České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Aplikace pro sledování letadel v reálném čase prostřednictvím rozšířené reality

Bakalářská práce

Vojtěch Böhm

Studijní program: Otevřená informatika
Obor: Počítačové hry a grafika
Vedoucí práce: Ing. David Sedláček, Ph.D.

Praha, Květen 2022

Vedoucí práce:
Ing. David Sedláček, Ph.D.
Katedra počítačové grafiky a interakce
Fakulta elektrotechnická
České vysoké učení technické v Praze
Technická 2
160 00 Praha 6
Česká Republika

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Praha, Květen 2022

.....
Vojtěch Böhm

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Böhm Vojtěch** Personal ID number: **483806**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Specialisation: **Computer Games and Graphics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Real-time plane tracking app via augmented reality

Bachelor’s thesis title in Czech:

Aplikace pro sledování letadel v reálném čase prostřednictvím rozšířené reality

Guidelines:

Bibliography / sources:

1) Steve Aukstakalnis. Practical Augmented Reality: A Guide to the Technologies, Applications, and Human Factors for AR and VR (Usability). Addison Wesley 2017.
2) Dieter Schmalstieg, and Tobias Hollerer. Augmented Reality: Principles and Practice (Usability), Addison Wesley 2016
3) Micheal Lanham. Augmented Reality Game Development. Packt Publishing, 2017.
4) Pablo Perea and Pau Giner. UX Design for Mobile. Packt Publishing, 2017.
5) T. Lowdermilk, User-Centered Design, O’Reilly Media, 2013

Name and workplace of bachelor’s thesis supervisor:

Ing. David Sedláček, Ph.D. Department of Computer Graphics and Interaction FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **09.02.2022** Deadline for bachelor thesis submission: **24.05.2022**

Assignment valid until: **30.09.2023**

Ing. David Sedláček, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Abstrakt

Tato práce popisuje návrh a vývoj mobilní aplikace pro sledování letadel v reálném čase prostřednictvím rozšířené reality. Zahrnuje rešerši existujících aplikací, zdrojů dat, technologie AR a geodézie, návrh uživatelského rozhraní a jeho následnou implementaci pomocí frameworků ARKit a SceneKit a uživatelské testování postupující dle metodiky User-Centered Design. Aplikace byla vyvinuta s důrazem na přehlednost a minimalismus pro mobilní platformu iOS.

Klíčová slova: Rozšířená realita, AR, ARKit, SceneKit, GPS pozicování, ADS-B, OpenSky Network, UI, UX, User-Centered Design.

Abstract

This thesis describes the design and implementation of a mobile app for real-time plane tracking via augmented reality. It includes research of existing apps, data sources, AR technology and geodesy, user interface design and its implementation using ARKit and SceneKit frameworks and user testing based on the User-Centered Design methodology. The app was built for the iOS mobile platform with lucidity and minimalism in mind.

Key words: Augmented reality, AR, ARKit, SceneKit, GPS positioning, ADS-B, OpenSky Network, UI, UX, User-Centered Design.

Title in english: Real-time plane tracking app via augmented reality.

Poděkování

Rád bych tímto poděkoval svému vedoucímu bakalářské práce, Ing. Davidu Sedláčkovi, PhD. za jeho cenné rady, vstřícnost a ochotu. Dále bych rád poděkoval své rodině, která mě při mém studiu vždy podporovala a také svým nejbližším kamarádům, kteří při mně stáli v mých nejhorších chvílích.

Seznam zkratek

ADS-B Automatic Dependent Surveillance-Broadcast. ix, 6–8, 12, 32

AMSL Above Mean Sea Level. 7, 9, 20

API Application Programming Interface. 12, 24, 29

AR Augmented Reality. ix, 1, 9, 10, 17, 19, 30, 32

DOF Degrees of freedom. 9–11, 25, 26

GLONASS Globalnaja navigacionnaja sputnikovaja sistěma. 8

GPS Global Positioning System. ix, 7–9, 14, 18, 21, 22

ICAO International Civil Aviation Organization. 12, 26

IDE Integrated Development Environment. 22, 40

IMU Inertial Measuring Unit. 11

JSON JavaScript Object Notation. 24

MEMS Micro-electromechanical System. 10

MVC Model View Controller. 22, 23

SLAM Simultaneous Localization and Mapping. 10, 11

UCD User Centered Design. 28, 32

UI User Interface. ix, 1, 20, 22, 32

URL Uniform Resource Locator. 24

UX User Experience. ix, 1, 19, 20, 28, 29, 32

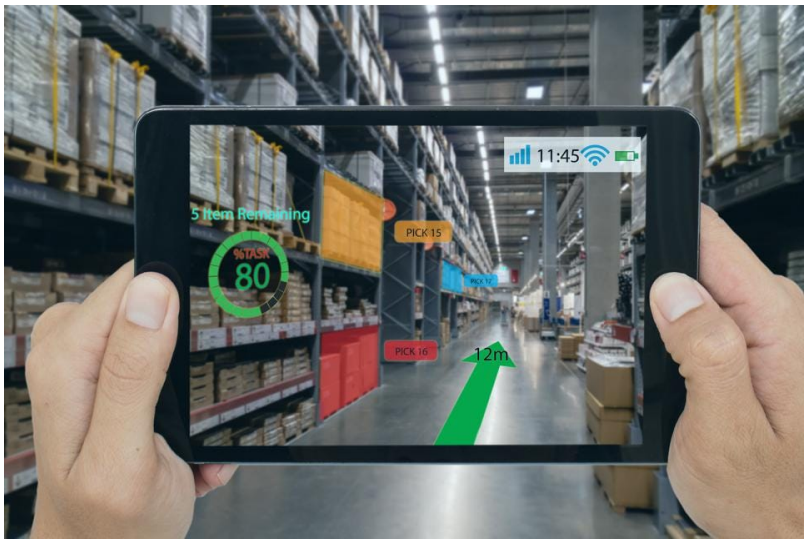
Obsah

Kapitola 1

Motivace

Jsem fanouškem otevřených dat a jejich vizualizace a taky nadšenec do letadel, leteckého provozu a avioniky. Inspirují mě projekty jako Flight Radar, OpenSky Network a Plane Finder. Tato práce měla původně být rozšířením produktu Aviee, na kterém jsem se 4 roky podílel v roli zakladatele, jakožto vizualizace letících letadel používající software Aviee. Nicméně již nejsem součástí produktu a proto jsem téma lehce pozměnil a vizualizovat budu veškerá letadla, která jsou dostupná z kolektivně sbíraných, otevřených dat. Zároveň jsem do projektu chtěl zakomponovat svou zkušenost s UI/UX designem a vypracovat přehlednou, minimalistickou aplikaci, která bude tato data stravitelně prezentovat.

Mou druhotnou motivací bylo naučit se pracovat s rozšířenou realitou - AR. Za poslední dekádu zažila tato technologie obrovský pokrok. Zatímco jsme byli zvyklí na počítačem generovaný obsah na monitoru, odpojený od našeho reálného světa, AR se daří tyto dva světy propojit [1]. Ukazuje se totiž, že lidem se premisa reálného světa doplněná o vhodné virtuální prvky hodí, je intuitivní a má mnoho praktických využití (1.1).

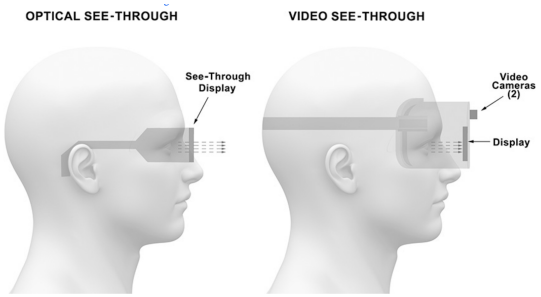


Obrázek 1.1: Příklad využití AR. Zdroj: [2]

Aby technologie byla považována za AR, musí splňovat tyto tři charakteristiky: kombinace reálného a virtuálního světa, interaktivita v reálném čase a zobrazení ve 3D. Definice ovšem nespecifikuj, jaké médium pro zobrazení obsahu se musí použít. I když si lidé můžou AR spojovat pouze s mobilními telefony a tablety, letci bojových stíhaček využívají například tzv. "head-up display"(1.2a) či "chytré brýle"(1.2b) [3].

Na trhu již roky existují takové brýle i pro standardního spotřebitele - např. Google Glass (1.2c) [4] - ty se ovšem neuchytily. Premisa byla taková, že není potřeba držet před sebou telefon, ale pouze mít brýle na očích, což velmi zpříjemní uživatelský požitek a dostane AR do většího povědomí a bude se rozrůstat počet praktických využití. Už roky se ale spekuluje o tom, že firma Apple pracuje na svých vlastních brýlích, které by měly přinést revoluci v oblasti rozšířené reality a splnit zmíněné očekávání. [5]. Apple již několikrát změnil technologický svět svými produkty a jak mnozí, včetně mě, věří, bude to i v případě zmíněných brýlí [6]. A jak historie napovídá, je dobré být připraven na technologickou vlnu a být mezi prvními, kteří ji úspěšně adoptují na zajímavých nápadech.

(a) Head-up display



(b) Chytré brýle

(c) Google Glass

Obrázek 1.2: Příklady médií pro zobrazení virtuálního obsahu. Zdroje: [1] [7]

Kapitola 2

Analýza

Pro správné uchopení projektu je třeba se podívat na trh na již existující aplikace, zhodnotit co dělají stejně či jinak, inspirovat se a utvořit si ucelený pohled na to, jak k projektu přistoupit. Vybral jsem si čtyři takové aplikace, které se nejvíce podobají mému zadání, z čehož jedna se dá považovat funkcionalitou za přímého "konkurenta". Dále je třeba prozkoumat technická a odborná specifika, která použiji při návrhu a následné implementaci aplikace.

2.1 Existující AR aplikace

2.1.1 Google Maps

Navigační aplikace od Google umožňuje uživateli vyhledat trasu z bodu A do bodu B a následně jí zobrazit v AR (v aplikaci nazváno "Live View"). Při prvním zapnutí mě překvapilo, jak rychle se aplikace dokázala zorientovat a lokalizovat mou orientaci v prostoru - cca 3 sekundy. Po následovném testování aplikace mohu vyzdvihnout pět hlavních aspektů.

První je 2D map overlay ve spodní části obrazovky, který napomáhá uživateli v orientaci. Je to pěkně vymyšlené UX - informace o pozici na mapě je klíčová a kompenzuje nedokonalosti AR navigace. Mapa zároveň nezabírá velkou část obrazovky, takže AR pohled je stále hlavním středem pozornosti (2.1a).

Druhý je krásný přechod mezi 2D mapou a Live View tím, že stačí zvednout telefon do vertikální polohy a aplikace automaticky přepne mezi těmito dvěma pohledy. Takzvaný "small touch - drobný detail", který je v profesionálním UX designu naprosto esenciální, oživuje aplikaci a uživateli dává dobrý pocit z objevení a zároveň nabízí další možnost přepnutí (spolu s klasickými tlačítky).

Třetí je šipka, která má uživatele poslat tím správným směrem. Často se bohužel ani neobjevila, objevila se na špatném místě, nebo se objevila příliš blízko obrazovky a tedy zabrala celý "výhled". Když se ale objevila správně, tak byla nápomocná a věřím, že spolu s 2D mapou by uživatel vždy úspěšně došel do cíle. Všiml jsem si, že šipka má občas pod sebou i správně promítnutý stín (2.1b) - zejména v dobře osvětlených oblastech.

Čtvrtý je poletující modrý view s názvem další ulice na trase. I když zprvu nejasné, po chvíli užívání jsem si na něj zvykl. View ukazuje vzdálenost k ulici velmi přesně, což je určitě díky GPS. Jediné co postrádám je okluze budovami - tedy když by ulice byla zakryta za budovou, odrazilo by se to na vzhledu view (například větší průhlednost).

Pátý, bohužel špatný, je banner, který uživateli nakazuje ať Live View používá pouze

když stojí na místě. Když jsem banner neuposlechl a pokračoval v chůzi, obrazovka mi zčernala a znemožnila mi používání Live View (2.1c). Nejspíše to bude z právních důvodů, aby se Google vyhnul odpovědnosti za případné nehody uživatelů, kteří nebudou dávat pozor na okolí, nicméně za mě je tímto Live View navigace nepoužitelná.

Celkový dojem z aplikace mám neutrální. Líbí se mi UI i UX a věřím, že bych byl schopný kamkoliv dojít, nicméně mi požitek z užívání kompletně ničí povinnost stát na místě.

(a) Live View

(b) Navigační šipka se stínem

(c) Blokování UI při pohybu

Obrázek 2.1: UI Google Maps AR

2.1.2 Flight Radar

Tato aplikace je funkcionálně nejblíže mému projektu. Umožňuje uživateli vidět civilní letadla která jsou vybavena vysílačem ADS-B po celém světě a v reálnem čase si zobrazit jejich pozici, směr letu a technické specifikace. Nejprve aplikaci rozeberu jako všechny ostatní a poté udělám srovnání s mým projektem.

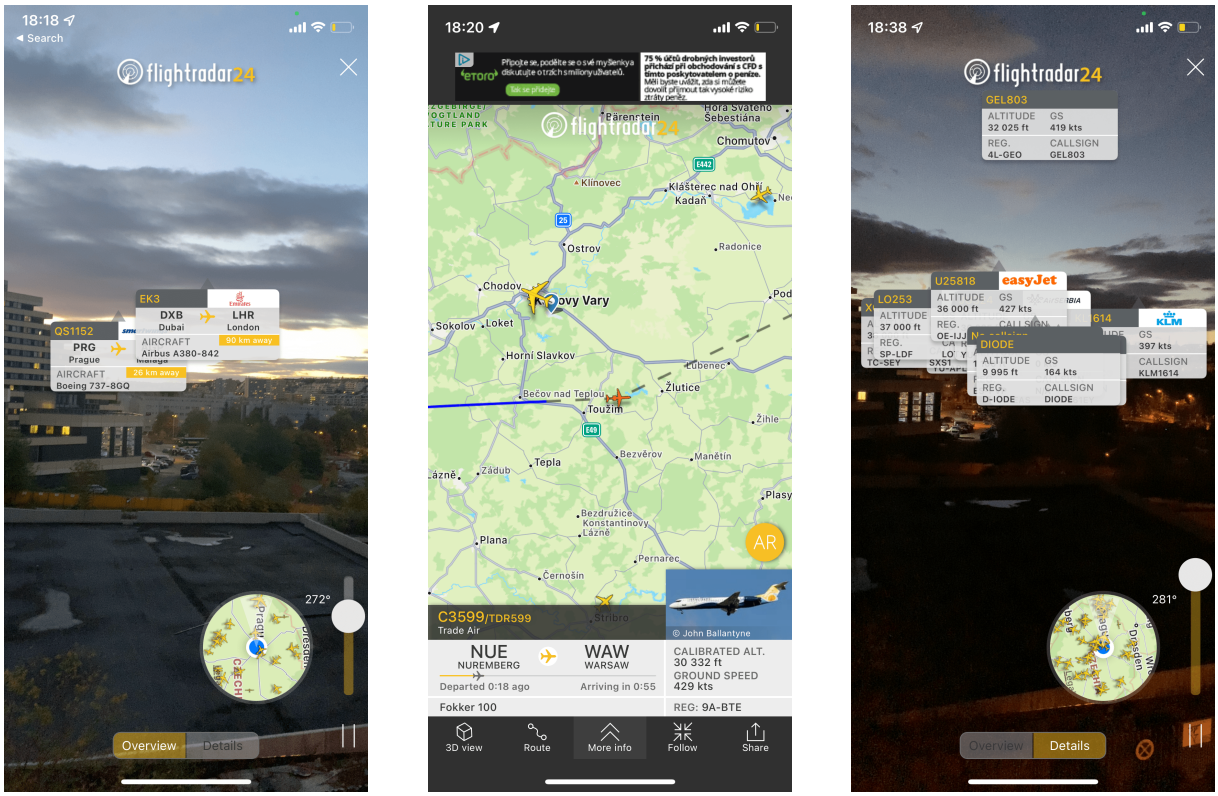
Aplikace mě překvapila svou rychlostí. Hned po zapnutí AR vieweru mě lokalizovala a zobrazila letadla. Každému z nich přísluší view, který visí ve vzduchu bez nějakého konkrétního kotevního bodu a působí to zvláštně (2.2a). Líbilo by se mi, kdyby jako kotvu dokreslili siluetu letadla. Ve view pro každé letadlo jsou zobrazeny základní data + vzdálenost od uživatele. K tomu je dole na obrazovce segmented control (overview x details), kterým se můžou přepnout data letadel ze základních na detailní, která obsahují například výšku AMSL. Klepnutím na libovolný view zobrazí novou obrazovku s detailem letadla a jeho dráhou promítnutou na 2D mapě (2.2b).

Kromě letadel je na obrazovce, stejně jako u Google Maps, malá 2D mapka zobrazující uživatelovu polohu včetně úhlu k magnetickému severu a letadla okolo něj. Opět je to skvělé UX, protože uživatel má ponětí o tom, kde se nachází a v jakém směru může očekávat viditelná letadla.

Překvapilo mě tlačítko "pauzy", které zastaví vykreslování a umožní uživatelovi zafixovat nějaký pohled. Přijde mi to jako zajímavý nápad - může se to hodit pokud se mu povede na obrazovku dostat zajímavý úkaz a bude si ho chtít lépe prohlédnout.

Také mě příjemně překvapilo, že aplikace pozná, kterým směrem se uživatel dívá a zvýrazní mu letadlo které je tomuto směru nejblíže.

V situaci kdy je na obloze mnoho letadel může dojít k zahlcení obrazovky (2.2c). Hodil by se filtr na základě vzdálenosti, případně počtu letadel.



