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

APLIKACE ROZŠÍŘENÉ REALITY: MĚŘENÍ ROZMĚRŮ OBJEKTŮ

APPLICATION OF AUGMENTED REALITY: MEASUREMENT OF OBJECT DIMENSIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MIROSLAV KARÁSEK**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Karásek Miroslav, Bc.**
Program: Informační technologie Obor: Bezpečnost informačních technologií
Název: **Aplikace rozšířené reality: Měření rozměrů objektů**
Application of Augmented Reality: Measurement of Object Dimensions
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou tvorby aplikací rozšířené reality na mobilních telefonech.
2. Prototypujte dílčí prvky uživatelského rozhraní a interakce v rozšířené realitě. Testujte prototypy na uživatelích.
3. Navrhněte inovativní aplikaci využívající rozšířenou realitu pro měření rozměrů předmětů.
4. Implementujte navrženou aplikaci, testujte ji na uživatelích a iterativně ji vylepšujte.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Android Developers: https://developer.android.com/index.html
- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Dieter Schmalstieg, Tobias Hollerer: Augmented Reality: Principles and Practice, ISBN: 978-0321883575

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 a 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Cílem této diplomové práce je navrhnout a implementovat aplikaci pro automatizované měření objektů v rozšířené realitě. Zaměřuje se na automatizaci celého procesu, aby uživatel prováděl co nejmenší počet manuálních akcí. Navržené rozhraní rozděluje měření do několika kroků, ve kterých dává uživateli instrukce pro postup do další fáze. Výsledkem je aplikace pro systém Android s technologií ARCore. Ta je schopná určit minimální rozměry kvádru pro obalení objektu obecného tvaru ležícího na vodorovné podložce. Chyba měření se v závislosti na okolních podmínkách pohybuje v řádu jednotek procent.

Abstract

The goal of this diploma thesis is design and implementation of an application for automated measurement of objects in augmented reality. It focuses on automating the entire process, so that the user carries out the fewest number of manual actions. The proposed interface divides the measurement into several steps in which it gives the user instructions to progress to the next stage. The result is an Android application with ARCore technology. Is capable of determining the minimal bounding box of an object of a general shape lying on a horizontal surface. Measure error depends on ambient conditions and is in units of percent.

Klíčová slova

Rozšířená realita, měření, ARCore, Android, mobilní aplikace, uživatelské rozhraní, odhad polohy, odhad osvětlení, Kotlin.

Keywords

Augumented reality, measure, ARCore, Android, mobile application, user interface, pose estimation, light estimation, Kotlin.

Citace

KARÁSEK, Miroslav. *Aplikace rozšířené reality: Měření rozměrů objektů.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Aplikace rozšířené reality: Měření rozměrů objektů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Karásek
20. května 2019

Poděkování

Rád bych poděkoval vedoucímu práce prof. Ing. Adamu Heroutovi, Ph.D. za odborné vedení a cenné rady při psaní této práce.

Obsah

Kapitola 1

Úvod

Někdy je nutné změřit nějaký objekt, ale není k dispozici metr či pravítko. Pokud postačují přibližné rozměry, pak je lze odhadnout porovnáním s objektem známé velikosti. Tato diplomová práce se zabývá návrhem a implementací aplikace, která umožňuje automaticky změřit rozměry objektu za použití komerčně dostupného mobilního zařízení. Uživateli pak stačí obejít měřený objekt a aplikace sama vypočítá jeho rozměry.

Typické využití takové aplikace je v případě, kdy uživatel potřebuje rozhodnout, jestli se konkrétní objekt vejde do stanoveného prostoru. Například pokud je potřeba rozhodnout, zda se přepravovaná krabice vejde do kufru automobilu, nebo pro ověření, zda rozměr cestovního zavazadla splňuje limit dopravce. Cílem je, aby navržené řešení bylo přesnější než odhad člověka.

Tato práce navrhuje proces měření, který využívá technologii rozšířené reality. Řeší i uživatelské rozhraní, které navržený proces zpřístupní uživateli. Výsledkem je aplikace pro mobilní zařízení se systémem Android, která uživateli umožňuje automatické měření reálných objektů. Aplikace byla v průběhu vývoje testována na skupině uživatelů a na základě pozorování a připomínek iterativně vylepšována.

Technologie pro rozšířenou realitu na mobilních zařízeních poskytují funkce, které propojují virtuální svět s reálným okolím uživatele. Pro svoji funkčnost mapují okolní prostředí mobilního zařízení. Z nasnímaných dat je možné zjišťovat rozměry objektů v okolí mobilního zařízení. Pro systém Android se tato technologie nazývá ARCore a její funkce jsou popsány v následující kapitole.

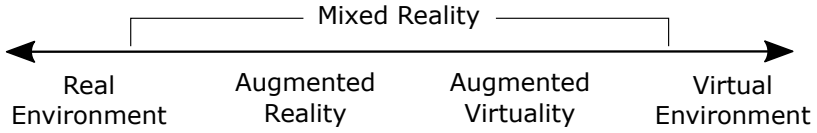
Kapitola 2

Rozšířená realita a technologie ARCore

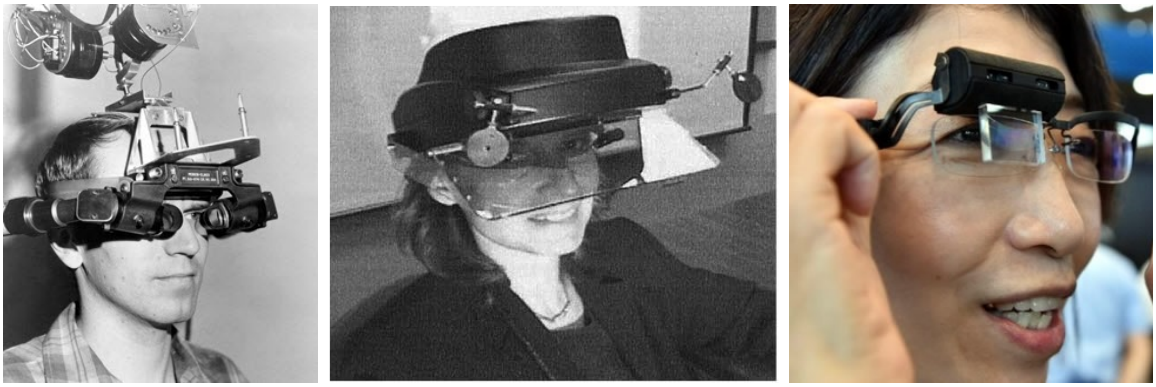
Tato kapitola nejprve definuje pojem rozšířená realita a popisuje, jaké technologie pro zobrazování se zde využívají. Další zmíněnou oblastí jsou technologie pro určování polohy uživatele a rozpoznávání reálného prostředí. Poslední část kapitoly je věnována vybraným technologiím a knihovnám pro práci s rozšířenou realitou. Nejvíce se zabývá knihovnou ARCore pro systém Android. Zaměří se i na technologii Google Tango, jež se dá považovat za předchůdce ARCore. Třetí technologii, o které se kapitola zmiňuje, je ARKit pro mobilní zařízení se systémem iOS.

Systém rozšířené reality [1] kombinuje reálné a virtuální (počítačem generované) objekty v reálném prostředí. Tento systém pracuje interaktivně v reálném čase. Oba typy objektů jsou mezi sebou zarovnány a koexistují ve společném prostoru. Kromě vizuální podoby mohou mít virtuální objekty podobu sluchovou, hmatovou či například čichovou. Mohou být buď tzv. konstruktivní, takové které doplňují reálný svět, nebo naopak tzv. destruktivní, které svým charakterem maskují některé části reálného světa.

V angličtině se můžeme kromě označení *Augmented Reality* setkat s obecnějším pojmenováním *Mixed Reality*. Vztah mezi těmito pojmy vysvětluje obrázek 2.1, který je převzatý z článku pana Milgrama [9]. Mezi reálným a virtuálním prostředím jsou dva stupně pojmenované *Augmented Reality* a *Augmented Virtuality*. Ty jsou souhrnně označovány jako *Mixed Reality*. Tato práce se bude zabývat pouze rozšířenou realitou, tedy pojmem *Augmented Reality*.



Obrázek 2.1: Taxonomie způsobů, jak lze kombinovat reálné a virtuální prostředí. Toto spojení nazýváme jako kombinovanou realitu (*Mixed Reality*). Převzato z [9].



Obrázek 2.2: Vývoj displejů umístěných na hlavu. Zleva: Shuherland 1968 [11], Rolland 2005 [10] a Konica Minolta 2014.

2.1 Technologie zobrazování

Rozlišujeme několik způsobů, jakými se virtuální realita zobrazuje uživateli. Ty se liší tím, jak se spojuje reálné prostředí s virtuálními objekty a umístěním zobrazovacího prvku vůči uživateli. Tato kapitola popisuje tři různé přístupy zobrazování. U každé zmiňuje princip a některé její výhody či nevýhody. Kapitola čerpá z článku Ronadla Azumy a kol. z roku 2001 [2].

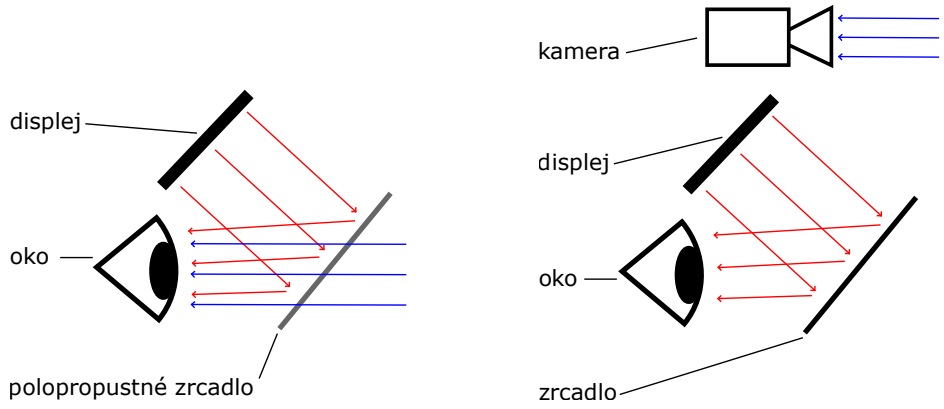
2.1.1 Displej umístěný na hlavu

V angličtině se lze setkat s pojmy *head-mounted display* nebo *head-worn display*, dále jen HMD. Vznik těchto displejů datujeme do roku 1968, kdy pan Sutherland [11] představil první z nich (obrázek 2.2 vlevo). Souhrnně označují technologii, kdy je displej (jeden či více) umístěn na hlavě uživatele a slouží k zobrazování či promítání virtuálních objektů před oči. Musí být zajištěno, aby uživatel viděl současně reálné prostředí a promítané objekty. K tomu lze využít dva základní principy:

- displej s poloprůhledným zrcadlem,
- displej se zrcadlem a kamerou.

První jmenovaný využívá polopropustné zrcadlo umístěné před očima uživatele. Skrz něj proniká obraz reálného světa a virtuální objekty jsou zobrazeny na displeji. Zrcadlo je vhodně natočeno, aby se obraz z displeje odrážel na sítnici člověka. Schéma tohoto principu je na obrázku 2.3 vlevo. Výhodou tohoto systému je, že uživatel neztrácí hloubkovou informaci o okolním prostředí. Prostorové vnímání virtuálních objektů je třeba řešit explicitně pomocí stereoskopie.

Druhý princip snímá reálné prostředí pomocí kamery umístěné v oblasti očí a spojení jejího obrazu s obrazem virtuálních objektů probíhá při vykreslení na displej. Princip znázorňuje obrázek 2.3 vpravo. Nevýhodou tohoto systému je, že snímáním okolního prostředí pomocí kamery ztrácí obraz hloubkovou informaci. Její zachování je třeba řešit explicitně, například využitím dvou kamer (každá pro jedno oko).



Obrázek 2.3: Dva principy využívané u displejů umístěných na hlavu. Ten vlevo využívá polopropustné zrcadlo pro spojení obrazu reálného prostředí a virtuálních objektů. Princip vpravo snímá okolní prostředí pomocí kamery a ke spojení dochází při zobrazení na displej.

Výhodou HMD je soukromí uživatele. Virtuální objekty vidí pouze on a u velmi malých displejů, které jsou například součástí dioptrických brýlí (obrázek 2.2 vpravo[1]), nemusí být na první pohled patrné, že uživatel tuto technologii aktuálně využívá. Je ovšem technologicky náročné displeje a související hardwarové vybavení takto zmenšit a zachovat výpočetní výkon i výdrž baterie. Jednou z cest je oddělit některé komponenty a odlehčit HMD na minimum. Propojení komponent pak může být řešeno kabelem či bezdrátově. To vede na složitější používání a konstrukci celého systému.

2.1.2 Displej mobilního zařízení

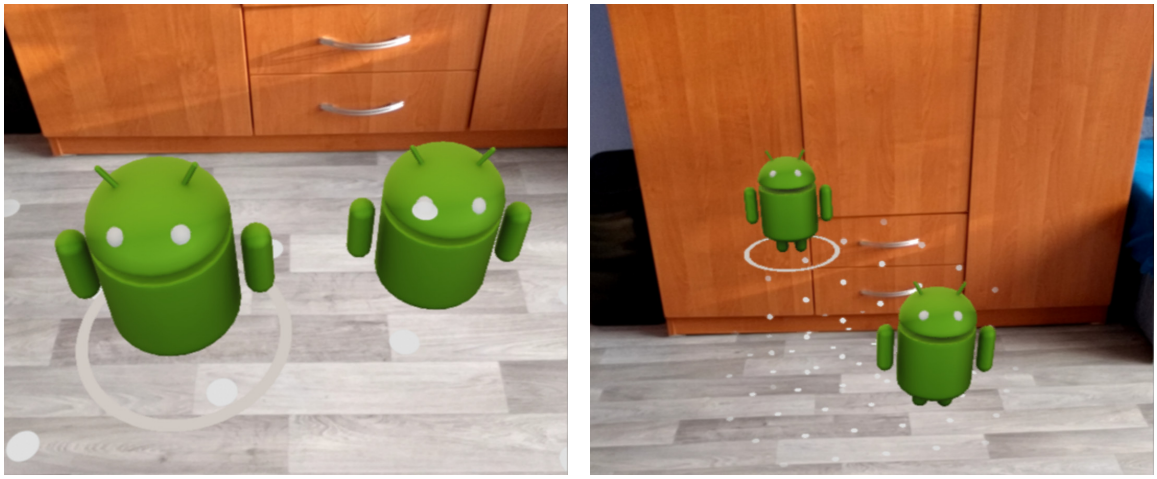
Za mobilní se často označuje takové zařízení, které je uzpůsobeno k nošení u sebe. V kontextu této práce je myšleno zařízení s displejem, se kterým se uživatel může volně pohybovat a při používání jej drží v rukou. Velikost displeje není omezena, ale obecně platí, že příliš malé displeje nejsou pro zobrazování vhodné. Dále je vyžadována kamera, jež snímá část reálného prostředí v závislosti na pozici zařízení. Obraz reálného prostředí a virtuálních objektů je před zobrazením na displej spojen.

Výhodou tohoto principu je dostupnost zařízení, která splňují kladené požadavky. Jako nevýhodu lze brát skutečnost, že je zobrazen pouze dvojrozměrný obraz okolního prostředí a virtuálních objektů. To může být v některých situacích matoucí. Obrázek 2.4 ukazuje příklad situace, u které je kvůli ztrátě hloubkového rozměru nemožné odhadnout vzdálenost a velikost virtuálního objektu.

2.1.3 Zobrazení pomocí projekce

Třetí způsob zobrazení virtuálních objektů ve spojení s reálným prostředím je pomocí projekce. V tomto případě dochází k promítání virtuálních objektů přímo na ty reálné. Využívá se k tomu jeden či více projektorů umístěných v místnosti. Ke spojení obrazu okolí a virtuálních objektů dochází na povrchu reálného prostředí. Výhodou tohoto systému je, že nevyžaduje nošení žádného zařízení na těle uživatele a implicitně umožňuje kooperaci více lidí.

[1] Zdroj: https://www.cdrinfo.com/d7/content/ceatec-2014-konica-minoltas-hologram-glasses-omrons-ping-pong-robot-and-more



Obrázek 2.4: Ukázka ztráty prostorové informace. Vlevo se virtuální objekty jeví jako přibližně stejně velké, ležící na podlaze. Vpravo je stejná scéna z jiného úhlu a je patrné, že objekty nejsou ve stejné výšce. Rozdíl velikosti se vyrovná různou vzdáleností od kamery. Díky tomu vypadají objekty stejně velké.

