukončení nastalo ve stejnou chvíli.

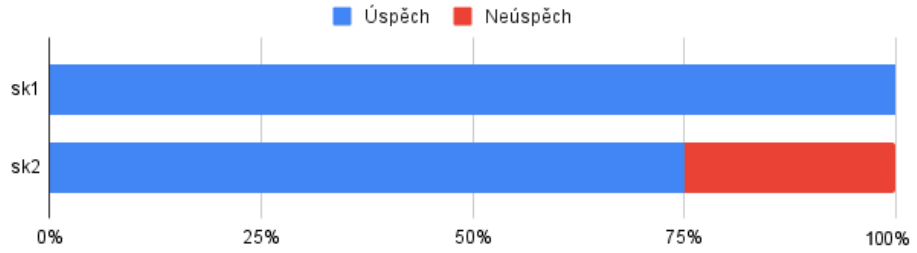
5.2 Uživatelské testování

Uživatelské testování se zaměřuje na uživatele a jeho interakci s aplikací. Cílem je ověřit, zda je aplikace intuitivní, snadno použitelná, poskytuje uživateli všechny potřebné informace

a splňuje jejich potřeby. Při testování se zaměřuje především na uživatelské rozhraní a celkovou uživatelskou zkušenost s aplikací.

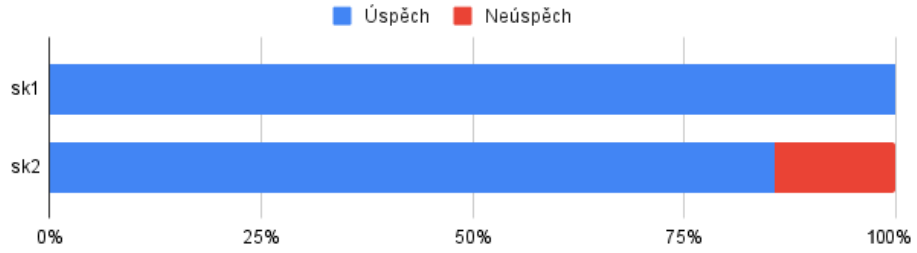
Testování probíhalo na dvou různých uživatelských skupinách. Všichni byli požádáni o splnění několika na sebe navazujících úkolů, které měly ověřit označení jednotlivých funkcí a jejich intuitivní použití. První skupinu (dále označovanou jako **sk1**) tvořili technicky zaměření jedinci ve věku kolem 20 let. Druhou byla věkově velmi různorodá skupina lidí (dále označovanou jako **sk2**) bez konkrétního zaměření, kteří byli osloveni jako běžní kolemjdoucí na veřejném prostranství.

1. Úkol: Načtení konfiguračních dat a graf na obrázku 5.2 – Po vysvětlení situace, ve které se jedinci, jako testovací subjekty, právě nacházejí, a po představení hlavního účelu aplikace, byli požádáni o import konfiguračních dat s upřesněním, že se jedná o komprimovaný ZIP soubor. U několika jedinců sk2 byl problém s neznalostí typu souboru, ale po doplnění některých informací všichni správně rozpoznali ikonu pro zobrazení menu a funkci hledali na správném místě.



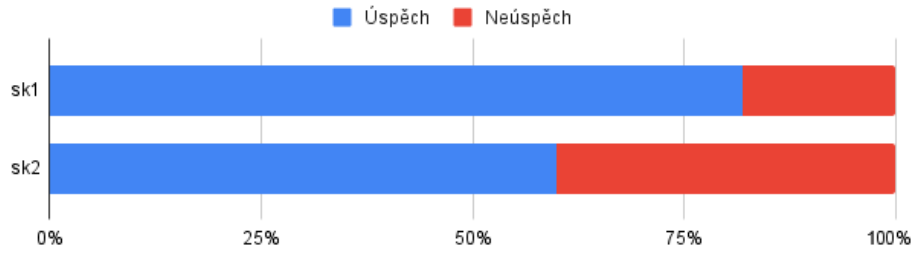
Obrázek 5.2: Úkol 1. Načtení konfiguračních dat.

2. Úkol: Změna barvy bodů a graf na obrázku 5.3 – Jedinci byli požádáni o změnu barvy vykreslených bodů, které se zobrazily po nahrání souborů. Až na jeden případ, kdy došlo k záměně tlačítek, byla správně zvolena funkce a následná volba barvy proběhla bez větších problémů.