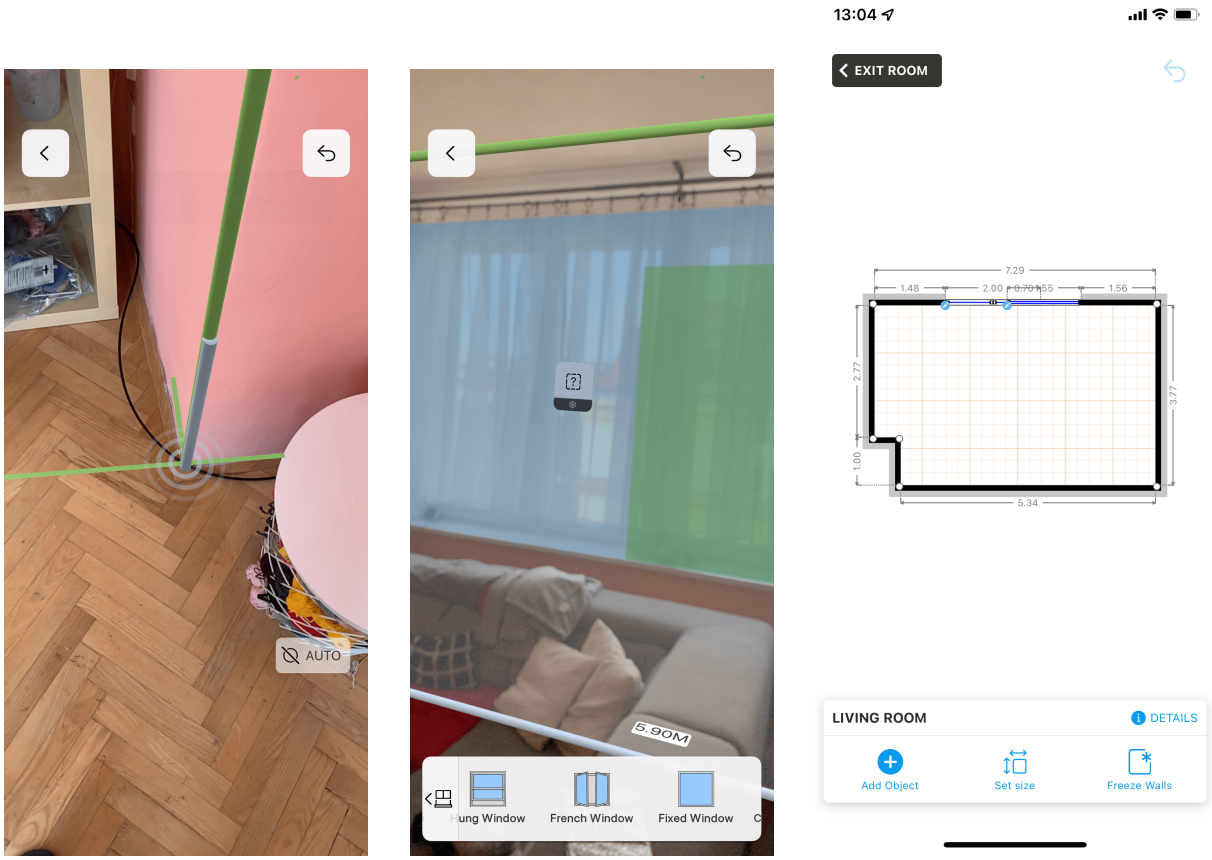
(a) AR pohled na letadla. (b) Detail letadla. (c) Příliš mnoho letadel.

Obrázek 2.2: UI Flight Radar

2.1.3 magicplan

Tato aplikace umožňuje uživateli naskenovat libovolný pokoj a následně vygenerovat půdorys, který je možno exportovat do .PDF formátu. Aplikace vyžaduje trochu pomalejší kalibraci - je nutno provádět "osmičky" se zařízením. Důvodem je kalibrace magnetometru (viz. 2.4). Poté může uživatel začít skenovat a umístit první roh místnosti (2.3a). Předpokládám, že aplikace chytře využívá detekci hran. V dobře osvětlené místnosti se to daří velmi pěkně. Rozměry místnosti celkem přesně odpovídaly realitě. Po dokončení musí uživatel ještě naskenovat okna, která to na několikátý pokus nakonec úspěšně samo detekovalo (2.3b). Hádám, že tomu bránily záclony a aplikace nebyla schopna poznat co

je stěna a co okno. Nakonec jsem vygeneroval profesionální půdorys, který jsem si poté exportoval (2.3c).



(a) Umístění rohu místnosti (b) Skenování oken (c) Výsledný půdorys

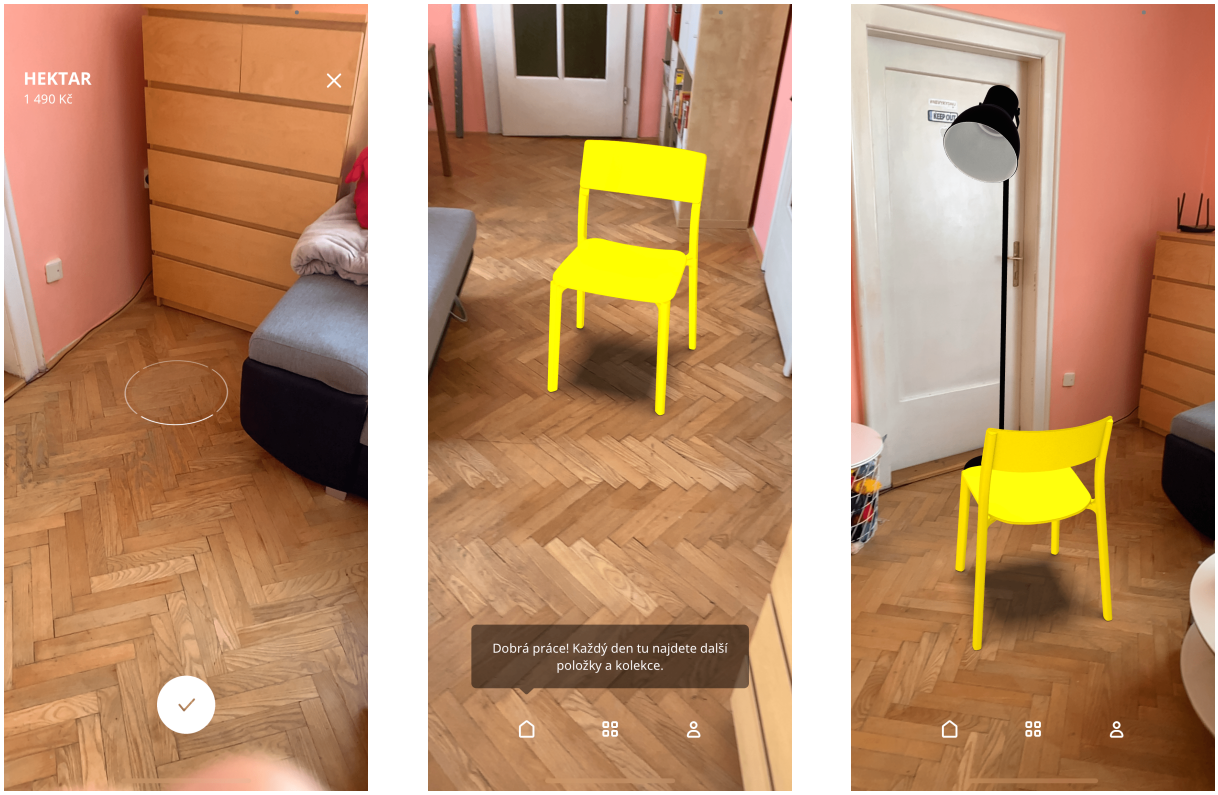
Obrázek 2.3: UI magicplan

2.1.4 IKEA Place

Aplikace umožňuje uživateli procházet katalog nábytku a poté jej umístit do místnosti v AR. Aplikace vyžadovala podobnou kalibraci jako v sekci (2.1.3). Z aplikace mám skvělý pocit. Po vybrání nábytku z katalogu se na obrazovce objeví umisťovací bod (2.4a), který kopíruje podlahovou rovinu. Umístěný objekt, který je možno libovolně přesouvat a otáčet, vypadá realisticky, čemuž napomáhá stín (2.4b). Třešničkou na dortu je správné řešení zakrývání objektů které jsou vzdálenější od uživatele objekty bližšími (2.4c). Toto chování je rozhodně žádoucí a zesiluje pocit realističnа.

2.2 ADS-B

ADS-B je relativně nová sledovací technologie, jejímž účelem je modernizace a unifikace leteckého provozu. Slouží jako levná náhrada konvenčního radaru a umožňuje řídicím věžím přesněji a na větší vzdálenosti kontrolovat provoz. Kombinací satelitů a vysílačů poskytuje přijímačům informace o pozici, rychlosti (a mnoho dalších datech) letadla [8]. Proces tedy funguje následovně:



(a) Umisťovací bod. (b) Objektů se stínem. (c) Zakrývání objektů.

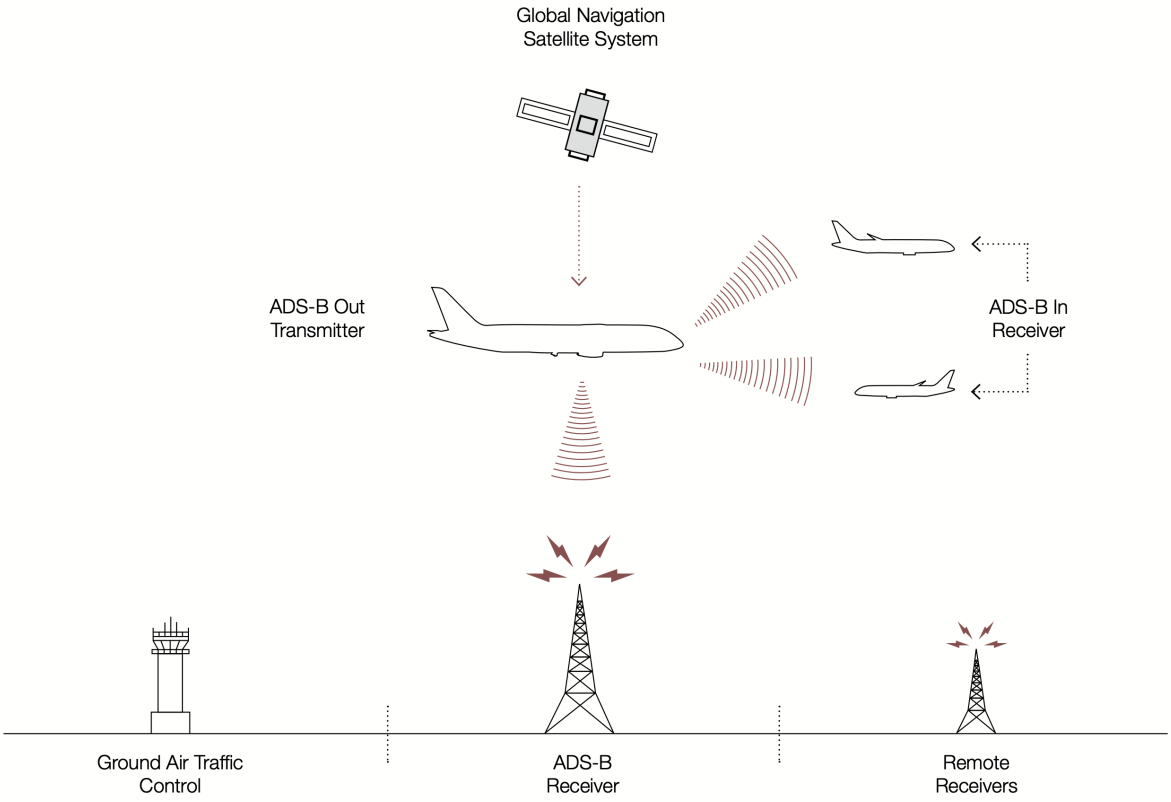
Obrázek 2.4: UI IKEA Place

1. Letadlo zjistí svou pozici pomocí GPS (viz. 2.3).
2. Spolu s pozicí, rychlostí a dalšími daty (jako je kurz, náklon a Above Mean Sea Level (AMSL) výška) vyšle signál který tato data přenáší.
3. Signál je zachycen ADS-B přijímačem či řídicí věží.

Na obrázku (2.5) je tento proces vizualizován. Privátní firmy si ale uvědomily druhořadé využití této technologie a vzali ji o kus dál - sběr a shromažďování dat na centralizovaný server. Libovolný nadšenec do letadel si dnes totiž může koupit ADS-B přijímač a přijatá data posílat na zmíněné servery, které pak tato data v reálném čase zobrazují na mapě a dále je poskytují k využití zdarma či za poplatek. Lidé po celém světě tak tvoří jeden velký celek, který sbírá a sdílí data [9]. A přesně tuto technologii využiji v této práci.

2.3 GPS

GPS je konstelace satelitů, obíhajících kolem Země, poskytující možnost lokalizace (a tedy i navigace) jakéhokoliv bodu na Zemi s různou přesností. Tyto satelity oběhnou celou Zemi dvakrát za jediný den. Každý z nich vysílá mikrovlnami svou časově zakódovanou pozici. GPS přijímač zpracuje signály těch satelitů, ke kterým má "přímou cestu"(tedy signálu nestojí v cestě například kopec, či vysoká budova) a pomocí trilaterace spočítá svou pozici, výšku a čas. K tomu je zapotřebí signálu alespoň ze tří, ideálně ze čtyř satelitů najednou. K trilateraci je totiž třeba znát vzdálenost k jednotlivým satelitům. Ta se počítá pomocí doby přenosu signálu atmosférou. Ve chvíli kdy je vzdálenost známa, je kolem satelitu

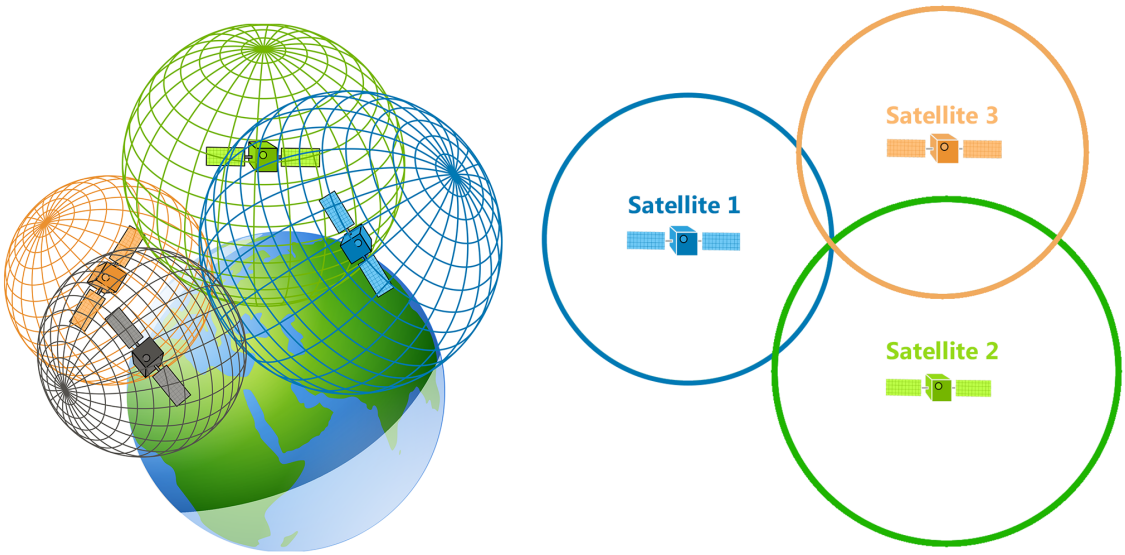


Obrázek 2.5: Vizualizace provozu ADS-B. Zdroj: [8]

vytvořena pomyslná sféra s potenciálními pozicemi přijímače. Jak je vidět na obrázku (2.6), abychom dostali ze všech možných pozic tu skutečnou, potřebujeme průnik více sfér najednou (přesněji 3). Když má přijímač signál z více (alespoň 4) satelitů, vyhýbá se potřebě atomových hodin, viz. dále [10].

V ideálních podmínkách má GPS přesnost cca 5 metrů. Existuje ale řada faktorů, které mohou přesnost ovlivnit. Jak bylo zmíněno, k určení doby trvání přenosu je zapotřebí znát aktuální čas satelitů. Ty jsou pro tento účel vybaveny velmi přesnými atomovými hodinami. Takové hodiny ale mohou být (a jsou) vystaveny časovým posunům za účelem snížení přesnosti (satelity jsou privátní a pro veřejnost je tímto záměrně přesnost omezena). Roli hrají také nepřesnosti hodin v přijímači, atmosférické podmínky (mraky mohou ovlivnit přenášený signál) a elektromagnetická pole. Největším problémem je ale tzv. "signal bouncing"- kdy se signál před dopadnutím do přijímače odrazí od okolních budov či kovových materiálů a tím uměle prodlouží délku signálu. Proto se tedy často stává že ve městech se k přesnosti 5 metrů nepřiblížíme [12].

Pojmem GPS se dnes chybě označuje jakýkoliv satelitní navigační systém, ale ve skutečnosti GPS je jen jedním z nich - patří Spojeným státum americkým. Mezi další systémy patří ruský Globalnaja navigacionnaja sputnikovaja sistěma (GLONASS), evropský Galileo nebo čínský BeiDou a dnešní moderní přijímače jsou schopny využívat signály z více systému najednou [13].



Obrázek 2.6: Trilaterace GPS přijímače. Zdroj: [11]

2.4 AR

AR jsme si představili v předchozí kapitole (1). Je to technologie, umožňující přidávat počítačem generovaný obsah do reálného světa prostřednictvím nějakého média, např. mobilního telefonu. K tomu, aby něco takového bylo možné, je potřeba vyřešit celou řadu komplexních problémů. Hlavní výzvou je tracking - lokace - uživatele, potažmo jeho zařízení v prostoru. Abychom totiž mohli ukotvit virtuální objekt k nějakému reálnému objektu, potřebujeme vědět jak uživatel zařízením hýbe t.j. mění pozici a orientaci (natočení). Tímto požadavkem definujeme tzv. stupně volnosti - Degrees of freedom (DOF). 3 pro změnu pozice ve směru osy x, osy y a osy z a 3 pro rotaci v těchto osách, celkem tedy 6. V AR je spousta technik jak trackovat uživatele/zařízení z nichž relevantní pro tuto práci je ta, která pracuje s možnostmi zařízení, jež je vybaveno senzory. Ty můžeme rozdělit na nevizuální a vizuální [3].