Nevýhodou je omezené využití této technologie, protože obraz promítaných objektů je typicky dvourozměrný. Toto omezení nevadí například při promítání uživatelského rozhraní (viz obrázek 2.5 vlevo[2]). Další možností je promítat obraz na reálné objekty a změnit tím pouze jejich texturu (viz obrázek 2.5 vpravo[3]).

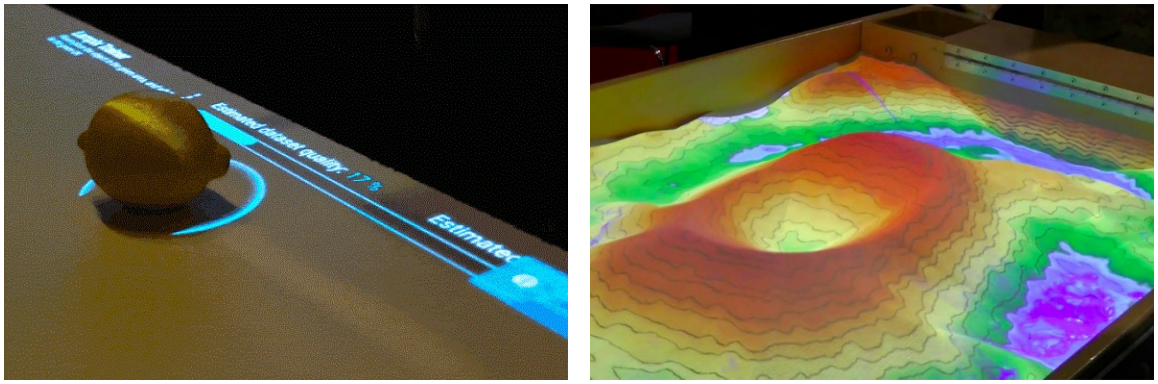
2.2 Sledování pozice a orientace uživatele

Pro fungování systému rozšířené reality je nezbytné určovat relativní polohu a orientaci uživatelů, kteří se v ní pohybují. Některé technologie umožňují implicitně sledovat více uživatelů, avšak ne vždy lze této funkcionality jednoduše dosáhnout. Dalším důležitým faktorem je určování pozice. Je nutné řešit, zda funguje pouze v omezeném, předem připraveném prostoru (například v uzavřené místnosti) nebo je možné určit pozici i v neohraničeném prostoru, tj. mimo budovy. Tato kapitola stručně popisuje rozdělení technologií, které vychází z článku pana You [12] a definuje pro jaké účely jsou vhodné a pro které nikoli. V praxi se pak kombinuje několik technologií, díky čemuž je lokalizace zpřesněna a je zvýšena její spolehlivost.

2.2.1 Aktivní technologie

Aktivní technologie se vyznačuje použitím vysílačů, které jsou umístěny na známých pozicích v reálném prostředí. Pro určování pozice se pak využije přijímač, který měří vzdálenost vůči několika dalším vysílačům a pomocí triangulace určí pozici. Pro určení orientace je třeba více přijímačů v konstantní vzájemné pozici. Signálem můžou být rádiové vlny, světlo či ultrazvuk. Vzdálenost může být určena například na základě doby letu signálu nebo jeho síle. Článek Erica Foxlina z roku 1998 [5] popisuje určování polohy na základě doby letu ultrazvuku.

[2] Zdroj: https://thehightechsociety.com/worldkit/
[3] Zdroj: http://projection-mapping.org/augmented-reality-sandtable-military-planning/



Obrázek 2.5: Ukázka zobrazování virtuálních objektů pomocí projekce. Vlevo jsou promítány prvky uživatelského rozhraní. Vpravo je interaktivní ukázka projekce textury na nepravidelný tvar. Jde o písek, na který jsou dle jeho reálné výšky promítány vrstevnice.

Výhodou tohoto způsobu je přesnost a spolehlivost. Implicitně podporuje sledování více uživatelů či více bodů jednoho uživatele. Nevýhodou je, že systém funguje pouze v omezeném a předem připraveném prostoru, který musí být opatřen velkým počtem systematicky rozmístěných vysílačů. Další nevýhodou aktivních vysílačů je, že pro svůj provoz vyžadují neustálý přísun energie.

2.2.2 Pasivní technologie

Pasivní technologie nevyžaduje vysílače, ale využívá existující signály prostředí. Může jít například o magnetické pole země. Nebo lze do prostředí přidat umělé pasivní signály. Ty bývají často optického charakteru ve formě značek, které lze technikami počítačového vidění efektivně lokalizovat. Výhodou je jednoduchost systému a implicitní podpora více zařízení. Nevýhodou je horší přesnost a u optické lokalizace závislost na kvalitě osvětlení.

2.2.3 Inerciální technologie

Tato technologie využívá senzory zrychlení v jednotlivých osách. Její dvojitou integrací se získává relativní pozice a rotace. Díky tomu může technologie pracovat v jakémkoliv prostředí, bez nutnosti jej předem připravit. Přesnost pozice je výrazně závislá na chybě měření u senzorů zrychlení. Tato chyba se totiž dvojitou integrací promítá kvadraticky do chyby polohy. Další nevýhodou je, že není možné vůči sobě lokalizovat více uživatelů bez jejich explicitní synchronizace.

2.3 Rozpoznávání okolního prostředí

V angličtině se setkáváme s pojmem *environment recognition*. V systému rozšířené reality se využívá pro zarovnání virtuálních objektů s reálným prostředím. To je podle Azumy [2] jedním z požadavků na systém rozšířené reality. V některých situacích je vhodné využít speciální senzor pro tento účel. Jako příklad poslouží hloubkový senzor. Jeho výstupem je hloubková mapa prostředí, která bývá reprezentována point cloudem. Ta lze použít pro detekci ploch, objektů či rekonstrukci scény. Tato kapitola čerpá mimo jiné z knihy *Computer Vision: Algorithms and Applications* [13].

Často lze rozpoznávání okolního prostředí kombinovat se sledováním pozice a orientace uživatele. Například u pasivní i aktivní technologie určování pozice typicky známe absolutní pozici v rámci připraveného prostředí. Pokud známe i reálné prostředí, tak už není třeba jej explicitně rozpoznávat.

Dále existují technologie, jejichž výstupem jsou současně informace o poloze i okolním prostředí. Souhrnně je nazýváme SLAM [4] (*Simultaneous Localization and Mapping*), což v překladu znamená „současná lokalizace a mapování". Zařízení pro rozšířenou realitu je v tomto případě agentem, který mapuje okolí a na základě jeho transformace určuje svoji relativní polohu. Pro potřeby rozšířené reality je nutné, aby SLAM probíhal v reálném čase. Robustní řešení, založené na snímcích z jednoho objektivu a určené přímo pro rozšířenou realitu, popsal pan Liu [8].

2.4 Technologie ARCore na systému Android

Současně používanou technologií na systému Android je ARCore [6]. Ta byla představena v březnu roku 2018. Je vyvíjena společností Google jako nástupce projektu Tango. Následující podkapitoly se zabývají hlavními funkcemi knihovny ARCore. Budou zde popsány hlavní funkce této knihovny a jejich možné využití.

Tuto technologii podporuje Android od verze 7. Kromě požadavků na software je podpora rozšířené reality závislá také na hardware mobilního zařízení. Zařízení musí získat certifikaci od společnosti Google. Při ní je kontrolována kromě výkonu procesoru také kvalita fotoaparátu a přesnost senzorů. Seznam podporovaných zařízení je dostupný na stránkách dokumentace[4].

2.4.1 Sledování pohybu

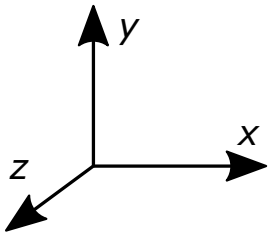
Sledování pohybu (*motion tracking*) využívá relativního určování polohy pomocí kombinace technologií, které byly blíže popsány v kapitole 2.2. Poloha zařízení je důležitá pro zobrazování virtuálních objektů. Ty musí působit dojmem, že jsou umístěny v reálném prostředí, přičemž mají splňovat požadavky na zarovnání reálných a virtuálních objektů.

Z hlediska programátora funguje ARCore na takovém principu, kdy je virtuální scéna (popřípadě graf scény) zarovnána s reálným světem v měřítku 1:1. Jinými slovy, po inicializaci subsystému rozšířené reality je zařízení umístěno na souřadnice (0, 0, 0) v globálním souřadném systému. Pokud zařízení posuneme o 1 metr v ose x, měla by knihovna rozšířené reality určit novou polohu zařízení (1, 0, 0). Je tedy třeba určovat relativní polohu zařízení s co největší přesností po celou dobu běhu aplikace, což znázorňuje obrázek 2.7. Díky této reprezentaci je pro vykreslování virtuálních objektů možné využít jakýkoliv 3d vykreslovací engine. Kamera ve scéně tohoto enginu se musí umisťovat na relativní pozici zařízení ve virtuální realitě. Na obrázku 2.6 je znázorněn souřadný systém, který ARCore využívá. Osa y je svislá a bude využita pro zjišťování výšky předmětů. Při označování os v celé práci budu vycházet z toho obrázku.

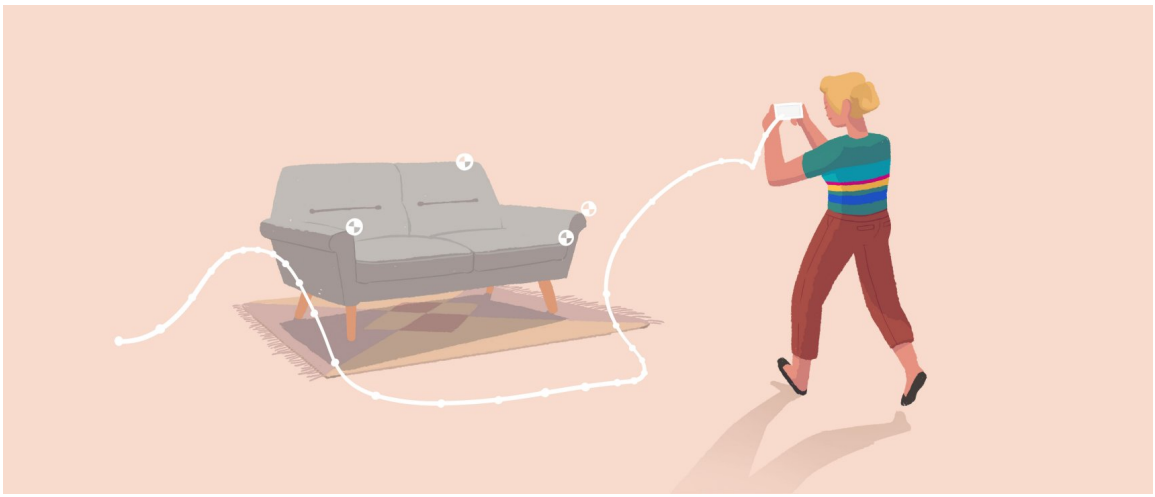
2.4.2 Detekce prostředí

Další funkcí, kterou technologie ARCore řeší je detekce okolního prostředí (*environmental understanding*) a jeho reprezentace v programu. Podporovaná je detekce vodorovných a horizontálních ploch. Pro každou detekovanou plochu je známa její pozice, typ a její velikost.

[4] https://developers.google.com/ar/discover/supported-devices



Obrázek 2.6: Souřadný systém používaný v knihovně ARCore. Důležitá je souřadnice y, která reprezentuje výšku ve scéně.



Obrázek 2.7: Vysvětlení funkce relativního určování polohy zařízení. Bílá křivka znázorňuje trajektorii mobilního zařízení. Kolečka v obraze jsou význačné body, které zařízení pozoruje. Převzato z dokumentace ARCore [6].

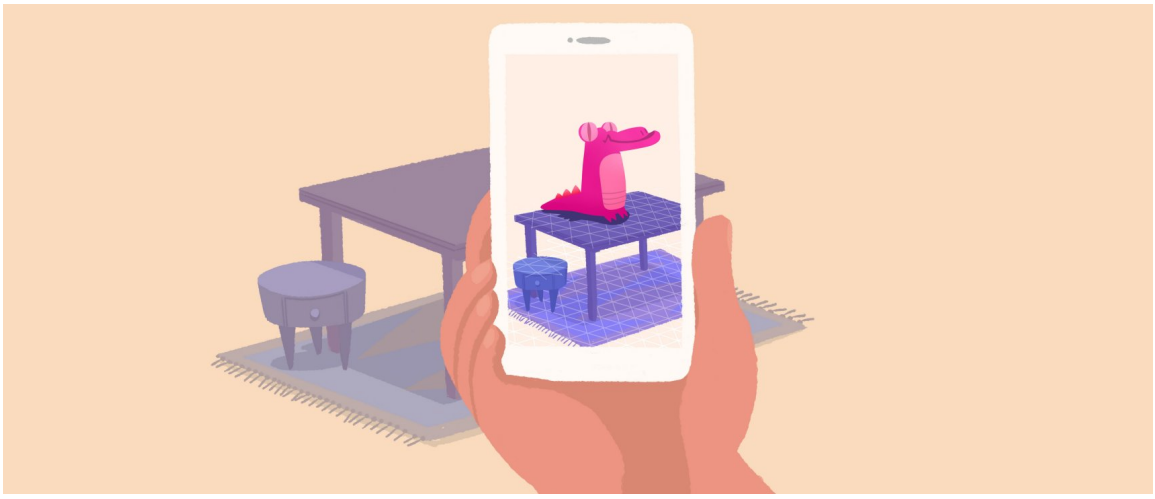
Pozice (třída `Pose`) v ARCore reprezentuje polohu pomocí souřadnic t_x, t_y a t_z. Dále pak rotace pomocí quaternionu q_x, q_y, q_z a q_w. Velikost je určena pomocí šířky w a výšky h.

Detekce je důležitá pro přirozenější umisťování virtuálních objektů do scény, což znázorňuje obrázek 2.8. Pomocí vrhání paprsku je možná interakce uživatele s virtuálními objekty tak, aby se umisťovaly zarovnané s detekovanou podlahou či deskou stolu. Další výhodou detekce ploch je možnost vykreslování stínů k virtuálním objektům. Detekované plochy jsou v tomto případě příjemci stínů. Ty potom vypadají, jako by byly součástí reálného prostředí a napomáhají tak věrnější reprezentaci virtuálních objektů.

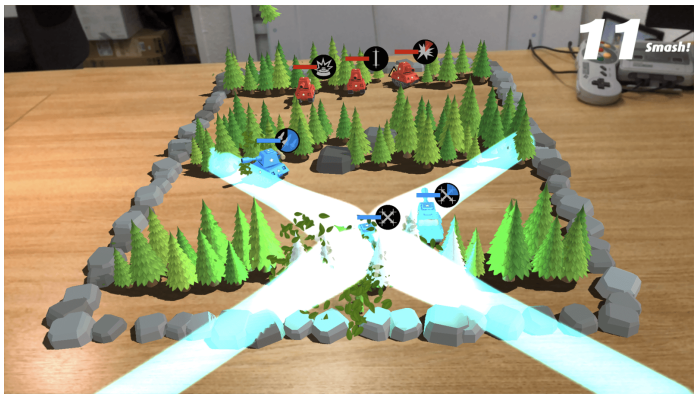
2.4.3 Ukotvení virtuálních objektů

Velmi častým požadavkem programátora je umístění virtuálního objektu na konkrétní místo v reálném prostředí. Obrázek 2.9 znázorňuje hru Smash Tanks![5] s 3d scénou, kterou si hráč pohybem telefonu může libovolně prohlížet a interagovat s ní. V takovém případě je nutné umístit scénu na pevně dané místo do reálného světa. Dalším požadavkem je, aby takto umístěná scéna věrně doplňovala prostředí uživatele. To znamená, aby se herní svět zarovnal s okolním prostředím. K tomu se v ARCore využívá koncept tzv. kotev (*Anchors*).

[5]Dostupné na: https://play.google.com/store/apps/details?id=com.dumpling.smashtanks



Obrázek 2.8: Znázornění detekovaných vodorovných ploch, které jsou zvýrazněny trojúhelníkovou sítí. Ty jsou využity pro umístění virtuálních objektů tak, aby byly zarovnané s reálným prostředím. Převzato z dokumentace ARCore [6].



Obrázek 2.9: Hra Smash Tanks! pro rozšířenou realitu. Jde o ukázku ukotvení herní scény na vodorovnou plochu.

2.4.4 Synchronizace kotev

V rámci platformy Firebase[6] je možné využít funkci Cloud Anchors[7]. Ta umožňuje vzájemnou synchronizaci a kooperaci uvnitř rozšířené reality v rámci stejného fyzického prostředí na více zařízeních současně. Využitelné je to zejména při vývoji her pro více hráčů nebo pro jiné aplikace zaměřené na kooperaci.