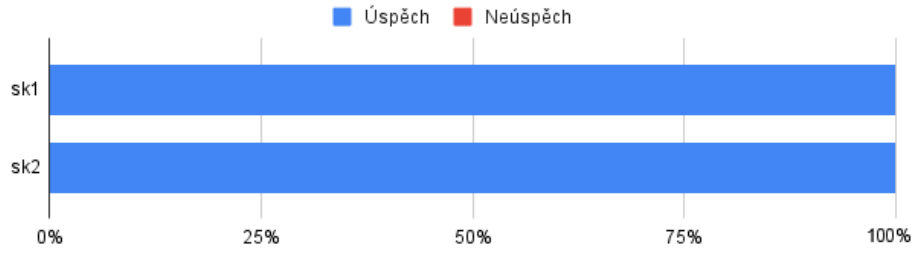
Obrázek 5.3: Úkol 2. Změna barvy bodů.

3. Úkol: Nastavení informačních položek a graf na obrázku 5.4 – Zde docházelo k největším problémům v rozpoznání odpovídající funkce. V některých případech jedinci volili tlačítko označené symbolem *i* pro informace, které slouží pouze ke skrytí nebo zobrazení již vybraných položek. Samotný výběr položek je nutné provést v menu aplikace. Na základě četnosti neúspěšného splnění zadání bylo upraveno grafické rozhraní, aby uživatel nejdříve musel vybrat položky přes menu a až potvrzením výběru se mu zobrazí tlačítko skrytí informací na hlavní obrazovce. Tím se předejde nejasnosti v prvotním výběru.



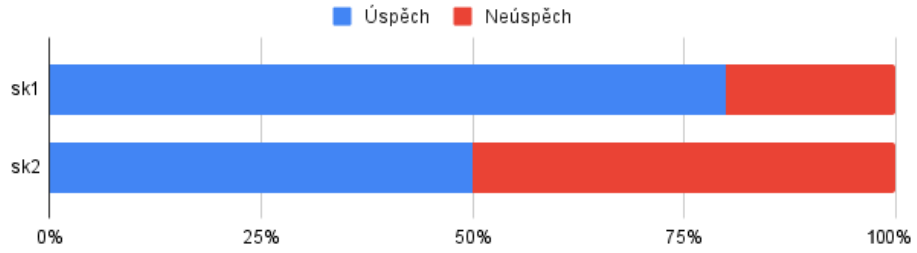
Obrázek 5.4: Úkol 3. Nastavení informačních položek.

4. Úkol: Spuštění a restartování animace a graf na obrázku 5.5 – Testování ovládání animace bylo stoprocentně úspěšné v obou testovacích skupinách



Obrázek 5.5: Úkol 4. Spuštění a restartování animace.

5. Úkol: Přetočení animace a graf na obrázku 5.6 – V tomto případě bylo jedincům zadáno, aby animaci nastavili na třicátý snímek v pořadí. Měli k tomu využít zobrazených informací z předchozího kroku, kde zvolili čítač snímků a aktuální hodnota se zobrazovala na hlavní obrazovce. Většina jedinců správně zvolila posuvník pro přetočení animace, což bylo cílem zadaného úkolu a intuitivní použití se ověřilo. Bohužel ale zadání nebylo úplně nejvhodněji definováno a použití posuvníku způsobovalo problémy z důvodu nepochopení významu hodnoty, kterou byli požádáni nastavit. Nejčastěji docházelo k záměně třicátého snímku a třicáté sekundy videa.



Obrázek 5.6: Úkol 5. Přetočení animace.

Ve slovní zpětné vazbě od uživatelů, především ze skupiny dvě, bylo nejčastěji zmiňováno umožnit přepnout aplikaci do českého jazyka, jelikož neznalost anglického jazyka byla překážkou v plnění úkolů. Celkově testování přineslo užitečné návrhy na vylepšení, které již byly implementovány, nebo se nabízejí k implementaci v rámci budoucího vývoje.