2.4.1 Nevizuální senzory

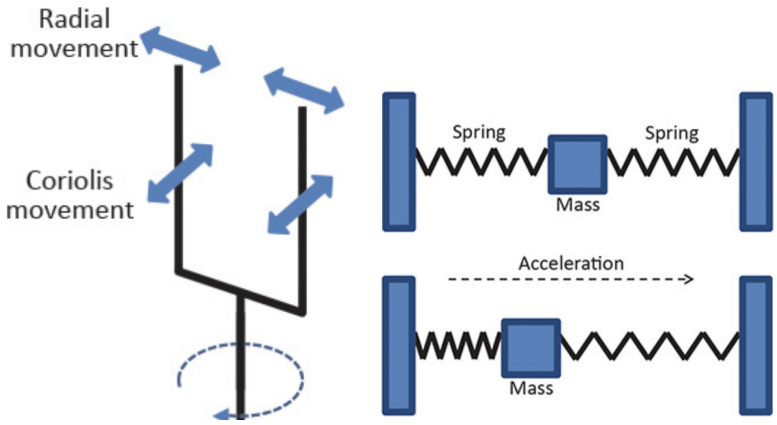
Každé moderní zařízení, ať už mobilní telefon či tablet je vybaveno (mimo jiné) čtyřmi důležitými senzory. Jejich funkčnost je silně omezena externími podmínkami (viz. dále), nicméně jejich rychlost (vzorkovací frekvence 50-100Hz) a konstantní dostupnost z nich dělá dobré kandidáty. Navíc jsou tyto senzory často velmi levné na výrobu a tak jsou přítomny i v těch nejlevnějších chytrých zařízeních:

- GPS senzor - přijímač, který umožňuje zjistit pozici, AMSL výšku a rychlost. Jeho funkcionalita byla detailně rozebrána v předchozí sekci (2.3).
- Magnetometr - také "elektronický kompas"- měří relativní pozici vůči magnetickému pólu Země. Funguje na principu Hallova jevu. Bohužel v praxi jsou tyto magnetometry ovlivněny lokálními magnetickými poli okolním subjektů jako jsou např. elektronická

či kovová zařízení a jsou tedy velmi nespolehlivé - běžně se můžeme setkat s nepřesností 30-40°. Tato nepřesnost se občas dá mitigovat speciálním krouživým pohybem ve tvaru osmičky naležato, což má za následek kalibraci.

- Gyroskop - měří úhlovou rychlost pomocí Coriolisovy síly působící na malé vibrující zařízení v senzoru (2.7a). S jejich pomocí jsme schopni spočíst relativní rotaci zařízení. Ve skutečnosti se kombinují tři ortogonální gyroskopy a dohromady tvoří tzv. Micro-electromechanical System (MEMS).
- Akcelerometr - pomocí změny pozice malého tělíska na pružině v lineárním akcelerometru jsme schopni měřit zrychlení v jedné ose (2.7b). Kombinace tří ortogonalních akcelerometrů opět tvoří MEMS. Jakmile známe akceleraci, jsme schopni dvojitou integrací spočítat změnu pozice od počáteční polohy. Nicméně, tím, že každý akcelerometr podléhá tzv. "driftu"neboli výchýlení, které narůstá v čase, kvůli dvojité integraci roste chyba vypočítané pozice kvadraticky [14].

Jak vidíme, tyto senzory dobře reflektují orientaci zařízení, nicméně na změny pozice se dají využít jen na velmi krátké vzdálenosti a ve velmi malém časovém okně, poté už kumulativní chyba přeroste snesitelnou hranici a výpočty pozbývají spolehlivosti. Jejich samostatné použití na tracking tedy není prakticky možné a proto se využívají spolu s visuálními (2.4.3) [3].



(a) Gyroskop a Coriolisova síla (b) Zrychlení v akcelerometru

Obrázek 2.7: Funkce nevizuálních senzorů. Zdroj: [3].

2.4.2 Visuální senzory, visuální odometrie

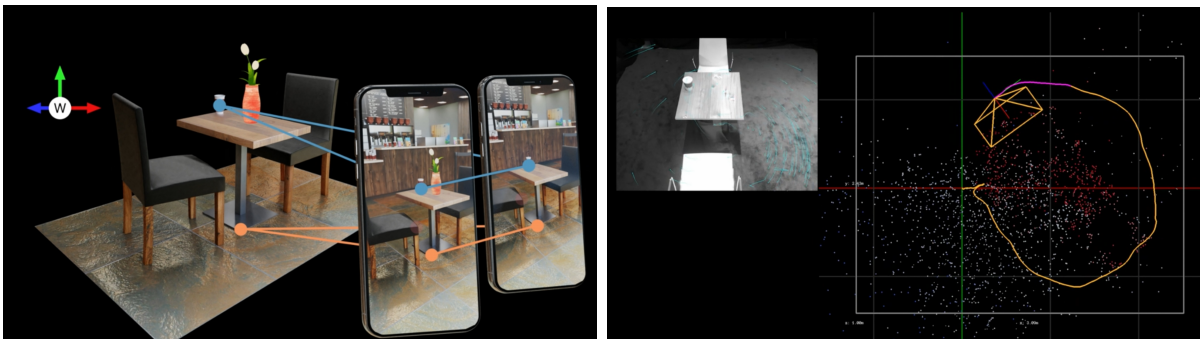
V oblasti AR je celá řada možností jak použít visuální senzor - optickou kameru - zařízení, mezi nimi například marker tracking, natural feature tracking by detection či incremental tracking. Nás bude ale hlavně zajímat tzv. Simultaneous Localization and Mapping (SLAM) (často ztotožňováno s pojmem visuální odometrie). Tento proces umožňuje kontinuální trackování 6DOF kamery. Složitost algoritmu je nad rámec této práce ale základní algoritmus by se dal velmi zjednodušeně popsat v následujících krocích:

1. Detekce orientačních bodů v prvním kamerovém snímku, například pomocí metody FAST. Taková úloha má svá omezení - je vyžadována dobře osvětlená místnost, s dostatečným detailem (různé změny hran, barev, textur). Pokud zařízení bude snímat bílou stěnu, algoritmus nebude schopen dobře provést lokalizaci.

2. V následujícím snímku detekce stejných bodů ze snímku předchozího.

3. Výpočet matice posunutí a rotace, což nám dává výslednou změnu polohy a orientace kamery, která nás zajímá (2.8a).

Metoda tak postupně vytváří mapu okolí (2.8b), zatímco v něm zároveň trackuje kameru - odtud název SLAM. Nevýhodu oproti nevisuálním senzorům má však kamera spolu s tracking algoritmem zřejmou - mnohonásobně vyšší spotřeba energie. [3].

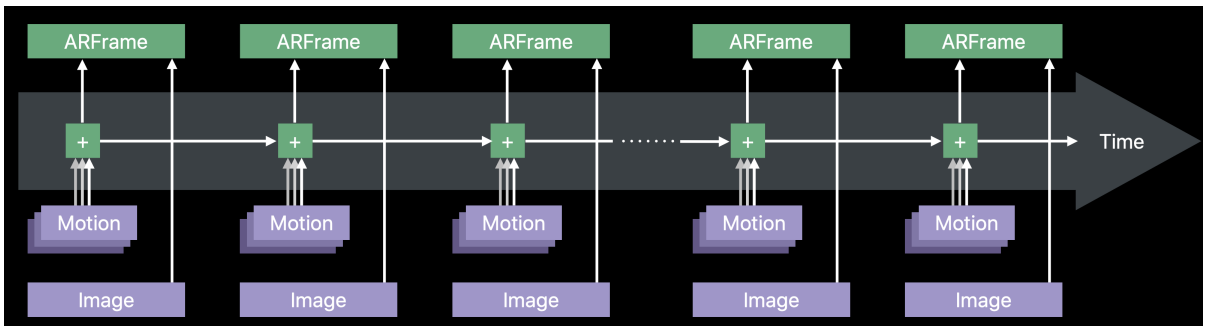


(a) Metoda SLAM (b) Mapování okolí

Obrázek 2.8: Visuální odometrie. Zdroj: [15].

2.4.3 Fúze senzorů

Vyjmenovali jsme si různé senzory, kterými je typické chytré zařízení vybaveno spolu s jejich pro a proti. Nabízí se jasné řešení - zkombinovat to nejlepší z obou světů a vytvořit velmi přesně a robustní řešení. Akcelerometr, magnetometr a gyroskop dohromady tvoří ortogonální systém označovaný jako Inertial Measuring Unit (IMU), jehož data jsou kombinována s těmi z visuální odometrie. Takové kombinaci senzorů se říká "Sensor fusion". S ní přichází další řada problému jako je synchronizace dat ze senzorů, z nichž každý má různou frekvenci jejich poskytování (2.9a), drift (který se obvykle vyhlazuje pomocí Kalmanova filteru) apod. Je zřejmé, že takové řešení převyšuje spotřebou energie předchozí způsoby, nicméně jeho klady značně převažují tuto skutečnost [3].



(a) Fúze senzorů. Zdroj: [15].

Fúze senzorů tedy splňuje původní požadavek na sledování 6DOF kamery a je to technologie, kterou využiji v této práci (resp. tato technologie funguje na pozadí frameworků, které budu používat).

2.5 Poskytovatelé dat

Existuje mnoho poskytovatelů aktuálních i historických dat. Rozeberu zde několik kandidátů, které bych mohl využít při implementaci aplikace. Všichni získávají základní data stejným způsobem pomocí ADS-B rozebraným v sekci 2.2 a další data jako jsou konkrétní informace o letu už získávají svým proprietárním způsobem.

2.5.1 FlightRadar24

Asi nejpoužívanější aplikace pro sledování letů v reálném čase (aktualizovaná každých 5 sekund) s největším objemem dat. Kromě standardních pozic letadel nabízejí čísla letů, registrační značky, volací značky a počáteční a cílová destinace (2.10). Mimo to nabízejí filtrování dle letek, letiště či zeměpisného regionu. Také poskytují informace o letových událostech jako vzlet, přistání, opuštění brány a.j. V neposlední řadě také umožňují nahlédnout do minulosti na historická data. Nevýhodou je, že jejich data jsou proprietární, nemají žádný ceník a člověk musí projít nepříjemným formulářem a vykomunikovat si přístup k jejich Application Programming Interface (API) aniž by věděl jestli to bude zpoplatněno či nikoliv [16].

2.5.2 OpenSky Network

Tato služba poskytuje základní, velmi omezená data z ADS-B - pozice, směr, rychlost a mezinárodní identifikátor International Civil Aviation Organization (ICAO) letadla. Obrovskou výhodou je, že data jsou otevřená a přístupná pro výzkumné a nekomerční využití a je k nim důkladně popsaná dokumentaci. Nijak své API neomezují a libovolný neregistrovaný uživatel může dostat aktuální data aktualizovaná každých 10, resp. pro registrované uživatele každých 5 sekund. Na základní funkcionalitu je toto výborný kandidát [17].

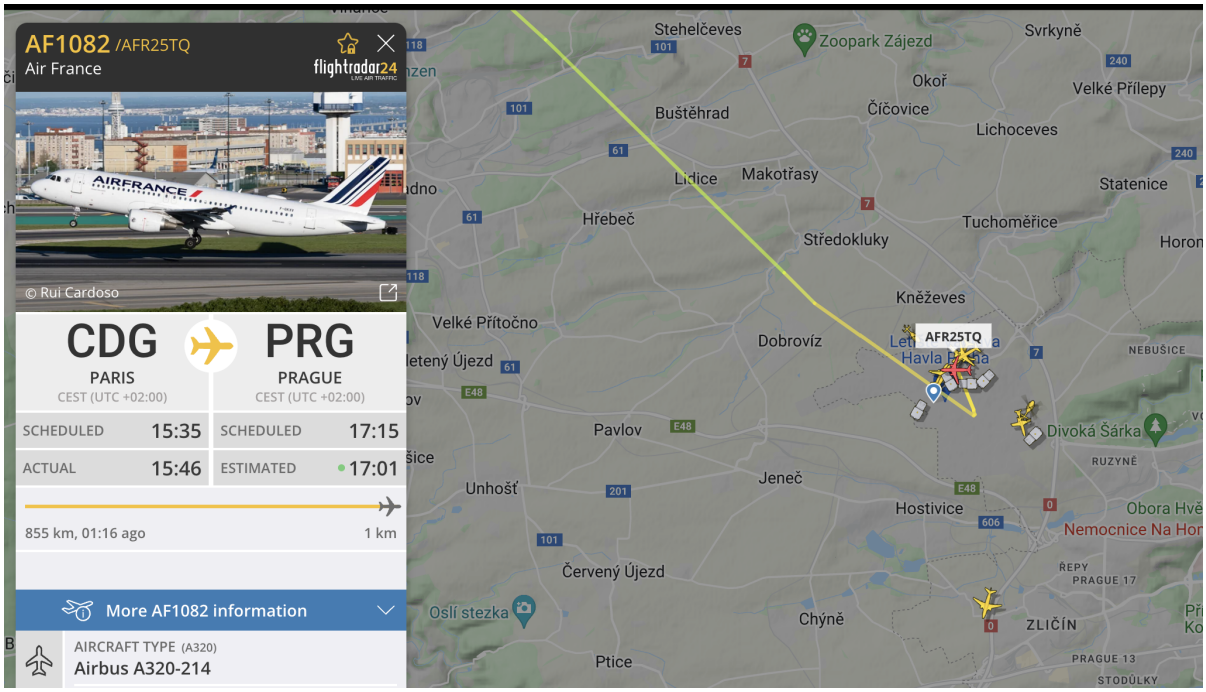
2.5.3 FlightAware

Ze všech poskytovatelů dat které jsem zkoumal má tento nejvíce doplňujících detailů o konkrétních letech jako je aerolinka, model letadla či celé názvy zemí odletu a příletu. Jejich ceník je trochu složitější - nabízejí určitou část dat zdarma, nicméně jsou omezena na 10 dotazů za minutu a každý dodatečný dotaz na doplňující informace je zpoplatněn. Nicméně je dobré o této stránce vědět, protože k datům se dá pomocí web scrapingu (proces kdy stáhneme zdrojový kód stránky a vytáhneme z něj data, která nás zajímají) přistoupit když známe konkrétní ICAO identifikátor letadla [18].

2.5.4 RadarBox

Obdobná funkcionalita jako FlightAware, ale oproti FlightAware nabízejí i fotky letadel, což by mohl být zajímavý dodatek do aplikace. Nenabízejí data zdarma, nejlevnější úroveň začíná na 120 dolarech za měsíc. Opět bych mohl využit techniku web-scrapingu a některá data z této stránky použít [19].

Abych zajistil, že data budou zdarma, dává mi největší smysl zkombinovat vícero poskytovatelů najednou a pomocí web scrapingu poskládat ucelený model, který budu v aplikaci využívat.



Obrázek 2.10: UI Flight Radar. Zdroj: [20].

2.6 Souřadné systémy

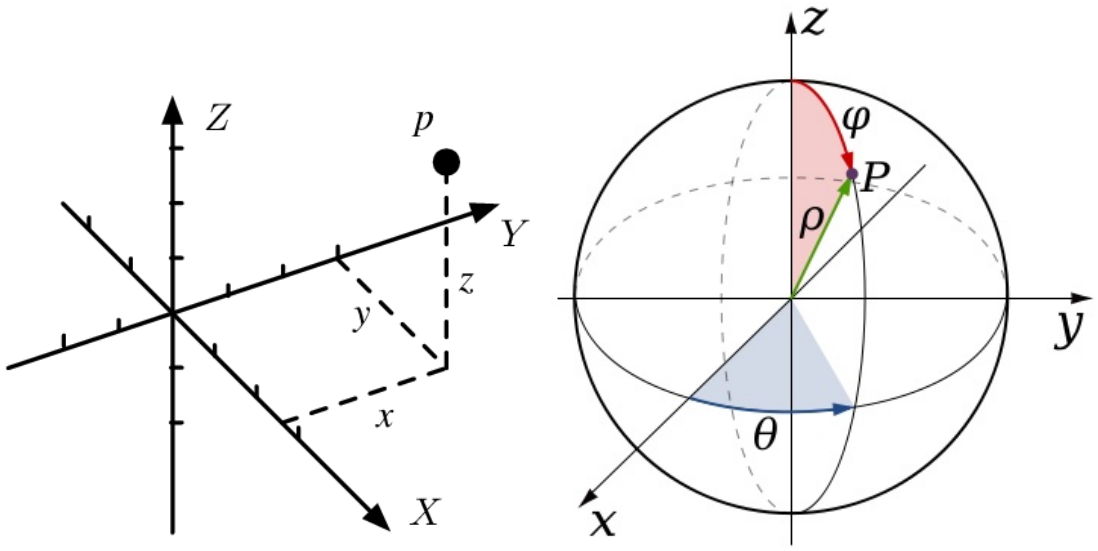
Abychom mohli jednoznačně popsat pozice, orientace a transformace v prostoru, potřebujeme souřadný systém. Ten každému bodu v prostoru jednoznačně přiřazuje číselné souřadnice (2.11a). Libovolný bod poté můžeme získat tak, že provedeme příslušnou transformaci os souřadného systému, kde jako koeficienty využijeme právě zmíněné souřadnice [21]. Souřadných systémů je celá řada, nás ale budou zajímat tři.

2.6.1 Kartézský souřadný systém

První ze systému je nám velmi blízký kartézský souřadný systém. Libovolný bod v trojrozměrném prostoru je jednoznačně určen 3 číselnými souřadnicemi kde každá udává koeficient lineární kombinace ortonormálních os x, y a z. Díky své přirozenosti je tento systém hojně využíván, mimo jiné, i v počítačové grafice a rozšířené realitě [22].

2.6.2 Sférický souřadný systém

Alternativní způsob jak popsat body v prostoru je představit si, že leží na sféře - odtud název systému. Libovolný bod pak můžeme popsat pomocí poloměru sféry r a dvou úhlů: θ, který udává úhel odklonu bodu od osy x a úhlu γ, který udává úhel odklonu bodu od osy z (2.11b). Pro různě vzdálené body od počátku systému přirozeně odpovídají sféry s různými poloměry r. Pokud se ovšem soustředíme na popis bodů na povrchu jedné konkrétní sféry, stane se z poloměru konstanta a bude nám stačit znát pouze dva zmíněné [23]. Přesně takto fungují zeměpisné souřadnice, kde r je poloměr Země. To by fungovalo dobře, kdyby Země byla perfektní matematická sféra, což bohužel není, její tvar je mnohem komplexnější.



