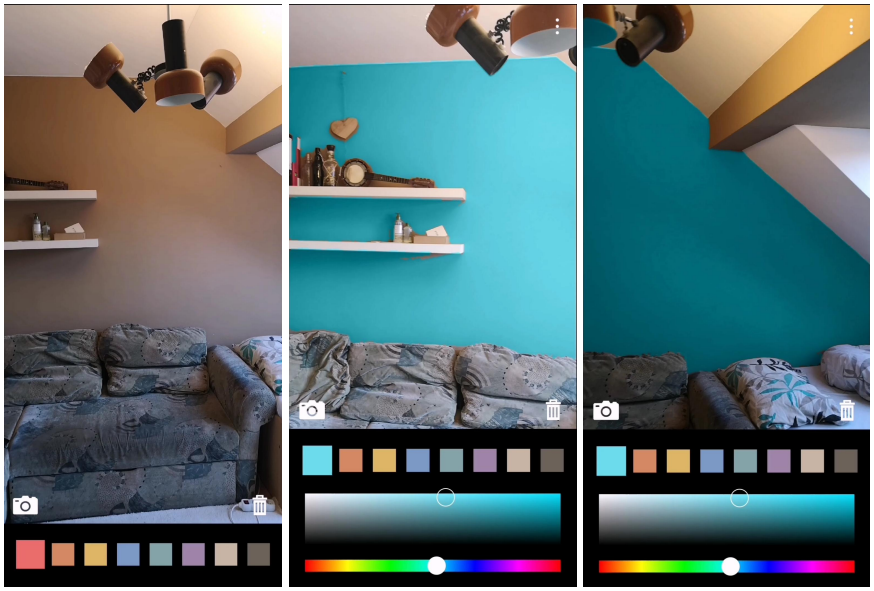
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

Náhľad farby na stene pomocou rozšírenej reality v mobile

Dominik Vagala



Abstrakt

Cieľom tejto práce je návrh a implementácia mobilnej aplikácie pre Android, ktorá by umožňovala meniť farby na stene pomocou rozšírenej reality. Užívateľ si tak môže vyskúšať rôzne farby priamo v miestnosti, kde sa nachádza a následne sa rozhodnúť, ktorá farba sa mu najviac páči na vymaľovanie stien. Na rozpoznanie hraníc steny je použitý Sobelov detektor hrán, kde sa ohraničený úsek steny vypĺňa farbou pomocou upraveného Queue-Linear Flood Fill algoritmu. 2D súradnice, kde užívateľ klikol na stenu, sa približne prepočítajú na 3D súradnice v priestore. Tie sa následne sledujú pomocou knižnice ARCore, vďaka čomu stena zostane zafarbená, aj keď sa užívateľ pohybuje po miestnosti.

Kľúčové slová: Paint my room — Change wall color — Náhľad farby na stene — Rozšírená realita — AR — ARCore — Mobilná aplikácia — Android