Kapitola 6

Navržená rozšíření pro budoucí vývoj

Tato kapitola se zaměřuje na návrhy rozšíření, která by mohla být implementována v budoucnosti s cílem vylepšit a optimalizovat současná řešení. V rámci budoucího vývoje je možné se zaměřit na různé části aplikace. Může se jednat o použití modernějších technologií za účelem urychlit výpočetní operace, případně upravit přístup provádění samotné projekce. Dále lze rozšířit uživatelské rozhraní, přidat funkce a ovládací prvky, které poskytnou uživateli větší volnost při konfiguraci a práci s daty.

Níže jsou uvedené návrhy, které svou implementací buď zdokonalí již implementované řešení, nebo přidají zcela novou funkcionalitu aplikace, čímž se rozšíří její využití.

6.1 Optimalizace rychlosti a využití GPU

Jedna z částí vhodná k dosažení významného zlepšení je projekce bodů a jejich vykreslení. Jelikož se jedná o dvě nejnáročnější operace celé aplikace, vyžadují hodně pozornosti a neustálé optimalizace.

Největší současný problém je zpracování nadměrně velkého množství dat z mračna bodů. Současné řešení pracuje s celým mračnem jako jedním celkem v jedné datové struktuře, ve které se nachází všechny body. Přitom po zpracování prvního snímku byla vykreslena pouze desetina celkového počtu bodů, ale v úvodní fázi výpočtu se vycházelo z celého datasetu.

Návrh na řešení tohoto problému spočívá v načítání mračna bodů po částech, ze kterých je seskládán. Tyto části obsahují daleko menší množství bodů a ke zpracování jsou vhodnější. Zajišťují, že dojde ke zpracování pouze vyhrazeného úseku bodů, ve kterém se kamera právě nachází, a body, které jsou zcela mimo úhel pohledu, nebudou do výpočtu ani zahrnuty. Tím dojde k výraznému zmenšení aktuálně zpracovávaného datasetu a zkrácení času, potřebného pro výpočet.

Projekce každého snímku je také výpočetně náročná, převážně z důvodu závislostí na výsledcích z předcházejících výpočtů, není možné provádět dílčí výpočty paralelně. Osvědčený způsob projekce implementovaný v současném řešení je využití více vláken k generování jednotlivých snímků, takže se paralelně zpracovává více snímků najednou, které se řadí do fronty na vykreslení na obrazovku.

Možnost rozšíření v tomto případě nabízí uložení všech vygenerovaných snímků k opětovnému použití. Aktuální implementace pomocí fronty ukládá pouze snímky, které následují po právě zobrazovaném snímku. Při vykreslení snímku dochází k jeho vyjmutí z fronty

a zpracovaná data se zahodí. Při restartování animace dojde k vyprázdnění fronty a projekce snímků musí být vypočítána znovu. Pokud by se generace snímků provedla pouze jednou, při nahrání vstupních souborů, ušetřilo by to spoustu výpočetního času spotřebovaného na opakované generování. Pro efektní řešení tímto přístupem musí být zvolen vhodný způsob ukládání vygenerovaných snímků. Jako možnost se nabízí vyhrazená část v úložišti zařízení. Je však potřeba brát ohled na to, že musí být umožněn rychlý přístup k těmto datům. Pro zachování původní rychlosti zobrazování to znamená deset přístupů za vteřinu. Vhodným indexováním jednotlivých snímků se zajistí jednodušší vyhledávání.

