Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Využitie eye trackera na analýzu zaujímavých oblastí na 3D objektoch

Bakalárska práca

2024
Veronika Horňáková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Využitie eye trackera na analýzu zaujímavých oblastí na 3D objektoch

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Zuzana Berger Haladová, PhD.

Bratislava, 2024
Veronika Horňáková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Veronika Horňáková
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Využitie eye trackera na analýzu zaujímavých oblastí na 3D objektoch
Utilizing an eye tracker for the analysis of salient areas on 3D objects

Anotácia: Cieľom práce je vytvoriť nástroj, ktorý bude, s využitím eyetrackera, zaznamenávať zaujímavé oblasti na 3D objekte a následne ich zobrazí vo forme textúry objektu.

Vedúci: RNDr. Zuzana Berger Haladová, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 24.09.2021

Dátum schválenia: 04.10.2023 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcela by som sa poďakovať mojej školiteľke RNDr. Zuzane Berger Haladovej, PhD., za jej pomoc, ochotu a všetky rady a nápady pri písaní práce. Poďakovanie tiež patrí mojej rodine a priateľom za podporu počas celého štúdia. Špeciálne ďakujem tým, ktorí mi pomohli so záverečným testovaním mojej práce.

Abstrakt

V tejto práci skúmame možnosť využitia eye trackerov na monitorovanie pohybu očí pri interakcii človeka s 3D objektami. Eye tracker je schopný zaznamenávať, kam pozorovateľ upriamil svoju pozornosť pri skúmaní 3D objektu. Dáta z eye trackerov nám dokážu poskytnúť hodnotné dáta o tom, na ktoré časti 3D objektov sú oči pozorovateľa najviac sústredené, a ktoré časti 3D objektov sú považované za zaujímavé. Cieľom tejto bakalárskej práce je vytvoriť nástroj, ktorý zaujímavé oblasti na 3D objektoch identifikuje a vizualizuje. V prvej časti práce je čitateľ oboznámený s teóriou pre lepšie pochopenie práce. V ďalších častiach je popísaný návrh aplikácie a podrobné detaily implementácie.

Kľúčové slová: 3D objekt, zaujímavé oblasti, eye tracker, sledovanie očí, rozšírená realita

Abstract

In this work, we investigate the possibility of using eye trackers to monitor eye movements during human interaction with 3D objects. The eye tracker is able to record where the observer rivets his attention while examining a 3D object. Data from eye trackers can provide us with valuable data on which parts of 3D objects the observer's eyes are most focused on, and which parts of 3D objects are considered interesting. The goal of this bachelor thesis is to create a tool that identifies and visualizes interesting areas on 3D objects. In the first part of the work, the reader is introduced to the theory for a better understanding of the work. The application design and detailed implementation details are described in the next sections.

Keywords: 3D object, areas of interest, eye tracker, eye tracking, augmented reality

Obsah

Zoznam obrázkov

Úvod

V dnešnej dobe sa 3D skenovanie stáva čoraz viac neoddeliteľnou súčasťou rôznych odvetví. Napriek technologickému pokroku však 3D skenovanie stále čelí množstvu výziev, ktoré môžu ovplyvniť kvalitu výsledných modelov. Hlavnou motiváciou tejto práce je odhaliť chyby pri 3D skenovaní pomocou ľudského vnímania, teda zistiť, ktoré oblasti na 3D skenoch najviac vadia ľuďom pri ich prezeraní, čo môže prispieť k odhaleniu chýb na týchto skenoch.

Cieľom tejto práce je vytvoriť nástroj, ktorý, s využitím eye trackera, zaznamená zaujímavé oblasti na 3D objekte. Získané dáta z eye trackera sa následne vizualizujú a poskytnú podrobný prehľad o oblastiach, ktoré upútali najväčšiu pozornosť pozorovateľa. Tento nástroj umožní lepšie pochopiť, ako ľudia vnímajú a skúmajú trojrozmerné objekty, a vizualizáciou zaujímavých oblastí môže upozorniť na potenciálne chyby na skenoch.

V prvej kapitole tejto práce sa oboznámime so základnými teoretickými poznatkami danej problematiky. Oboznámime sa s princípom sledovania očí, popíšeme, ako funguje hardvérové zariadenie, ktoré slúži na monitorovanie očí, aké dáta vieme získať a akým spôsobom ich dokážeme spracovať. Predstavíme rôzne spôsoby na vizualizáciu získaných dát a vysvetlíme pojem marker based reality, pomocou ktorej budeme s konkrétnymi 3D objektami pracovať.

Druhá kapitola obsahuje návrh aplikácie, popisuje fungovanie aplikácie, z akých častí sa aplikácia skladá, aké technológie v nich využívame a s akými dátami pracujeme.

Tretia kapitola sa venuje implementácii. V tejto kapitole sú podrobne popísané všetky implementačné detaily každej časti práce vrátane použitých knižníc a funkcií.

Štvrtá kapitola sa venuje testovaniu aplikácie. Obsahuje detaily o testovaní aplikácie a zobrazuje výsledky práce v podobe vizualizácií sledovaného objektu.

Piata kapitola sa zaoberá možnou modifikáciou a rozšíreniami tejto práce, ako aj jej ďalším využitím.