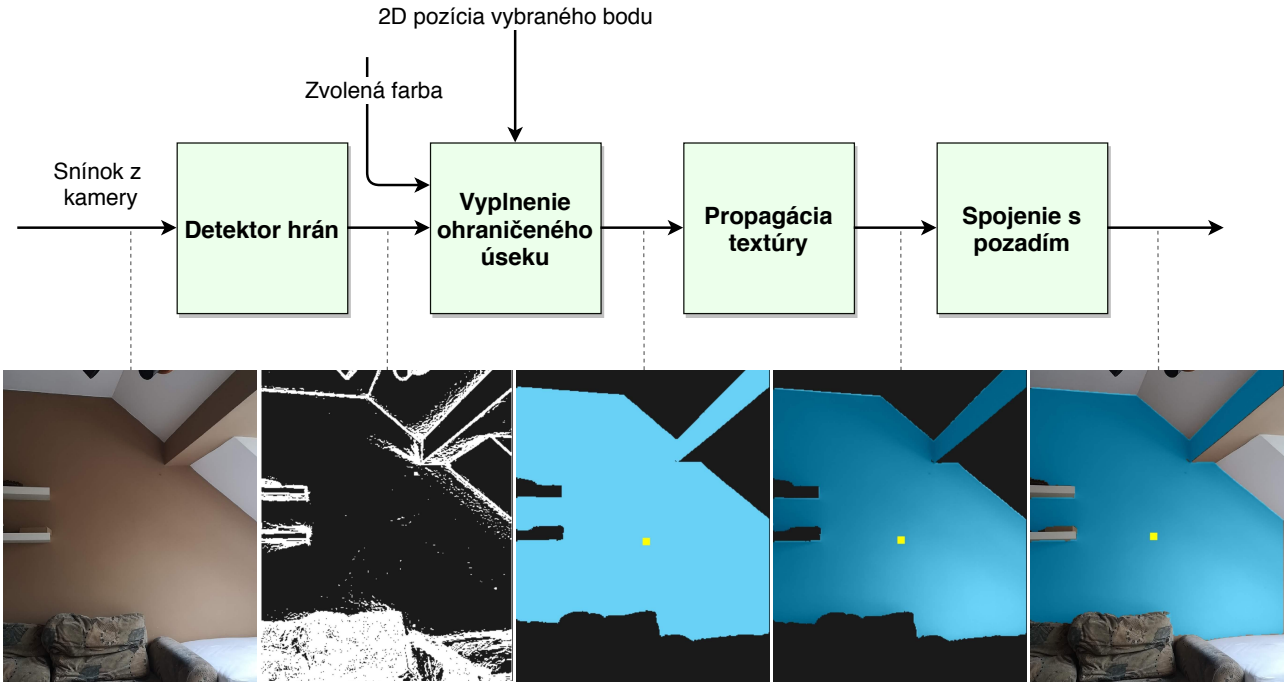
Priložené materiály: Aplikácia na Google Play, Demonštračné video

*xvagal00@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Ľudská vizuálna predstavivosť má svoje limity, a preto sú častokrát potrebné riešenia, ktoré vedia očakávanú víziu zobraziť. Moja aplikácia ponúka riešenie v modelovej situácii, keď si človek chce ísť vymaľovať izbu, a nevie sa rozhodnúť, akú farbu si vybrať.

V aplikácií riešim, čo najviac realistické zobrazenie farby na stene, ktorú si užívateľ zvolil. Je to kľúčový faktor, pretože ak vizualizácia farby nebude vyzerať dostatočne reálne, užívateľ si nebude vedieť predstaviť, ako by miestnosť skutočne vyzerala vymaľovaná s danou farbou. Výsledná aplikácia by preto



Obrázok 1. Bloková schéma vykresľovania.

mala spoľahlivo rozlíšiť hranicu steny, aby sa farbou neprekrývali okolité objekty, a aby boli v prekrývanej farbe zachované tiene a odlesky steny. Ďalším dôležitým faktorom je, aby vizualizácia prebiehala v reálnom čase, čiže užívateľ sa môže voľne pohybovať po miestnosti a sledovať vyfarbené steny z rôznych uhlov. Užívateľ si vždy najskôr klikne na stenu, ktorú si chce vymaľovať.

V blokovej schéme na obrázku 1 je zobrazený zjednodušený postup, ktorý sa aplikuje na každý nový snímok. Na zistenie hraníc steny je použitý detektor hrán, ktorého výstupom je binárna maska. Na sledovanie bodov, ktoré si užívateľ vybral, je použitá knižnica ARCore [1]. Z týchto bodov sa následne vychádza pri vyfarbovaní steny, kedy sa vyplní ohraničený segment v binárnej maske, kde sa daný bod nachádza.

2. Existujúce riešenia

Existuje niekoľko mobilných aplikácií na Android, ktoré vedia zmeniť farbu steny na fotke, ale len dve, ktoré riešia tento problém v reálnom čase: Dulux Visualizer[1] a ColorSnap[2] (na väčšine zaradení spadne). V obidvoch prípadoch je priestor na zlepšenie. Pri pohybe sa vyfarbovaný úsek nepropaguje ďalej, ale zmizne hneď, ako sa prejde za bod, kde používateľ klikol na stenu. Tiež je v užívateľskom rozhraní potrebných príliš veľa krokov na to, aby si užívateľ zmenil zvolenú farbu steny. Je dôležité, aby bol tento proces čo najrýchlejší a užívateľ mohol jednoducho alternovať medzi farbami.