Synchronizace vynucuje, aby jednotlivá zařízení odesílala údaje o svém okolí do Firebase. To by mělo zajistit, že server bude mít dostatek informací o okolním prostředí ve chvíli, kdy se do scény umístí kotva. Pokud následně připojíme do Firebase druhé zařízení a umístíme jej do stejného prostředí, pak ARCore rozpozná známé prostředí a umístí kotvu na stejné místo. Komunikaci s Firebase řeší knihovna ARCore sama. Pro programátora je to jen otázka zapnutí této funkcionality a správného nakonfigurování knihovny.

[6] https://firebase.google.com
[7] https://developers.google.com/ar/develop/java/cloud-anchors/overview-android

Obrázek 2.10: Rozdílné osvětlení v různých částech scény pro zvýšení realističnosti vykreslovaných virtuálních objektů. Převzato z dokumentace ARCore [6].

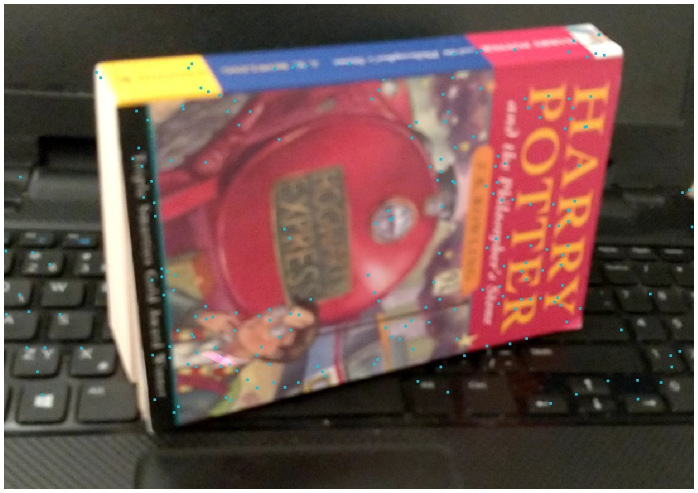
2.4.5 Odhad osvětlení

Pro vykreslování virtuálních objektů je třeba v 3d enginu nastavit osvětlení. To je možné řešit buď manuálně, tj. vložením zdrojů světla do scény, nebo automaticky (anglicky se tato funkce nazývá *light estimation*), což je jedna z funkcionalit ARCore. Výsledkem by mělo být, že v různých částech scény se osvětlovací podmínky blíží podmínkám v realitě. Díky tomu virtuální objekty vypadají ve scéně více realisticky a lépe zapadají do reálného světa. Rozdíl osvětlení v různých částech scény znázorňuje obrázek 2.10. Problematikou automatického odhadu osvětlení z obrazu se zabývá například článek [3], jehož autory jsou Caudell a Mizell.

2.4.6 Získání point cloudu

Point cloud definujeme množinu velkého množství bodů, která bývá výstupem například 3d skeneru. U rozšířené reality jej chápeme také jako detekované body fyzického prostředí, přičemž se jejich počet pohybuje v řádech desítek. V rámci technologie ARCore je možné v každém snímku (třída `Frame`) získat kolekci aktuálně detekovaných bodů. Ta pro každý z nich definuje souřadnice x, y, z a jednoznačný identifikátor id, který slouží k provázání stejných bodů mezi snímky. Příklad vizualizace point cloudu ukazuje obrázek 2.11. Je zde zobrazen jako soubor jednotlivých azurových bodů detekujících převážně blízké objekty.

Point cloud lze využít pro pokročilejší detekci okolního prostředí. Jak již rozebírá kapitola 2.4.2, ARCore provádí pouze detekci horizontálních a vertikálních ploch. Detekci obecných objektů je třeba řešit explicitně a point cloud je vhodný prostředek pro předání informací o okolním prostředí k dalšímu zpracování. Často se využívá pro rekonstrukci tvaru scény do polygonální struktury [14].



Obrázek 2.11: Příklad získaného mračna bodů vizualizovaného jako světlé body v obraze.

2.4.7 Podporované platformy

Knihovna ARCore je primárně určena pro Android. Její rozhraní je dostupné pro jazyk Java (popřípadě Kotlin[8]) a pro C/C++ s využitím Android NDK[9]. Dále existují rozšíření pro herní enginy Unity[10] a Unreal[11]. Zajímavá je částečná podpora na systému iOS. Nejedná se však o náhradu technologie ARKit, nýbrž o její rozšíření, které slouží pro zpřístupnění funkce Cloud Anchors (viz kapitola 2.4.4). To umožňuje vzájemnou synchronizaci objektů rozšířené reality pomocí platformy Firebase mezi zmíněnými operačními systémy.

2.5 Starší technologie – Projekt Tango

Tuto technologii představil Google již v červnu 2014, avšak příliš se nerozšířila. Existují pouze dvě komerční zařízení, která ji podporují. Těmi jsou Lenovo Phab 2 Pro a Asus Zenfone AR. Neúspěch lze připisovat kromě malého počtu zařízení i vysoké pořizovací ceně v době vydání a větším rozměrům telefonů (velikost displeje přes 6 palců).

Projekt Tango pro svoji funkčnost vyžaduje speciální snímače, které se na běžných zařízeních nenacházejí. Pro určování polohy se používá dedikovaný fotoaparát s širokoúhlým objektivem (tzv. *fish eye*). Dalším specializovaným snímačem je hloubkoměr. Pomocí nich určuje knihovna Tango relativní polohu zařízení a umožnuje detekci vodorovných ploch. Díky hloubkoměru má programátor k dispozici velké množství bodů okolí uložených v point cloudu. Ty lze využít k pokročilejší detekci prostředí, protože uchovávají informaci o hloubce obrazu v jednotlivých pixelech. Tyto informace lze použít pro řešení viditelnosti při vykreslování virtuálních objektů. To demonstruje obrázek 2.12, kde vlevo je řešení viditelnosti vypnuté a vpravo zapnuté. Obrázky byly převzaty z aplikace Kiwi.com[12].

[8] https://kotlinlang.org
[9] https://developer.android.com/ndk
[10] https://unity3d.com
[11] https://www.unrealengine.com
[12] Dostupné na: https://play.google.com/store/apps/details?id=com.skypicker.main

Obrázek 2.12: Příklad využití dat z hloubkoměru pro řešení viditelnosti čar virtuálního kvádru. Vlevo je řešení viditelnosti vypnuto a vpravo zapnuto. (Zdroj: aplikace Kiwi.com)

Nevýhodou při programování aplikací bylo, že Google ke knihovně Tango nedodával žádnou podpůrnou knihovnu pro vykreslování scény rozšířené reality. Bylo tedy nutné využít některou z knihoven třetí strany či přímo OpenGL. V březnu 2018 byla kvůli nedostatečnému zájmu ze strany uživatelů i vývojářů podpora této technologie ukončena.

2.6 Srovnání ARCore (Android) s technologií ARKit (iOS)

Obě technologie nabízejí programátorovi podobné funkce. Umožňují sledování zařízení v prostoru, rozpoznávání ploch v okolním prostředí a odhad osvětlení. Z hlediska přesnosti záleží kromě implementace knihovny také na hardwaru samotném. Porovnat pouze knihovny není možné z důvodu nekompatibility obou platforem (nelze je obě zprovoznit na stejném hardwaru). Kvůli tomu lze knihovny srovnat jen velmi laicky, avšak je vypozorováno, že Apple se svým ARKitem je na tom s přesností lépe. To se připisuje především tomu, že má kontrolu nad vývojem softwaru i hardwaru. Z důvodu poměrně malého počtu modelů mobilních zařízení, které firma Apple nabízí, lze software precizně vyladit pro každý z nich. Situace na Androidu je těžší v tom, že ARCore není open source a tedy výrobci mobilních zařízení nemají příliš volnosti pro optimalizaci přesnosti na jimi poskytovaných zařízeních.

Kapitola 3

Návrh aplikace pro měření reálných objektů v rozšířené realitě

Změřit celkové rozměry objektu obecného tvaru je i pro člověka poměrně složitý úkon. Obzvlášť když vezmeme v potaz poměrně nepravidelný tvar, jakým je například hrnek s uchem, který má tělo ve tvaru válce. U takových objektů pak není jednoznačně určeno, které míry by měly reprezentovat jeho celkové rozměry. Někdy stačí určit výšku a průměr. Jindy může být důležitá velikost ucha a pro některé účely je třeba znát i tloušťku stěny, tloušťku ucha, pozici ucha a podobně.

Čím podrobněji zkoumáme daný objekt, tím více rozměrů potřebujeme pro jeho popis. Ty jsou ale závislé na konkrétním objektu. Pokud bychom vzali jiný hrnek, například s tělem ve tvaru komolého kužele, bylo by oproti tomu válcovému potřeba znát i průměry dolní a horní podstavy. Díky tomu je téměř nemožné algoritmicky zjistit a změřit všechny relevantní rozměry. Proto se návrh aplikace omezí na rozměry nejmenšího kvádru, kterým lze tento objekt obalit (*minimal bounding box*).

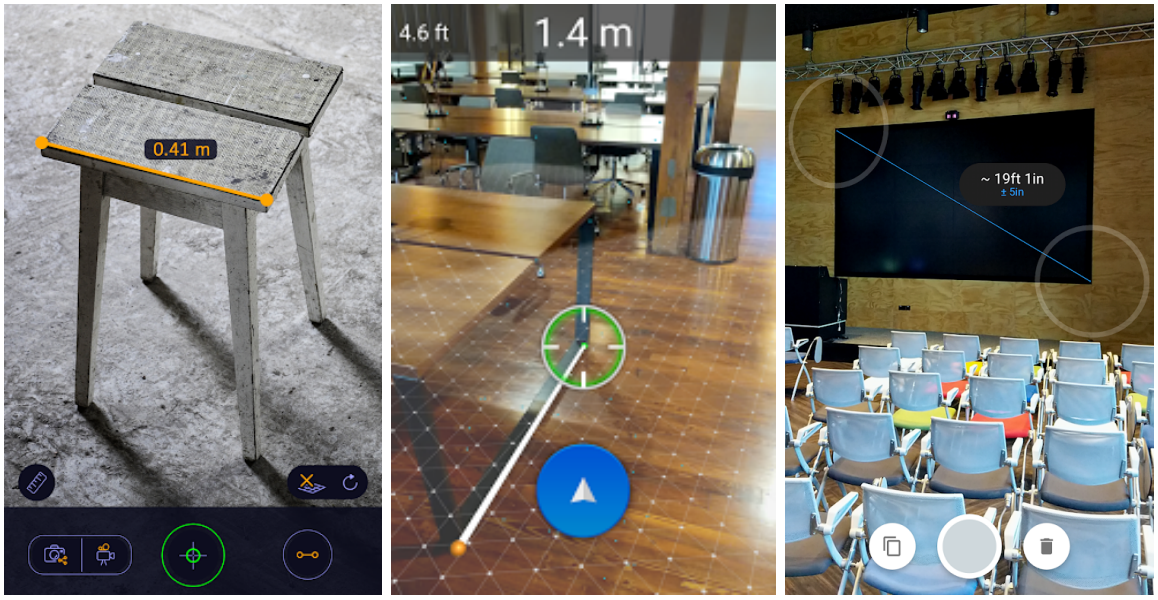
3.1 Analýza existujících aplikací

Před samotným návrhem byly analyzovány existující aplikace, které dokáží měřit pomocí rozšířené reality. Většina jich simuluje použití fyzického měřícího prostředku (svinovací metr, pravítko apod.). Jejich principem je, že uživatel vybere dva body na obrazovce mobilního zařízen, mezi kterými chce měřit vzdálenost. Příklady těchto aplikací jsou na obrázku 3.1, kde vlevo je aplikace AR Ruler App[1], uprostřed Air Measure[2] a vpravo aplikace Measure[3]. Práce má za cíl vylepšit tento koncept a zaměřit se při tom na automatizaci celého měření. Aby například stačilo obejít měřený objekt s mobilním telefonem a ten by sám určil jeho celkové rozměry. Tato kapitola se zabývá návrhem zmíněné technologie a uživatelským rozhraním, které by ji vhodně prezentovalo uživateli.

3.2 Návrh procesu měření

Před návrhem samotné aplikace bylo třeba vymyslet proces, jak budou získávány rozměry objektu. Protože ARCore umí v prostředí detekovat pouze vertikální a horizontální plo-

[1]Dostupné na: https://play.google.com/store/apps/details?id=com.grymala.aruler
[2]Dostupné na: https://play.google.com/store/apps/details?id=com.laan.AirMeasure
[3]Dostupné na: https://play.google.com/store/apps/details?id=com.google.tango.measure



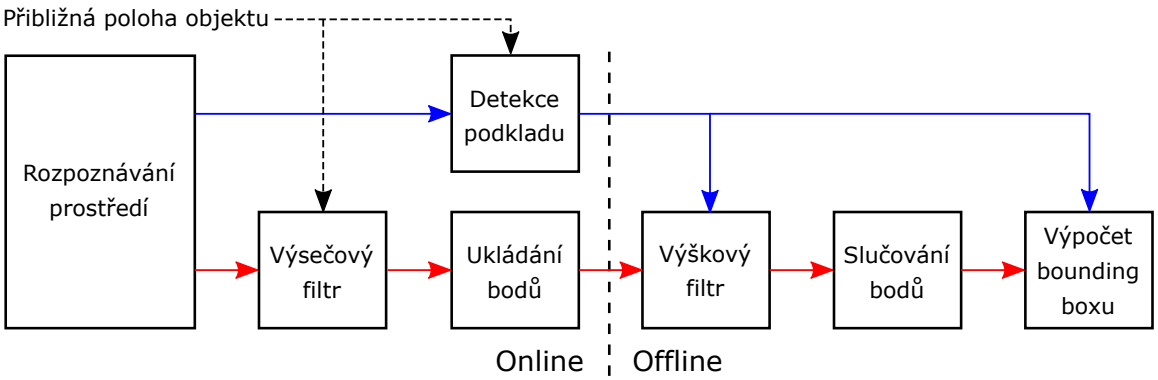
Obrázek 3.1: Existující aplikace pro měření v rozšířené realitě, kde uživatel vybírá dva body a technologie ARCore určí vzdálenost mezi nimi. Vlevo aplikace AR Ruler App, uprostřed Air Measure a vpravo aplikace Measure.

chy, není možné jej použít pro detekci objektů obecného tvaru. Detekci ploch bude možné použít pouze pro rozpoznání podkladu objektu. Pro objekt jako takový bude vhodné použít body z point cloudu. Následující podkapitoly popisují postupné získávání a zpracování dat z knihovny rozšířené reality, které jsem pro tuto práci navrhl. Na obrázku 3.2 je znázorněn mnou navržený proces měření a následující podkapitoly jej postupně popíší. Blok *rozpoznávání prostředí* reprezentuje technologii ARCore, která se využívá pro detekci ploch a získávání point cloudu okolí. Návrh jsem prováděl experimentálně a využíval jsem při tom příklady použití knihovny ARCore[4]. Pomocí nich se daly vyzkoušet možnosti této technologie, díky čemuž návrh nevycházel pouze z teoretických znalostí, ale i z praktických zkušeností.

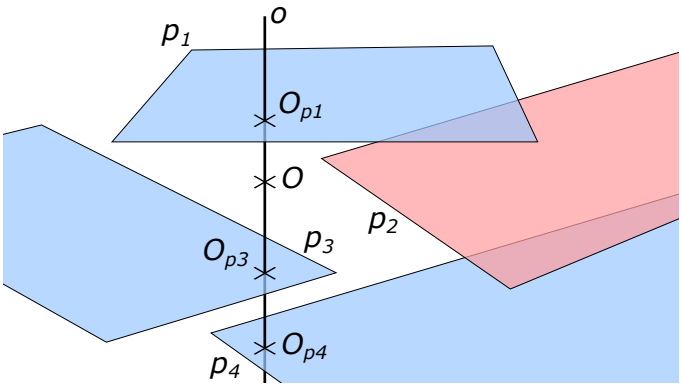
3.2.1 Detekce vodorovného podkladu

Technologie ARCore nabízí funkci detekce vodorovných a svislých ploch v reálném prostředí. Proto jsem se rozhodl ji využít při detekci podkladu měřeného objektu. Experimentálně jsem zjistil, že detekce velké plochy, jako je například podlaha, trvá jen několik sekund. Rychlost závisí především na pohybu zařízení. Nejvhodnější je zařízení nasměrovat na jedno místo a mírně jím kroužit. Přesnost detekce vodorovné plochy je závislá na její vzdálenosti, textuře, intenzitě osvětlení a nejvíce ji ovlivňují okolní předměty. Například při pokusu detekovat desku psacího stolu, kde ležela klávesnice a další drobnosti, se vypočítalo umístění plochy v úrovni výšky klávesnice. Z experimentů provedených za účelem detekce vodorovných ploch plyne, že by měřený objekt měl ležet na rovné, dostatečně velké podložce. Při měření by měly být dobré světelné podmínky a nejvíce se osvědčil povrch s patrnou texturou.