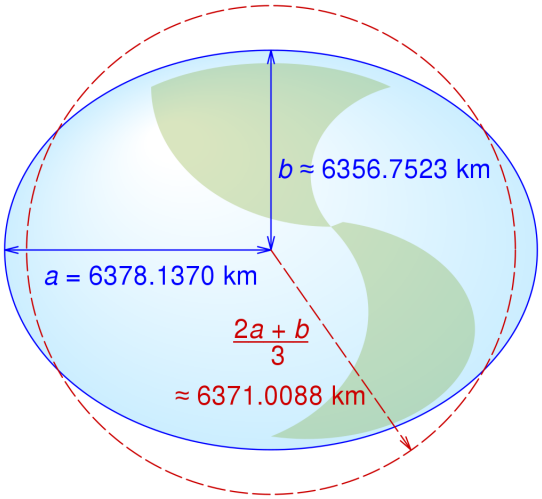
(a) Kartézský souřadný systém. Zdroj: [24]. (b) Sférický souřadný systém. Zdroj: [25].

Obrázek 2.11: Souřadné systémy.

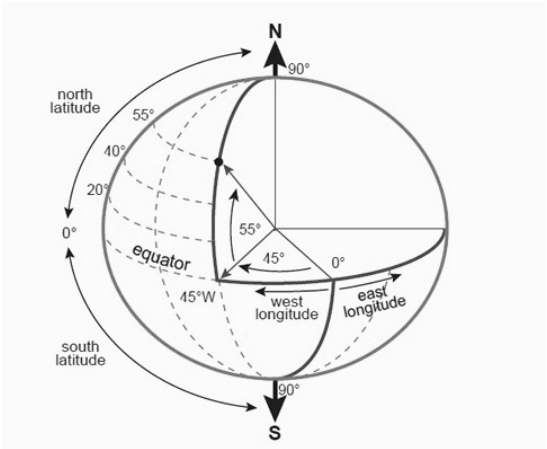
2.6.3 World Geodetic System, zeměpisné souřadnice

Geodeti z celého světa desítky let pracují na vytvoření standardu, který umožní pracovat se Zemí jako s matematickým modelem za účelem navigace v principu sférického souřadného systému. Jedním z těch nejpoužívanějších je americký standard WGS - World Geodetic System, jehož nejnovější varianta je z roku 1984, odtud WGS 84. Definuje řadu proměnných jako hmotnost, úhlovou rychlost či gravitační parametr Země. Nás ale zajímá poloměr. Standard WGS 84 definuje Zemi jako tzv. referenční elipsoid. Je to z toho důvodu, že elipsoid nejlépe popisuje zploštělou Zemskou sféru a zároveň je to model se kterým se dobře pracuje. Rotace Země kolem své osy má totiž za následek zploštění na pólech kvůli odstředivé síle. Hlavní osa Země je standardem určena (přibližně) jako 6378,137 km a vedlejší osa jako 6356,752 km (2.12a). Průměrná hodnota je definována jako 6371,008 km [26] [27].

S tímto standardem pracují i GPS souřadnice. Rozdíl od sférického souřadného systému je hlavně v tom, že druhý úhel udává odchylku od osy y, nikoliv z. Prvnímu úhlu se také říká zeměpisná délka - anglicky longitude (úhel odklonu od osy x). Body se stejnou zeměpisnou délkou leží na stejném tzv. poledníku. Bod s nulovou zeměpisnou délku se nazývá nultý poledník. Nabývá hodnot 0°-180° pro pravou hemisféru, pro levou je rozsah stejný, akorát se záporným znaménkem. Obdobně je pak druhý úhel nazývaný zeměpisná šířka - anglicky latitude a udává odklon od bodů s nulovou zeměpisnou šířkou = rovníku. Nabývá hodnot od 0° na rovníku po 90° na pólu pro severní hemisféru a stejných hodnot s opačným znaménkem pro tu jižní (2.12b). Je zvykem psát souřadnice jako uspořádanou dvojici, kde první je zeměpisná šířka a druhá je zeměpisná délka. Praha má souřadnice 50.0755° severní šířky a 14.4378° východní délky [28].



(a) WGS referenční model. Zdroj: [29].



(b) Zeměpisné souřadnice. Zdroj: [30].

Obrázek 2.12: WGS model a souřadnice.

2.7 Projekce

Pro zobrazení povrchu sféry (případně naší Země) na dvourozměrnou rovinu, musíme provést projekci (či mapovací transformaci). Výsledkem je tzv. mapa. S mapou se nám lépe pracuje a staletí jsme zvyklí ji používat k navigaci. Pro sféru můžeme předpokládat 4 základní charakteristiky:

1. Všechny plochy jsou korektně reprezentovány.
2. Všechny vzdálenosti jsou korektně reprezentovány.
3. Všechny úhly jsou korektně reprezentovány.
4. Tvary oblastí jsou věrohodné.

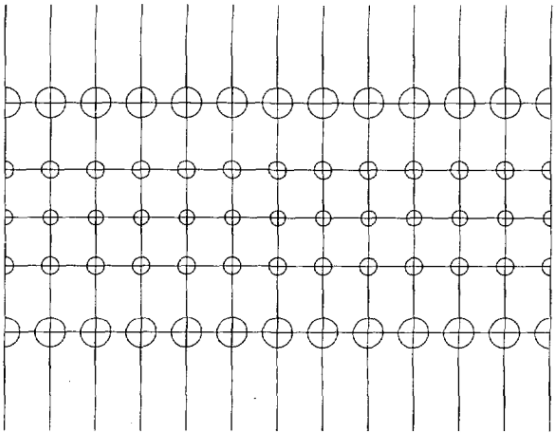
Kvůli fundamentální odlišnosti povrchu sféry a roviny je ale nevyhnutelné, že při mapování dojde k určitému zkreslení a ne všechny 4, výše uvedené, charakteristiky budou zachovány. Z tohoto důvodu vznikla celá řada projekcí, každá se svými výhodami a nevýhodami, které se liší právě svými charakteristikami. Opět z této plejády projekcí vyzdvihnu dvě, které jsou relevantní pro tuto práci [31].

2.7.1 Mercator projekce

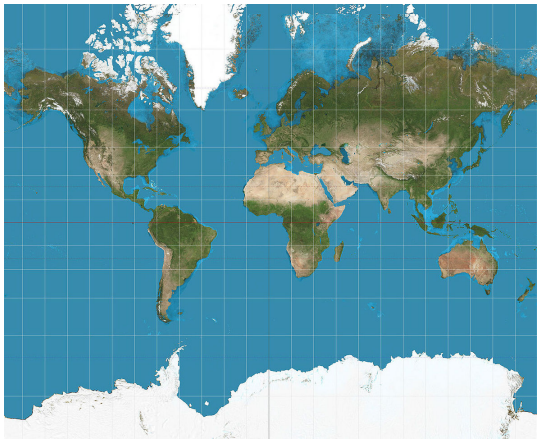
První z nich je cylindrická (mající navzájem kolmé poledníky a čáry zeměpisných šířek) Mercator projekce, která zachovává úhly. To má využití zejména v mořské či letecké navigaci, protože stejný úhel na mapě odpovídá stejnému úhlu při cestě na Zemi, neboli cesta po přímce na mapě odpovídá konstantnímu kurzu plavidla či letadla. Druhou výhodou této projekce je zachování tvarů.

Hlavní nevýhodou je, že s narůstající vzdáleností od rovníků zkresluje (zvětšuje) plochy a tedy i vzdálenosti (2.13b). Oblasti na pólech není schopna zobrazit vůbec. I přes tento zásadní fakt se nesprávně používá v podstatě na všech moderních mapách jako Google

Maps, Mapy.cz, Apple Maps, OpenStreetMaps apod., což má za následek chybné vnímání velikostí jednotlivých zemí. Například Grónsko se na mapě vytvořené Mercator projekcí jeví přibližně stejně velké jako Afrika, přitom má cca. 15x menší rozlohu (2.13b) [31].



(a) Zvětšení ploch při projekci. Zdroj: [31].

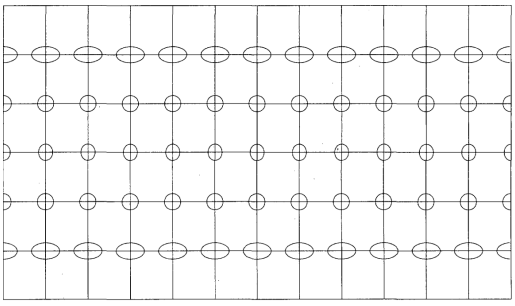


(b) Zkreslená mapa světa. Zdroj: [32].

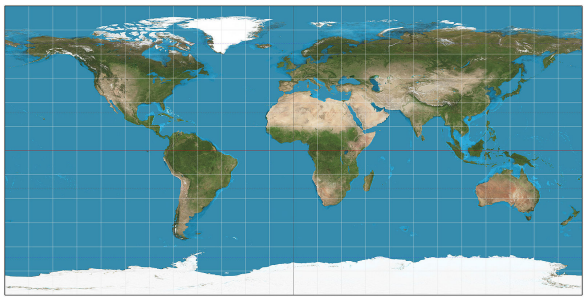
Obrázek 2.13: Mercator projekce.

2.7.2 Ekvidistantní válcová projekce

Druhá je opět cylindrická, ekvidistantní projekce (zachovávající vzdálenost mezi libovolným bodem a ohniskem mapy). Má stejně vzdálené všechny poledníky a rovnoběžné čáry zeměpisné šířky. Nezachovává tvary, plochy ani úhly (2.14a). Z toho důvodu nemá využití pro navigaci. Naopak je vhodná pro počítačovou grafiku, protože každá pozice na sféře odpovídá unikátním souřadnicím x, y v rastrovém obrázku (2.14b). Pro oblasti dostatečně blízko u sebe a zároveň dostatečně blízko rovníku se dá využít na transformaci rozdílu pozic daných zeměpisnými souřadnicemi na rozdíl pozic v kartézském souřadném systému. Pro potřeby této práce je tento předpoklad splněn (letadla se zobrazují většinou do vzdálenosti 100 km což v měřítku Země je "dostatečně blízko"). Tyto znalosti využijeme v sekci 2.9 [31].



(a) Zvětšení ploch při projekci. Zdroj: [31].



(b) Zkreslená mapa světa. Zdroj: [33].

Obrázek 2.14: Ekvidistantní válcová projekce.

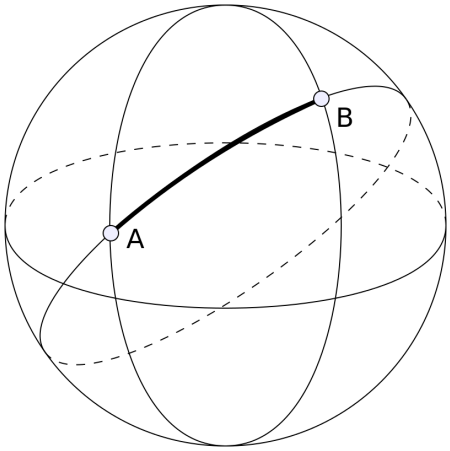
2.8 Vzdálenosti na sféře

Jelikož uživatelovo zařízení a letadla v jeho okolí jsou popsány zeměpisnými souřadnicemi, musíme být schopni spočítat jejich vzájemnou vzdálenost. Jinými slovy nás zajímá délka ortodromy (také oblouk hlavní kružnice), což je nejkratší spojnice dvou bodů na ploše sféry (2.15).

Existuje vícero způsobu jak tuto vzdálenost spočítat a nejzákladnější z nich je obdoba kosinovy věty z trigonometrie, aplikovaná na sférickou trigonometrii (pro výpočet délek stran polárního trojúhelníku). Častěji používaný způsob výpočtu je ale tzv. Haversine formula [34]:

$$
\begin{aligned}
a &= \sin^2(\Delta\varphi/2) + cos(\varphi_1) \cdot cos(\varphi_2) \cdot \sin^2(\Delta\lambda/2) \\
c &= 2 \cdot atan2\left(\sqrt{a}, \sqrt{1-a}\right) \\
d &= R \cdot c
\end{aligned}
$$

kde φ_1 a φ_2 jsou zeměpisné šířky dvou bodů na sféře, $\Delta\varphi$ je rozdíl jejich zeměpisných délek, $\Delta\lambda$ je rozdíl jejich zeměpisných šířek a R je poloměr Země. Pro poloměr využijeme hodnoty 6371 km (viz. sekce 2.6.3). Chyba této formule narůstá pouze s rozdílem zeměpisných šířek a dosahuje maximálních hodnot 0.5%. Je tedy více než dostatečná pro potřeby této práce. Je rovněž využívána v mapových podkladech od Google Maps či OpenStreetMaps. Existuje ještě velmi precizní a komplikovaná iterativní Vincentyho metoda která je schopná určit délku ortodromy s přesností až na půl milimetru (vůči které se počítají chyby ostatních metod). Využívá se v geodezii pro velmi přesné vědecké výpočty. Ta je ovšem nad rámec této práce [35].



Obrázek 2.15: Ortodroma. Zdroj: [36].

2.9 Umisťování letadel, transformace

Klíčovým problémem této práce je správné umístění letadel do virtuálního světa kamery uživatelova zařízení tak, aby jejich AR pozice (udávaná v kartézských souřadnicích, viz. 2.6.1) korespondovala s tou v reálném světě (udávanou ve sférických souřadnicích, viz. 2.6.2). Roli tedy hraje nejen uživatelova pozice, ale i směr, kterým se dívá (vůči magnetickému

severu), jelikož translace (posunutí) letadel ve scéně vůči nějakému počátku záleží na směru pohledu. Konkrétně: když se uživatel dívá na sever, může mít letadlo nalevo (na západě), ale když se dívá na jih, má to samé letadlo napravo.

Problém jsem tedy rozpadnul na dvě části:

1. Transformace souřadného systému kamery tak, aby vždy směřoval na sever. To je vcelku jednoduché, stačí rotovat systém o opačný úhel, než kterým se uživatel dívá (když se dívá na západ, tedy o 90° vlevo od mag. severu, otočím systém o 90° vpravo). Ten lze zjistit z magnetometru. Samozřejmě taková rotace bude jen tak přesná jako data z magnetometru a tedy celková přesnost a věrohodnost umístění letadel do scény je jím omezena (viz. sekce 2.4).

2. Spočítání translace letadla ve scéně vůči uživateli (přesněji vůči souřadnému systému kamery). K tomu je zapotřebí aby uživatelova aktuální pozice daná zeměpisnými souřadnicemi korespondovala s počátkem souřadné soustavy kamery, která je v kartézských souřadnicích. Neboli GPS pozice uživatele (φ, λ) odpovídá $(0, 0, 0)$ v systému kamery. V tuto chvíli už jen stačí spočítat translaci pozice letadle od uživatele ve sférických souřadnicích a tu použít jako translaci v systému kamery. Díky ekvidistantní projekci (viz. sekce 2.7.2) můžeme zformovat pravoúhlý trojúhelník ve sférických souřadnicích a příslušné délky odvěsen jsou hledané položky translace [34].

Oba body korespondují k afinním zobrazením, které lze reprezentovat maticemi - matice rotace R a matice translace T. Složením těchto dvou matic vznikne výsledná transformační matice, kterou když aplikujeme na souřadný systém kamery, dostaneme správně umístěný souřadný systém letadla, ve kterém už můžeme vykreslit siluetu či model s informacemi o letadle. Jedná se o maticové násobení $R * T$. Pořadí je důležité, posunutí musí být vzhledem k rotovanému systému kamery [21] [37].

Kapitola 3

Návrh řešení

Aby technologie byla považována za AR, musí splňovat tyto tři charakteristiky: kombinace reálného a virtuálního světa, interaktivita v reálném čase a zobrazení ve 3D. [3]. To jsou základní pilíře na kterých budu aplikaci stavět. Dále jsem si z aplikací probraných v sekci 2.1 odnesl tři body, na kterých bych rád projekt postavil:

- 2D mapa - kombinace pěkného vzhledového kabátku z Google Maps (2.1a) a funkcionality z Flight Radaru (2.2a). Mapa bude zobrazovat pozici uživatele a směr, kterým se dívá, spolu s pohledovým kuželem aby měl přehled o tom, kolik může vidět letadel. Na mapě budou vizualizována letadla ve vzduchu, natočená a pohybující se směrem jejich kurzu. Mapa se bude otáčet jako u FlightRadaru - uživatel s kuželem je stále na místě a "svět se točí" kolem něj.

- Realistické umístění objektů do světa. Rád bych do obrazovky umístil 3D objekty letadel natočené správným směrem letu, které budou škálované na základě blízkosti k uživateli. Vzdálenější letadla budou mít menší siluetu. Zde bude potřeba vyřešit přesnost pozice a ideálně to otestovat v provozu na skutečném letadle na které budu mít vizuální dohled.

- Správné řešení hloubky - viz. IKEA place (2.4c), která vykresluje objekty správně tak, že bližší zakrývá vzdálenejší. Pokud v jednom směru bude za sebou více letadel, zobrazí se chytře jen to nejbližší, abychom předešli problému, který nastává u Flight Radaru (2.2c). Případně budu limitovat množství zobrazených letadel v jednom směru.

V poslední řadě bych k aplikaci rád přidal následující rozšíření, které zpříjemňují UX a rozšiřují implementaci:

- Prezentování dat v lehce stravitelném formátu. Uživatel by neměl být kognitivně zahlcen a v hlavním view by mělo být zobrazeno opravdu limitní množství informací.