[1] https://play.google.com/store/apps/details?id=com.akzonobel.cz.dulux

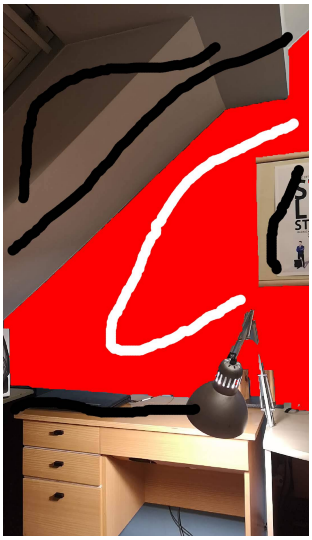
[2] https://play.google.com/store/apps/details?id=com.colorsnap

3. Rozpoznávanie steny

Základným problémom, ktorý bolo treba riešiť, je rozpoznanie úseku steny, ktorý má byť zafarbený.

Prvotný návrh bol, robiť toto rozpoznanie len na základe farby jednotlivých pixelov v snímke (obrázok 2). Od miesta, na ktoré užívateľ klikol na stenu, by sa postupne vyplňovacím algoritmom prechádzalo jeho okolie, a ak je daný pixel v určitej farebnej tolerancii, zafarbí sa. Vyskúšal som rôzne prístupy porovnania farieb, euklidovskú vzdialenosť v RGB, alebo CIE94.

Obrázok 2. Vľavo: Pôvodný snímok z kamery. **Vpravo:** Nekvalitné rozpoznanie steny – porovnávanie zložky U a V.



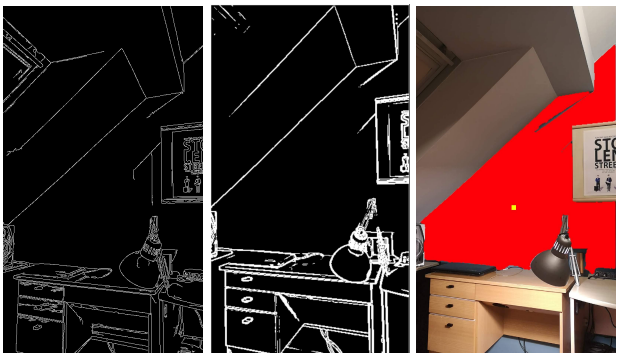
Obrázok 3. Rozpoznanie steny – algoritmus GrabCut s čiarami, ktorými užívateľ označil úsek steny.

Počas experimentovania som ale zistil, že najvhodnejšie je porovnávať vo farebnom modeli YUV [2] iba zložky U (intenzita modrej) a V (intenzita červenej), pričom Y (jas) sa ignoruje. To trochu pomohlo riešiť problém s tieňmi a odleskami na stene, avšak stále to nebolo postačujúce.

Ďalej som zvážil použitie segmentačných algoritmov. Vyskúšal som algoritmus GrabCut [3] implementovaný v knižnici OpenCV. Výsledok bol najlepší z použitých prístupov. Problém bol v tom, že nestačilo iba kliknúť na stenu, ale užívateľ by musel celkom detailne vyznačiť úsek steny, aby algoritmus vyprodukoval kvalitný výstup. Na obrázku 3 je to znázornené čiarami: biela – tento úsek vyber, čierna – tento úsek zahoď. Na statickom snímku by sa to dalo použiť, ale pri použití v reálnom čase, by bolo problém sledovať tieto vyznačené čiary. Navyše doba výpočtu sa pohybovala rádovo v sekundách.