[4] https://github.com/google-ar/arcore-android-sdk/tree/master/samples



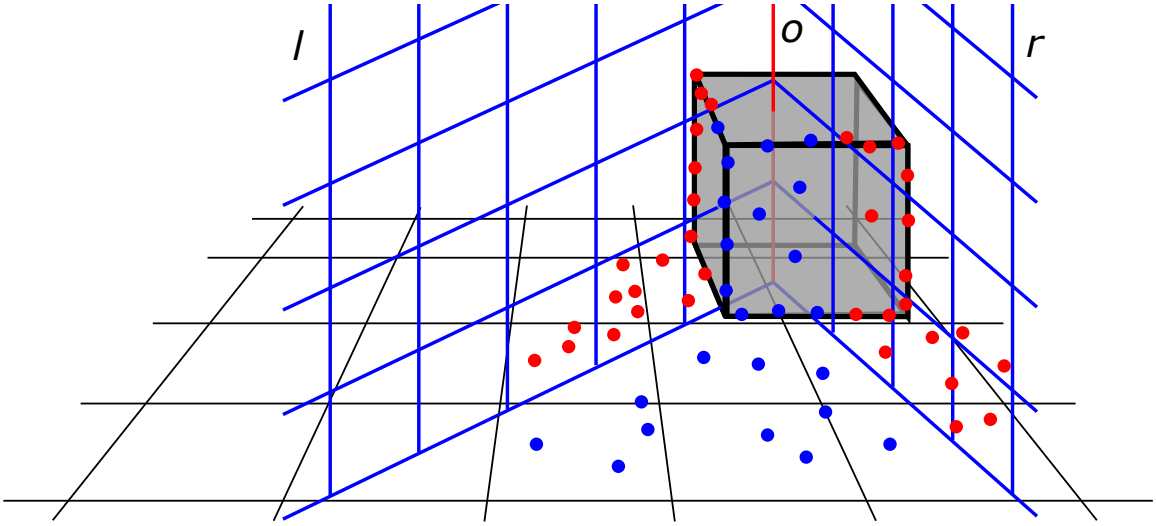
Obrázek 3.2: Vizualizace navrženého procesu měření. Online část se provádí pro každý snímek a offline pouze ve chvíli, kdy je třeba přepočítat rozměry objektu. Modrou barvou je znázorněna větev pro zpracování detekovaných ploch a červená větev zpracovává detekované body.



Obrázek 3.3: Vizualizace výběru podkladu. Bod O je přibližná pozice měřeného objektu, pomocí níž je sestrojena svislá přímka o. Detekované plochy jsou označeny jako p_1 až p_4 a O_{p_1} až O_{p_4} jsou průsečíky mezi danou plochou a přímkou o. Plocha p_2, která neprotíná přímku o, je z výběru vyřazena.

Z detekovaných vodorovných ploch je pak potřebné vybrat pouze jednu, která bude co nejpřesněji kopírovat skutečný podklad měřeného objektu. Při experimentech se ukázalo, že pokud má měřený objekt na horní straně přibližně vodorovnou plochu, tak pravděpodobně bude součástí detekovaných ploch. Dalším problémem je, že větší plochy jsou typicky detekovány jako několik menších.

Navržený postup pro detekci podkladu vyžaduje znalost přibližné pozice měřeného objektu O. Bodem O se sestrojí přímka o rovnoběžná s osou y souřadného systému. Tato přímka je tedy kolmá na detekované vodorovné plochy (p_1 až p_n) a je možné získat jejich vzájemné průsečíky $O_{p_1}, O_{p_2}, \ldots O_{p_m}$, kde O_{p_i} je průsečík s plochou p_i. Platí, že $m \leq n$, protože plochy mají konečnou velikost, takže dojde k průniku jen s některými z nich. Výběr detekovaných ploch se díky tomu zúží pouze na ty, které leží v blízkosti měřeného objektu. Poté lze vybrat plochu s největšími rozměry a tu považovat za podklad měřeného objektu. Obrázek 3.3 ukazuje příklad pro 4 vodorovné plochy. Plocha p_4 je v tomto případě největší a byla by tedy vybrána jako podklad.



Obrázek 3.4: Vizualizace výsečového filtru. Modré body uvnitř výseče jsou zachovány a červené body, které nejsou ve výseči, jsou zahozeny. Červená přímka o reprezentuje osu, která prochází přibližnou polohou měřeného objektu a která je kolmá na podkladovou plochu.

3.2.2 Shromažďování bodů

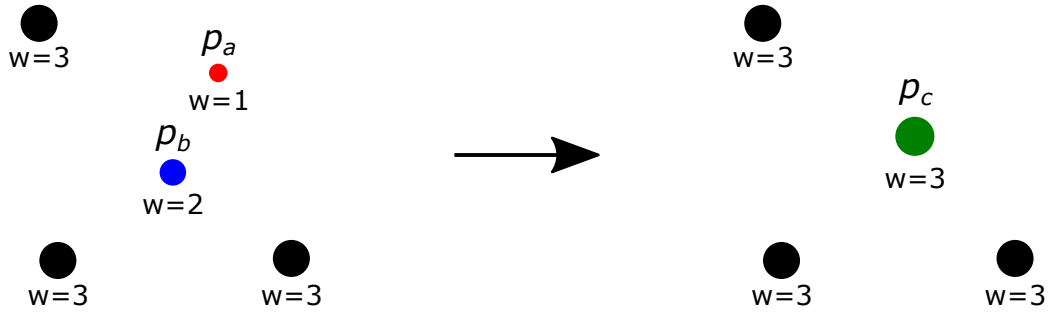
ARCore umožňuje v každém snímku získat množinu detekovaných bodů, což bylo blíže popsáno v kapitole 2.4.6. Dle experimentů závisí počet bodů na osvětlení, členitosti povrchu a nejvíce na pohybu zařízení pomocí kterého jsou snímány. Při nulovém pohybu klesl počet získaných bodů v jednom snímku k nule. Pokud se v průběhu detekce se zařízením hýbalo, pak se jejich počet nacházel řádově v desítkách.

Při shromažďování bodů je třeba pro každý bod rozhodnout, zda je relevantní či ne. Například body patřící jinému než měřenému objektu, by měly být z měření vyloučeny. V navrženém procesu na obrázku 3.2 se o to stará blok *výsečový filtr*. Ten pro svou funkci potřebuje znát přibližnou pozici měřeného objektu O. Pak lze sestrojit přímka o, která je kolmá k podkladové ploše a prochází bodem O. Dále předpokládejme, že známe polohu uživatele U. *Výsečový filtr* pak dělí prostor na dvě výseče. Ty vznikají na základě dvou ploch r a l, které se protínají v ose o. Úhel mezi plochami byl experimentálně zvolen na 90°. Tato výseč je natočená směrem k poloze uživatele U. Přesněji funkci tohoto filtru vysvětluje obrázek 3.4. Modré body v přilehlé výseči budou filtrem přijaty, červené budou zahozeny. Červená přímka znázorňuje osu o.

Aby bylo možné získat body celého měřeného objektu, je třeba jej nasnímat ze všech úhlů. V praxi bude nutné, aby uživatel měřený objekt obešel. Postupným filtrováním a *ukládáním bodů* vznikne point cloud, který bude mít body rovnoměrně rozprostřené po povrchu objektu.

3.2.3 Eliminace nadbytečných bodů

Nasbíraný point cloud zahrnuje i body, které nenáleží měřenému objektu. Typicky to jsou zaznamenané části podložky a šum. Před výpočtem rozměrů měřeného objektu je třeba tyto body eliminovat. Navržený proces nejprve aplikuje tzv. *výškový filtr*. Ten filtruje body (p_1 až p_n) na základě jejich výškové souřadnice y_{p_i} a vyžaduje znalost polohy podkladové plochy



Obrázek 3.5: Příklad jedné iterace slučování bodů. Levý obrázek ukazuje bod s minimální váhou p_a, který by měl být sloučen s nejbližším bodem p_b. Na pravém obrázku je výsledný bod p_c.

měřeného objektu f. Byl stanoven práh t, který určuje nejmenší přípustnou souřadnici y. Filtr tedy zachovává pouze body pro které platí, že $y_{p_i} > t$.

Ukázalo se, že přístup kdy $t = y_f$ neeliminuje všechny nežádoucí body protože nasbírané body podlahy mají poměrně velký rozptyl souřadnice y. Je třeba filtrovat i body kousek nad podkladem. Experimentálně bylo zjištěno, že konstantní posunutí prahu t (například o 5 cm) nefunguje dobře pro širokou škálu měřených objektů. Jinými slovy, malé objekty nemohly být změřeny vůbec a pro velké byla filtrace nedostatečná. Proto jsem navrhl volbu prahu relativně vůči výšce měřeného objektu. Je třeba znát přibližnou výšku objektu h. Tu lze určit z nasbíraných bodů podle vztahu:

$$h = max\{y_{p_i} | 1 \leq i \leq n\} - y_f.$$

Práh t je poté vypočten podle vzorce:

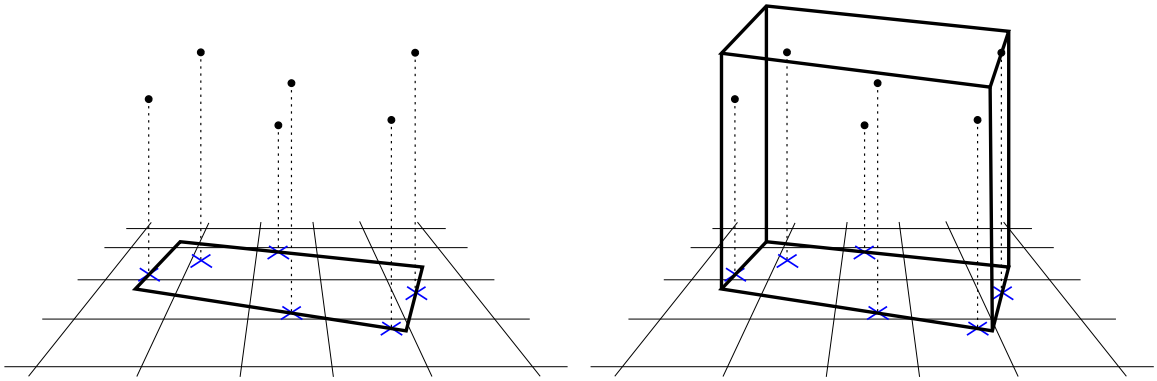
$$t = y_f + k \cdot h \ ,$$

kde k určuje, jak velká část bodů v poměru k velikosti měřeného objektu bude odfiltrována a platí: $0 \leq k \ \leq 1$. Experimentálně jsem zjistil, že hodnota $k = \ 0,3$ spolehlivě odfiltruje podlahu a neovlivní většinu testovaných objektů.

Dalším problémem je, že knihovna ARCore občas chybně detekuje bod někde v prostoru, kde se nenachází měřený, ani jiný objekt. Současně dochází k poměrně velkému rozptylu bodů na povrchu měřeného předmětu. To znamená, že detekované body leží i několik centimetrů od povrchu. Pro vyřešení obou problémů jsem navrhl jejich eliminaci slučováním (na obrázku 3.2 jde o blok *slučování bodů*). Ta každému bodu $p_i \in P$ přidává váhu w_i, která má výchozí hodnotu $w = \ 1$. Dále je stanoven parametr w_{min}, který určuje minimální váhu, kterou musí mít všechny body na výstupu slučování. Algoritmus má iterativní charakter, kde dokud platí podmínka:

$$\exists p : p \in P \wedge w_p < w_{min},$$

tak se provede krok slučování. Ten spočívá ve vybrání bodu s minimální váhou p_a a následném určení jeho nejbližšího souseda p_b. Slučováním vznikne bod p_c, jehož pozice je určena váženým průměrem pozic bodů p_a a p_b. Váha w_c je rovna součtu vah w_a a w_b. Následně jsou z množiny P odebrány body p_a a p_b a místo nich je vložen bod p_c. Obrázek 3.5 ilustruje jeden krok slučování.



Obrázek 3.6: Ukázka výpočtu bounding boxu nad detekovanými body. Vlevo je výpočet 2d boxu za situace, kdy je ignorována souřadnice y. Vpravo je výsledný bounding box i s výškou, určenou na základě maximální souřadnice y všech bodů.

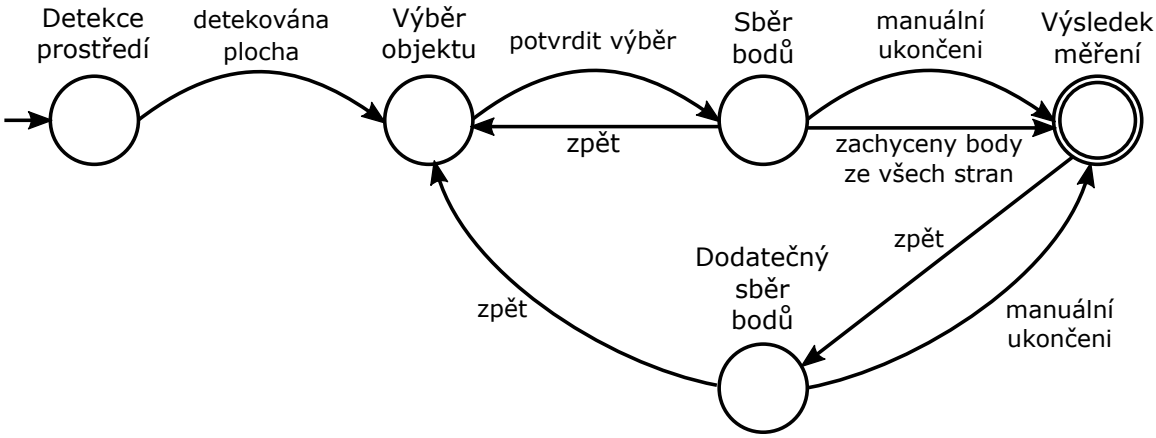
3.2.4 Převod bodů na rozměry objektu

Za předpokladu, že zachytíme body ležící na povrchu měřeného objektu, může být kolem nich vypočten tzv. *minimal bounding box* (dále jen box). Jeho rozměry budou přibližně odpovídat rozměrům měřeného objektu. Protože známe i polohu vodorovné plochy, na které objekt leží, můžeme při výpočtu boxu počítat s tím, že jeho spodní plocha leží na této vodorovné ploše. Díky tomu lze zjednodušit problém výpočtu minimálního bounding boxu ve 3d prostoru na jednodušší problém výpočtu 2d bounding boxu. Toho lze dosáhnout tak, že bude ignorována souřadnice y. Bounding box bude vypočítán jen na základě souřadnic x a z. To ukazuje obrázek 3.6 vlevo. Velikost boxu v ose y bude odspodu omezená podkladovou plochou a shora maximální souřadnicí y vstupního point cloudu. Výsledný bounding box ukazuje obrázek 3.6 vpravo.

3.3 Návrh uživatelského rozhraní

Při návrhu uživatelského rozhraní jsem vycházel z výše navrženého procesu měření. Cílem bylo především celý proces co nejvíce automatizovat, aby uživatel musel dělat co nejméně manuálních zásahů, ale zároveň měl měření pod kontrolou. Návrh probíhal iterativně a na základě zpětné vazby od uživatelů byl vylepšován. Průběžně jsem vyvíjel prototypy aplikace, které implementovaly aktuálně navrženou část uživatelského rozhraní a na základě zpětné vazby od několika nezávislých uživatelů jsem návrh upřesňoval.

Protože navržený proces (obrázek 3.2) má jasně definované závislosti jednotlivých bloků na ostatních, bude při implementaci potřebné respektovat pořadí těchto operací. Zároveň jsem se chtěl vyhnout tomu, abych tento proces pouze transformoval 1:1 na uživatelské rozhraní. Mou snahou bylo to, aby byl proces, prezentovaný uživateli, srozumitelný a logicky členěný do několika fází. Následující kapitoly popíší, jak návrh probíhal a jak byl měřící proces zasazen do uživatelského rozhraní.



Obrázek 3.7: Konečný automat popisující stavy uživatelského rozhraní v průběhu měření.

3.3.1 Fáze měření

V okamžiku inicializace knihovny ARCore je třeba počítat s tím, že zařízení nemá zmapováno okolí. Jako první fázi jsem navrhl inicializaci, při níž aplikace detekuje prostředí (stav *detekce prostředí* na obrázku 3.7). Ve chvíli, kdy bude detekována aspoň jedna vodorovná plocha, lze přejít do druhé fáze.