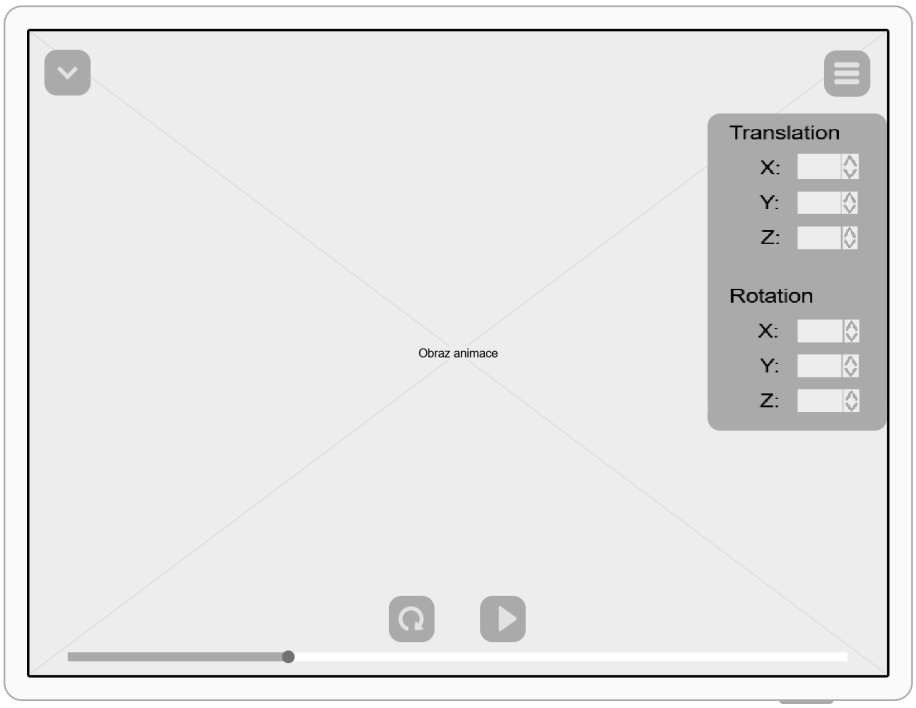
Pro grafické vykreslení se využívá modulu *kivy.graphics*, kvůli kterému byl Kivy framework zvolen k implementaci. Tento modul provádí velkou většinu vykreslování za pomocí GPU. Využívá k tomu rozhraní OpenGL, které umožňuje s grafickým hardwarem efektivně komunikovat. Vylepšení v tomto případě spočívá v použití `FrameBuffer` objektu, zkráceně fbo. Je to nástroj pro pokročilé grafické operace na GPU. Představuje vyrovnávací paměť – buffer, alokovanou na grafické kartě. Umožňuje vykreslení základních geometrických tvarů, ale i složitějších Kivy objektů mimo hlavní obrazovku, tedy že se obraz hned nezobrazí, ale čeká ve vyrovnávací paměti. Výhodou je, že z jakéhokoli množství vykreslených objektů do fbo se stane jedna výsledná textura, která se při požadavku vykreslí na hlavní obrazovku jako jeden zjednodušený celek. Tento přístup by mohl umožnit přípravu následujícího snímku ve vyrovnávací paměti na grafické kartě a v požadovanou chvíli vykreslit předpřipravenou texturu na obrazovce. Současné řešení implementuje přímé vykreslování do textury na hlavní obrazovce, což je proces, který také probíhá na GPU. Urychlení tímto způsobem oproti současnému řešení by proto muselo být nejdříve ověřeno.

6.2 Online streamování dat

Dalším významným rozšířením je umožnit načítat a zpracovávat data v reálném čase. K vizualizaci a přenosu dat ze senzorů autonomních vozidel je možné využít protokolu XVIZ (Extended Visualization Protocol), který je určený k práci se složitými datovými strukturami velkého objemu. Lze uvažovat nad dvěma hlavními způsoby získání dat. Při prvním způsobu se data přijímají rovnou ze zařízení, která je získávají. Tím je zajištěn sběr dat v reálném čase a umožňuje jejich okamžité zpracování i vykreslení. Nevýhodou tohoto přístupu by mohla být rychlost zpracování dat. Pokud nebude odpovídat rychlosti přijímání, tak komunikace v reálném čase bude postrádat smysl. Druhý přístup využívá server, na který jsou data ze senzorů streamována a může tam docházet i k jejich zpracování. Pokud se jedná o výkonný server, mohlo by zpracování dat probíhat tam a následně by do aplikace docházelo k přenosu pouze výsledných snímků, které by bylo jednodušší zobrazit. Tento způsob snižuje výpočetní náročnost v aplikaci.

6.3 Rozšíření uživatelského rozhraní

Na dostupných datech k vývoji při synchronizaci vygenerovaných snímků a obrazu na pozadí dochází k drobným nesrovnalostem v podobě mírného posunutí nebo natočení a vzniká tím rozladění obrazu. Proto bylo nutné provést korekci manuálně pro konkrétní dataset, aby se co nejvíce zamezilo odchylkám. Pro uživatele, který chce nahrát vlastní data, je tato korekce překážkou, která mu omezí podobu vizualizace.