Kapitola 1

Teoretické východiská

1.1 Princíp sledovania očí

Princíp sledovania pohybu očí („eye tracking“) je jedným zo základných prístupov na určovanie významných oblastí v pozorovanej scéne. Pojmom „významná oblasť“ sú označované oblasti v scéne, na ktoré ľudia upriamujú svoju vizuálnu pozornosť. Vizuálnou pozornosťou sa zaoberajú vedci z rôznych odvetví, pretože zistenie, kam ľudia zameriavajú svoju pozornosť, je v mnohých aplikáciách kľúčové. Detegovanie významných oblastí má široké využitie v praxi, a preto je potrebné, aby bola táto detekcia čo najpresnejšia a najrýchlejšia.

Eye tracking je založený na sledovaní očí pozorovateľa a ich pohybu. Zaznamenávaním pohybov očí sa zisťuje, na akú oblasť v scéne upriamuje svoju pozornosť. Táto technika sa využíva najmä v oblasti interakcie medzi človekom a počítačom alebo pri ovládaní rôznych systémov, napríklad elektronického vozíka a televízneho ovládača. Môže tiež byť použitá pri čoraz obľúbenejšom biometrickom rozpoznávaní na základe dúhovky používateľa. Sledovanie pohybu očí sa využíva aj v automobilovom priemysle, konkrétne v systémoch nasadených do áut, ktoré dokážu varovať vodiča v prípade, že počas jazdy zaspí.

Monitorovanie pohybu očí je možné vykonávať viacerými spôsobmi. Najznámejšími sú sledovanie pohybu očí pomocou senzorov a sledovanie pohybu očí pomocou techník počítačového videnia. Sledovanie pohybu očí pomocou senzorov je založené na využívaní elektrických potenciálov, ktoré sú zaznamenávané elektródami umiestnenými v blízkosti očí. Druhý spôsob, sledovanie pohybu očí pomocou techník počítačového videnia, využíva kamery na monitorovanie očí. [1]

1.1.1 Sledovanie pohybu očí pomocou techník počítačového videnia

Tento spôsob sledovania pohybu očí využíva kameru, ktorá sníma oči, a techniky počítačového videnia, ktoré slúžia na riešenie dvoch hlavných problémov:

- určovanie pozície očí vzhľadom k pozorovanej scéne
- sledovanie pohybu očí

Tieto problémy sa dajú riešiť rôznymi prístupmi. Jedným z nich sú techniky rozpoznávania obrazcov a redukcie dimenzie príznakov pre lepšiu detekciu oka.

Ďalší prístup využíva kruhový tvar zreničky a detekciu hrán. Najprv sa deteguje tvár, overí sa s geometrickým vzorom štruktúry tváre a následne sa aplikujú štyri Gaborove filtre s dosadenými hodnotami 0, $\pi/4$, $\pi/2$, $3\pi/4$ na oblasti s pravdepodobným umiestením očí. Niektoré systémy tiež využívajú algoritmus detekcie najdlhšej čiary alebo kruhovú Houghovu transformáciu.

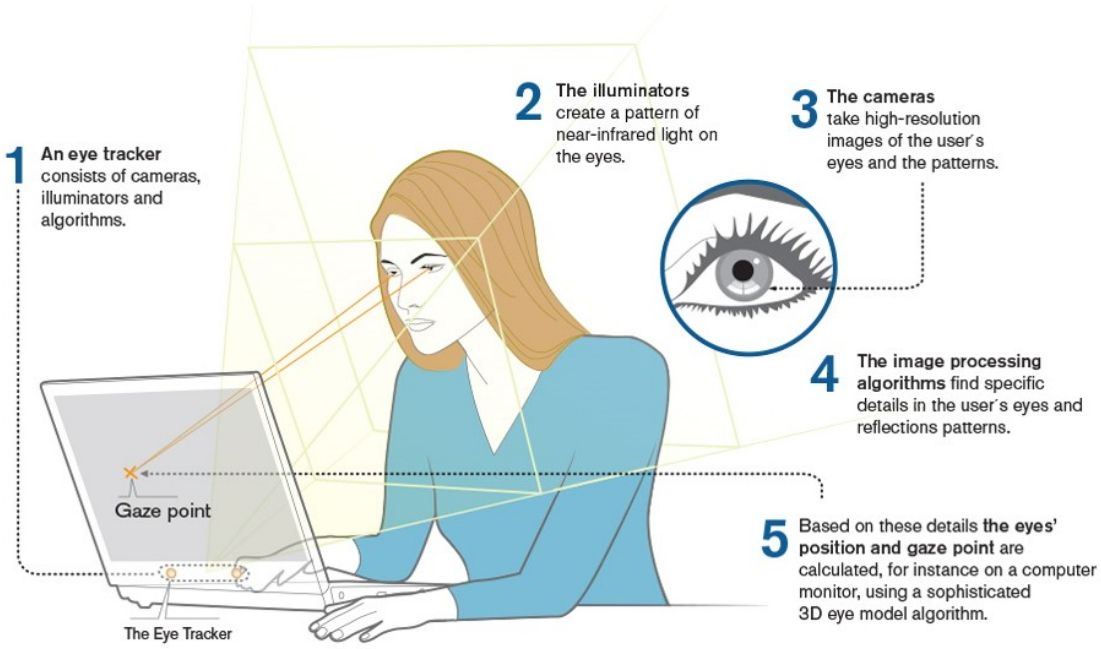
Na detekciu pohybu očí sa využíva aj určovanie polohy zreničky pomocou metódy tmavej zreničky alebo metódy svetlej zreničky. Metóda tmavej zreničky funguje na základe kontrastu medzi zreničkou a dúhovkou. Tento kontrast môže byť nízky pri veľmi tmavých očiach, čo môže spôsobovať problémy. Metóda svetlej zreničky eliminuje problém s kontrastom využitím odrazu infračerveného svetla od sietnice, vďaka čomu sa zrenička javí ako biela. [1]