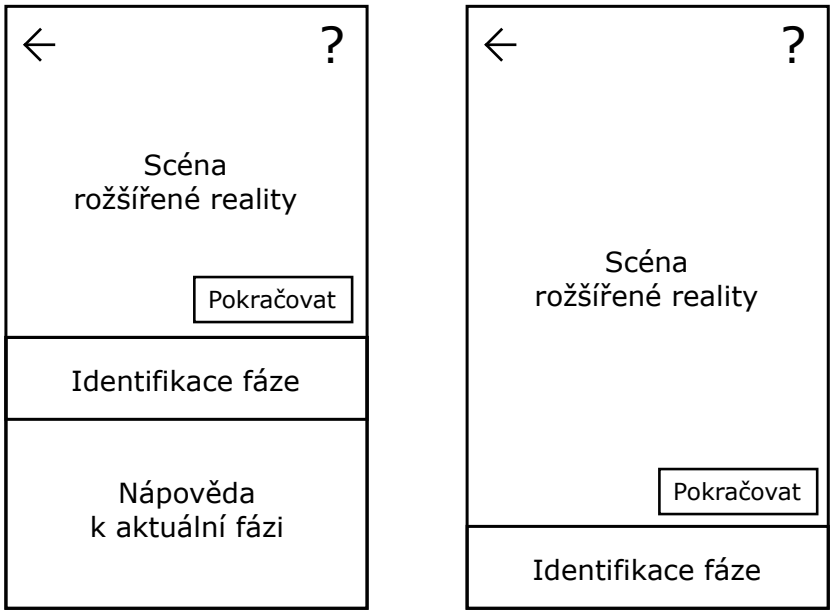
Proces měření vyžaduje znát přibližnou polohu měřeného objektu a zároveň plochu na které leží. Z pohledu uživatele lze tuto informaci zadat současně. Konkrétně jsem pro tento účel navrhl fázi s názvem *výběr objektu*. Její konkrétní podoba bude nastíněna v kapitole 3.3.4.

Třetí fází je *sběr bodů*, který odpovídá online větvi pro zpracování bodů na obrázku 3.2. Tuto fázi bude vhodné automaticky ukončit ve chvíli, kdy bude nasbírán dostatek bodů. Dále musí být uživatel informován o průběhu měření, aby bylo zřejmé, jakou část objektu změřil. Uživatel má možnost manuálně přejít do další fáze a tím ukončit sběr bodů, aniž by musel čekat na automatické ukončení. Díky tomu bude mít tento proces pod kontrolou. Kromě toho se uživatel může vrátit zpět a změnit měřený objekt.

Čtvrtá a poslední fáze uživateli zobrazí *výsledek měření*, odkud se může vrátit zpět k *dodatečnému sběru bodů*. Ten byl z pohledu konečného automatu na obrázku 3.7 navržen jako oddělený stav, avšak z pohledu uživatelského rozhraní se liší pouze tím, že již nemá automatický přechod do poslední fáze. Toto řešení jsem zvolil za předpokladu, že uživatel ví co dělá a automatický přechod na výsledek by mohl jeho snahy znepříjemnit. Dále by fáze *dodatečný sběr bodů* měla průběžně zobrazovat aktualizovaný výsledek měření. Uživatel pak může kdykoliv přejít opět do čtvrté fáze, čímž bude sběr bodů zastaven. Z návrhu těchto fází plynou některé požadavky na ovládací prvky aplikace. Na některých obrazovkách bude potřebné zobrazovat tlačítko pro přechod do další fáze, případně i tlačítko pro návrat k předchozí fázi.

3.3.2 Základní rozvržení aplikace

Při návrhu rozložení aplikace jsem se snažil řídit zvyklostmi na platformě Android. Protože jde o aplikaci s rozšířenou realitou, je třeba nechat co nejvíce prostoru pro vykreslování scény. Aplikace se díky tomu podobá aplikacím typu „fotoaparát“ a proto se bude částečně inspirovat jejich rozložením ovládacích prvků. Od začátku návrhu jsem nejvíce promýšlel,



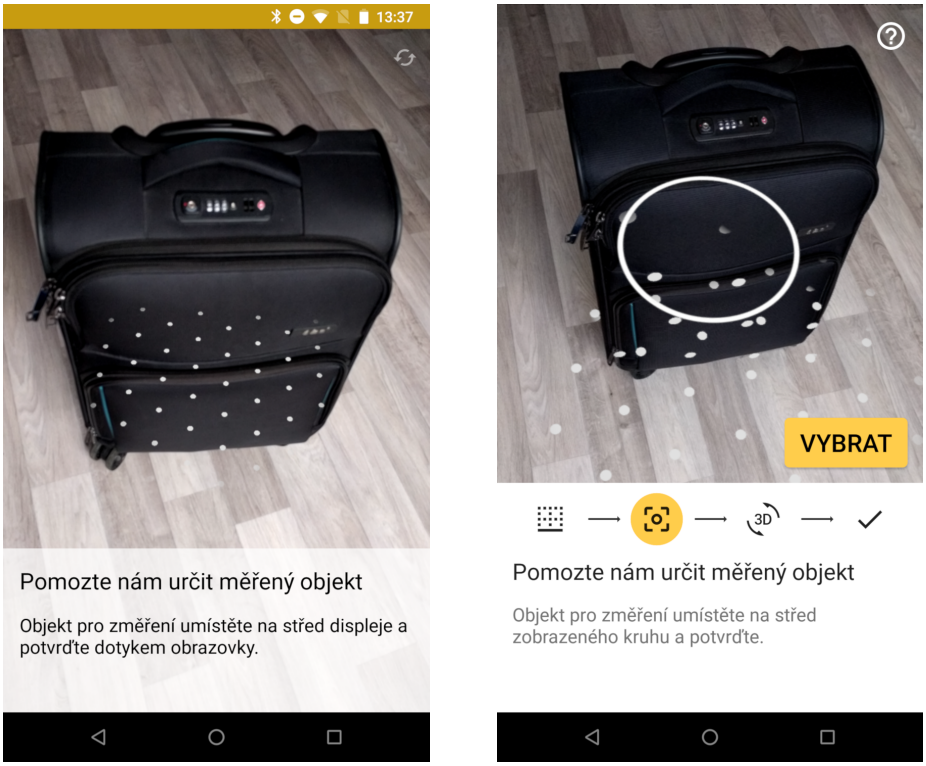
Obrázek 3.8: Základní rozložení aplikace a ovládacích prvků. Levý obrázek znázorňuje rozšířenou variantu s nápovědou k aktuální fázi. Pravá varianta šetří místo skrytím nápovědy.

jak co nejlépe uživatele informovat o krocích, které je nutné udělat pro správné změření požadovaného objektu. První návrh pracoval pouze se stručnými popisky, které uživateli poskytovaly nápovědu, jak se dostane do další fáze.

Pokyny pro uživatele, i přes snahu o jejich stručnost, zabíraly nezanedbatelnou část obrazovky. Tento problém jsem vyřešil tím, že jsem popis fáze rozdělil na dvě části. První z nich slouží pouze k *identifikaci fáze* a druhá obsahuje textovou *nápovědu k aktuální fázi*, jak je vidět na obrázku 3.8 vlevo. Potom bylo možné skrýt rozsáhlejší *nápovědu* v případech, kdy uživatel aplikaci používal již po několikáté (vpravo na obrázku 3.8). Návrh počítal také s tím, že si uživatel může tuto nápovědu podle potřeby zobrazit či skrýt. To by umožňoval symbol otazníku (na obrázku 3.8 vpravo nahoře).

Všechny Android zařízení mají systémové tlačítko „zpět“, které může být hardwarové a nebo dotykové. Kromě toho mohou mít aplikace vlastní tlačítko „zpět“, které bývá v levém horním rohu aplikace. Jejich sémantika je mírně odlišná. To v levém horním rohu se využívá k návratu na předchozí obrazovku. Systémové „zpět“ může provádět i menší kroky. Typicky jde o skrytí klávesnice, zavření dialogu a případně jiné operace „zpět“ v rámci jedné obrazovky. Při návrhu aplikace jsem se rozhodl využít tlačítko „zpět“ v levém horním rohu aplikace, i když fáze měření nejsou oddělené obrazovky. Chtěl jsem tím zdůraznit, že je možné se vrátit o krok zpět.

Některé fáze vyžadují manuální potvrzení akce, které uživatele posune k další fázi. Pro tento účel je součástí návrhu tlačítko v pravém dolním rohu. Na obrázku 3.8 je to rámeček s textem *pokračovat*. Toto tlačítko bude zobrazeno, jen v případě, kdy bude mít uživatel možnost přejít do další fáze manuálně.



Obrázek 3.9: Ukázka prototypů uživatelského rozhraní, kde vlevo jsou aktuální fáze popsány pouze textově. Vpravo jsou přidány ikony pro reprezentaci všech fází měření. Aktuální fáze je podbarvena oranžově. Dále je na pravém obrázku vidět navržený kruh pro výběr objektu.

3.3.3 Vizualizace fází měření

Základní rozložení na obrázku 3.8 neřeší konkrétní rozlišení fází. První prototyp identifikoval fáze pouze podle jejich názvu. To se ukázalo jako nepřehledné. Při průběžném testování nebyl tento způsob pro uživatele příjemný a často si ani nevšiml, že se text změnil. Dále nebylo jasné, kolik fází bylo provedeno a kolik jich ještě zbývá. Proto byl pouhý název fáze nahrazen řadou ikon, kde ke každé fázi byla vybrána ikona, která ji co nejlépe vystihuje. Dále je potřebné, aby aktuálně probíhající fáze byla zvýrazněna. Obrázek 3.9 ukazuje prototypy uživatelského rozhraní pro obě zmíněné varianty.

Pro zdůraznění přechodů mezi jednotlivými fázemi jsem zvolil vibrace, které uživatele upozorní na každou změnu fáze krátkým zavibrováním. Změnu textu nápovědy jsem zdůraznil animovaným odsunem vlevo, kdy se nový text přisune zprava.

3.3.4 Návrh virtuální scény

Důležitým prvkem aplikací s rozšířenou realitou jsou virtuální objekty. Ty lze v podstatě považovat také za část uživatelského rozhraní. Stěžejním prvkem, který je třeba takto vizualizovat, je měřící bounding box. Pro přehledné informování o jeho rozměrech bude vhodné zobrazovat jednotlivé popisky přímo u odpovídajících hran. Navržený bounding box je zobrazen na obrázku 3.10 vlevo.

Pro fázi *výběr objektu* je třeba vizualizovat samotný výběr. Výstupem této fáze by měla být přibližná pozice měřeného objektu a pozice jeho podložky. Pro tento účel jsem navrhl

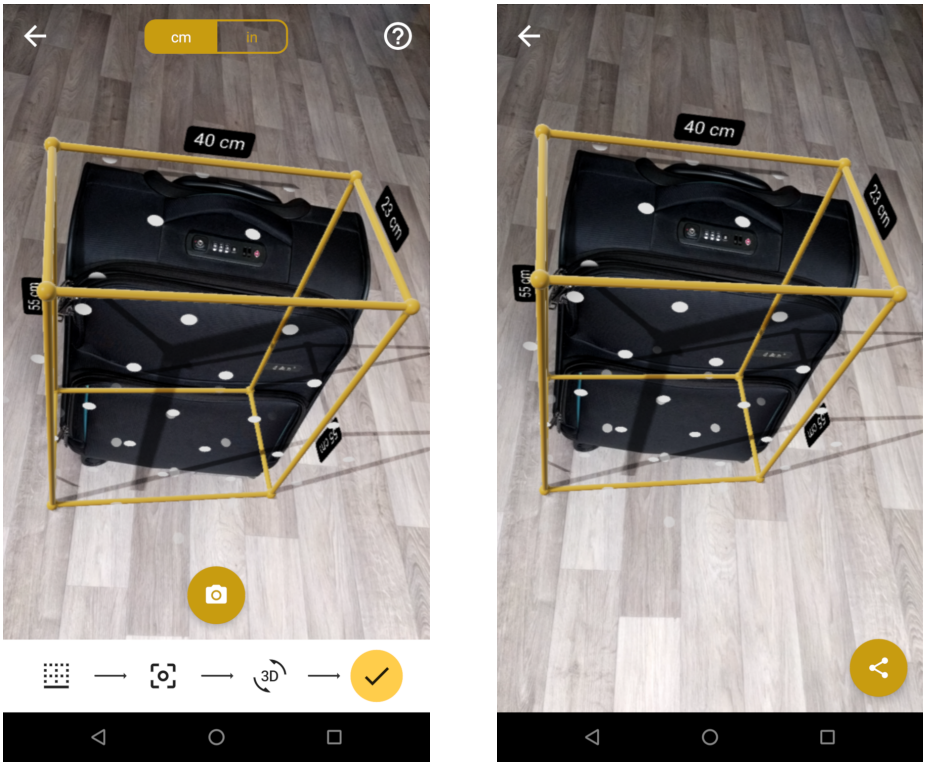
Obrázek 3.10: Navržené vizualizace virtuálních objektů rozšířené reality. Vlevo je navržený měřící box s popisky hran. Vpravo je vizualizace nasbíraných bodů. Kruh pro výběr objektu v tomto případě slouží k vizualizaci průběhu měření.

výběr pomocí kruhu, který je vidět na obrázku 3.9 vpravo. Kruh má konstantní průměr 50 cm. To proto, aby se dala odhadnout hloubková informace a uživatel viděl, zda kruh leží na podlaze, či například na horní vodorovné ploše měřeného objektu. Kruh je automaticky umisťován na střed obrazovky na největší detekovanou plochu. Uživatel tedy mění jeho pozici pouze pohybem mobilního zařízení.

Dále jsem navrhl, že bou zobrazovány i detekované body, které mohou uživateli přiblížit princip, kterým aplikace měří objekty. Uživatel si může kontrolovat, jestli jsou body rovnoměrně rozprostřené po celé ploše objektu a případně podle potřeby lépe naskenovat určitou část. První prototyp aplikace zobrazoval všechny nasbírané body (výsledek bloku *ukládání bodů* na obrázku 3.2). Výhodou tohoto řešení je, že zobrazené body přibývají téměř s každým snímkem. Avšak při testech na větších objektech se frekvence vykreslování snímků, z důvodu obrovského množství bodů, snížila pod únosnou mez a aplikace se stala v praxi nepoužitelnou. Proto jsem se rozhodl zobrazovat jen body, které jsou výstupem algoritmu eliminace. Pro jejich vizualizaci jsem použil koule s poloměrem 1 cm. Jejich výslednou podobu ukazuje obrázek 3.10 vpravo. Je na něm vidět i kruh pro výběr objektu, který je v tomto případě umístěn staticky a oranžová barva na něm vizualizuje průběh měření.

3.3.5 Výběr jednotek rozměrů

Jednou z prvních relevantních připomínek ze strany široké veřejnosti byla absence převodu metrických jednotek na imperiální. První verze zobrazovaly rozměry pouze v centimetrech. Snažil jsem se navrhnout jednoduché přepínaní jednotek, které by zapadalo do rozložení aplikace. První nápad byl určovat jednotky podle lokalizace sytému. Tento přístup mi nebyl mým vedoucím doporučen, protože dost lidí používá zařízení v anglické lokalizaci, ale chtějí metrické jednotky. Proto jsem navrhl manuální přepínání, které je na obrázku 3.11. Viditelnost tohoto přepínače je omezena pouze na fáze, kdy jsou zobrazeny rozměry. Výchozí hodnota přepínače je zvolena podle lokalizace. V případě manuálně změněné hodnoty si aplikace toto nastavení zapamatuje.