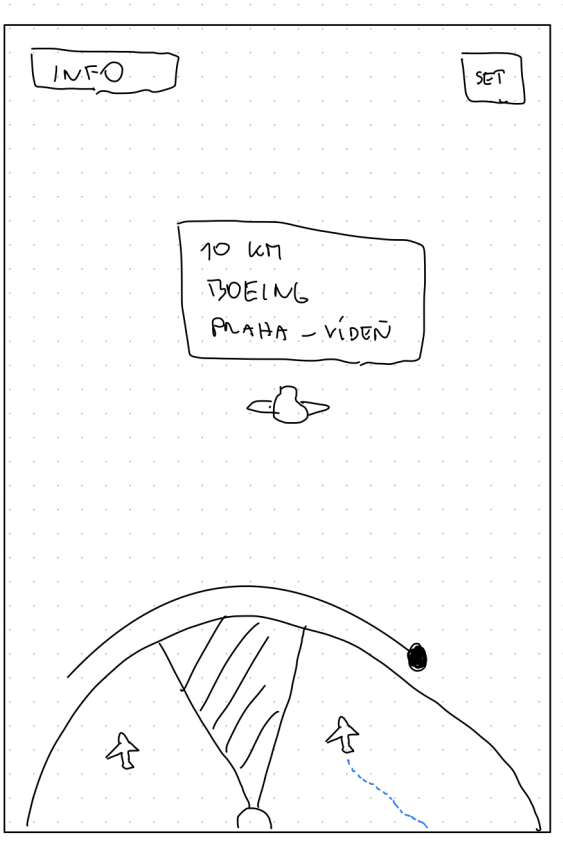
- Detail dráhy letu po zvolení konkrétního letadla. Na 2D mapě a možná se pokusit o experimentaci se zobrazením 3D dráhy i v AR view.

- Detail letadla, který bude obsahovat informace jako volací značka, rychlost, směr, výška.

- Interpolace dat tak, ať jsou letadla plynule interpolována na obrazovce.

3.1 UI a UX

Při návrhu funkcionality začínám tak, že nejprve načrtnu tzv. wireframe - hrubý nákres toho, jak by aplikace mohla vypadat. Klíčový není vzhled, nýbrž kde budou jaké grafické elementy a jaká bude jejich funkce [38]. Abych se řídil body uvedenými výše, do wireframu jsem zakomponoval mapu s letadly v okolí, z nichž jedno zvolené zobrazuje svou dráhu, umístění letadel, posuvník na filtrování letadel v závislosti na vzdálenosti od uživatele a dva pomocné elementy pro zobrazení dodatečných informací jako jsou ladící výpisy a tlačítko na zobrazení nastavení (3.1).

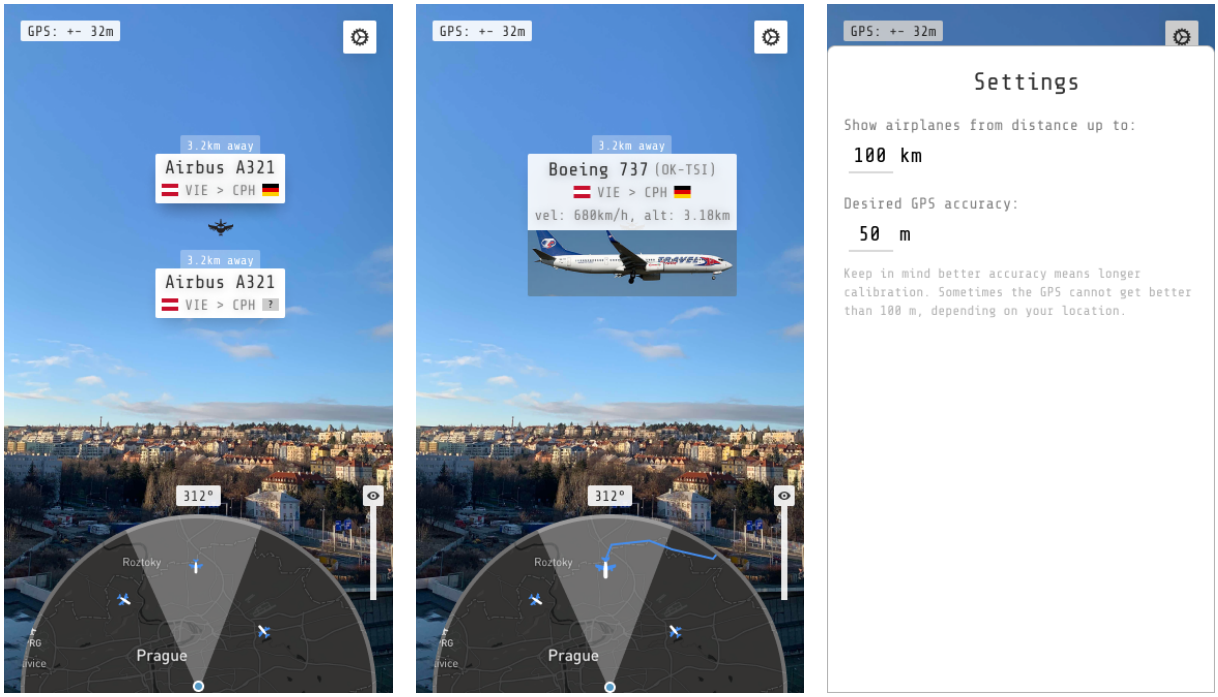


Obrázek 3.1: Wireframe - hrubý vzhled aplikace

Když jsem s wireframem spokojený je čas z něj vytvořit tzv. high-fidelity prototype neboli prototyp se všemi detaily. Na to použiji moderní a velmi rozšířený designovací nástroj Figma. Na prvním obrázku (3.2a) je vidět návrh na základě wireframu. Přijde mi relevantní ukázat i směr, kterým uživatel hledí v podobě kompasu. Přesunul jsem posuvník na filtraci letadel dle vzdálenosti do pravé části obrazovky. Dále mi nepřijde relevantní pro uživatele zobrazovat ladící výpisy, tak jsem do levého horního rohu místo toho dal přesnost (resp. odchylku) GPS. Kartička letadla je velmi minimalistická, zobrazuje jen model letadla, jaká je jeho vzdálenost od uživatele a odkud kam letí. Jako drobný detail jsem přidal vlaječky zemí, protože uživatel nemusí být obeznámen se zkratkami letišť. Také jsem navrhl zástupnou ikonku pro případ, že vlajka země nebude dostupná z dat.

Na druhém obrázku (3.2b) je znázorněná obrazovka po zvolení konkrétního letadla. Zobrazí se detailnější kartička, obsahující registrační značku, rychlost, AMSL výšku a fotku letadla. Zároveň se na mapě ikonka letadla zvětší a zobrazí se dráha letu.

Na třetím obrázku (3.2c) je vidět velmi jednoduché nastavení, kde si uživatel může zvolit z jaké maximální vzdálenosti se mu mají letadla zobrazovat a také si může upravit požadovanou přesnost GPS.



(a) Hlavní obrazovka (b) Detail letadla (c) Nastavení

Obrázek 3.2: High fidelity prototyp aplikace

Kapitola 4

Implementace

V této kapitole rozeberu technické detaily při realizaci navrhovaného řešení na základě analýzy.

4.1 Platforma a použité technologie

Aplikaci budu tvořit pro mobilní platformu iOS od společnosti Apple, jelikož s ní mám předešlé zkušenosti. Zdrojový kód budu psát v jazyce Swift, což je moderní open-source nástupce jazyka Objective-C a k vývoji použiji proprietární Integrated Development Environment (IDE) Xcode, rovněž od společnosti Apple.

Swift nabízí nabízí řadu Kitů/Core frameworků, což jsou samostatné knihovny/celky s jednou konkrétní funkcionalitou, s jejichž pomocí si může programátor aplikaci modulárně postavit. Stěžejním je pro mě ARKit spolu se SceneKitem, které jsou zodpovědné za detekci uživatele v prostoru a následné umisťování objektů do reálného světa. Pro tvorbu UI použiji standardní UIKit spolu s Auto-layoutem k responsivnímu pozicování prvků na obrazovce. Krom toho ještě využiji MapKit k načtení, zobrazení a manipulaci s mapou, CoreLocation pro přístup k GPS pozici zařízení a CoreFoundation pro základní stahování dat přes protokol HTTP [39].

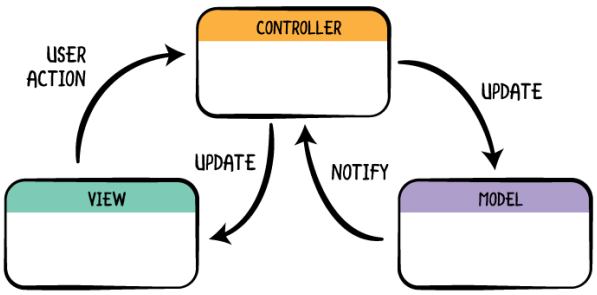
4.2 Architektura aplikace

Vzhledem k jednoduchosti navigace a struktury aplikace zde postačí použít standardní Model View Controller (MVC) architekturu (4.1). Model obsahuje tzv. "byznys logiku" neboli struktury a objekty, které aplikace využívá. Hlavními modely jsou `Record` - jedno letové datum obsahující čas, souřadnice, výšku, rychlost a kurz letadla a `Flight` což je struktura obsahující informace o letu letadla jako je volací značka, model a fotka letadla, historie `Record`ů a dráha letu.

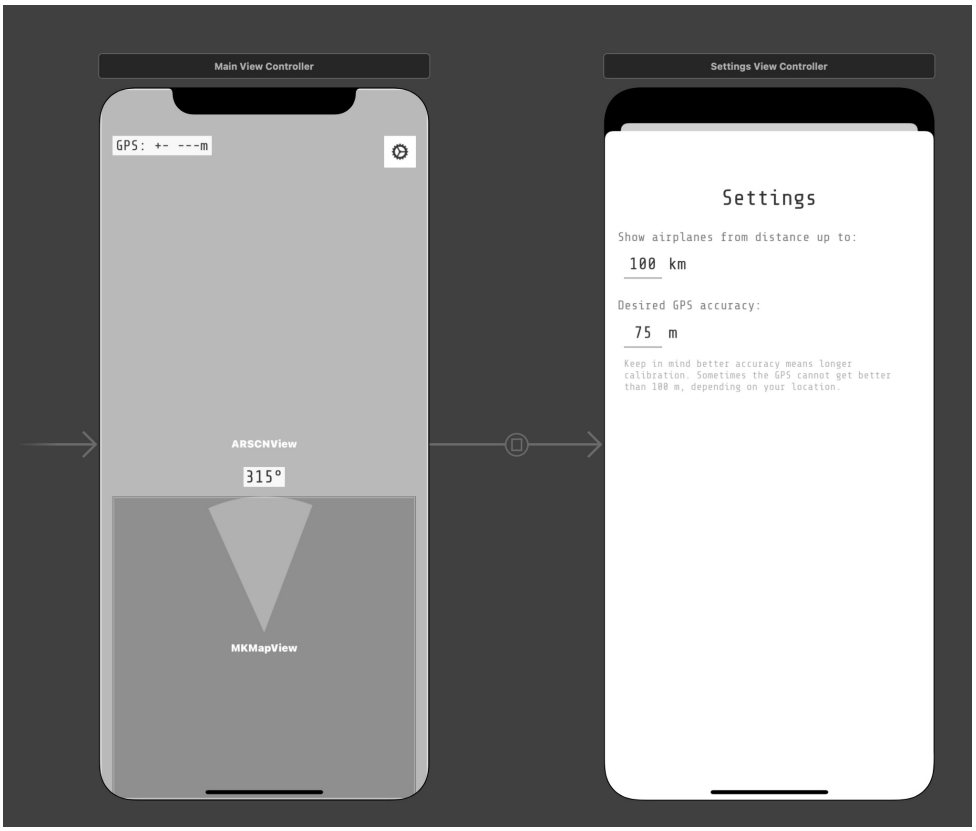
View jsou specializované třídy zodpovědné za vykreslovaní hodnot, které obdrží z Controlleru, na obrazovku a poté za zpracovávaní uživatelských vstupů (dotyk na obrazovku), které následně předají Controlleru, který modifikuje a uloží Model. Měly by být bezestavové a všechna byznys logika by měla být umístěná v Controlleru.

Controlleru se ve světě Apple říká ViewController a je úzce spjatý s tzv. storyboardem - souborem ve formátu XML, ve kterém si programátor může tvořit prvky UI (Views) chytrým WYSIWYG editorem Interface Builder (4.2) a ty pak napojit do zmíněného

ViewControlleru, kde je může v kódu měnit či aktualizovat. Setkal jsem se se synonymním pojmem code-behind ve WPF od firmy Microsoft.



Obrázek 4.1: Návrhový vzor MVC. Zdroj: [40].



Obrázek 4.2: Interface Builder v Xcode, flow aplikace.

Aplikace má dvě obrazovky:

- `MainViewController` - zobrazující Mapu + AR scénu. Je zodpovědný za správné zobrazení dat jak na mapě, tak ve scéně. Ty dostane ze specializované třídy `FlightFeed`, která data stahuje pomocí HTTP protokolu ze stránek poskytovatelů rozebraných v sekci 2.5.
- `SettingsViewController` - zodpovědný za zobrazování, nastavování a ukládání uživatelských preferencí. Lze se na něj dostat přes tlačítko nastavení v předchozí obrazovce. Zpět se uživatel vrátí gesturou zavření (pohyb prstem dolů - nativní gestura pro platformu iOS).

Obě obrazovky jsou navrženy pomocí výše zmíněného Interface Builderu (4.2).

4.3 Zdroj dat

4.3.1 Primární zdroj - OpenSky Network

Po zanalyzování poskytovatelů dat v sekci 2.5 jsem se rozhodl použít OpenSky Network jako primární zdroj základních dat o letech. Data jsou dostupná s časovými rozestupy 5 sekund, čili `FlightFeed` se každých 5 sekund dotáže na endpoint `GET /states/all`. V dokumentaci je ale uvedeno, že data nemusí být pokaždé nová a že může nastat kompletní výpadek konkrétního letu, takže API bude vracet duplicitní data. O tom jsem se přesvědčil i v praxi - občas letadlo stálo na místě i 30 sekund. To je bohužel limitace, kterou lze obejít jedině změnou poskytovatele, kde ovšem rovněž nemusí být spolehlivost dat zajištěna [17].

Abych nestahoval data o všech letech na světě, což je soubor v řádech jednotek MB, je možné přidat hraniční obdélník (bounding box) jako součást parametrů dotazu. Obdélník vymezuje oblast na Zemi pomocí 4 parametrů `lamin, lomin, lamax, lomax` které udávají minimální a maximální zeměpisnou šířku, resp. délku. Dává tedy smysl vytvořit obdélník okolo uživatelovy pozice (uživatel je střed obdélníku). S použití jednoduché trigonometrie a znalosti sférických souřadnic jsem vytvořil funkci, která pro libovolnou zeměpisnou pozici a vzdálenost (velikost) vytvoří příslušný hraniční obdélník - viz A.

Velikost obdélníku odpovídá uživatelově preference vzdálenosti, kterou si může volit - když si zvolí letadla do maximální vzdálenosti 80 km, vytvoří se kolem něj obdélník který má délku strany 160km, 80km doprava, 80km doleva. Díky této optimalizaci se stahují jen lety z okolí uživatele (filtrace na základě vzdálenosti), soubor s daty má jen pár jednotek kB a jeho parsování trvá necelou milisekundu.

API vrací poziční data ve formátu JavaScript Object Notation (JSON). Napsal jsem velmi jednoduchý parser, který JSON přečte a vytvoří pole `Record`ů. Z dat si beru jen část, konkrétně `icao24, callsign, lastContact, longitude, latitude, onGround, velocity, trueTrack, geoAltitude`. Třída `FlightFeed` si udržuje pole letů `Flight`, kde postupně nově přibývající `Record`y přiřazuje ke korespondujícím letům a každý let má ještě doplňující informace (viz další sekce 4.3.2) [17].

Pro jednodušší práci s JSON a HTML formáty používám open-source knihovny `Swift-Soup` a `SwiftyJSON`.

4.3.2 Doplňková data

Abych mohl zobrazit dodatečná data jako je fotka, volací značka či destinace potřebuji ještě další poskytovatele. Po prozkoumání zdrojového kódu stránek FlightAware jsem objevil data, která zobrazují na svém webu v proměnné `trackpollBootstrap`, pro konkrétní let na adrese `https://flightaware.com/live/flight/registracni-Znacka-Letu`. Proměnná obsahuje jeden dlouhý řetězec, který je ve vyhovujícím JSON formátu. V něm jsem našel potřebné informace o výrobci a modelu letadla a názvy letišť odletu a příletu spolu s názvy příslušných zemí. V datech je také pole všech předchozích pozic zadaných v zeměpisných souřadnicích, která ukládám interně do proměnné `flightPath` a která reprezentují předchozí trajektorii letu.

Obdobným způsobem jsem ještě získal Uniform Resource Locator (URL) odkaz na obrázek letadla ze zdrojového kódu stránky `https://www.radarbox.com/data/mode-`

`s/icao-Letadla` poskytovatele RadarBox v HTML značce s identifikátorem `image-container`.

Může se samozřejmě stát, že žádná z těchto doplňujících dat nebudou dostupná a proto mám tyto případy v aplikaci ošetřené nahrazujícími texty "—" v případě, že chybí číselná či textová hodnota. Když chybí fotka letadla, nezobrazí se uživatelovi po zvolení konkrétního letadla.

Pro implementaci vlaječek příslušných zemí využívám open-source knihovny `FlagKit`, ve které jsem upravil mapování názvu zemí na příslušné 2 písmenné kódy, které korespondují k jednotlivým vlaječkám:

```
"Andorra": "AD",
"United Arab Emirates": "AE",
"Afghanistan": "AF",
"Antigua & Barbuda": "AG",
...
```

A pro stahování a jednoduché ukládání obrázku do mezipaměti a následné zobrazení využívám open-source knihovnu `KingFisher`.

4.4 Mapa

Pro implementaci mapy využívám již zmíněného frameworku MapKit. Jednotlivé letadla tam lze umístit velmi jednoduše prostřednictvím objektu `MKAnnotation`, kterému lze přiřadit zeměpisnou pozici a MapKit už zařídí, že se na mapě správně zobrazí. Pokaždé když přijde nová pozice letadla, aktualizuji příslušný objekt na mapě.

`MKAnnotation` rovněž umožňuje vykreslit libovolný obrázek na své pozici - použil jsem tedy ikonku letadla otočenou ve směru letu (proměnná `trueTrack`).