1.2 Eye tracker

„Eye tracker“ je hardvérové zariadenie na sledovanie polohy a pohybu očí. Pri sledovaní pohybu očí pomocou techník počítačového videnia sa používajú kamery, ktorých postavenie môže byť rôzne. Kamera môže byť pripevnená k hlave pozorovateľa v blízkosti očí. Väčšinou sa využíva na sledovanie okolia, pretože umožňuje pozorovateľom voľne sa pohybovať. Druhá možnosť je fixná kamera, ktorá sa využíva na sledovanie obrazov na monitore. [1]

1.2.1 Tobii

Tobii je spoločnosť, ktorá bola založená v roku 2001 troma švédskymi podnikateľmi za účelom zlepšovať svet pomocou technológie, ktorá chápe ľudskú pozornosť. Tieto technológie predstavujú významný krok pre zdokonalenie interakcie medzi človekom a počítačom. [2] Využívajú na to umelú inteligenciu, strojové učenie a poznatky z oblasti počítačového videnia. Dnes je Tobii svetový líder v oblasti sledovania očí. Vyrábajú



Obr. 1.1: Screen-based eye tracker [4]

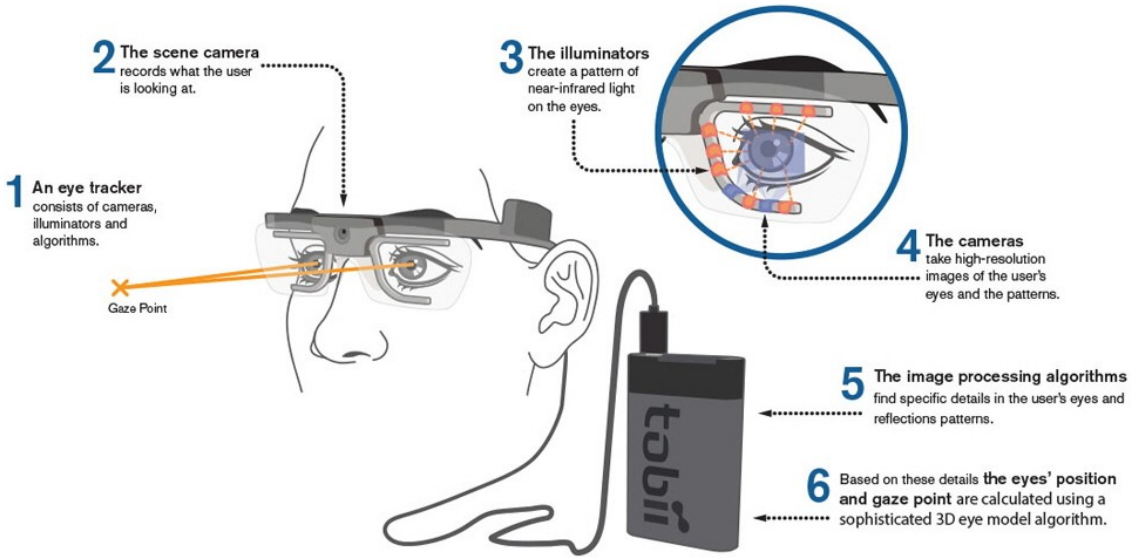
fixné, tzv. „screen-based“ eye trackery obr. 1.1, aj eye trackery vo forme okuliarov, tzv. „wearables“ eye trackery obr. 1.2. [3]

Na detekciu a sledovanie pohybov očí využívajú techniku „pupil center corneal reflection“ (PCCR), ktorej základom je osvetliť oko a zachytiť obraz oka. Tento obraz oka slúži na určenie stredu zrenice a odrazu osvetľovača na rohovke. Tieto poznatky sa používajú na výpočet pohľadu pozorovateľa.

Fixný (screen-based) eye tracker obr. 1.1 aj eye tracker vo forme okuliarov (wearable eye tracker) obr. 1.2 pozostáva z kamier, osvetľovačov a procesorovej jednotky obsahujúcej pokročilé algoritmy spracovania obrazu. Jedna kamera sníma pozorovanú scénu a ďalšie kamery snímajú oči. Osvetľovače vytvárajú na očiach vzor infračerveného svetla, kamery zhotovujú snímky očí a ich vzorov vo vysokom rozlíšení a algoritmy sa používajú na výpočet polohy oka a bodu pohľadu v priestore. [4]

Tobii Pro Glasses 3

V tejto práci pracujeme so zariadením Tobii Pro Glasses 3. Ide o „wearable“ eye tracker, teda eye tracker vo forme okuliarov. Eye tracker zaznamenáva rôzne typy dát. Pre účely tejto práce je dôležité video, ktoré zobrazuje scénu a zaznamenáva sa prednou kamerou umiestnenou na eye trackeri, a súbor, ktorý obsahuje informácie o jednotlivých bodoch pohľadu.