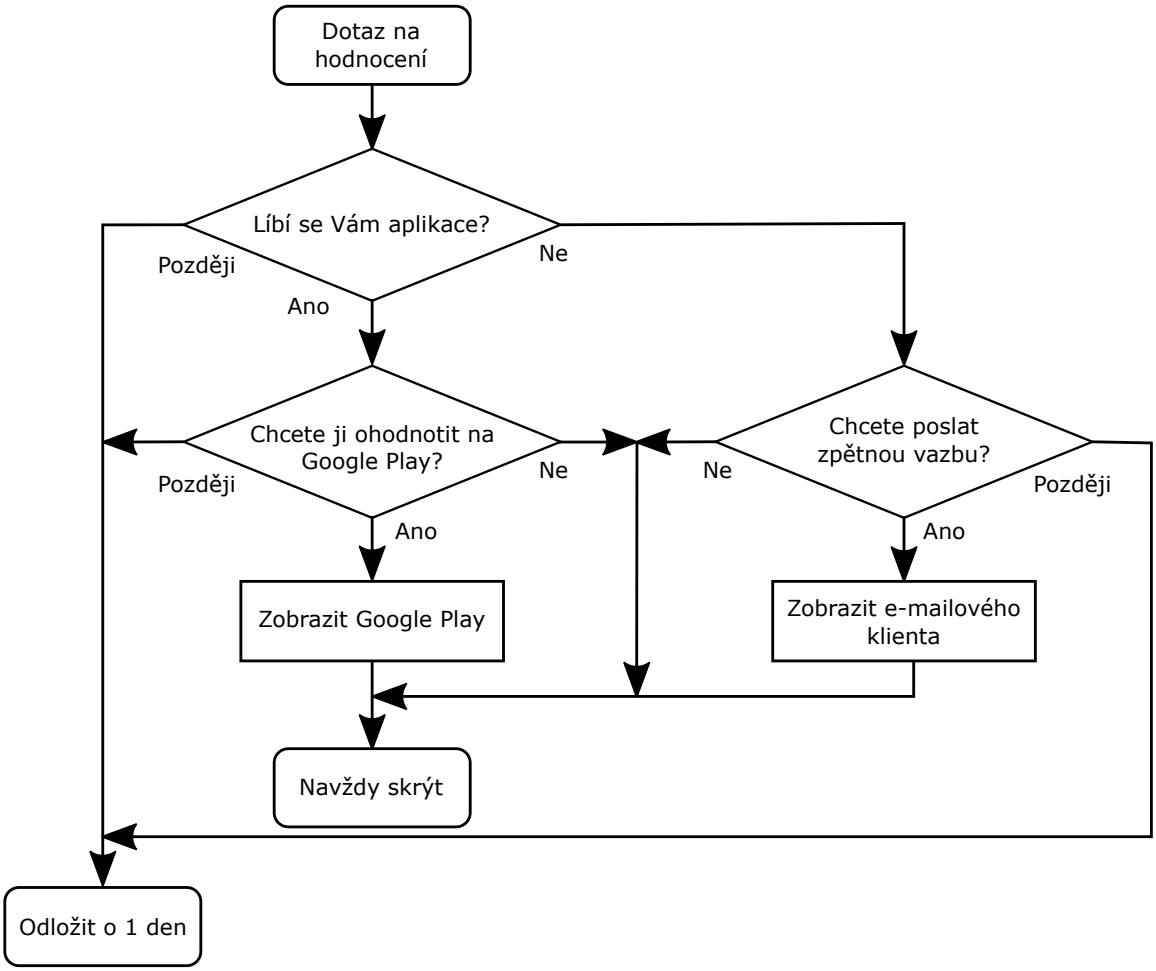


Obrázek 3.11: Obrazovka vlevo ukazuje fázi zobrazení výsledku měření, kde kulaté tlačítko s ikonou fotoaparátu slouží pro zachycení snímku měřeného objektu. Obrazovka vpravo ukazuje již zachycený snímek spolu s tlačítkem pro sdílení.

3.3.6 Uložení a sdílení výsledku měření

Při experimentech se ukázalo, že je užitečné mít možnost pozastavit vykreslování virtuální reality. Například ve chvíli, kdy uživatel změří objekt a potřebuje si rozměry někam zapsat nebo je ukázat kamarádovi. Je sice možné udělat snímek obrazovky, ale to pak vyžaduje přechod do aplikace pro prohlížení obrázků a navíc bude obrázek obsahovat i prvky uživatelského rozhraní, které jsou v tomto případě nadbytečné. Proto jsem navrhl přidání tlačítka pro pořízení snímku, které se zobrazí jen ve fázi *výsledku měření*. Pořízený snímek se pak zobrazí přes celou obrazovku aplikace a bude zobrazovat pouze scénu s virtuálními objekty. Uživatelské prostředí bude ignorováno.

Jako další vylepšení se nabízí mít možnost sdílet zachycený snímek změřeného objektu. Systém Android umožňuje vyvolat dialog, kde si uživatel vybere, v jaké aplikaci chce obrázek otevřít. V něm se typicky zobrazují aplikace pro zasílání zpráv, e-mailoví klienti či aplikace cloudového úložiště (Google Drive, OneDrive atd.). Obrázek 3.11 ukazuje navrhované funkce. Vlevo je vidět fáze zobrazení výsledných rozměrů, kde kulaté tlačítko s ikonou fotoaparátu slouží k zachycení scény. Obrázek vpravo pak ukazuje již zachycený snímek a tlačítko pro sdílení.



Obrázek 3.12: Vývojový diagram pro získání zpětné vazby od uživatele. Každý podmíněný blok znamená z pohledu uživatelského rozhraní systémový dialog.

3.3.7 Výzva k ohodnocení aplikace

Až po vydání aplikace (bude popsáno v kapitole 5) se ukázalo, že uživatelé sami od sebe nehodnotí aplikaci na Google Play. Proto jsem navrhl zobrazování posloupnosti dialogů, která by od uživatele žádali zpětnou vazbu. Nejprve bylo potřeba navrhnout, za jakých podmínek dialog zobrazovat, aby to uživatele neotravovalo a zároveň, aby se dialog zobrazil větší části uživatelů. Navrhl jsem, že uživatel musí nejprve provést 3 měření a teprve poté se mu dialog zobrazí. Jednou z možných akcí by mělo být odložení zpětné vazby na pozdější dobu, aby uživatel mohl dialog rychle zavřít, když by se mu zobrazil například v nevhodnou chvíli.

Navrženou posloupnost dialogů popisuje vývojový diagram na obrázku 3.12. Podmíněný blok diagramu bude uživateli zobrazen jako systémový dialog se třemi volbami. Každý z nich bude mít volbu „Později“, které provede odložení dotazu o jeden den. První dialog se ptá uživatele, jestli se mu aplikace líbí. Při kladné odpovědi bude uživatel požádán o hodnocení na Google Play. Při negativní odpovědi bude požádán o zpětnou vazbu formou e-mailu. Obě žádosti může odmítnout. Ať už uživatel přijme či odmítne zpětnou vazbu, tak se dialog zavře a neměl by být znovu zobrazován.

Kapitola 4

Implementace mobilní aplikace

Aplikace byla vyvíjena již v průběhu návrhu, jak již zmiňuji v předchozí kapitole. Nejdříve šlo o prototypy, které implementovaly jen nějakou konkrétní funkcionalitu a byly určeny k jejímu otestování. Při implementaci jsem se snažil využít moderní technologie, které vedly k jednoduššímu vývoji a pomohly předejít častým chybám, jež se v Android aplikacích vyskytují. U těchto technologií bude stručně popsán jejich účel a budou vyzdvihnuty jejich vhodné vlastnosti. Informace o nich byly čerpány především z oficiální dokumentace, přičemž odkazy na ni jsou k dispozici v poznámkách pod čarou. Tato kapitola také zmiňuje, jakým konkrétním způsobem jsou implementovány navržené funkce a na jaké zajímavé problémy jsem při tom narazil. Mnou implementované řešení je často doplněno o ukázky kódu.

4.1 Programovací jazyk Kotlin

Jako programovací jazyk byl zvolen moderní, silně typovaný a bezpečný jazyk Kotlin, který byl vydán v roce 2016. Od roku 2019 je preferovaným jazykem pro vývoj aplikací pro systém Android. Oproti jazyku Java, který byl v oblasti mobilních aplikací jeho předchůdcem, přináší mnoho pozitivních vlastností. Jeho návrháři se zaměřili především na vylepšení často opakovaných konstrukcí, které bylo třeba psát neustále dokola. I když je Kotlin pokrokovějším jazykem, tak je s Javou interoperabilní a lze je na úrovni tříd využít současně v jednom programu. Oba jazyky lze kompilovat do tzv. *bytecode*, který je interpretován pomocí JVM (*Java Vitual Machine*). Kotlin kromě toho podporuje kompilaci do JavaScriptu nebo přímo do nativního kódu dané platformy.

Konkrétním příkladem, kde Koltin vhodně odstraňuje nadbytečný kód, je testování na hodnotu `null`. Výpis 4.1 obsahuje ukázky kódů se stejnou sémantikou. Nahoře je ukázka v Javě a dole v Kotlinu. Testování na `null` se v Javě musí psát velmi často a zapomenutí může vést ke snaze o přístupu přes `null` referenci a následnému pádu programu.

Kotlin nejenže testování na `null` zjednodušuje, ale také nedovoluje jeho opomenutí. To lze díky tomu, že součástí definice datového typu proměnné, je i informace o tom, zda je proměnná tzv. *nullable* (může nabývat hodnoty `null`). Rozdíl mezi používáním *nullable* a tzv. *non-nullable* proměnné demostruje ukázka 4.2. K bezpečnému přistupování přes *nullable* referenci se používá operátor „`?.`“. Použitím tohoto operátoru pro volání metody `length()`, jejíž návratovým typem je `Int`, pak výsledkem výrazu `proměnná?.length()` je typ `Int?`. Pokud má `proměnná` hodnotu `null`, pak se metoda `length()` nevolá a výsledkem celého výrazu je `null`.

```
// Java
if (promenna != null) {
  promenna.volanaMetoda();
}

// Kotlin
promenna?.volanaMetoda()
```

Výpis 4.1: Ukázka sémanticky stejného kódu v jazycích Java (nahoře) a Kotlin (dole). Demonstruje testování proměnných na hodnotu `null`. Výrazem s využitím operátoru `?.` šetří Kotlin programátorovi práci.

```
var nullablePromenna: String? = "Test"
var nonNullablePromenna: String = "Test"

nullablePromenna.length()           // Skončí chybou překladu
nullablePromenna?.length()          // Lze přeložit
nullablePromenna = null             // Lze přeložit

nonNullablePromenna.length()        // Lze přeložit
nonNullablePromenna?.length()       // Lze přeložit, ale s varováním,
                                    // že operátor ?. není potřeba
nonNullablePromenna = null          // Skončí chybou překladu
```

Výpis 4.2: Ukázka rozdílného použití proměnných tzv. typu *nullable* oproti *non-nullable*. Překladač striktně hlídá jejich použití a nedovolí přeložení potencionálně nebezpečného kódu.

Často je třeba převést *nullable* proměnnou na její *non-nullable* variantu. To se hodí v případech, kdy existuje nějaká výchozí hodnota, která má být použita v případě, že je hodnota `null`. K tomu lze využít operátor „`?:`“ (tzv. „elvis“ operátor). Například máme-li proměnnou `x` typu `String?`, tak potom výraz `x ?: "Výchozí hodnota"` je typu `String`. Podobných vylepšení Kotlin nabízí mnohem více. Například není nutné ručně psát všechny *gettery* a *settery*, které jsou důležité k zapouzdření objektů. Zjednodušuje i přetypování tříd, které dědí od společné třídy. Všechna tato vylepšení a mnohá další jsou popsány v knize *Kotlin In Action* [7], odkud čerpá i tato kapitola.

4.2 Asynchronní zpracování pomocí ReactiveX

Rozhraní s názvem ReactiveX[1] slouží pro asynchronní programování a je formou knihoven dostupné pro různé platformy a programovací jazyky. Pro JVM platformu se tato knihovna nazývá RxJava[2]. Dále existuje rozšíření RxKotlin[3], díky kterému je možné využít všechny výhody Kotlinu. Využívá se zde návrhový vzor *observer* (v češtině známý jako pozorovatel). Z něj jsou převzaty entity `subject` (pozorovaný) a `observer` (posluchač). `Subject`

[1] http://reactivex.io
[2] https://github.com/ReactiveX/RxJava
[3] https://github.com/ReactiveX/RxKotlin

tzv. emituje položky, které může odebírat 0 až N `observerů`. Tento koncept je rozšířený o operátory[4], které mohou položky transformovat, filtrovat a případně je spolu kombinovat.

Velmi užitečnou kategorií operátorů jsou tzv. plánovací operátory. Konkrétně se jedná o `observeOn` a `subscribeOn`. Ty umožňují určit, na jakém vlákně se mají provádět jednotlivé operace. To se hodí například v případě, kdy nějaký operátor provádí složitou operaci a její výsledek má být zobrazen v uživatelském rozhraní. To znamená, že se nesmí provádět na vlákně obsluhující události od uživatele (v Androidu tzv. `mainThread`), protože by to způsobovalo zamrzání aplikace. Zároveň nelze položky uživatelského rozhraní měnit z jiného než hlavního vlákna. Z toho vyplývá, že je nutné spustit výpočet na jiném vlákně a po jeho ukončení ho předat hlavnímu vláknu (`mainThread`). To vše lze udělat za pomoci plánovacích operátorů několika řádky kódu. Konkrétní příklad ukazuje výpis 4.3. Je ze kompletní ukázka metody `uzivatelKliklNaTlacitko()`, která může být vyvolána přímo událostí tlačítka. Ukázka demonstruje provedení metody `slozitеZpracovani()` na výpočetním vlákně (`computation`) a aplikaci výsledku na hlavním vlákně (`mainThread`).

```
fun uzivatelKliklNaTlacitko() {

  val hodnotaVstupu = vstup.text

  Observable.just(hodnotaVstupu)
    .observeOn(Schedulers.computation())        // Vše níže bude provedeno
                                                // na výpočetním vlákně
    .map { hodnotaVstupu ->
       sloziteZpracovani(hodnotaVstupu)
    }
    .observeOn(AndroidSchedulers.mainThread())  // Vše níže bude provedeno
                                                // na hlavním vlákně
    .subscrube { zpracovanaHodnota ->
      vystup.text = zpracovanaHodnota
    }
}
```

Výpis 4.3: Ukázka metody volané na základě akce uživatelského rozhraní, které provádí složité zpracování (metoda `slozitеZpracovani()`) na výpočetním vlákně (`computation`). Její výsledek je pak aplikován na hlavním vlákně (`mainThread`).

4.3 Testovací knihovny JUnit, Mockito a Powermock

Při programování je dobrým zvykem psát unit testy, které slouží pro otestování jednotlivých tříd programu. K tomu na JVM platformě slouží knihovna JUnit[5]. Jejím použitím lze testovat pouze třídy, které nejsou závislé na systému Android. Typickým představitelem tříd, které jsou závislé na Androidu, jsou vlastní prvky uživatelského rozhraní. Takové třídy vyžadují mokování (*mocking*) těchto Android závislostí. To je proces, při kterém se vytváří nová třída, která má stejné rozhraní jako ta kterou mokujeme, ale typicky nedělá žádnou funkcionalitu.

[4] http://reactivex.io/documentation/operators.html
[5] https://junit.org/

K mokování slouží rozšiřující knihovny Mockito[6] a Powermock[7]. Mockito vytváří *mock* tak, že vytvoří třídu, která dědí od mokované třídy a přepíše všechny `public` a `protected` metody prázdnými metodami. Problém nastává u tzv. `final` tříd, tedy u tříd, od kterých nelze dědit. Dále nelze mokovat statické metody či žádné části kódu, které by nešly mokovat manuálně prostředky daného programovacího jazyka. Oproti tomu Powemock využívá složitější techniky mokování a díky tomu jím lze namokovat téměř cokoliv. Jeho funkce je založena na modifikaci programu na úrovni *bytecode*, takže není omezen možnostmi jazyka a omezeními kompilátoru. Nevýhodou je pomalejší vyhodnocení testů, protože modifikace jsou prováděny až při jejich běhu.

Při implementaci některých algoritmů jsem použil přístup s názvem *Test Driven Development* (testováním řízený vývoj), který nejprve definuje vstup a očekávaný výstup implementovaného problému a teprve pak je prováděna samotná implementace, dokud se výstup neshoduje s očekávaným. Tento přístup byl výhodný zejména u operací nad 3d body, kde bylo snadné určit vstup a očekávaný výstup. Naopak otestovat je na reálných datech, které nedeterministicky produkovala knihovna ARCore, bylo téměř nemožné a hledání drobných chyb, například ve znaménku, trvalo hodiny. Přitom napsání funkčního unit testu a následná oprava chyby trvala řádově jednotky minuty. Příklad jednoho konkrétního testu ukazuje výpis 4.4, kde je testováno slučování bodů, které bylo popsáno v kapitole 3.2.3.

```
@Test
fun mergeTwoPointsWithSameWeight() {

  val point1 = Point(
    position = Vector3(1f, 1f, 1f),
    weight = 1
  )

  val point2 = Point(
    position = Vector3(3f, 3f, 3f),
    weight = 1
  )

  val merged = PointsMerge.merge(point1, point2)

  assertEquals(2f, merged.position.x)
  assertEquals(2f, merged.position.y)
  assertEquals(2f, merged.position.z)
  assertEquals(2, merged.weight)
}
```

Výpis 4.4: Ukázka testu slučování dvou bodů. Body mají stejnou váhu, takže se očekává, že souřadnice výsledného bodu budou aritmetickým průměrem souřadnic vstupních bodu a výsledná váha součtem vstupních vah.