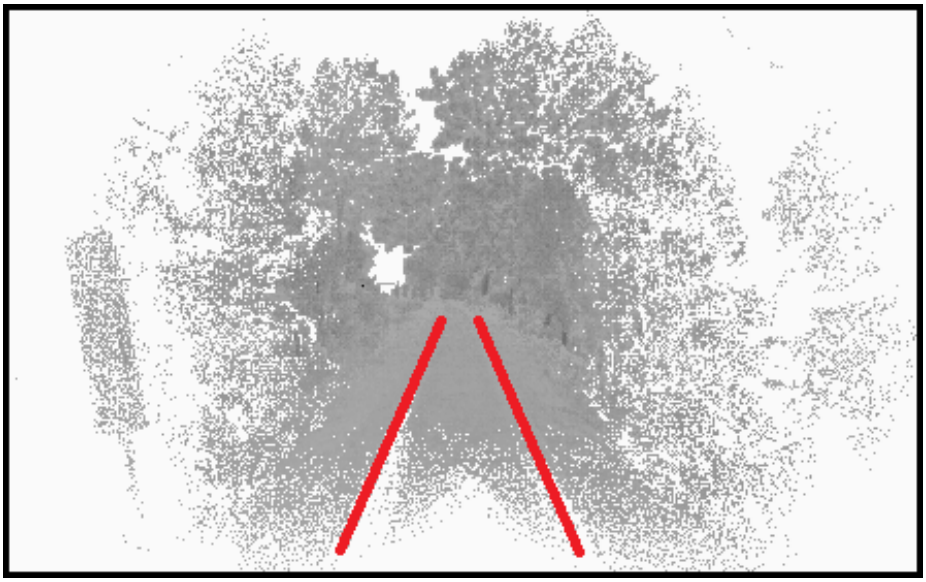


Obrázek 6.1: Návrh uživatelského rozhraní s parametry pro ovládání projekce.

Nejjednodušší řešení tohoto problému je umožnit uživateli si ručně nastavit parametry ovlivňující projekci bodů přímo z uživatelského rozhraní v podobě číselných vstupů. Mezi tyto parametry patří posunutí a otočení, přičemž každý se skládá ze tří složek x, y, z představujících osy pohybu. Navíc podél každé osy se lze pohybovat ve dvou směrech, pro posunutí je to doleva a doprava, což odpovídá snižování nebo zvyšování hodnoty, a u otočení je to pohyb po směru nebo proti směru pohybu hodinových ručiček. Návrhu grafického rozhraní odpovídá obrázek 6.1

6.4 Vizualizace vektorových dat

Dalším směrem, kterým je možné aplikaci rozšířit, je zaměření se na analýzu zobrazovaných dat a implementaci vektorové vizualizace některých objektů na obrazovce. V prvním kroku by se mohlo jednat o rozpoznání kolejí. Jejich pravidelný a výrazný tvar by mohl podpořit jednodušší detekci. Po úspěšném rozpoznání by se dalo objekty ověřit vykreslením zvýrazňující úsečky podle jejich průběhu. Jelikož výsledné snímky jsou ve 2D podobě, nabízí se řešení s využitím knihovny OpenCV, která poskytuje velké množství různých funkcí pro zpracování právě 2D snímků. Jako příklad je možné uvést Houghovu transformaci pro detekci přímek, případně je možné použít některou z jejich obměn pro zakřivené čáry. Výsledný snímek by mohl vypadat podobně jako uvádí obrázek 6.2 níže.