Finálne riešenie spočíva v použití detektoru hrán. Algoritmus je založený na Sobelovom detektore hrán [4] (obrázok 4), ktorého výstup je prahovaný, aby sa získala binárna maska. Následne sa z miesta, kde užívateľ klikol na stenu vyplňovacím algoritmom, vyplní táto ohraničená časť. Hlavný problém tohto prístupu je, že niekedy ohraničený úsek steny obsahuje medzery a tým pádom sa vyplnia aj úseky, ktoré nemajú byť vyplnené. To spôsobuje nepríjemné preblikovanie chybne vyfarbených častí. Snažil som sa to riešiť rozšírením hrán v maske (dilation) s následným stiahnutím (erosion), aby sa tieto medzery vyplnili. Vo výsledku to ale spôsobilo, že malé parazitné hrany na stene, spôsobené tieňmi sformovali súvislé bloky, čiže tam zostala stena nevyfarbená. Preto zatiaľ jediným riešením je vhodne určiť prah tohto detektoru.

Vyskúšal som aj algoritmus Canny Edge Detection [5] (obrázok 4) v OpenCV, no je výpočetne náročnejší a medzery v ohraničenom úseku sa vyskytovali ešte častejšie.



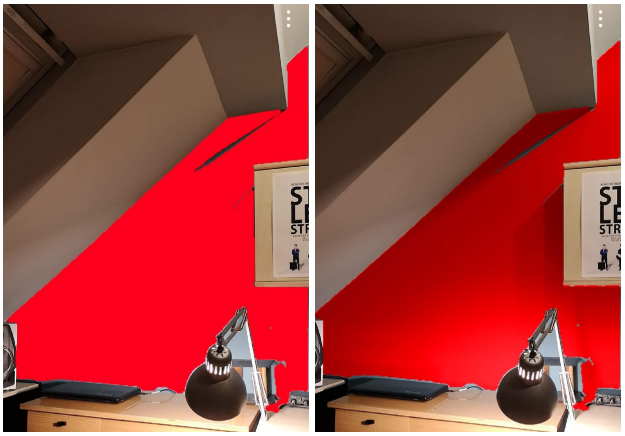
Obrázok 4. Porovnanie detektorov hrán. **Vľavo:** Canny edge detector. **V strede:** Sobel Operator. **Vpravo:** Sobel operator – vyplnenie úseku – finálne riešenie.

4. Vyplnenie ohraničeného úseku

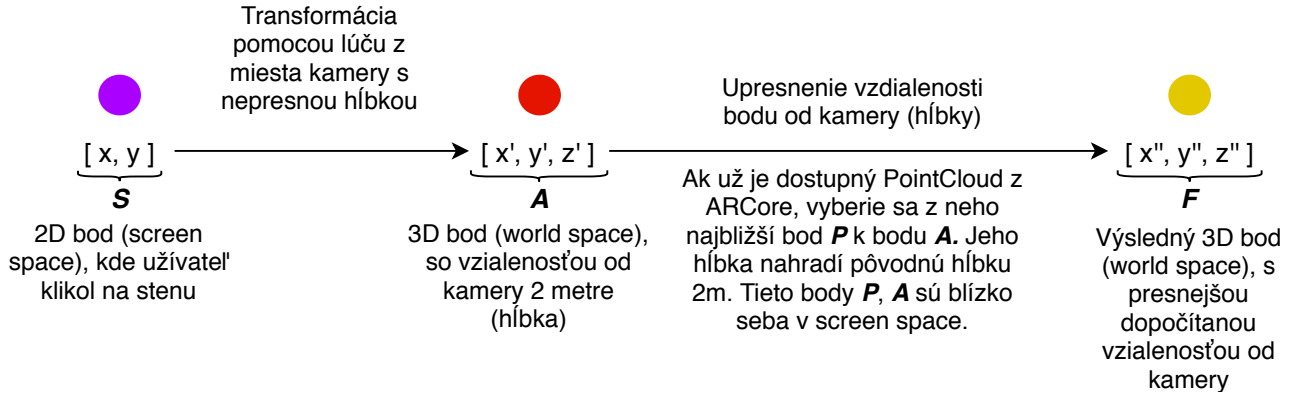
Na vyplnenie ohraničeného úseku, ktorý poskytne detektor hrán je použitý trochu upravený algoritmus Queue-Linear Flood Fill [6]. Je pomerne rýchly a nie je pamäťovo náročný. Tento vyplnený úsek je následne rozšírený o fixnú veľkosť, aby farba doliehala presne až ku okrajom steny a aby sa zakryli prípadné parazitné hrany vo vnútri steny.