Obr. 1.2: Wearable eye tracker [4]

1.3 Eye trackingové dáta

Sledovaním pohybu očí dokážeme nielen identifikovať miesto, na ktoré majú ľudia upriamenú pozornosť v konkrétnom časovom okamihu, ale dokážeme aj zistiť, akú dlhú dobu sa venujú sledovaniu určitého miesta, a tiež zaznamenať trasu, ktorú sledujú ich oči. Pri tomto sledovní máme k dispozícii množstvo dát, ktoré je potrebné spracovať. [5]

1.3.1 Metriky

Pre vytvorenie úplného obrazu pozorovanej scény sa ľudské oko neustále pohybuje. Tento proces pozostáva z dvoch fáz - fixácie a sakády. Fixácia je proces, kedy sa pohyb oka na určitý čas preruší a pohľad sa zameria na konkrétnu oblasť zorného poľa. Tieto zastavenia bývajú veľmi krátke, zvyčajne trvajú 100 až 600 milisekúnd. Pohyby oka medzi jednotlivými fixáciami sa nazývajú sakády. Sakády pomáhajú zabezpečiť, aby oči zachytili kompletnú scénu toho, na čo sa pozorovateľ pozerá. Pomocou sakád dokážeme tiež určiť poradie pozorovania jednotlivých objektov. Fixácie a sakády nám poskytujú obraz o tom, ako jednotlivec vníma pozorovanú scénu.

Pre pochopenie správania pohybu očí pozorovateľa sú dôležité atribúty, ako sú poloha, trvanie a pohyb. Poloha sa zaznamenáva pomocou fixácií, ktoré poskytujú informácie o oblastiach, ktoré upútali pozornosť pozorovateľa a ktoré sleduje počas dlhšieho časového obdobia. Trvanie reprezentuje časové rozpätie, počas ktorého sa zaznamenáva fixácia na určitej oblasti. Pohyb očí pozorovateľa je dosiahnutý prostredníctvom sakád,

Obr. 1.3: Heat map [5]

teda rýchlym striedaním fixácií.

Fixácie a sakády sú bežné merania zachytené pri sledovaní očí. Medzi najbežnejšie metriky eye trackingu považujeme frekvenciu fixácie, teda ako často sa pozorovateľ na dané miesto pozeral, dĺžku fixácie, teda ako dlho sa na dané miesto alebo objekt pozeral, atď. [5]

1.3.2 Vizualizácia dát

Eye trackery sú vybavené softvérovými balíkmi, ktoré umožňujú rýchle vytváranie vizualizácií údajov získaných z eye trackingu. Tieto vizualizácie zachytávajú správanie pohybu očí prostredníctvom troch atribútov, ktoré sme spomenuli vyššie. Môžeme teda pomocou nich detegovať oblasti záujmu, zistiť, na aké oblasti sa pozorovateľ pozeral, ako dlho sa na tieto oblasti pozeral (trvanie fixácie), ako často (frekvencia fixácie) a na akú oblasť sa pozrel potom (poradie fixácií). Medzi najpoužívanejšie vizualizácie eye trackingových dát patrí „heat map“ (tepelná mapa) obr. 1.3 a „gaze plot“ (graf sekvencií pohľadu) obr. 1.4. [5]

Heat map

Pri vizualizácii pomocou tepelnej mapy obr. 1.3 sa využívajú farby, ktoré reprezentujú počet fixácií alebo dĺžku fixácií v jednotlivých oblastiach. Červenou farbou sa zaznamenávajú oblasti s vysokým počtom fixácií alebo dlhou dobou fixácie. Sú to oblasti s

Obr. 1.4: Gaze plot [5]

vysokou úrovňou záujmu. Oblasti označené zelenou farbou predstavujú oblasti s najmenším počtom fixácií, a teda aj s najmenšou úrovňou záujmu. V tepelných mapách sa nachádzajú aj oblasti bez farby, čo nemusí nutne znamenať, že sa na túto oblasť pozorovateľ nepozrel. Oblasť bez farby mohol vidieť periférne alebo sa na ňu pozrel na veľmi krátku dobu, neprišlo k fixácii a eye tracker tento pohľad nezaznamenal. [5]

Gaze plot

Graf sekvencií pohľadu obr. 1.4 je druh vizualizácie, ktorá zaznamenáva poradie fixácií a sakády medzi jednotlivými fixáciami. Fixácie sú znázornené bodkami, môžu byť očíslované a môžu mať rôznu veľkosť. Číslo určuje poradie fixácie a veľkosť bodky je priamo úmerná trvaniu fixácie. Sakády sú znázornené čiarou spájajúcou dve bodky. Tento druh vizualizácie je užitočný pri skúmaní cesty, ktorou sa oči pohybovali. [5]

1.4 Rozšírená realita

Rozšírená realita (AR) je technológia, ktorá kombinuje skutočný svet s digitálnymi dátami. Ide o zobrazenie reality, ktorá je rozšírená o prvky grafickej povahy. Rozšírená realita spája oblasti počítačového videnia a počítačovej grafiky. Počítačové videnie vo vzťahu k AR zahŕňa detekciu a sledovanie markerov, sledovanie pohybu, analýzu obrazu a mnohé ďalšie. Počítačová grafika zahŕňa napríklad fotorealistické vykresľovanie objektov. Ronald Azume [6] definoval rozšírenú realitu ako systém, ktorý:

- kombinuje skutočné a virtuálne
- je interaktívny v reálnom čase

- je registrovaný v 3D

Jednoduchý systém rozšírenej reality sa skladá z kamery, výpočtovej jednotky a displeja. Kamera zachytáva obraz prostredia a výpočtová jednotka vypočíta správnu polohu a orientáciu objektov, na základe toho vykreslí virtuálne objekty a zobrazí výsledok na displeji.