Při zvolení konkrétního letadla na mapě ikonku zvětším a zároveň zobrazím trajektorii letu, kterou mám k dispozici v proměnné `flightPath`. Vykreslení opět řeší MapKit, kterému stačí dát pole zeměpisných pozic a framework je po částech spojí ortodromami. Já už jen dospecifikuji tloušťku a barvu trajektorie.

Uživatel je vždy ve středu mapy - kdykoliv obdržím novou GPS pozici, vycentruji na ní mapu pomocí metody `mapView.setCenter(location.coordinate, animated: false)`. Dojem pohledového kuželu vytvářím tak, že otáčím celou mapou, stejně jako aplikace FlightRadar (viz. 2.1.2), na základě natočení zařízení dle magnetometru. Stejnou hodnotu natočení zároveň zobrazuji nad mapou. Mapa se přibližuje automaticky tak, aby vždy byl vidět hraniční obdélník a uživatel nemusel vlastnoručně přibližovat prsty. Kvůli tomu, že se mapa automaticky sama pozicuje mi přišlo vhodné zablokovat manuální posouvání a přibližování, protože se stejně po chvilce vrátí do automatické polohy.

MapKit je velmi omezený co se týče stylování mapy, proto jsem ponechal standardní barvy, které poskytuje, místo původní navržené šedivé varianty (3.1) [41].

4.5 ARKit, umisťování letadel do scény

Nativní framework ARKit funguje na principu relace. Vytvoří se objekt `ARSession` s nějakou konfigurací typu `ARSessionConfiguration`. Každá z nich odemyká relaci odlišné schopnosti, od trackování orientace zařízení až po detekci obrázků a obličejů. Z celkově osmi konfigurací jsem po rozsáhlé analýze vybral `ARWorldTrackingConfiguration`, která umožňuje tracking 6DOF kamery (detaily v sekci 2.4). Také nabízí celou řadu dalších

možností od detekci ploch až po detekci obličejů, ale pro účely této práce nám stačí její schopnost trackingu. Ostatní konfigurace neposkytovaly mnou kýžené vlastnosti. `ARGeoTrackingConfiguration` funguje jen v USA a `ARPositionalTrackingConfiguration` sice umožňuje 6DOF kamery, ale Apple ji doporučuje používat spíše v aplikacích pro virtuální realitu [15].

K vykreslení a přidání objektů do scény využívám framework SceneKit. Objekty lze přidávat buď pomocí kotev, které ARKit sám vytváří (případně se dají vytvořit pomocí hit-testů) nebo přímo pomocí specifikování transformace vůči světovému souřadnému systému. Druhá možnost je ta, která mě zajímá. ARKit totiž umožňuje v libovolný okamžik přistoupit k nejnovějšímu snímku videa `ARFrame`, který mimo jiné obsahuje transformaci kamery (uživatelova zařízení) vůči světovým souřadnicím. Spolu se znalostmi ze sekce 2.9 jsem vytvořil funkci, která vytvoří rastrový obrázek letadla (view obsahující informace o letu) a tento obrázek umístí do scény vůči kamerovému souřadnému systému - detailněji rozebráno v příloze B.

Zároveň se v pozadí kontinuálně aktualizuje pozice zařízení pomocí objektu `CLLocationManager`. Vždy když přijde přesnější lokace (proměnná `horizontalAccuracy`) tak tuto novou pozici prohlásím za nový počátek scény pomocí metody `sceneView.session .setWorldOrigin(relativeTransform: translation)`, kde `translation` je aktuální posunutí kamery vůči starému počátku a znovu překreslím všechna letadla ve scéně [42].

Letadlo může uživatel zvolit dvěma způsoby: zvolením ikonky na mapě nebo dotykem na jeho příslušné view. To je realizováno pomocí hit-testingu `sceneView.hitTest(point)` kde `point` je bod dotyku na obrazovce. Výsledkem je node, u kterého si zjistím příslušné ICAO letu a poté letadlo označím jako zvolené.

4.6 Interpolace pozic

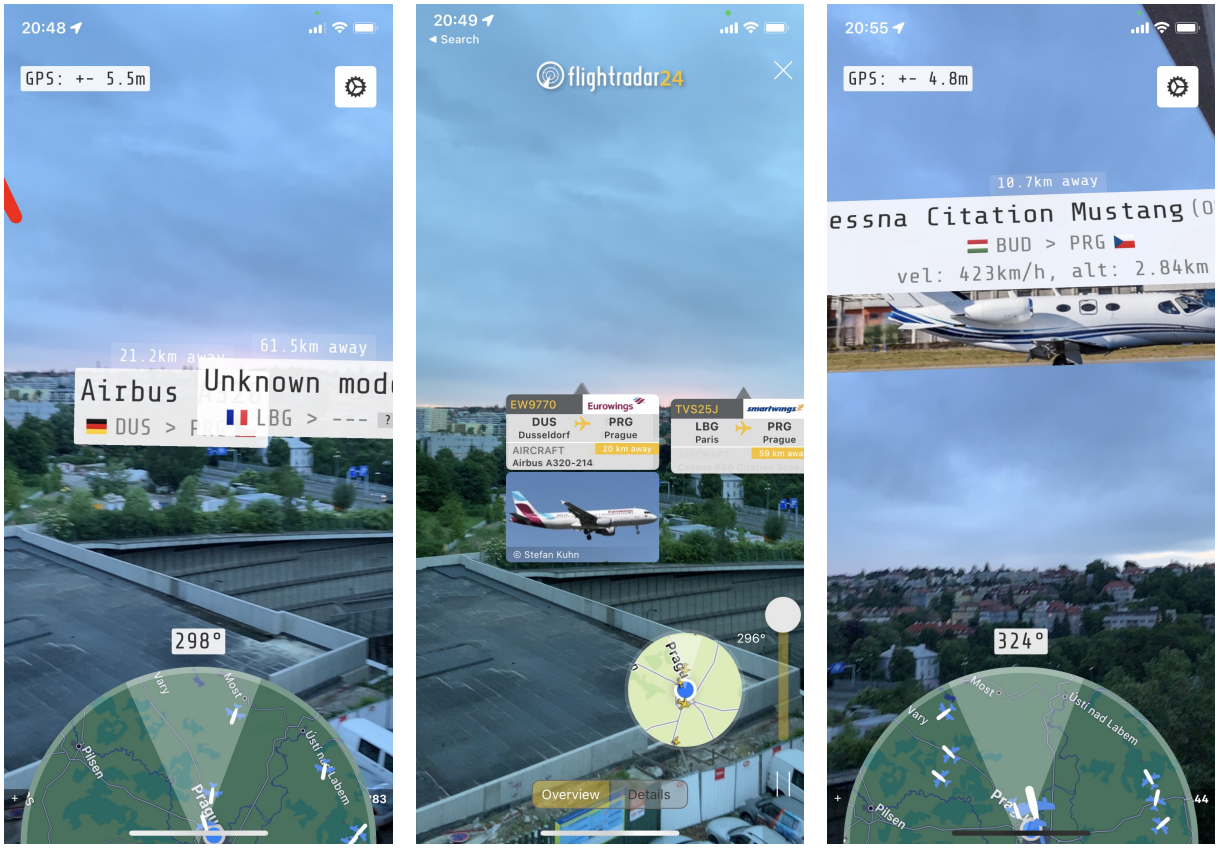
Vzhledem k tomu, že data jsou dostupná s minimálním rozestupem 5 sekund a často i déle, jsou letadla na obrazovce stále na jedné pozici a aplikace působí zamrzlá i když tomu tak není. Napadlo mě, že bych mohl pozice interpolovat, protože k tomu mám dispozici všechna data co potřebuji. Nejjednodušší formě takové interpolace se v navigaci říká "Dead reckoning", kde se odhaduje budoucí pozice jen ze znalosti kurzu, rychlosti a času. Obdobu této interpolace jsem implementoval ve své aplikaci (detaily v příloze C) s časovým krokem $100ms$ a dosáhl jsem kýženého výsledku. Letadla se plynule pohybují na obrazovce a aplikace už působí živě [43].

Taková interpolace je ale pouze aproximací a její chyba s časem narůstá, čili když přijde nová pozice letadla, může se stát, že letadlo ve scéně poskočí z interpolované pozice na tu správnou. Nicméně je to problém se kterým se potýkají i existující aplikace, které jsem testoval a ideální by bylo, kdyby letadla/poskytovatelé posílali pozice častěji.

4.7 Finální stav

Aplikace je v provozním stavu, správně zobrazuje letadla v okolí uživatele - viz. srovnání s aplikací FlightRadar (4.3a, 4.3b) a filtruje je na základě preferované vzdálenosti. Dále zobrazuje mapu s letadly správně natočenými ve směru letu, umožňující selekci a následné zobrazení trajektorie. Aplikace splňuje estetičnost a minimalismus dle navrženého designu (3.1) a zobrazuje informace o letu ve stravitelné podobě, s možností selekce konkrétního letu za účelem získání detailnějších informací (4.3c). Nepovedlo se mi implementovat 3D

trajektorii letu, protože jsem nenašel zdroj dat, který by poskytoval historické pozice včetně výšek a který by byl zdarma.



(a) Pohled skrze mou aplikaci. (b) Pohled přes Flight Radar. (c) Detail letadla v aplikaci.

Obrázek 4.3: Výsledná aplikace

Z časových důvodu jsem bohužel jen částečně stihl implementovat systém pro zobrazování 3D modelů - zasekl jsem se na rotaci modelů ve směru letu se společným pozicováním pod informační view letadla ve scéně. Výsledkem byly dobře umístěné modely, které se ale všechny otáčely na kameru, což není korektní chování a tak jsem se rozhodl tuto funkcionalitu vyřadit, i když jsem systém pro načítání modelů napsal a našel jsem pěkné modely letadel na internetu.

Podcenil jsem komplexitu úkolu filtrování letadel v blízkém okolí (které se překrývají) a nepodařilo se mi z časových důvodů vyřešit detekci těchto případů a následné mapování na jednotlivé úrovně. Algoritmus, který jsem zamýšlel, měl fungovat tak, že bych pro každé view zjistil, kolik dalších sousedů ho zakrývá a přiřadil mu index na základě vzdálenosti od kamery od 1 do počtu zakrývajících sousedů. Následně bych vzal maximum z těchto čísel a tím bych měl definován interval úrovní od 1 do N. Tuto úroveň by pak uživatel mohl měnit posuvníkem dle návrhu (3.1) a pro konkrétní hodnotu by se zobrazily jen letadla dané úrovně. Výsledkem by byla scéna, kde se nepřekrývají žádné objekty.

Kód je dostupný na adrese https://gitlab.fel.cvut.cz/bohmvojt/plane-ar/-/tree/main/. Návod ke spuštění je v příloze D.

Kapitola 5

Testování

Aplikace postupující dle metodiky User Centered Design (UCD) by se měla zaměřit na přímou interakci s uživateli. Základní myšlenka této metodiky je umístit uživatele do středu vývoje skrze zapracování jejich zpětné vazby k produktu. Komunikací s uživateli zvyšujeme šance na to, že jim aplikace skutečně usnadní život, případně přinese zajímavou funkcionalitu, za kterou budou ochotni i platit a nebudeme vyvíjet aplikaci, kterou nikdo nebude používat. Zároveň šetříme čas na vývoj a místo asumpcí postupuje na základě ověřených připomínek. Iterativní cyklus - vývoj, zpětná vazba a vylepšení - se může opakovat mnohokrát, dokud nejsou obě strany spokojeny, případně dokud nedojdou časové či finanční prostředky.

Jednou z technik UCD je uživatelské testování, kdy je uživateli představen testovací scénář, kterým musí v aplikaci projít a vývojář či UX designér sleduje jak s aplikací interaguje, za jak dlouho (jestli vůbec) se mu podaří scénář splnit, všímá si věcí co dělá uživatel jinak a jestli nenastane nějaká nežádoucí chyba. Je dobré testování završit dotazníkem, ve kterém mohou poskytnout dodatečnou zpětnou vazbu a sdělit své pocity [44].

5.1 Testovací scénář

Scénář testuje, zda aplikace splnila svůj fundamentální cíl a je dostatečně intuitivní pro uživatele. Scénář jsem vytvořil následující:

1. Najděte na obloze letadlo které vás zajímá.
2. Najděte to stejné letadlo i v aplikaci a zjistěte o jaký model se jedná.
3. Zjistěte, jakou rychlostí letadlo letí, případně v jaké výšce se nachází.
4. Zobrazte si dráhu letu.

Pro tento scénář jsem 5 testovacích uživatelů, u každého si poznamenal jeho věk, profesi, jestli scénář splnil a svoje pozorování:

Uživatel 1

- Věk: 23.
- Profese: Student softwareového inženýrství.

- Pozorování: Našel letadlo na obloze a spustil aplikaci. Okamžitě zkoušel třepat zařízením a pokoušel se "rozbít"aplikaci. Ta ale rychlé pohyby ustála. Letadlo úspěšně našel, i když bylo lehce posunuté. Ihned se dotkl příslušného view a zobrazil si detail aniž bych mu řekl další kroky. Zobrazil si tedy dodatečné info i dráhu letu. Poté zkoušel klikat na mapu, ale nešlo mu s ní interagovat. Vypadal trochu zklamaně, že mu to mapa neumožňuje.
- Splnil scénář: Ano.

Uživatel 2

- Věk: 29.
- Profese: CEO startupu.
- Pozorování: Lokalizoval letadlo na obloze a namířil na něj telefon. Bohužel se mu nic neukázalo, což je nejspíše následek toho, že OpenSky pro toto letadlo nemělo žádná data. V okolí žádné další viditelné letadlo nebylo, tak si zobrazil to, které nebylo vidět na obloze, ale v aplikaci ano. Dráhu i dodatečné informace si bez problému zobrazil.
- Splnil scénář: Částečně.

Uživatel 3

- Věk: 26.
- Profese: Datový analytik.
- Pozorování: Vybral si letadlo na obloze a pokusil se jej najít v aplikaci. Najít se mu sice povedlo, ale pozice letadla na obrazovce byla zamrzlá několik desítek sekund. Opět je to následek OpenSky a výpadku, před kterým varuje jejich API a může běžně nastat (4.3.1). Detail si zobrazil bez mého pobídnutí, ovšem dráhy si nevšiml, neboť byla v opačné části mapy, která byla zrovna zakrytá kvůli natočení. To bych ale nepovažoval za UX chybu, to je přímý důsledek návrhu. Nicméně by asi bylo dobré mít možnost zobrazit mapu přes celou obrazovku, což mi sdělil i uživatel.
- Splnil scénář: Ano.

Uživatel 4

- Věk: 27.
- Profese: CTO startupu.
- Pozorování: Nalezl letadlo na obloze ale nepovedlo se mu najít v aplikaci ve změti letadel okolo, která nebyla vidět, ale zakrývala to jeho zvolené. Instruoval jsem ho k dalším úkolům na libovolném letadle a ty bez obtíží a mé pomoci splnil.
- Splnil scénář: Částečně.

Uživatel 5

- Věk: 53.
- Profese: Zdravotní sestra.
- Pozorování: Našla si letadlo na obloze, ale nebyla si jistá co má dělat, jelikož předtím AR nikdy neviděla. Po mé instruktáži letadlo na obloze nalezla, ale pozice příliš neodpovídala, mohlo to být v důsledku nepřesnosti kompasu. Nebyla si jistá jak zobrazit detail letu a musel jsem ji navigovat lehkými nápovědami. Obdobně probíhalo i zobrazení letové trajektorie.
- Splnil scénář: Ne.

5.2 Zpětná vazba, dotazník

Je dobré udělat dotazník krátký a výstižný, jelikož uživatelův čas je cenný a je žádoucí, ať je jeho zpětná vazba kvalitní. Když je počet otázek v dotazníku přehnaný, uživatel má tendenci dotazník odbýt, čímž se snižuje jeho relevance. Uživatelům z předchozí sekce (5.1) jsem na závěr testování poslal formulář prostřednictvím aplikace Google Forms s těmito otázkami:

1. Jak dobře bylo vámi zvolené letadlo umístěno v aplikaci na obrazovce? Odpovědi číselně v rozsahu 1 (přesně na pozici letadla) - 5 (úplně mimo).
2. Jak přehledná pro vás byla aplikace? Odpovědi číselně v rozsahu 1 (naprosto přehledná) - 5 (absolutně nepřehledná).
3. Uveďte jednu věc která se vám na aplikaci nejvíce líbila - krátká textová odpověď.
4. Uveďte jednu věc, která se vám na aplikaci nejvíce nelíbila - krátká textová odpověď.
5. Uveďte jednu věc, kterou byste rádi v aplikaci uvítali - krátká textová odpověď.

Výsledky dotazníku jsou uvedeny formou snímků obrazovky v příloze E.

5.3 Reflexe

Uživatelské testování se dle mého názoru povedlo, získal jsem cennou zpětnou vazbu a ověřil jsem si, zda jsem udělal použitelnou aplikaci.

Když jsou dobré podmínky - tedy přesná data, magnetometr a v okolí není mnoho letadel, je aplikace dobře použitelná, i když pozice občas nesedí - zejména výškově. První bod by se dal vyřešit přechodem na jiný zdroj dat, ale nejsem si jist, zda-li bych se nepotýkal se stejným problémem. Druhý bod by se dal mitigovat nějakou instruktážní animací, která ukáže uživateli, jakým způsobem zkalibrovat magnetomer. Třetí bod mě nepřekvapil, už při návrhu jsem věděl, že změť letadel bude problém způsobující nepřehlednost, určitě by stálo za to dodělat posuvník na filtraci letadel (viz. 4.7). A posuny výšek letadel by bylo dobré vyřešit chytřejším pozicováním, které nepromítá objekty na sféru okolo uživatele, ale třeba na zploštěnou sféru. Nicméně sféra byla matematicky i implementačně zvládnutelná,

jiné modely budou vyžadovat mnohem rozsáhlejší rešerši. Design a celkové zpracování aplikace se mi zřejmě povedlo.

Nemyslím si, že aplikace z otevřených dat by měla být monetizována, ale souhlasím s návrhem interaktivní mapy přes celou obrazovku.