5. Propagácia textúry steny

Aby vyfarbený úsek steny pôsobil reálne, musí obsahovať tiene a odlesky pôvodnej steny (obrázok 5). V použitom algoritme sa z pôvodného pixelu steny, zoberie komponenta jasu (Luminance) vo farebnom modeli YUV a primieša sa do prekrývanej farby.



Obrázok 5. Vľavo: Bez propagácie textúry. **Vpravo:** S propagáciou textúry.



Obrázok 6. Transformácia vybraného bodu.

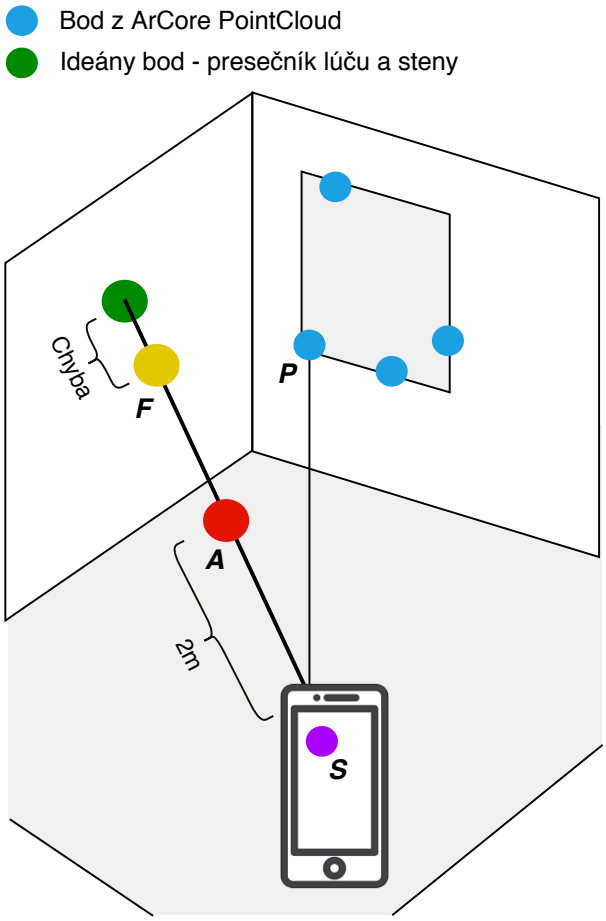
6. Sledovanie vybraných bodov

V mieste, kde užívateľ klikne na stenu, sa začína s vypĺňovaním ohraničeného úseku. Aby sa užívateľ mohol voľne pohybovať po miestnosti, je potrebné tieto 2D súradnice previesť na bod v 3D priestore a sledovať ho. Postup tejto transformácie je načrtnutý v obrázkoch 6 a 7. Tento 3D bod (world space) je pri každom novom snímku prevedený späť na bod v 2D (screen space), ktorý sa odovzdá vypĺňovaciemu algoritmu. Táto transformácia sa vykoná pomocou aktuálnych matíc *projection* a *view* z ARCore.

Spomenutý PointCloud [7] sú body v priestore, ktoré poskytuje ARCore. Tieto body sú typicky dostupné až vtedy, keď sa užívateľ začne pohybovať po miestnosti a sú na miestach, ktoré vie ARCore jednoducho odlíšiť od okolia. Preto sa nevyskytujú na holej stene, ale napríklad na nábytku s textúrou alebo na okrajoch objektov (obrázok 7). Kvôli tomuto faktu som nemohol použiť Plane z ARCore a robiť iba jej priesečník s lúčom z kamery. Užívateľ by musel najprv detailne prejsť celý priestor, aby sa mu Plane spočítala a navyše by mu vykreslená prekážala vo výhľade na stenu.