S vývojom technológií bolo možné preniesť rozšírenú realitu aj na lacné a bežne dostupné zariadenia, ako sú mobilné telefóny. Vďaka tomu je na vrchole svojej popularity, stáva sa bežnou súčasťou spotrebiteľských aplikácií a nové aplikácie rozšírenej reality neustále pribúdajú. Používa sa na vizualizáciu interiéru aj exteriéru, v interiérovom dizajne si používatelia môžu otestovať, ako sa k sebe hodia jednotlivé kusy nábytku. Aplikácie s rozšírenou realitou sú užitočné aj pri montáži a údržbe na zobrazovanie pokynov. Najväčšie využitie má v hernom priemysle, pretože umožňuje interakciu s hráčom a prostredím a prináša nové spôsoby ovládania pomocou polohy a 3D pohybu, čím zlepšuje herný zážitok a hry sa zdajú byť atraktívnejšie. [6]

Podľa [7] poznáme 2 základné typy rozšírenej reality, v závislosti od toho, či používajú alebo nepoužívajú značku:

- Marker augmented reality - rozšírená realita založená na značkách. V tejto práci využívame práve tento typ rozšírenej reality, a preto sa jej podrobnejšie venujeme v nasledujúcej časti.

- Markerless augmented reality - rozšírená realita bez značiek. Tento typ rozšírenej reality by sa dal nazvať aj ako Location-based AR, teda rozšírená realita založená na polohe. Objekty do reálneho sveta umiestňuje na základe geografickej polohy - zemepisnej šírky, zemepisnej dĺžky a nadmorskej výšky.

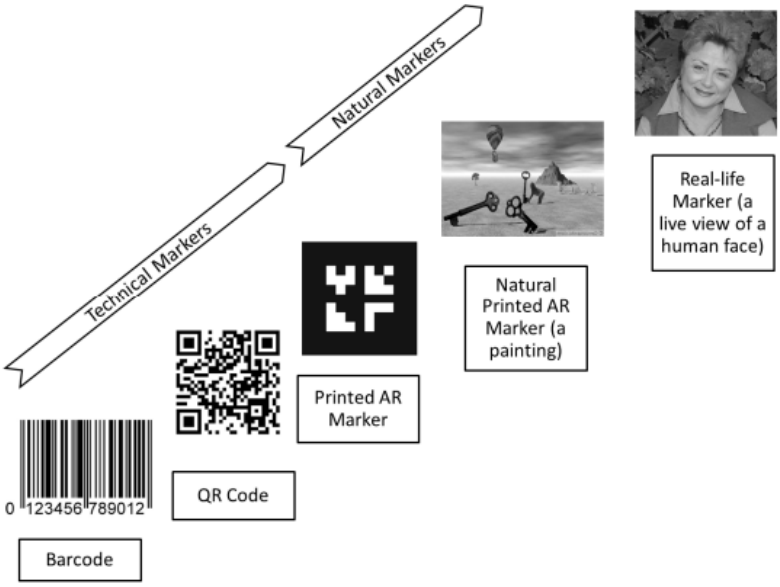
V inej literatúre sa toto rozdelenie môže mierne líšiť.

1.4.1 Rozšírená realita založená na značkách

Značka (marker) v kontexte rozšírenej reality je znak, ktorý je v obraze ľahko detekovateľný metódami spracovania obrazu, rozpoznávania vzorov a technikami počítačového videnia. Po detekcii tejto značky systém dokáže vypočítať polohu a orientáciu kamery a na základe toho vykresliť virtuálny objekt na vrch značky. [6]. Označuje sa aj ako rozšírená realita založená na rozpoznávaní.

S vývojom technológie rozpoznávania obrazu pribúdajú nové podoby značiek. Na obr. 1.5 sú zobrazené rôzne typy značiek ako aj vývoj ich rozpoznávania. V aplikáciách rozšírenej reality sa dajú všetky typy značiek použiť rovnakým spôsobom, ale náročnosť ich detekcie sa líši. [7]

Jadro systému rozšírenej reality tvorí sledovací modul, ktorý počíta relatívnu polohu kamery v reálnom čase, teda 3D umiestnenie a 3D orientáciu objektu a práve



Obr. 1.5: Príklady značiek a vývoj značiek v technológii rozpoznávania obrazu [7]

použitie značiek poskytuje najjednoduchší spôsob na tento výpočet. 3D objekty sú vykreslené pomocou renderovacieho modulu. V počítačovej grafike sa virtuálna scéna premieta na obrazovú rovinu pomocou virtuálnej kamery a táto projekcia sa potom vykresľuje. V prípade rozšírenej reality je kľúčové použitie virtuálnej kamery, ktorá je identická so skutočnou kamerou systému. Tým sa dosiahne, že virtuálne objekty sa na scéne premietajú podobne ako reálne objekty, čo vytvára presvedčivý výsledok. Pre napodobnenie skutočnej kamery musí systém poznať optické vlastnosti kamery, čo sa nazýva kalibrácia kamery. [6]

Kalibrácia kamery

Kalibrácia kamery je proces, pri ktorom sa zisťujú optické vlastnosti kamery. Zisťujú sa ňou dva typy parametrov kamery: vnútorné a vonkajšie. V tejto práci používame metódu kalibrácie zo známej scény s použitím kalibračného objektu so známou geometriou, konkrétne so šachovnicou. [1]

Kamery spôsobujú skreslenie obrazu, a preto je dôležité pri kalibrácii kamery zistiť okrem vnútorných a vonkajších parametrov aj päť parametrov skreslenia. Označujú sa ako koeficienty skreslenia. Konkrétne ide o tri koeficienty radiálneho skreslenia a dva koeficienty tangenciálneho skreslenia. Radiálne skreslenie spôsobuje zakrivenie rovných čiar, pričom skreslenie je výraznejšie smerom k okrajom obrazu. Tangenciálne skreslenie spôsobuje, že niektoré časti obrazu sa javia bližšie, než sú v skutočnosti.