Velmi mě překvapilo, že nikdo z uživatelů nešel do nastavení. Předpokládám tedy, že důvěřují aplikaci v tom, že jim zobrazí relevantní data z rozumné vzdálenosti. Taky by to ale mohlo být způsobeno tím, že už šli do aplikace se specifickým úkolem a jen na to neměli čas.

Poslední uživatel si s aplikací nevěděl rady, ale přisuzuji to obecné technické nezdatnosti, která se i pojí s věkem. Zároveň z mých osobních zkušeností jsou nadšenci do letadel za jedno s technikou a věřím že mé cílové skupině aplikace nebude dělat problém.

Kapitola 6

Závěr

Cílem práce bylo navrhnout a vytvořit aplikaci pro zobrazování letadel v reálném čase s využitím rozšířené reality.

V kapitole 2 byla provedena analýza existujících řešení, byla rozebrána funkcionalita AR, visuálních i nevizuálních senzorů mobilních zařízení, technologie ADS-B a byl prozkoumán matematický základ potřebný pro realizaci aplikace, zahrnující geodézii, souřadné systémy a trigonometrii. Kapitola rovněž obsahuje postup (matematickou část) jak přistoupit k implementaci řešení.

V kapitole 3 bylo navrženo UX a UI řešení umožňující vizualizaci letadel v okolí uživatele, dle směru natočení jeho mobilního zařízení. Dále byla navržena logika filtrování letadel na základě těsné blízkosti a intuitivní vizualizace doplňkových dat letu jako je 2D letová trajektorie, cílová destinace či nadmořská výška.

V kapitole 4 byla podrobně rozebrána implementace navrhovaného řešení spolu s přiloženými matematickými podklady a úryvky kódu. Také zde byly vyjmenovány všechny použité technologie, spolu s jejich klady i zápory.

V kapitole 5 bylo postupováno dle metodiky UCD a aplikace byla otestována na testovacím scénáři s uživateli. Od nich byla poté poptána zpětná vazba na základě které se provedla reflexe a zhodnocení práce.

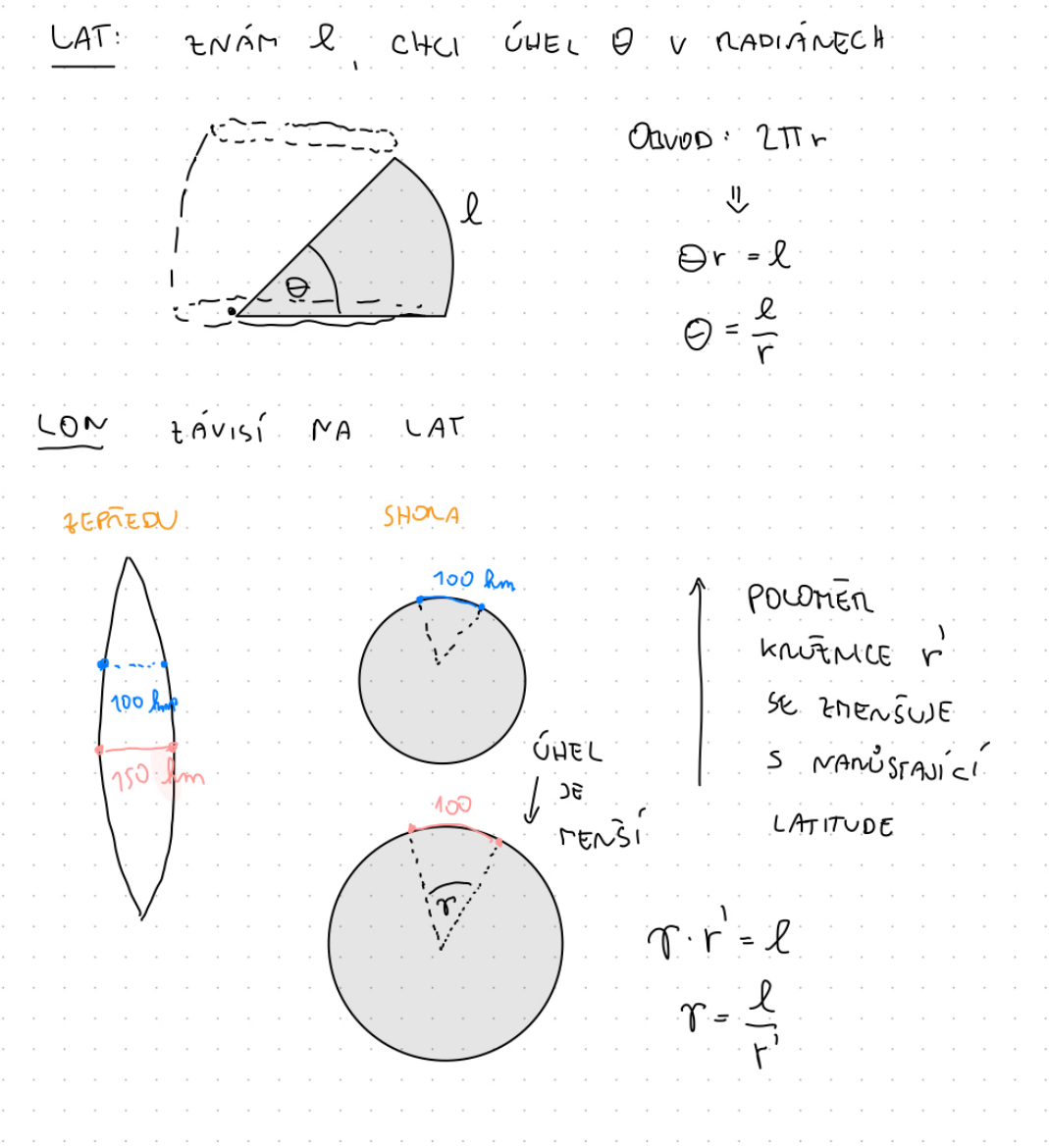
Aplikace aktuálně není v dokonalém stavu, zejména by si zasloužila dokončené filtrování letadel a lepší zdroj dat, nicméně je použitelná, splnila nejen svůj prvotní cíl, ale i ten druhotný - naučil jsem se pracovat s rozšířenou realitou a rozšířil jsem si obzory v oblasti geodézie.

Příloha A

Výpočet hraničního obdélníku

Dle nákresu (A.1) jsem vytvořil funkci, která pro libovolnou zeměpisnou pozici a vzdálenost (velikost) vytvoří příslušný hraniční obdélník:

```
1  static func createBoundingBox(for coordinate: CLLocationCoordinate2D, with meters:
     CLLocationDistance)
2      -> (latMin: CLLocationDegrees, latMax: CLLocationDegrees, lonMin:
     CLLocationDegrees, lonMax: CLLocationDegrees) {
3
4          // we want to take the center coordinate and move in each direction by x km
5          let kilometers = meters / 1000
6          let radius = 6378.0 // average Earth's radius
7
8          // differences between latitudes don't change anywhere on earth so the
     calculation is simple
9          // (a fraction of a circumference of a circle -> theta * radius = arc length)
10         var angleInRad = kilometers / radius // arc length / r = theta
11         let dLat = angleInRad * 180.0 / .pi // difference in latitude by x km in
     degrees
12
13         // longitude difference is a bit harder since it depends on latitude
14         // therefore the radius of the circle for which we compute the degrees
15         // depends on how much we are away from the equator in terms of latitude
16         // specifically, when we are at the equator (latitude is 0) the radius is the
     earth's radius
17         // and when we are at the north poole (latitude is 90) the radius is 0
18         // therefore we will use cos(latitude) * earth's radius
19         let smallCircleRadius = cos(coordinate.latitude * .pi / 180) * radius //
     latitude must be converted to radians
20         angleInRad = kilometers / smallCircleRadius // again, arc length except for
     the small circle
21         let dLon = angleInRad * 180.0 / .pi
22
23         // Now return the bounding box (simple arithmetic)
24         return (
25             latMin: coordinate.latitude - dLat,
26             latMax: coordinate.latitude + dLat,
27             lonMin: coordinate.longitude - dLon,
28             lonMax: coordinate.longitude + dLon
29         )
30
31     }
```



Obrázek A.1: Výpočet hraničního obdélníku.

Příloha B

Pozicování letadel v ARKitu

Pozicování je prováděno v SceneKitu a je rozděleno na dvě částí:

- Vytvoření virtuálního objektu reprezentující letadlo: Pro každé letadlo nejdřív vytvořím v paměti `FlightView`. Pokud je letadlo zvolené, vytvořím detailnější `FlightDetailView`. Poté vytvořím rovinu, jejíž difúzní obsah je rasterizovaná reprezentace vytvořeného view. Na tuto rovinu připnu billboard constraint, které zajistí, že bude vždy otočena na kameru.

- Umístění do scény: dle výpočtů ze sekce 2.9 spočítám translaci letadla vzhledem k pozici uživatelova zařízení. Jediný rozdíl je, že v SceneKitu směřuje nahoru osa y, nikoliv z. Po složení rotační a translační transformace vznikne výsledná transformace, která je předána do proměnné `planeNode.transform` [45].

Problém který jsem řešil při umisťování letadel do scény byl ten, že objekty byly příliš vzdálené od kamery a nebyly vidět. Je to z toho důvodu, že letadla jsou od uživatele vzdálena jednotky až desítky kilometrů což korespondovalo k translačním vektorům s obrovskou délkou. Proto jsem všechny tyto vektory normalizoval tak, ať se promítnou na kouli s poloměrem 100 metrů okolo uživatele. 100 metrů mi z osobních pokusů vycházelo jako vhodná konstanta.

```
1  private func addNode(for flight: Flight, relativeTo locationPoint: CLLocation,
       isSelected: Bool) {
2          guard let camera = sceneView.pointOfView else { return }
3
4          // Generate a view for the aircraft
5          let distance = flight.currentPosition!.haversineDistance(from: locationPoint.
       coordinate)
6          let view: UIView
7          if isSelected {
8              view = FlightDetailView(flight: flight, frame: .zero, distance: distance)
9          } else {
10             view = FlightView(flight: flight, frame: .zero, distance: distance)
11         }
12         view.frame.size = view.intrinsicContentSize
13         let viewScale: CGFloat = 1 / 5 // arbitrary scale that makes the views look
       good on the screen
14         let geometry = SCNPlane(width: view.frame.width * viewScale, height: view.
       frame.height * viewScale)
15         geometry.materials.first?.diffuse.contents = view.image
16         geometry.materials.first?.lightingModel = .constant
17
```

```
18          // Create a node which will host the view and will be placed in the real world
19          let planeNode = SCNNode(geometry: geometry)
20          sceneView.scene.rootNode.addChildNode(planeNode)
21          // Constraint, only axis is the Y one. This makes sure the view will always
    face the camera
22          let constraint = SCNBillboardConstraint()
23          constraint.freeAxes = [.Y]
24          planeNode.constraints = [constraint]
25          planeNode.name = flight.id
26
27          // We bring items that are far away to a circle with this radius:
28          let maxDistance = 100.0 // in meters
29          // Farther points are scaled more
30          let scale = maxDistance / distance
31          var (xDif, yDif, altDif) = locationPoint.displacement(from: flight.
    currentPosition!, at: flight.latestRecord.altitude!)
32
33          if abs(xDif) > maxDistance { xDif *= scale }
34          if abs(yDif) > maxDistance { yDif *= scale }
35          if abs(altDif) > maxDistance { altDif *= scale }
36
37          // When we are positioning the planeNodes, we always need to be sure
38          // we are doing it relative to the true north.
39          // However, the user will generally not face the north, therefore
40          // we need to take the device's heading and rotate the camera
41          // frame by the offset to the north
42          let heading = currentHeading.trueHeading // true north
43          let offset = 360.0 - Float(heading) // in degrees, the offset of the camera's
    device from the north
44
45          let rotation = SCNMatrix4MakeRotation(-offset.toRadians, 0, 1, 0) // and we
    rotate in the opposite direction of the user to shift it back to north
46          let rotatedCameraFrame = SCNMatrix4Mult(camera.transform, rotation)
47
48          // We construct a translation matrix based on the lon/lat/alt changes
49          // Differences in altitude are the shift in the Y axis (* 1.1 to put the view
    slightly above the real position)
50          // xDif is the difference in longitudes (x axis)
51          // yDif is the difference in latitudes, but -1 because the camera is looking
    in the negative z direction (z axis)
52          let translation = SCNMatrix4MakeTranslation(Float(xDif), Float(-altDif) * 1.1,
     Float(-yDif))
53          // Finally, the plane's transform is the translated and rotated current camera
     frame
54          planeNode.transform = SCNMatrix4Mult(translation, rotatedCameraFrame)
55      }
```

Příloha C

Interpolace pozic

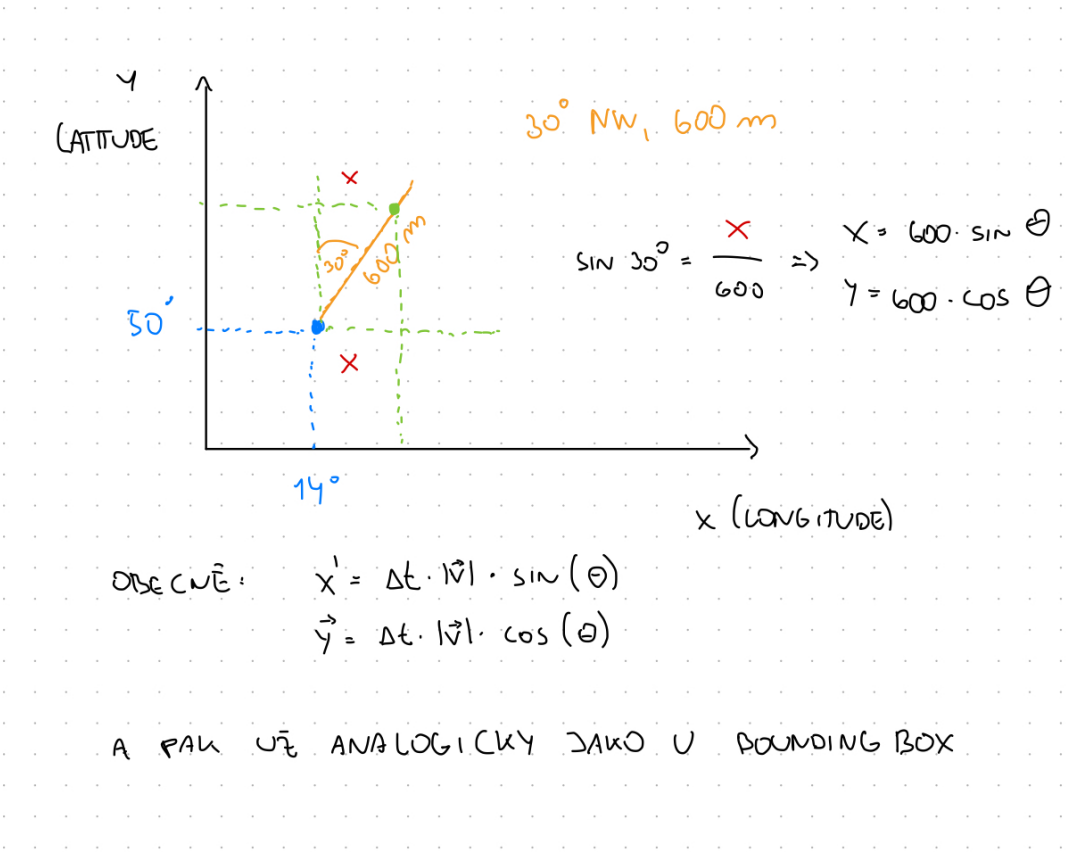
Interpolace se provádí na dvě části:

- Interpolace vertikální pozice. Stačí znát výšku letadla a přičíst k ní uraženou vzdálenost, což je jen rovnoměrný přímočarý pohyb $s = v \cdot \Delta t$, kde v je rychlost stoupání letadla, která je k dispozici v datech a Δt je časový parametr interpolace.
- Interpolace horizontální pozice. Zde jsem použil jednoduchou goniometrii (viz. C.1) pro získání posunu v metrech ve směru zeměpisné šířky i délky. Potom už se postupuje analogicky jako při výpočtu bounding boxu (A), akorát se k pozici letadla přičítá vzdálenost jen v jednom směru.

```
1  private func interpolate(flight: Flight, at deltaTime: TimeInterval) {
2          // We take the position, velocity (speed + heading) and deltaTime to calculate
       new position
3          // The interpolation is broken down into interpolating lat/lon & altitude
4          let newPosition = interpolateHorizontalPosition(for: flight, at: deltaTime)
5          let newAltitude = interpolateVerticalPosition(for: flight, at: deltaTime)
6
7          // Make sure there were some new computations, otherwise just return
8          guard newPosition != nil || newAltitude != nil else { return }
9          // Create new record with the newly interpolated data
10         let newRecord = Record(icao: flight.latestRecord.icao,
11                                callSign: flight.latestRecord.callSign,
12                                lastUpdated: Int(Date().timeIntervalSince1970.rounded()
       ),
13                                longitude: newPosition?.longitude ?? flight.
       latestRecord.longitude,
14                                latitude: newPosition?.latitude ?? flight.latestRecord.
       latitude,
15                                isOnGround: flight.latestRecord.isOnGround,
16                                velocity: flight.latestRecord.velocity,
17                                trueHeading: flight.latestRecord.trueHeading,
18                                verticalRate: flight.latestRecord.verticalRate,
19                                altitude: newAltitude ?? flight.latestRecord.altitude)
20         // Update and inform the delegate
21         flight.updateLatestRecord(with: newRecord)
22         delegate?.didUpdate(flight: flight)
23     }
24
25     private func interpolateHorizontalPosition(for flight: Flight, at deltaTime:
       TimeInterval) -> CLLocationCoordinate2D? {
```

```
26          // The data must be present, otherwise we cannot calculate new position
27          guard let velocity = flight.latestRecord.velocity, let heading = flight.
        latestRecord.trueHeading else { return nil }
28          // Again, based on equirectangular projection, we use trigonometry to
        approximate the new position
29          // we calculate the x (longitude) and y (latitude) offsets in meters based on
        simple trigonometry
30          let distance = velocity * deltaTime // the distance the airplane will travel
        with the velocity at deltatime
31          let xDelta = distance * sin(heading.toRadians) // distance traveled in the x
        direction
32          let yDelta = distance * cos(heading.toRadians) // distance travelled in the y
        direction
33
34          // And now we add these deltas to the current position (meters convert to
        degrees of lat and lon)
35          return flight.currentPosition!.add(offset: (latitude: yDelta, longitude:
        xDelta))
36      }
37
38      private func interpolateVerticalPosition(for flight: Flight, at deltaTime:
        TimeInterval) -> Double? {
39          // If current altitude is not available at all, return nothing, we cannot
        create data out of thin air
40          guard let currentAltitude = flight.latestRecord.altitude else { return nil }
41          // If the vertical rate is not available, just return the current altitude
42          guard let verticalRate = flight.latestRecord.verticalRate else { return
        currentAltitude }
43
44          return currentAltitude + verticalRate * deltaTime // s = v*t :]
45      }
46
47      func add(offset: (latitude: CLLocationDistance, longitude: CLLocationDistance)) ->
         CLLocationCoordinate2D {
48          // The calculation is identical as when creating the bounding box,
49          // but now we just add the offset (in meters) to the coordinate
50
51          let radius = 6378.0 // average Earth's radius
52
53          // differences between latitudes don't change anywhere on earth so the
        calculation is simple
54          // (a fraction of a circumference of a circle -> theta * rad = arc length)
55          var angleInRad = (offset.latitude / 1000) / radius // arc length (offset) / r
        = theta
56          let dLat = angleInRad * 180.0 / .pi // difference in latitude by x km in
        degrees
57
58          // longitute difference is a bit harder since it depends on latitude
59          // therefore the radius of the circle for which we compute the degrees
60          // depends on how much we are away from the equator in terms of latitude
61          // specifically, when we are at the equator (latitude is 0) the radius is the
        earth's radius
62          // and when we are at the north poole (latitude is 90) the radius is 0
63          // therefore we will use cos(latitude) * earth's radius
64          let smallCircleRadius = cos(self.latitude * .pi / 180) * radius // latitude
        must be converted to radians
65          angleInRad = (offset.longitude / 1000) / smallCircleRadius // again, arc
        length except for the small circle
```

```
66          let dLon = angleInRad * 180.0 / .pi
67
68          return CLLocationCoordinate2D(
69              latitude: self.latitude + dLat,
70              longitude: self.longitude + dLon
71          )
72
73      }
```



Obrázek C.1: Interpolace pozice.