Vďaka prístupu, ktorý som použil, môže užívateľ používať aplikáciu aj bez toho, aby musel vopred zmapovať okolie. Ak sa kamera nehýbe, alebo ak rotuje, umelá vzdialenosť (2 m) neprekáža, pretože z pohľadu kamery je na tom istom mieste. Ak sa kamera začne pohybovať, tak sa vzdialenosť upresní, pretože je už dostupný PointCloud.

Z tohto prístupu je zrejmé, že môže nastať situácia, kedy bude mať žltý bod **F** veľkú *chybu* (obrázok 7), a kamera môže zmeniť svoju pozíciu tak, že bod bude na chybnom úseku steny a vyfarbí sa. Riešenie spočíva v tom, že v momente ako sa v mieste snímku daného 2D súradnicou bodu **F** vyskytne iná farba, ako tá, ktorá tam bola pri vzniku tohto bodu, tento bod sa zahodí. Táto farebná podobnosť je založená na algoritme popísanom v kapitole 3.

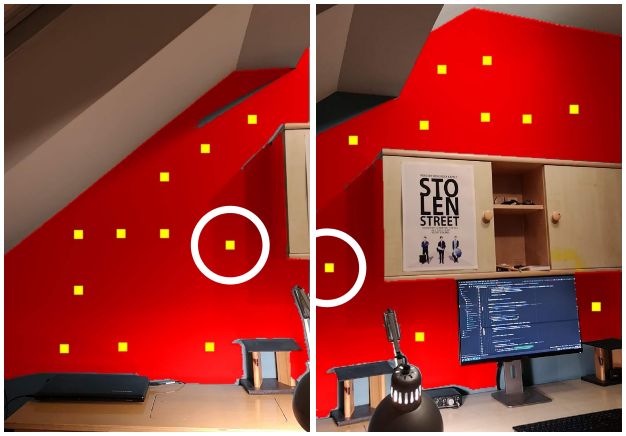


Obrázok 7. Lúč vychádzajúci z kamery, zobrazenie jednotlivých bodov.

7. Propagácia vybraných bodov

Pre jednoduchosť bolo doteraz popisované, že vybrané body sa vytvárajú iba užívateľom, ktorý klikne na stenu. V skutočnosti sa ale vždy k takémuto bodu pridajú ďalšie automaticky generované body v primeraných rozostupoch. Tie sa generujú iba na mieste, kde je stena už zafarbená. Pre výpočet týchto bodov platia rovnaké princípy ako som popisoval doteraz. Tieto body sa automaticky generujú v prípade, že v danom momente je viditeľných menej vybraných bodov ako

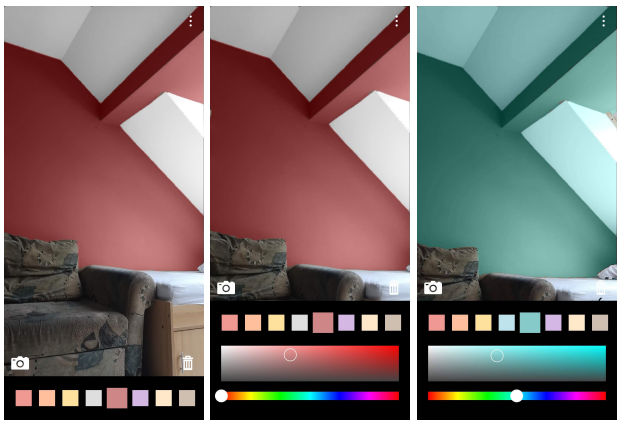
určitá konštanta. Vďaka tomu dochádza ku automatickej propagácii vybraných bodov pri pohybe (obrázok 8).



Obrázok 8. Propagácia vybraných bodov pri zmene uhlu kamery. Bod v krúžku znázorňuje jediný bod, ktorý bol vytvorený užívateľom kliknutím na stenu. Ostatné boli generované automaticky.