[6] https://site.mockito.org
[7] https://powermock.github.io

4.4 Cloudová služba Firebase

Službu Firebase[8] nabízí firma Google programátorům různých platforem. Jde o serverovou platformu, která implementuje funkce typické pro mobilní aplikace, které vyžadují serverovou infrastrukturu. Mezi ně patří autentizace, databáze, cloudové úložiště, vzdálená konfigurace, analýza na základě používání aplikace, odesílání informací o pádech aplikace či například spouštění automatizovaných testů.

Navrhovaná aplikace pro měření objektů funguje bez nutnosti serverové infrastruktury, ale protože je experimentálního charakteru, tak bude užitečné odesílat informace o pádech aplikace. Dále je vhodné odesílat některé události, dle kterých bude patrné, jak s aplikací uživatelé pracují. Jejich analýzou lze odhalit slabá místa implementace a navrhnout zlepšení.

Funkce pro odesílání informací o pádech aplikace se nazývá Crashlytics[9]. Její přidání do aplikace je snadné. Stačí do projektu přidat její knihovnu a následně se již sama stará o odesílání nezachycených výjimek na server. Případně je ukládá lokálně a odešle je až ve chvíli, kdy se zařízení připojí k Internetu. Služba pak v reálném čase posílá e-mailové upozornění na problémy v aplikaci. Součástí reportu jsou informace o typu zařízení, jeho verzi systému a verzi aplikace. Při implementaci jsem narazil pouze na problém, že se posílaly reporty i z právě vyvíjené aplikace, kde se občasným pádům nedalo vyhnout. To ovšem šlo vyřešit snadno tak, že se knihovna importovala pouze v `release` režimu překladu.

Knihovna Analytics[10], která slouží pro odesílání událostí o chování uživatele, řeší veškerou komunikaci se serverem také automaticky. Programátor pouze volá metodu `logEvent()`, které se předává název události a je možné přidat i kolekci doplňujících atributů. Jde o třídu `Bundle`, která se v Androidu používá například pro předávání dat mezi obrazovkami. Data se v ní identifikují klíčem typu `String` a umožňuje uložení dat různých datových typů. Pro posílání událostí jsou podporovány jen některé a to: `Double`, `Long` a `String`. Například typ `Boolean` je interně převeden na celočíselnou hodnotu.

Při implementaci jsem se rozhodl každou událost reprezentovat jednou třídou. Název události bude definován názvem dané třídy a odesílané atributy budou odpovídat jejím atributům. Příklad definice události, která informuje o tom, že uživatel přešel do fáze zobrazující výsledek měření, ukazuje výpis 4.5. Její atributy zahrnují především informace o rozměrech měřeného objektu a atribut `isManualAction` specifikuje, zda se uživatel k výsledku dostal manuální akcí (kliknutím na tlačítko) nebo automaticky. K převodu obecného objektu na `Bundle` využívám reflexi.

```
class MeasureResultPhaseEntered (
  val isManualAction: Boolean = false,
  val xSize: Float = 0f,
  val ySize: Float = 0f,
  val zSize: Float = 0f,
  val volume: Float = 0f
): BaseEvent()
```

Výpis 4.5: Příklad definice události, kdy uživatel přejde do fáze zobrazení výsledků měření. Atributy reprezentují výslednou velikost měřeného objektu a atribut `isManualAction` specifikuje, zda se uživatel k výsledku dostal manuální akcí.

[8] https://firebase.google.com
[9] https://firebase.google.com/docs/crashlytics/
[10] https://firebase.google.com/docs/analytics

4.5 Architektura aplikace a Android ViewModel

Při vytváření Android aplikací je základním stavebním prvkem aktivita (třída, která dědí od třídy `Activity`). Z pohledu uživatele aktivita tvoří jednu obrazovku. Z toho plyne, že každá aplikace, která má uživatelské rozhraní musí mít alespoň jednu aktivitu. Přepínání mezi aktivitami či mezi různými aplikacemi je možné pomocí speciálních systémových volání.

Aktivita tvoří vstupní bod pro komunikaci se systémem. Řídí se tzv. životním cyklem aktivity, který definuje její stavy a akce v návaznosti na různé fáze její existence. Jednotlivé metody životního cyklu jsou provolávány systémem automaticky. Pro stručnost zde uvedu jen ty nejzákladnější. Při vytváření aktivity se volá metoda `onCreate()`. Dále následuje `onStart()`, která se volá před jejím zobrazením. Ve chvíli, kdy je aktivita na popředí, volá se `onResume()`. V případě, že již aktivita není na popředí, například při částečném překrytí aktivity dialogem, se volá metoda `onPause()`. Pokud je aktivita skryta, bude zavoláno `onStop()`. Další metody a popis celého životního cyklu je definován v dokumentaci Androidu[11].

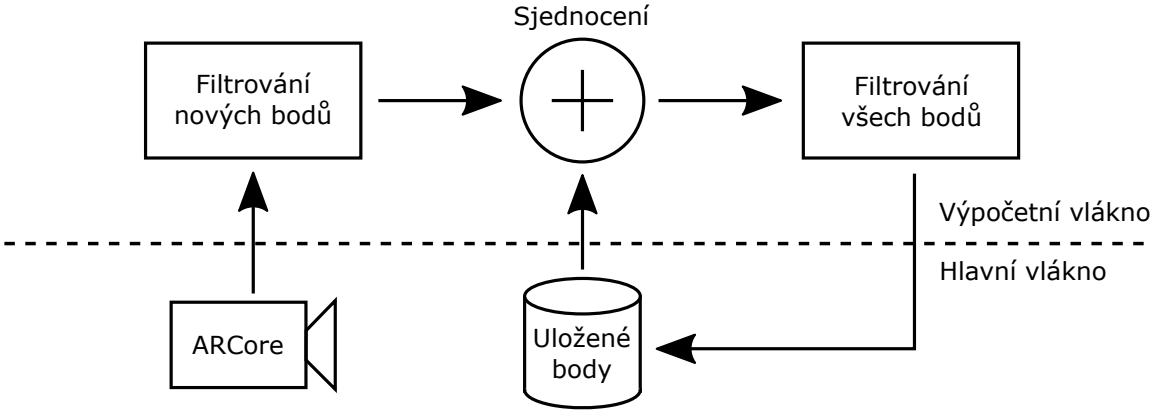
V současnosti se při tvorbě aplikací často využívá architektura MVVM, tedy *Model*, *View*, *ViewModel*. Aktivity spadají do sekce *View*, kam patří i další třídy určené pro zobrazování. Sekce *Model* se stará o získávání a uchovávání dat. Pokud k návrhu používáme ER diagram, tak mezi třídy modelu patří i navržené entity. *ViewModel* tyto dvě části propojuje. Jeho součástí je především logika pro zobrazování. Dále je zde uchován stav uživatelského rozhraní. Důležitou podmínkou, je že *ViewModel* není závislý na *View*, ale naopak *View* má referenci na *ViewModel*. Při použití architektury MVVM by měla aktivita obsahovat minimum logiky, která je přesunuta do části *ViewModel*. Jednou z vlastností Androidu je fakt, že při rotaci obrazovky je aktivita zničena a následně je vytvořena její nová instance. Tento problém se dá vyřešit pomocí uložení stavu do *ViewModelu*.

Architektura MVVM je výhodná především u rozsáhlejších aplikací s velkým počtem obrazovek. Pokud například některé obrazovky mají podobnou funkci, lze sdílet jejich implementaci na úrovni jejich *ViewModelu* (pomocí dědičnosti). Při implementaci aplikace jsem použil tuto architekturu i přes to, že je navrhovaná aplikace, co se počtu obrazovek týče, poměrně malá. Důvodem je striktní oddělení logiky od uživatelského rozhraní, které dle mého názoru zpřehledňuje výsledný kód. Pro implementaci byla použita knihovna Android ViewModel[12], která je součástí Android Jetpack[13].

4.6 Shromažďování a zpracování nasnímaných bodů

Při implementaci zpracování nasnímaných dat bylo třeba dbát na optimalizaci implementace. Zejména operace filtrování a ukládání detekovaných bodů, které se provádí v každém snímku, jsou kritické. Současné mobilní telefony mají výkonné mnoho jádrové procesory, takže pro plné využiti jejich potenciálu, je nutné použít vícevláknové zpracování. K tomu jsem využil výše zmíněnou knihovnu ReactiveX, která umožňuje například i zpracování každé emitované položky na novém vlákně.

[11] https://developer.android.com/guide/components/activities/activity-lifecycle
[12] https://developer.android.com/topic/libraries/architecture/viewmodel
[13] https://developer.android.com/jetpack



Obrázek 4.1: Řetězec pro zpracování dat z knihovny ARCore v každém snímku. Je zde ukázáno, jak probíhá sjednocení již detekovaných bodů s nově detekovanými. Dále je určeno, jaké operace lze provádět na výpočetním vlákně, a které musí být prováděny na hlavním vlákně.

Ke zpracování a ukládání detekovaných bodů jsem implementoval řetězec, který je nakreslen na obrázku 4.1. Pro každé nově detekované body je proveden blok *filtrování nových bodů*. Získaná množina dat je *sjednocena* s již *uloženými body* a postupuje do bloku *filtrování všech bodů*. Takto zpracované body jsou následně *uloženy*. Obrázek také ukazuje, jaké fáze lze dělat na *výpočetním vlákně*, a které musejí být provedeny na *hlavním vlákně*. Výpis 4.6 ukazuje, jak je tento řetězec v aplikaci implementován, a jak je zajištěno přepínání vláken. Třída `CaptureData` je mnou navržená struktura, která ukládá: detekované body, plochy, pozici uživatele, pozici objektu a časové razítko. Je vytvořena pro každý snímek za pomoci dat z knihovny ARCore. Data se do řetězce posílají metodou `captureSubject.onNext(newData)`, kde `newData` jsou data získaná v aktuálním snímku.

```
val captureSubject = PublishSubject.create<CaptureData>()

captureSubject.observeOn(Schedulers.computation())
  .map { newData ->
    val filtered = filterNewCaptureData(newData)// Filtrování nových bodů
    val merged = mergeOldCaptureData(filtered)  // Sjednocení
    return@map filterAllCaptureData(merged)     // Filtrování všech bodů
  }
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe(this::applyCaptureData)            // Zpětné uložení bodů
```

Výpis 4.6: Implementace řetězce pro zpracovaní bodů. Je zde ukázáno, jak se řeší přepínání vláken. Veškeré zpracování je prováděno na výpočetním vlákně (`computation`) a výsledek je aplikován na hlavním vlákně (`mainThread`).

Kapitola 5

Zhodnocení výsledné aplikace s názvem AR measure

Aplikace byla vydána 23. 11. 2018 na Google Play v programu beta testování. Od té doby dostala 10 aktualizací a 17. 3. 2019 byla vydána její produkční verze[1]. Tato kapitola se zaměří právě na produkční verzi aplikace, kterou považuji za plně funkční. Budou zde zhodnoceny její výsledky při měření objektů známé velikosti, dále zhodnotím zpětnou vazbu od uživatelů a statistiky používání aplikace. Uvedu možná vylepšení, která většinou vyplynula ze zpětné vazby či pozorování uživatelů, kteří nebyli s aplikací seznámeni.

5.1 Měření objektů známé velikosti

Již při vývoji jsem využíval několik běžných objektů známé velikosti, pomocí nichž jsem ověřoval přesnost. Cílem bylo měřit objekty, které by reálný uživatel mohl chtít změřit a bylo by vhodné je použít i při propagaci aplikace. Obrázek 5.1 ukazuje tyto referenční objekty. Jedná se zleva o zásuvkový kontejner, cestovní kufr, dřevěnou stoličku, kartonovou krabici a odpadkový koš.

Tato kapitola se zaměří na zhodnocení přesnosti měření těchto objektů pomocí produkční verze. Tabulka 5.1 ukazuje sadu referenčních objektů a jejich skutečné rozměry, které budou později srovnány s naměřenými. Chyby měření budu zkoumat odděleně pro každý rozměr. Je to z důvodu využití rozdílných principů určování výšky předmětu oproti zbylým rozměrům.

Objekt	**Výška**	**Šířka**	**Hloubka**
Kartonová krabice	33 cm	22 cm	10 cm
Cestovní kufr	54 cm	36 cm	23 cm
Dřevěná stolička	32 cm	31 cm	31 cm
Zásuvkový kontejner	68 cm	40 cm	49 cm
Odpadkový koš	40 cm	25 cm	25 cm

Tabulka 5.1: Skutečné rozměry referenčních objektů, které jsou použity pro vyhodnocení přesnosti měření.

[1] Dostupné na: https://play.google.com/store/apps/details?id=ml.kari.armeasure

Pro vyhodnocení přesnosti aplikace jsem každý referenční objekt změřil 5×. Pokaždé bylo měření provedeno s nově spuštěnou aplikací, abych předešel ovlivnění předchozími pokusy. Měření jsem prováděl za dobrých světelných podmínek v místnosti se světlou podlahou s motivem dřeva (viz obrázek 5.1). Výsledky jsou zaznamenávány v tabulce 5.2.

Objekt		**Měření**					**Průměrná chyba**	
		1	2	3	4	5		
Kartonová krabice	v	35 cm	31 cm	33 cm	35 cm	31 cm	1, 2 cm	3, 64 %
	š	22 cm	23 cm	23 cm	23 cm	23 cm	0, 8 cm	3, 64 %
	h	9 cm	13 cm	11 cm	10 cm	9 cm	1, 2 cm	12, 00 %
Cestovní kufr	v	55 cm	63 cm	51 cm	56 cm	52 cm	3, 4 cm	6, 30 %
	š	37 cm	38 cm	37 cm	38 cm	38 cm	1, 6 cm	4, 44 %
	h	24 cm	26 cm	22 cm	25 cm	24 cm	1, 6 cm	6, 96 %
Dřevěná stolička	v	28 cm	39 cm	34 cm	31 cm	32 cm	2, 8 cm	8, 75 %
	š	30 cm	30 cm	29 cm	30 cm	29 cm	1, 4 cm	4, 52 %
	h	31 cm	29 cm	26 cm	29 cm	30 cm	2, 0 cm	6, 45 %
Zásuvkový kontejner	v	74 cm	72 cm	83 cm	69 cm	69 cm	5, 4 cm	7, 94 %
	š	39 cm	40 cm	43 cm	41 cm	46 cm	2, 2 cm	5, 50 %
	h	49 cm	50 cm	50 cm	51 cm	54 cm	1, 8 cm	3, 67 %
Odpadkový koš	v	36 cm	44 cm	35 cm	39 cm	35 cm	3, 8 cm	9, 50 %
	š	25 cm	26 cm	15 cm	25 cm	25 cm	2, 2 cm	8, 80 %
	h	23 cm	23 cm	26 cm	22 cm	21 cm	2, 4 cm	9, 60 %
		Průměrná chyba výšky					3, 3 cm	7, 22 %
		Průměrná chyba šířky					1, 6 cm	5, 38 %
		Průměrná chyba hloubky					1, 8 cm	7, 74 %

Tabulka 5.2: Výsledky měření referenčních objektů. Bylo provedeno 5 měření, kde byla měřena odděleně výška (v), šířka (š) a hloubka (h). Vpravo je průměrná chyba všech měření vůči referenčním rozměrům (absolutní v cm a relativní v %).

Je patrné, že chyba při měření výšky objektu, je výrazně větší než u ostatních rozměrů. Průměrná chyba napříč všemi měřeními výšky je 3, 3 cm. Oproti tomu u šířky a hloubky je to 1, 6 cm a 1, 8 cm. Dále je patrné, že relativní chyba je vyšší pro menší rozměry. Tedy že absolutní chyba je nezávislá na rozměrech objektů. Pohybuje v rozsahu $1 - 5$ cm pro šířku a hloubku a $1 - 15$ cm pro výšku.

5.2 Uživatelské testování a zpětná vazba

Za uživatelské testování v tomto případě považuji situaci, kdy jsem dal produkční verzi aplikace otestovat jiné osobě, kterou jsem při používání pozoroval. Jednalo se o kamarády, rodinu a největší skupinu tvořili návštěvnici mého plakátu na konferenci Excel@FIT[2]. Většina ze vzorku testovaných uživatelů viděla aplikaci poprvé. Před samotným testováním jsem uživatelům krátce vysvětli k čemu aplikace slouží. Dále jsem měl připravený referenční objekt. Abych předešel zbytečným chybám a celý proces urychlil, tak jsem měřený objekt umístil do prostoru a vysvětlil uživateli, proč je to důležité. Aplikaci jsem připravil tak, aby byla viditelná podrobná nápověda.