Příloha D

Spuštění projektu

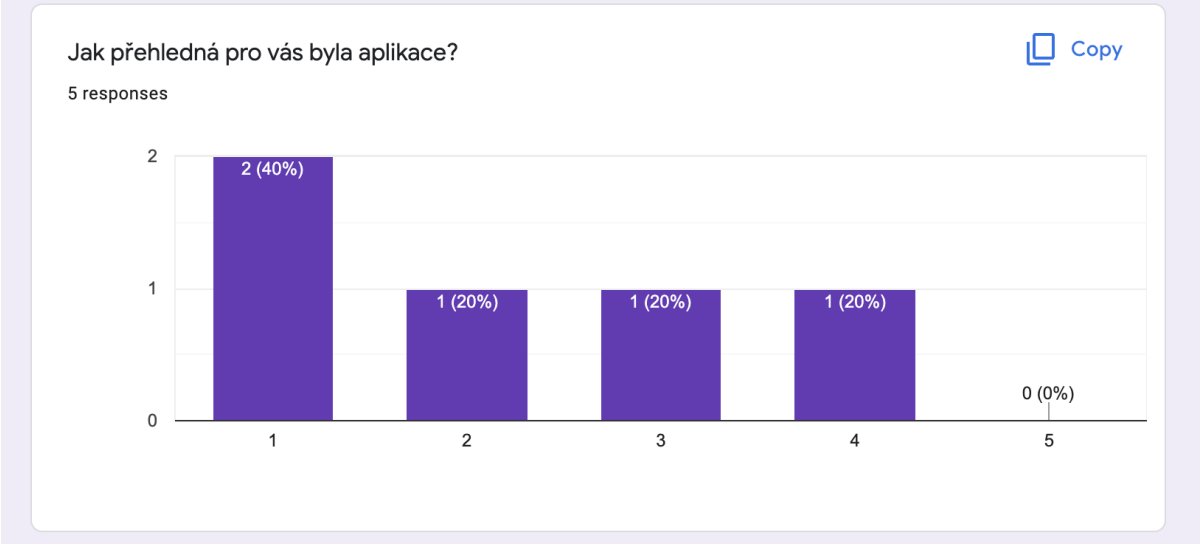
Ke spuštění je třeba počítač s operačním systémem macOS na kterém je nainstalováno IDE Xcode:

1. Stáhněte si projekt z této adresy: https://gitlab.fel.cvut.cz/bohmvojt/plane-ar/-/tree/main/. Buď jako .zip nebo pomocí programu git.
2. Otevřete soubor PlaneAR.xcodeproj v programu Xcode.
3. Vyberte cílové zařízení a zahajte pro něj kompilaci a následné spuštění v levém horním rohu obrazovky, případně pomocí zkratky CMD+R.

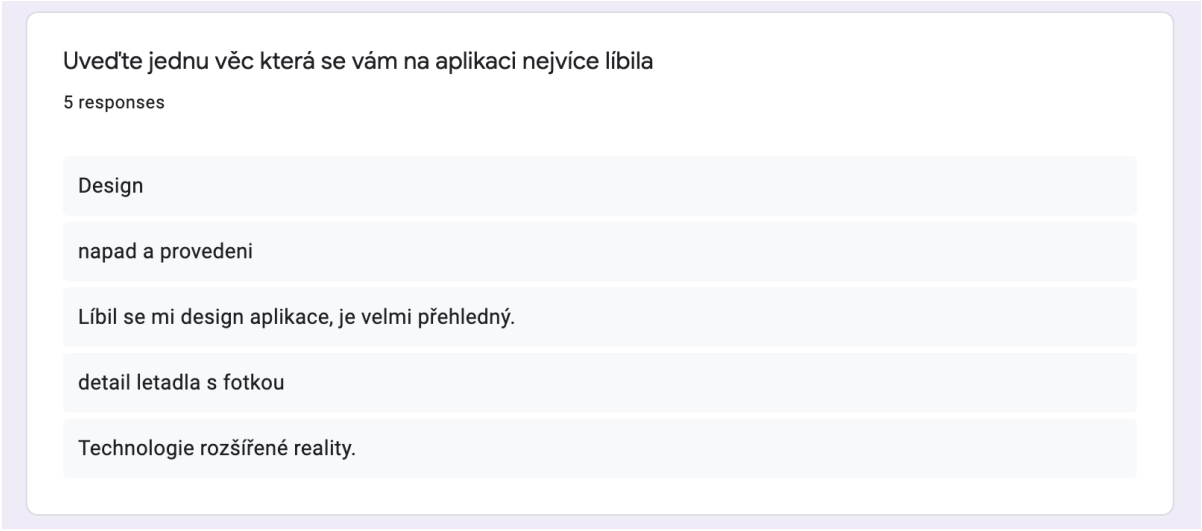
Příloha E

Výsledky dotazníku

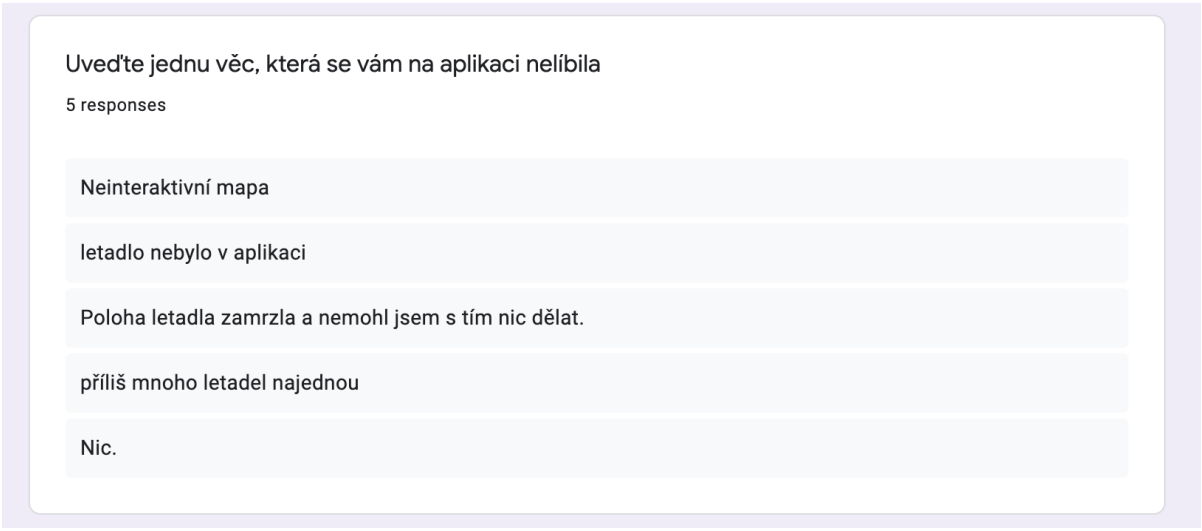
Obrázek E.1: Otázka číslo 1.



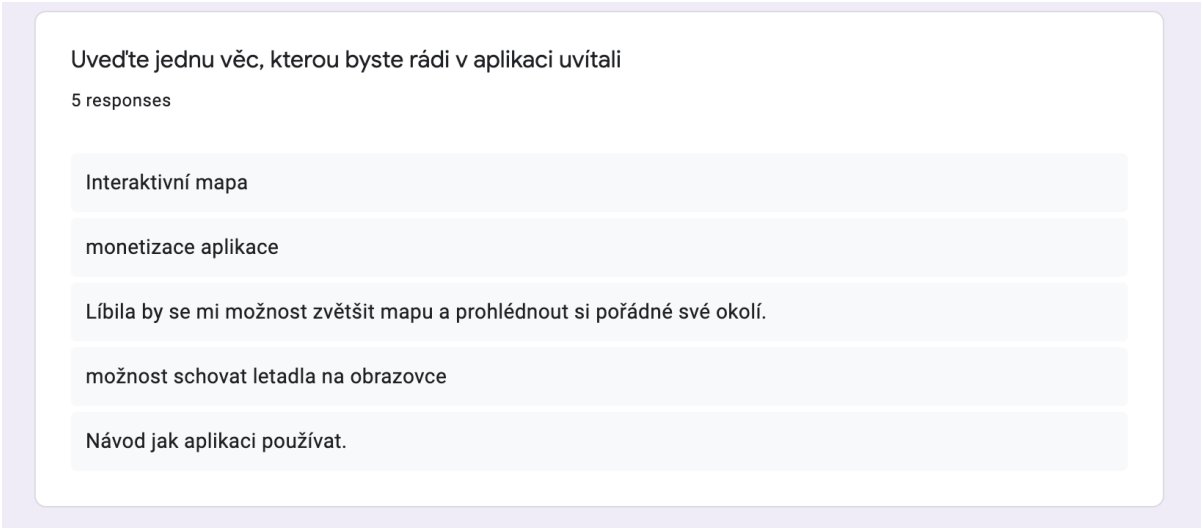
Obrázek E.2: Otázka číslo 2.



Obrázek E.3: Otázka číslo 3.



Obrázek E.4: Otázka číslo 4.



Obrázek E.5: Otázka číslo 5.

Literatura

[1] S. Aukstakalnis, *Practical Augmented Reality: A Guide to the Technologies, Applications, and Human Factors for AR and VR (Usability)*. Addison Wesley, 2017.

[2] [online] https://imageio.forbes.com/specials-images/dam/imageserve/1126094461/960x0.jpg?format=jpg&width=960, navštíveno: 2022-17-05.

[3] D. Schmalstieg a T. Hollerer, *Augmented Reality: Principles and Practice (Usability)*. Addison Wesley, 2016.

[4] Google, *Google Glass*, [online] https://www.google.com/glass/start/, navštíveno: 2022-16-05.

[5] T. T. Mango, *Everything we know about Apple Glass*, [online] https://www.thetealmango.com/technology/apple-glasses-everything-we-know-so-far/, navštíveno: 2022-16-05.

[6] R. Hanssen, *Why Apple Glass will change computing*, [online] https://www.rubenhanssen.com/mixed-reality/why-apple-glass-will-change-computing/, navštíveno: 2022-16-05.

[7] [online] https://lh3.googleusercontent.com/Qvp46ZPEQuOXHwxDeSnR6452OC0-TDGF4TXchQoEBrHSBUZ56KR02DiT-npQ_ivN9xkkUGcYs1y49l9r6-Hklzln2ILZ91-jyIMgOvgiQ=s1440, navštíveno: 2022-17-05.

[8] Boeing, *New Air Traffic Surveillance Technology*, [online] https://www.boeing.com/commercial/aeromagazine/articles/qtr_02_10/pdfs/AERO_Q2-10_article02.pdf, navštíveno: 2022-14-04.

[9] F. R. 24, *How flight tracking works*, [online] https://www.flightradar24.com/how-it-works/, navštíveno: 2022-23-04.

[10] F. A. Administration, *Satellite Navigation - GPS - How It Works*, [online] https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks, navštíveno: 2022-03-04.

[11] [online] https://gisgeography.com/wp-content/uploads/2016/11/GPS-Trilateration-Feature.png, navštíveno: 2022-18-05.

[12] M. Lanham, *Augmented Reality Game Development*. Packt Publishing, 2017.

[13] P. Rapant, *Družicové polohové systémy*. VŠB-TU Ostrava, 2002.

[14] S. M. LaValle, *Virtual Reality*. Cambridge University Press, 2019.

[15] Apple, *Understanding ARKit Tracking and Detection*, [online] https://developer.apple.com/videos/play/wwdc2018/610/, navštíveno: 2022-03-27.

[16] FlightRadar24, *FlightRadar Data Services*, [online] https://www.flightradar24.com/commercial-services/data-services, navštíveno: 2022-03-24.

[17] O. Network, *The OpenSky Network API*, [online] `https://openskynetwork.github.io/opensky-api`, navštíveno: 2022-05-02.

[18] FlightAware, *FlightAware APIs*, [online] `https://flightaware.com/commercial/data/`, navštíveno: 2022-04-29.

[19] RadarBox, *Flight Insights on Demand API*, [online] `https://www.radarbox.com/api/pricing`, navštíveno: 2022-04-29.

[20] [online] `https://www.flightradar24.com/`, navštíveno: 2022-19-05.

[21] J. Velebil, *Abstraktní a konkrétní lineární algebra*. ČVUT, katedra matematiky, 2022.

[22] J. Solà, "Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: a Geometric and Probabilistic Approach", 2017.

[23] W. MathWorld, *Spherical Coordinates*, [online] `https://mathworld.wolfram.com/SphericalCoordinates.html`, navštíveno: 2022-04-12.

[24] [online] `https://www.researchgate.net/profile/Joan-Sola-2/publication/37243878/figure/fig2/AS:669520322654228@1536637492794/A-three-dimensional-Cartesian-coordinate-frame.png`, navštíveno: 2022-04-12.

[25] [online] `https://qph.cf2.quoracdn.net/main-qimg-9d30b330dc430ca6c396-acc0953080cf`, navštíveno: 2022-04-12.

[26] [online] `http://www.jpz.se/Html_filer/wgs_84.html`, navštíveno: 2022-04-12.

[27] [online] `https://www.icao.int/NACC/Documents/Meetings/2014/ECARAIM/REF08-Doc9674.pdf`, navštíveno: 2022-04-12.

[28] [online] `https://www.ordnancesurvey.co.uk/documents/resources/guide-coordi-nate-systems-great-britain.pdf`, navštíveno: 2022-04-13.

[29] [online] `https://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/WGS84_mean_Earth_radius.svg/1024px-WGS84_mean_Earth_radius.svg.png`, navštíveno: 2022-04-12.

[30] [online] `https://kartoweb.itc.nl/geometrics/Bitmaps/geographic-coordinate-system.gif`, navštíveno: 2022-04-12.

[31] U. S. G. Survey, *An Album of Map Projections*, [online] `https://pubs.usgs.gov/pp/1453/report.pdf`, navštíveno: 2022-05-12.

[32] [online] `https://en.wikipedia.org/wiki/File:Mercator_projection_Square.JPG`, navštíveno: 2022-04-20.

[33] [online] `https://en.wikipedia.org/wiki/File:Equirectangular_projection_SW.jpg`, navštíveno: 2022-04-20.

[34] M. T. Scripts, *Calculate distance, bearing and more between Latitude/Longitude points*, [online] `http://www.movable-type.co.uk/scripts/latlong.html`, navštíveno: 2022-05-03.

[35] Mapbox, *Fast geodesic approximations with Cheap Ruler*, [online] `https://blog.mapbox.com/fast-geodesic-approximations-with-cheap-ruler-106f229ad016`, navštíveno: 2022-05-03.

[36] [online] `https://commons.wikimedia.org/wiki/File:Orthodrome_globe.svg`, navštíveno: 2022-05-03.

[37] P. Felkel, *Transformace v OpenGL*, [online] `https://cent.felk.cvut.cz/courses/PGR/lectures/04_Transform_1.pdf`, navštíveno: 2022-05-03.

[38] P. Perea a P. Giner, *UX Design for Mobile*. Packt Publishing, 2017.

[39] Apple, *Technologies*, [online] `https://developer.apple.com/documentation/technologies`, navštíveno: 2022-04-04.

[40] [online] `https://koenig-media.raywenderlich.com/uploads/2019/01/01-MVC-Diagram.png`, navštíveno: 2022-04-13.

[41] ——, *MapKit*, [online] `https://developer.apple.com/documentation/mapkit/`, navštíveno: 2022-05-15.

[42] ——, *CLLocationManager*, [online] `https://developer.apple.com/documentation/corelocation/cllocationmanager`, navštíveno: 2022-04-28.

[43] Skybrary, *Dead Reckoning*, [online] `https://skybrary.aero/articles/dead-reckoning-dr`, navštíveno: 2022-05-12.

[44] T. Lowdermilk, *User-Centered Design*. O'Reilly Media, 2013.

[45] Apple, *SceneKit*, [online] `https://developer.apple.com/documentation/scenekit/`, navštíveno: 2022-05-10.