Obrázek 6.2: Snímek z projekce s detekovanými kolejemi.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo vytvořit uživatelskou aplikaci k vizualizaci agregovaných dat ze senzorů umístěných na čele vlaku. V rámci práce se provedl výzkum možností a dostupných nástrojů pro vizualizaci dat z mobilního mapování v podobě mračna bodů. Také je uvedeno, jak pomocí perspektivní projekce mračno bodů zpracovat do podoby 2D snímků představujících záběry z jednotlivých umístění kamery při pohybu v daném úseku trati. Dále proběhl návrh systému, který umožní vizualizaci získaných dat právě s využitím projekce. Efektivního zobrazení se dosáhlo aplikováním metody hloubkového ořezávání pro ošetření viditelnosti objektů v mračně bodů. Pro pohodlnější práci s daty a lepší vizualizaci byly navrženy další funkce, které uživatelskou aplikaci rozšiřují. Dle návrhu byla provedena implementace v prostředí Python s frameworkem Kivy pro grafické uživatelské rozhraní. Všechny požadované funkcionality podle uvedených specifikací byly úspěšně implementovány a otestovány. Testování uživatelské přívětivosti aplikace probíhalo ve spolupráci s reálnými jedinci, kteří na zadaných úkolech ověřili intuitivní rozhraní a poskytli návrhy ke zlepšení interakce s aplikací. Provedením výše uvedených bodů bylo dosaženo stanoveného cíle, proto zadání bakalářské práce považuji za splněné. Přínosem bakalářské práce je důkaz, že zpracování velkých objemů dat z mobilního mapování je realizovatelné i na běžně používaných přenosných zařízeních, jako jsou tablety. Je umožněno prohlížení agregovaných dat ze senzorů umístěných na čele vlaku při průjezdu železniční tratí. S dodatečnou optimalizací bude možné dosáhnout zpracování v reálném čase a skutečnou rychlostí.

Řešení práce mi poskytlo příležitost zpracovat reálná data, která se využívají k vývoji autonomních vozů. Díky tomu jsem se seznámila se zpracováním dat ve 3D podobě, což si vyžádalo systematický přístup a trpělivost k dosažení správných výsledků. Řešení některých problémů bylo opravdu náročné, protože bylo potřeba využít přesných postupů. Zjistila jsem, že při zpracování takto velkého množství dat je nutné dbát na efektivitu jednotlivých operací a šetřit výpočetním časem, co nejvíc je to možné, protože je pevně omezený. Tato skutečnost mě přinutila využít specializovanější přístup a zaměřit se na vícevláknové zpracování. Tímto přístupem bylo za určitých podmínek dosaženo původní rychlosti kamery, což považuji za úspěch.

Přesto je stále možné aplikaci rozšiřovat a vylepšovat. Prvním krokem v budoucím vývoji by určitě měla být optimalizace s využitím grafického procesoru k vykreslování bodů. Tím se urychlí celý proces zpracování a umožní rychlejší načítání snímků při animaci. Urychlení projekce bodů by také do jisté míry mohlo pomoci zmíněné postupné načítání jednotlivých částí mračna bodů místo jednoho velkého celku.

Literatura

[1] *AŽD má autonomní vůz EDITA pro vývoj nových technologií AŽD* online. Dostupné z: https://acri.cz/2024/06/03/azd-ma-autonomni-vuz-edita-pro-vyvoj-novych-technologii-azd/. [cit. 2025-01-23].

[2] *About BeeWare* online. Russel Keith-Magee, 2025. Dostupné z: https://beeware.org/project/about/. [cit. 2025-01-7].

[3] *Success stories* online. Russel Keith-Magee. Dostupné z: https://briefcase.readthedocs.io/en/stable/background/success.html. [cit. 2025-01-19].

[4] Brodský, J. *Využití mobilního mapování ve městech*. Brno, CZ, 201. Diplomová práce. Masarykova univerzita přírodovědecká fakulta. Dostupné z: https://is.muni.cz/th/w8axl/DP_-_Jan_Brodsky.pdf.

[5] Stoller, C. *Deflectouch* online. 2011. Dostupné z: https://github.com/stocyr/Deflectouch. [cit. 2025-03-20].

[6] Hartley, R. a Zisserman, A. *Multiple View Geometry in Computer Vision*. 2. vyd. Cambridge University Press, 2003. ISBN 0-521-54051-8.

[7] *Kivy Philosophy* online. Dostupné z: https://kivy.org/doc/stable/philosophy.html#philosophy. [cit. 2025-01-19].

[8] Barot, P. *Comparing Kivy and BeeWare: Understanding the Key Differences* online. 17. 9. 2022. Dostupné z: https://www.botreetechnologies.com/blog/kivy-vs-beeware-difference/. [cit. 2025-01-19].

[9] DaneshFar, F. *Kivy vs BeeWare: Choosing the Best Python Framework* online. 18. 6. 2024. Dostupné z: https://www.thecyberiatech.com/blog/mobile-app/kivy-vs-beeware/. [cit. 2025-01-19].

[10] Berk, R. *MathPath Console* online. 2024. Dostupné z: https://mathpathconsole.github.io/. [cit. 2025-03-20].