8. Návrh užívateľského rozhrania

Návrh užívateľského rozhrania (obr. 9) som priebežne testoval na užívateľoch, aby som zistil, ktoré prvky sú nejasné. Najväčší problém bol, ako docieliť, aby užívateľ mohol používať viaceré farby naraz na jednotlivé úseky steny a zároveň, aby mohol rýchlo tieto farby meniť a zisťovať, ktorá sa mu páči najviac. Použité widgety na výber farby som si navrhoval a implementoval sám, s existujúcimi riešeniami som nebol spokojný.



Obrázok 9. Návrh užívateľského rozhrania. Ukážka vysúvacej palety.

9. Optimalizácie

Všetky použité algoritmy sú pomerne rýchle. V čase písania tohto článku dosahuje vykresľovanie 30 fps na zariadení Huawei p20 pro. Čiže zhruba 33 ms trvá spracovanie jedného snímku. Väčšina algoritmov v mojom riešení prebieha na jednom vlákne v pozadí, iba niektoré sa dali implementovať na GPU. Do budúcna chcem zvážiť rozdelenie týchto sekvenčných výpočtov do viacerých vlákien. Chcel by som tiež optimalizovať detektor hrán, pretože je to najslabší článok celého riešenia. Do aplikácie ešte pridám uloženie fotky vyfarbenej miestnosti, mať možnosť meniť farby aj na statickej fotke a zobrazenie kódu danej farby, aby si ju užívateľ mohol v predajni kúpiť.

10. Zverejnenie aplikácie na Google Play

Aplikácia *Paint my Room* bola najskôr zavedená v uzavretom Alfa kanáli, kde som ju testoval prostredníctvom rôznych užívateľov a na rôznych zariadeniach. Teraz je pridaná do otvoreného Beta kanálu[3], kde sa dá aplikácia voľne stiahnuť.

11. Záver

Cieľom tejto práce bol návrh a implementácia mobilnej aplikácie pre Android, ktorá umožňuje meniť farby na stene pomocou rozšírenej reality. Vďaka tomu si užívateľ môže farbu lepšie predstaviť a vybrať naozaj takú, aká sa mu hodí do jeho miestnosti.

Hlavnou časťou tejto práce bolo experimentovanie s rôznymi prístupmi a algoritmami, ktoré sa dajú použiť v reálnom čase. Neoddeliteľnou súčasťou bol taktiež návrh intuitívneho užívateľského rozhrania. Výsledkom je teda funkčná aplikácia, zverejnená na Google Play v otvorenom Beta kanáli[3].

Vo vývoji aplikácie pokračujem ďalej, chcem sa zamerať na optimalizovanie rýchlosti vykresľovania a pridanie možnosti uložiť a zdieľať snímku vyfarbenej miestnosti (viď kapitolu 9). Po pridaní týchto funkcií a otestovaní, bude aplikácia zverejnená do produkčného kanálu, kde bude voľne dostupná užívateľom.

Poďakovanie

Rád by som sa poďakoval môjmu vedúcemu prof. Ing. Adamovi Heroutovi, PhD. za cenné rady a čas strávený na konzultáciách.

Literatúra

[1] Arcore overview — google developers. https://developers.google.com/ar/discover.

[2] Yuv color system. https://www.hisour.com/yuv-color-system-25916/.

[3] https://play.google.com/store/apps/details?id=com.bonc.paintmyroom

[3] Opencv: Interactive foreground extraction using grabcut algorithm. https://docs.opencv.org/trunk/d8/d83/tutorial_py_grabcut.html.

[4] Ashish. Understanding edge detection (sobel operator) - data driven investor - medium. https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900, Sep 2018.

[5] Canny edge detection step by step in python — computer vision. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123, Jan 2019.

[6] J. Dunlap. Queue-linear flood fill: A fast flood fill algorithm - codeproject. https://www.codeproject.com/Articles/16405/Queue-Linear-Flood-Fill-A-Fast-Flood-Fill-Algorith, Nov 2006.

[7] Pointcloud — arcore — google developers. https://developers.google.com/ar/reference/java/arcore/reference/com/google/ar/core/PointCloud.