[2] http://excel.fit.vutbr.cz/submissions/2019/049/49_poster.pdf

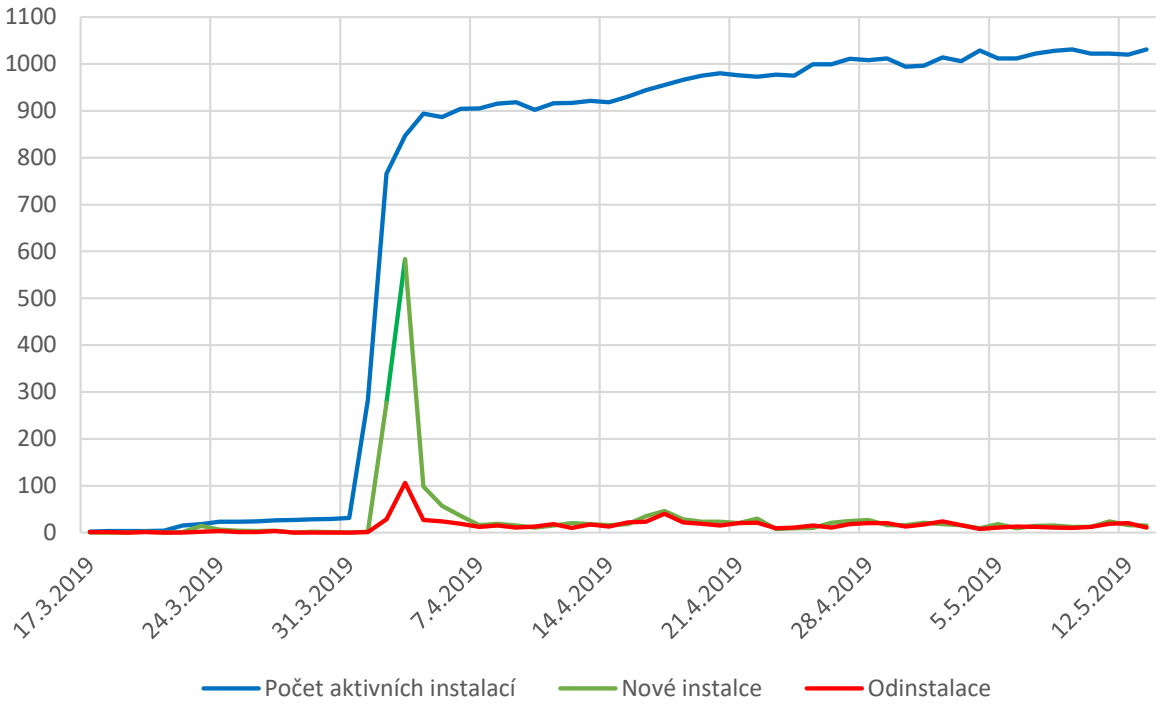
Obrázek 5.1: Referenční objekty použité při vyhodnocení přesnosti měření. Jedná se zleva o zásuvkový kontejner, cestovní kufr, dřevěnou stoličku, kartonovou krabici a odpadkový koš.

Někteří uživatelé sami postupovali podle pokynů v aplikaci. Někteří ale věnovali více pozornosti mému vysvětlování, než popiskům v aplikaci. Těmto uživatelům jsem poradil, ale snažil jsem se pokyny formulovat podobně, jak jsou zmíněny v nápovědě aplikace. Při testování jsem si nedělal žádné statistiky, takže tato kapitola zmíní jen některé zajímavé postřehy samotných uživatelů a poznatky, které jsem načerpal při pozorování jejich chování.

Jedním z častých problémů bylo, že uživatel nevěděl, jak měření zopakovat. Aplikace nenabízí žádné tlačítko pro okamžité opakování měření. Je třeba se vrátit 2× zpět do fáze *výběru objektu*. Další související problém spočíval v tom, že si uživatel neuvědomil, že se může vrátit zpět. Šipka zpět, v levém horním rohu, totiž není vidět na světlém pozadí. Několik uživatelů se zkoušelo vracet zpět kliknutím na ikonu předchozí fáze, ale s tímto jsem při návrhu a implementaci nepočítal a proto návrat tímto způsobem není možný.

Několik uživatelů se snažilo manipulovat se scénou rozšířené reality. Šlo o gesto pro přiblížení obrazu, které je známé z aplikací fotoaparátu a prohlížení fotografií. Dále se například snažili posunout kruh pro výběr objektu gestem. Další uživatelé se pokoušeli smazat některé nasnímané body, které nepatřily měřenému objektu, ale byly omylem zachyceny. Všechny tyto snahy byly nefunkční, protože jsem při návrhu nepočítal s interaktivní scénou.

Zmíněné problémy, které jsem u uživatelů vypozoroval, byly způsobeny hlavně tím, že aplikace neimplementuje více způsobů pro řešení stejného problému. Až při sledování chování různých uživatelů se objevili další logické způsoby, jak tyto problémy řešit. Tyto poznatky mi byly inspirací pro další funkce, které by někteří uživatelé intuitivně využili.



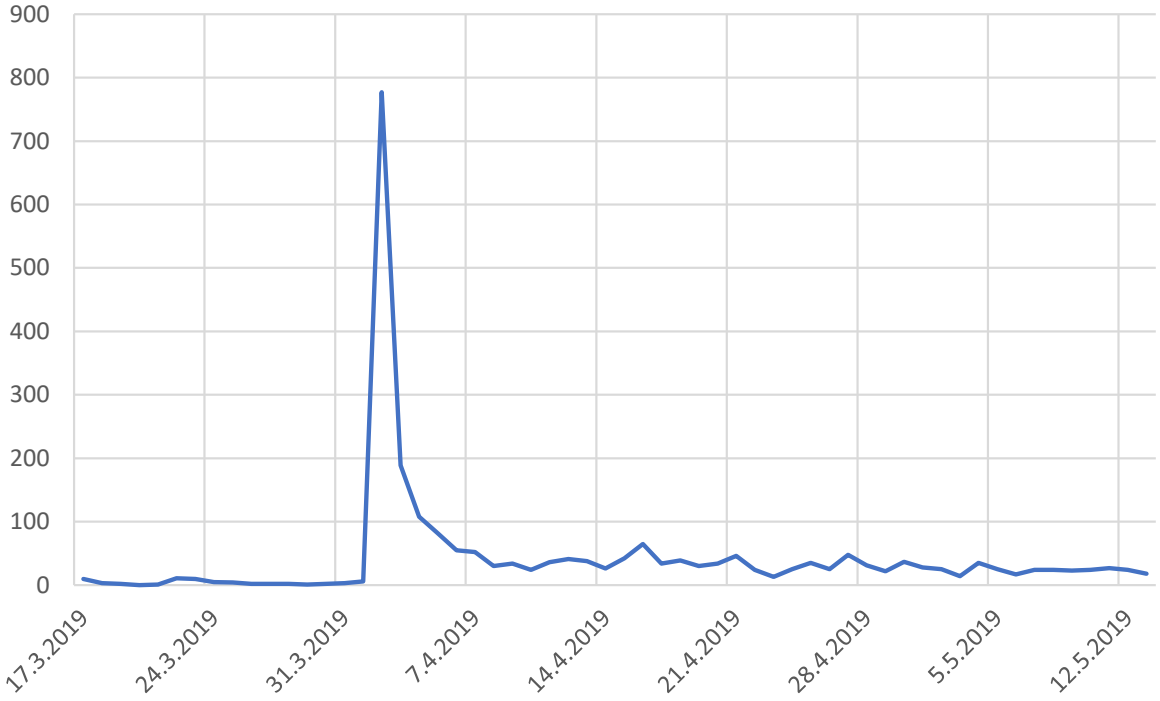
Obrázek 5.2: Graf počtu instalací, odinstalací a aktivních instalací z přibližně dvouměsíčního období, začínajícího vydáním produkční verze aplikace.

5.3 Analýza stahování a používání aplikace

Aplikace implementuje odesílání uživatelských událostí pomocí Firebase Analytics, jak již bylo zmíněno v kapitole 4.4. Další údaje lze vyčíst z Google Play Console, která ukazuje statistiky týkající se stahování aplikace a hodnocení od uživatelů. Tato kapitola bude z těchto statistik vycházet a budou zde shrnuty zajímavé poznatky, související s používáním aplikace.

Aplikaci si od jejího vydání do 15. 5. 2019 stáhlo přibližně 1 840 uživatelů a za tuto dobu bylo zaznamenáno přibližně 830 odinstalací. Aktuálně je tedy aplikace nainstalována na více než tisíci zařízeních. Vývoj této situace shrnuje graf na obrázku 5.2. Je zde patrný nejvyšší růst s maximem 2. 4. 2019. Ten je způsobený neplacenou propagací na různých sociálních sítích. Je vidět, že propagace měla účinnost ještě přibližně týden. Od té doby je počet stažení konstantní. Počet stažení v ustálené oblasti je průměrně 18, 5 denně. U odinstalací se průměr pohybuje kolem 16 denně.

Druhý graf na obrázku 5.3 ukazuje počet aktivních uživatelů v jednotlivých dnech. V ustáleném období od 7. 4. 2019 je průměrný počet aktivních uživatelů 30 denně. Uvážíme-li, že ve stejném období bylo přibližně 18, 5 instalací za den, tak spuštění od stávajících uživatelů muselo být kolem 11, 5 denně.



Obrázek 5.3: Graf počtu aktivních uživatelů denně. Přibližně dvouměsíční období, začínající vydáním produkční verze aplikace.

Další zajímavá čísla jsem získal z události konce měření, jejíž součástí jsou naměřené rozměry. Průměrné rozměry měřených objektů jsou $51 \times 54 \times 57$ cm (první je výška). Dále ze statistik vyplývá, že 70 % přechodů do fáze *výsledku měření* je provedeno manuální akcí uživatele. Fáze *sběr bodů* trvá obvykle 21 sekund a je nasbíráno přibližně 1 100 bodů. Průměrné trvání jednotlivých fází měření je shrnuto v tabulce 5.3. Tabulka dále uvádí průměrnou četnost jednotlivých fází v rámci jednoho sezení (jedno spuštění aplikace) na jednoho uživatele.

Fáze	**Doba trvání**	**Počet na sezení**	**Počet na uživatele**
Detekce prostředí	15 sekund	1, 6	2, 3
Výběr objektu	19 sekund	2, 9	4, 6
Sběr bodů	21 sekund	2, 0	3, 2
Výsledek měření	29 sekund	1, 5	2, 9
Dodatečný sběr bodů	6 sekund	1, 2	2, 7

Tabulka 5.3: Průměrná doba trvání jednotlivých fází měření získaná ze statistik používání aplikace. Tabulka ukazuje i průměrnou četnost jednotlivých fází v rámci jednoho sezení (jedno spuštění aplikace) na jednoho uživatele.

5.4 Možná vylepšení

Na základě pozorování uživatelů, kteří testovali mnou implementovanou aplikaci, a podle jejich zpětné vazby jsem vymyslel některá vylepšení. Tato kapitola zmíní pouze ty, které jsem nestihl implementovat, protože hotová vylepšení jsou popsána v předchozích kapitolách prostřednictvím iterativního návrhu aplikace.

Z výsledků přesnosti měření je patrná velká chyba při měření výšky objektů. Ta je způsobena tím, že při návrhu jsem se příliš nezabýval problematikou výběru podkladu objektu. Zvolil jsem poměrně jednoduché řešení, které je v některých případech nepřesné. Někdy se dokonce stane, že vybraná plocha je i o několik desítek centimetrů níže, než reálná podložka. Ačkoliv díky konstantní velikosti kruhu pro výběr je patrné, že vybraná plocha neodpovídá realitě, ukázalo se, že si toho všimne pouze zkušený uživatel. Uživateli, který aplikaci používá poprvé, se měření nepodaří, nebo skončí s velkou chybou. Pro vylepšení přesnosti měření by teď bylo nutné navrhnout a implementovat lepší proces pro výběr podložky měřeného objektu.

Z hlediska uživatelského rozhraní je třeba promyslet tlačítka pro návrat zpět. Bylo by vhodné přidat tlačítko pro opakování celého měření a umožnit návrat zpět kliknutím na ikonu nějaké předchozí fáze. Dále se nabízí udělat virtuální scénu interaktivní. Aby se například dalo gesty zvětšovat obraz, či nějak ovlivňovat měření. Například častým problémem jsou chybně detekované body. Pokud jich je více, tak mohou výrazně ovlivnit měření. Proto by bylo dobré mít možnost je odstranit. Aktuálně je pro získání správného výsledku nutné celé měření zopakovat.

Kapitola 6

Závěr

Tato diplomová práce se zabývala návrhem a implementací inovativní aplikace, která pomocí technologie rozšířené reality dokáže měřit reálné objekty. Inovativnost spočívá v tom, že oproti existujícím aplikacím stačí obejít měřený objekt a aplikace sama určí celkové rozměry objektu. Práce navrhuje celý proces měření, který z dat knihovny ARCore sbírá informace, pomocí nichž jsou poté určeny rozměry. Uživatelské rozhraní jsem vyvíjel iterativně pomocí prototypů a na základě zpětné vazby uživatelů jsem jej vylepšoval.

Aplikaci jsem implementoval pro operační sytém Android a publikoval prostřednictvím Google Play. Za dva měsíce si ji stáhlo přes 1 800 uživatelů a denně ji používá průměrně 30 lidí. Díky tomu získala na studentské konferenci Excel@FIT 2019 ocenění odborné komise „za inovativní a dotaženou aplikaci, která má již nyní reálné uživatele“.

Součástí práce je i zhodnocení výsledné aplikace, které se zabývá vyhodnocením přesnosti měření. Ukázalo se, že přesnost měření výšky předmětů je výrazně horší než přesnost měření ostatních rozměrů. Zabývalo se i zpětnou vazbou od uživatelů, kteří aplikaci používali poprvé. Z ní vzniklo několik nápadů na funkce, které v aplikaci chybí. Bylo odhaleno několik slabých míst návrhu uživatelského rozhraní, které vedly k dezorientaci uživatele. Získané poznatky jsem popsal a vychází z nich navržená vylepšení aplikace.

Literatura

[1] Azuma, R.: A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, ročník 6, 1996.

[2] Azuma, R.; Baillot, Y.; Behringer, R.; aj.: Recent advances in augmented reality. IEEE Comput Graphics Appl. *Computer Graphics and Applications, IEEE*, ročník 21, 2001: s. 34 – 47.

[3] Caudell, T. P.; Mizell, D. W.: Augmented reality: an application of heads-up display technology to manual manufacturing processes. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, ročník 2, 1992, s. 659 – 669.

[4] Durrant-Whyte, H.; Bailey, T.: Simultaneous localization and mapping: part I. *IEEE Robotics Automation Magazine*, ročník 13, č. 2, 2006: s. 99 – 110.

[5] Foxlin, E.; Harrington, M.; Pfeifer, G.: Constellation: A Wide-range Wireless Motion-tracking System for Augmented Reality and Virtual Set Applications. In *Annual conference, Computer graphics: SIGGRAPH 98*, 1998, s. 371 – 378.

[6] Google: ARCore API Reference. [Online; navštíveno 13.01.2019].
URL https://developers.google.com/ar/reference/

[7] Jemerov, D.; Isakova, S.: *Kotlin in Action*. Manning Publications, první vydání, 2017, ISBN 9781617293290.

[8] Liu, H.; Zhang, G.; Bao, H.: Robust Keyframe-based Monocular SLAM for Augmented Reality. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2016, s. 1 – 10.

[9] Milgram, P.; Kishino, F.: A Taxonomy of Mixed Reality Visual Displays. *IEICE Trans. Information Systems*, ročník vol. E77-D, no. 12, 1994: s. 1321 – 1329.

[10] Rolland, J.; Biocca, F.; Hamza Lup, F.; aj.: Development of Head-Mounted Projection Displays for Distributed, Collaborative, Augmented Reality Applications. *Presence*, ročník 14, 2005: s. 528 – 549.

[11] Sutherland, I. E.: A Head-mounted Three Dimensional Display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, 1968, s. 757 – 764.

[12] Suyam, Y.; Neumann, U.; Azuma, R.: Hybrid inertial and vision tracking for augmented reality registration. In *Proceedings IEEE Virtual Reality (Cat. No. 99CB36316)*, 1999, s. 260 – 267.

[13] Szeliski, R.: *Computer Vision: Algorithms and Applications*. Berlin, Heidelberg: Springer-Verlag, první vydání, 2010, ISBN 1848829345, 9781848829343.

[14] Vosselman, G.; Dijkman, S.: 3D building model reconstruction from point clouds and ground plans. In *Proceedings of the ISPRS Workshop*, XXXIV-3/W4, International Society for Photogrammetry and Remote Sensing (ISPRS), 2001, s. 37 – 43.