Vnútorné parametre kamery sú špecifické pre konkrétnu kameru a zahŕňajú ohniskovú vzdialenosť (f_x, f_y) a optický stred (c_x, c_y). Tieto parametre sa využívajú na

vytvorenie matice kamery (1), ktorá je použitá na odstránenie skreslenia kamery spôsobeného šošovkami kamery. Medzi vonkajšie parametre patria vektory rotácie a vektory translácie, ktoré transformujú 3D súradnice bodu do súradnicového systému kamery. [8]

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

Detekcia značky a výpočet polohy

Pri detekcii značky je prvým krokom nájdenie obrysov potenciálnych značiek a ich rohov. Následne detekčný systém overí, že skutočne ide o značku a identifikuje ju. Predspracovanie obrazu sa vykonáva jednou z metód spracovania obrazu, napr. prahovanie, detekcia hrán, atď. Pri hľadaní značiek systém overuje, či potenciálne značky majú presne štyri rovné čiary a štyri rohy. Na overenie, že systém skutočne identifikoval značku, sa využívajú ďalšie metódy, ktoré závisia od podoby značky.

Výpočet polohy kamery znamená nájsť jej umiestnenie a orientáciu vo svetových súradniciach. Poloha je vyjadrená tromi translačnými súradnicami (x, y, z) a orientácia troma rotačnými uhlami (α, β, γ) okolo súradnicových osí. Poloha kalibrovanej kamery sa dá jednoznačne určiť z aspoň štyroch bodov. Teda polohu značky v 3D súradniciach vzhľadom na fotoaparát dokáže systém vypočítať pomocou štyroch rohových bodov značky v súradniciach obrazu.

Transformácia $\mathbf{T}$ medzi kamerou a značkou sa nazýva transformačná matica kamery a je vyjadrená ako (2), kde $\mathbf{X}$ je bod vo svetových súradniciach a $\mathbf{x}$ je jeho projekcia v súradniciach obrazu. Pozostáva z vektora translácie $\mathbf{t}$ a 3x3 rotačnej matice $\mathbf{R}$. V homogénnych súradniciach sa dá vyjadriť ako (3).

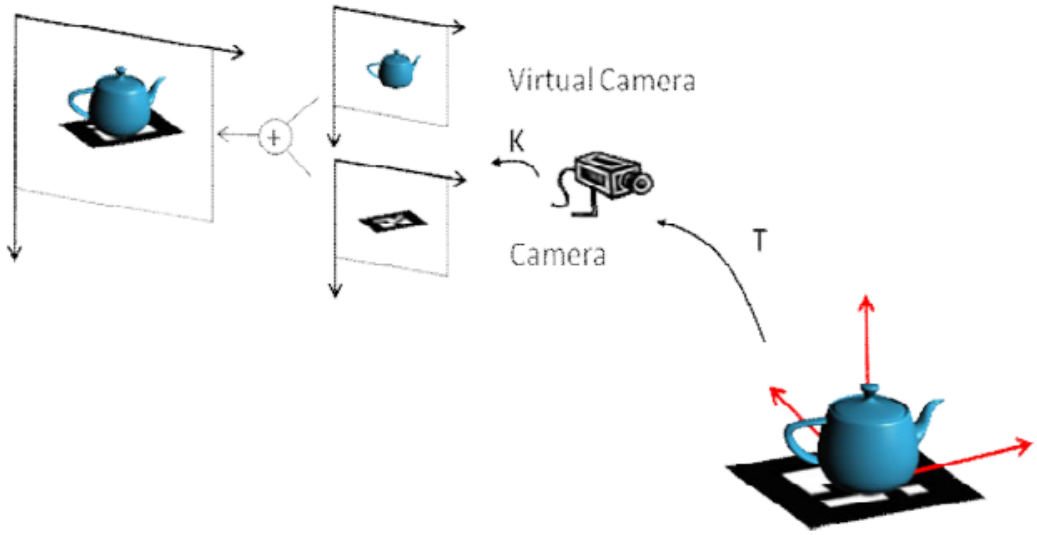
$$\mathbf{x} = \mathbf{TX} \tag{2}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r_1 & r_2 & r_3 & t_x \\ r_4 & r_5 & r_6 & t_y \\ r_7 & r_8 & r_9 & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

Pre výpočet polohy je tiež potrebná kalibračná matica kamery $\mathbf{K}$ (4) získaná kalibráciou kamery.

$$K = \begin{bmatrix} f_x & 0 & p_x & 0 \\ 0 & f_y & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4}$$

Body v súradniciach kamery sú označené $\mathbf{x}$ a im prislúchajúce body vo svetových súradniciach sú označené $\mathbf{X}$. Detekciou značky získame štyri rohové body v súradniciach



Obr. 1.6: Rozšírená realita založená na značkách [6]

obrazu: $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}$. Pre každý tento bod $\mathbf{x}_\mathrm{i}$, $i = 1, 2, 3, 4$ platí (5), čo sa dá vyjadriť ako (6).

$$\mathbf{x}_i = \mathbf{KTX}_i \tag{5}$$

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & t_x \\ r_4 & r_5 & r_6 & t_y \\ r_7 & r_8 & r_9 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{6}$$