[11] Andrés Rodríguez, A. B. a contributors, K. *FileManager* online. 2022. Dostupné z: https://kivymd.readthedocs.io/en/1.1.1/components/filemanager/index.html. [cit. 2025-03-26].

[12] *Mobilní mapování* online. GEOVAP, spol. s r.o., 2019. Dostupné z: https://www.geovap.com/cs/mobilni-mapovani. [cit. 2024-12-09].

[13] *Napari: a fast, interactive viewer for multi-dimensional images in Python* online. Created 2025 using Sphinx 7.4.7. Dostupné z: https://napari.org/stable/index.html. [cit. 2025-01-19].

[14] *Open3D documntation* online. 2018. Dostupné z: https://www.open3d.org/docs/release/introduction.html. [cit. 2024-12-26].

[15] *About OpenCV* online. 2025. Dostupné z: https://opencv.org/about/. [cit. 2025-01-06].

[16] *Gui Features in OpenCV* online. Generated 2025 by doxygen. Dostupné z: https://docs.opencv.org/4.x/dc/d4d/tutorial_py_table_of_contents_gui.html. [cit. 2025-01-06].

[17] Pierre, B.; Aurélie, B.; Jean François, A. a Mathieu, B. Visibility Estimation In Point Clouds With Variable Density. In:. 2019. Dostupné z: https://hal.science/hal-01812061v2.

[18] *PyQt5 - Introduction* online. 2023. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html. [cit. 2025-01-19].

[19] Fitzpatrick, M. *PyQt vs. Tkinter* online. 13. 8. 2023. Dostupné z: https://www.pythonguis.com/faq/pyqt-vs-tkinter/. [cit. 2025-01-19].

[20] Rusu, R. B. a Cousins, S. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: [b.n.], May 9-13 2011. Dostupné z: https://pointclouds.org/assets/pdf/pcl_icra2011.pdf.

[21] *Slovník geodézie: Mračno bodů* online. VISIONPLAN-3D s.r.o., 2021. Dostupné z: https://www.visionplan.cz/slovnik-geodezie-mracno-bodu/. [cit. 2024-12-09].

[22] Sullivan, C. B. a Kaszynski, A. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*. The Open Journal, may 2019, sv. 4, č. 37, s. 1450. Dostupné z: https://doi.org/10.21105/joss.01450.

[23] Keith Magee, R. *Travel Tips* online. 2018. Dostupné z: https://apps.apple.com/au/app/travel-tips/id1336372310. [cit. 2025-01-19].

[24] *What is Mobile Mapping* online. Celantur GmbH, červen 2020. Dostupné z: https://www.celantur.com/blog/getting-started-with-mobile-mapping/. [cit. 2024-12-09].

Příloha A

Vysvětlení zkratek

ADB – Android Debug Bridge (Ladící nástroj připojených Android zařízení)

API – Application Programming Inteface (Aplikační programové rozhraní)

APK – Android Application Package (Formát instalačního balíčku Android)

AŽD – Automatizace železniční dopravy

BGR – Blue, Green, Red (Modrá, Zelená, Červená)

CPU – Central Processing Unit (Centrální procesorová jednotka)

EDITA – Experimentální Drážní vozidlo pro Inovativní Technologie AŽD

FBO – FrameBuffer Object (Objekt vyrovnávací paměti)

FPS – Frames Per Second (Snímků za sekundu)

GNSS – Global navigation satellite system (Globální družicový polohový systém)

GUI – Graphical User Interface (Grafické uživatelské rozhraní)

JNI – Java Native Interface (Rozhraní k propojení s Java programy)

JSON – JavaScript Object Notation (JavaScriptový objektový zápis)

KNN search – K-Nearest Neighbor search (Vyhledávání k-nejbližších sousedů)

PCL – Point Cloud Library (Knihovna pro zpracování mračna bodů)

RGB – Red, Green, Blue (Červená, Zelená, Modrá)

SAF – Storage Access Framework (Rozhraní pro přístup k uložišti)

URI – Uniform Resource Identifier (Jednotný identifikátor zdroje)

VTK – Visualisation Toolkit (Nástroj pro vizualizaci 3D a 2D dat)

XML – Extensible Markup Language (Rozšířitelný značkovací jazyk)

XVIZ – Extended Visualization Protocol (Rozšířený vizualizační protokol)