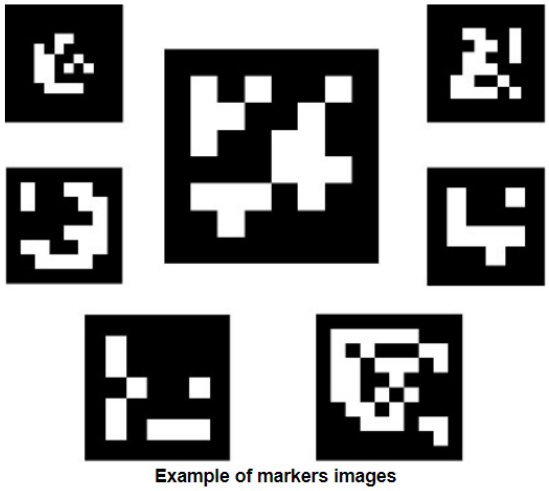
Vypočítaná poloha kamery slúži na vykresľovanie virtuálneho objektu v správnej perspektíve. Virtuálny objekt sa vykresľuje pomocou transformačnej matice T a matice kamery K v orientácii, ktorá zodpovedá súradnicovým osiam obr. 1.6. [6]

Pre podrobnejšie informácie o detekcii značky a vysvetlení výpočtu polohy odporúčame knihu [6].

1.4.2 ArUco značky

ArUco značky sú jedny z najpoužívanejších typov značiek, pretože ich štyri rohy poskytujú dostatok korešpondencie medzi bodmi v reálnom prostredí a ich 2D projekciou obrazu na získanie polohy kamery. Majú tvar štvorca a sú zložené zo širokého čierneho okraja a vnútornej binárnej matice obr. 1.7. Čierny okraj umožňuje rýchlu detekciu značky v obraze a binárna matica určuje identifikátor značky.

Značky použité v danej aplikácii sú uložené v slovníku. Identifikátor značky je index danej značky uloženej v slovníku. Veľkosť slovníka je určená počtom značiek, ktoré sú v ňom uložené, a veľkosť značky určuje veľkosť vnútornej matice značky. [9]



Obr. 1.7: Príklad ArUco značiek [9]

Na implementáciu rozšírenej reality sme sa rozhodli použiť ArUco značky kvôli jednoduchej práci s modulom aruco a využívame preddefinované slovníky, ktoré modul aruco obsahuje.

1.4.3 OpenCV

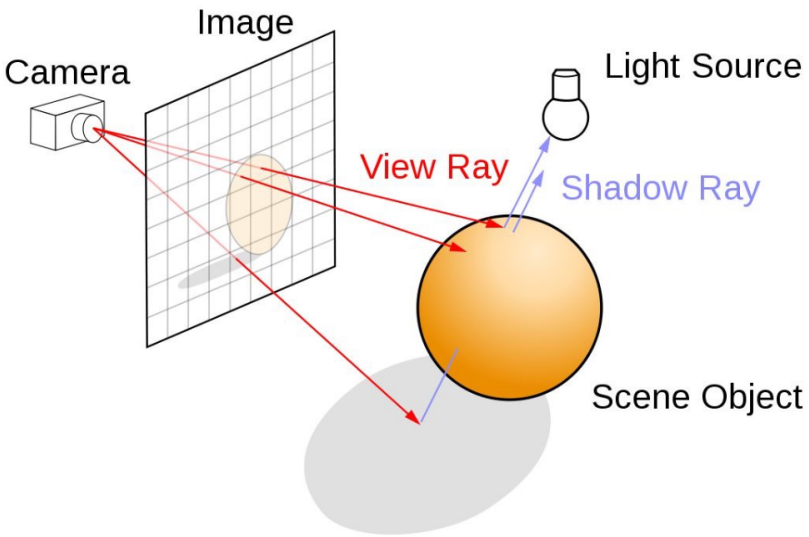
OpenCV je open-source knižnica, ktorú založil Gary Bradsky v Inteli v roku 1999, a prvá verzia bola uvedená na trh v roku 2000. Obsahuje veľké množstvo algoritmov z oblasti počítačového videnia a strojového učenia. Knižnica je dostupná na rôznych platformách a podporuje rôzne programovacie jazyky, ako napríklad C++, Python a Java. Python je populárny programovací jazyk vďaka svojej jednoduchosti a prehľadnosti kódu, preto v tejto práci pracujeme s knižnicou OpenCV v jazyku Python. OpenCV-Python tiež využíva knižnicu Numpy používanú pre rýchle numerické operácie. [10]

1.5 Homografia

Homografia je perspektívna transformácia medzi dvoma 2D obrazmi, ktorá mapuje pixely zo zdrojového obrazu na pixely cieľového obrazu podľa vzťahu (7). V tomto vzťahu je x=(u,v,1) bod v zdrojovom obraze, x'=(u',v',1) je bod v cieľovom obraze a homografia je definovaná maticou M (8) veľkosti 3x3.

Na výpočet homografie sú potrebné aspoň štyri páry korešpondujúcich bodov, teda je potrebné poznať štyri body zo zdrojového obrazu a k nim korešpondujúce body cieľového obrazu. [11]

$$\mathbf{x}' = \mathbf{M}\mathbf{x} \tag{7}$$



Obr. 1.8: Metóda Ray casting [12]

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \tag{8}$$

Pre viac informácií o homografii odporúčame [11].

1.6 Ray casting

Ray casting je metóda z oblasti počítačovej grafiky na vykresľovanie 3D scén do 2D obrazu. Základom tejto metódy je zostrojenie lúča do 3D scény smerujúceho z kamery cez každý pixel v rovine obrazu. Následne sa overuje, či tento lúč pretína nejaký objekt v scéne. Ak existuje prienik lúča s objektom, vypočíta sa vzdialenosť lúča od kamery k objektu a získa sa informácia o farbe objektu, ktorá určuje farbu pixelu. Obrázok 1.8 ilustruje fungovanie metódy ray castingu. [12]

1.7 Podobné práce

Patrik Modrovský sa vo svojej bakalárskej práci [13] zaoberal registráciou 3D skenov a chybami na týchto skenoch. Jeho cieľom bolo navrhnúť a vytvoriť nástroj na zaznamenávanie spôsobu, akým ľudia sledujú objekty pri kontrole, a vytvorené dáta spracovať pre ďalšie využitie. 3D objekt nezobrazoval v rozšírenej realite, ale na sledovanie a otáčanie objektu využíval Unity. V rámci jeho práce sa mu nepodarilo vizualizovať pohľad pozorovateľa na 3D objekt. Hoci sú tieto dve práce odlišné, využíval podobné technológie ako v tejto práci, konkrétne eye tracker a spracovával eye trackingové dáta.

Kapitola 2

Návrh

Praktická časť práce zahŕňa tieto časti:

- vytvorenie 3D modelu a prostredia na pozorovanie
- vytvorenie a zbieranie dát
- spracovanie dát
- vizualizácia dát

Na obrázku 2.1 je grafické znázornenie jednotlivých častí praktickej časti práce, ktoré definuje technológie použité v každej časti, opisuje ich funkcie a prenos dát medzi nimi.

V nasledujúcich podkapitolách bližšie popíšeme jednotlivé časti a postupne vysvetlíme fungovanie praktickej časti práce.

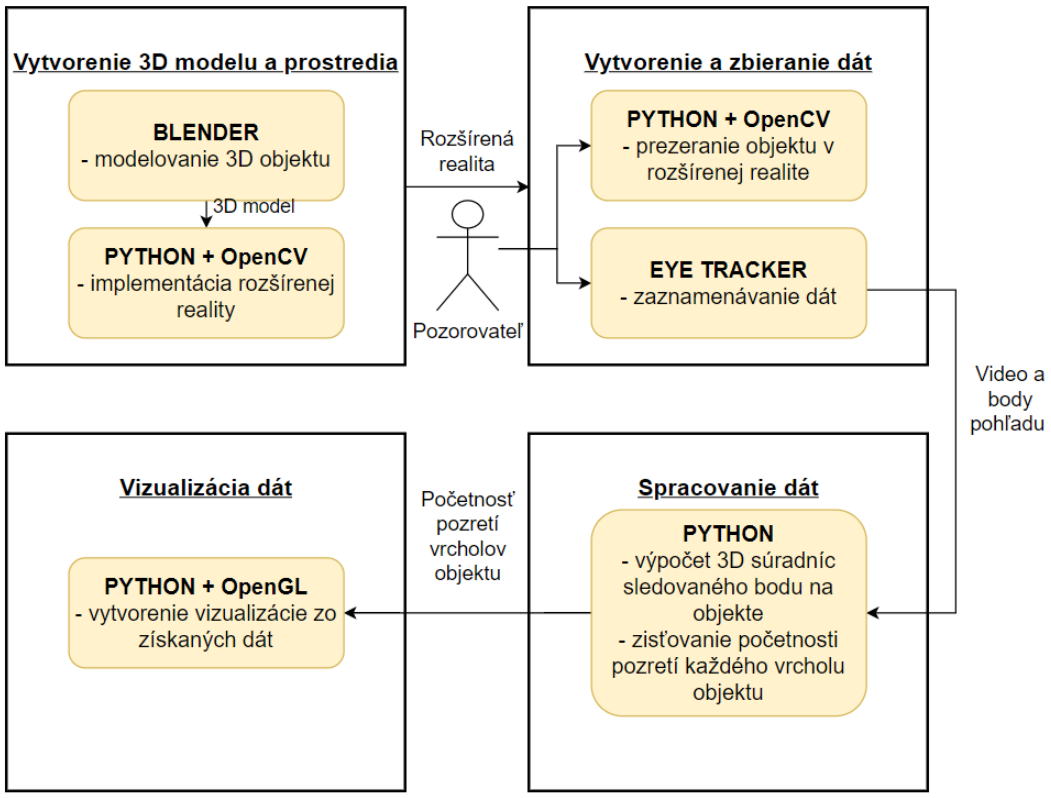
2.1 Vytvorenie 3D modelu a prostredia na pozorovanie

Ako prvé je potrebné vytvoriť 3D model, ktorý bude pozorovateľ skúmať. Na vytvorenie 3D modelu sme zvolili Blender, čo je aplikácia na 3D modelovanie.

Pozorovateľ sleduje 3D objekt v rozšírenej realite založenej na značkách, ktorá je implementovaná v Pythone s využitím knižnice OpenCV. 3D model sa po detekcii značky nezobrazuje priamo na značke, ale je posunutý, aby bola značka stále viditeľná. To je potrebné pre opakovanú detekciu značky na videu z eye trackera.

2.2 Vytvorenie a zbieranie dát

Táto časť zahŕňa pozorovanie 3D objektu a zaznamenávanie sledovaných bodov eye trackerom. Na vytvorenie a zbieranie dát je potrebné spustiť Python aplikáciu, ktorá



Obr. 2.1: Jednotlivé časti aplikácie, ich funkcie a prenos dát medzi nimi

implementuje rozšírenú realitu, a eye tracker, ktorý zaznamenáva dáta potrebné na ďalšie spracovanie. Medzi tieto dáta patrí video nasnímané kamerou eye trackera a súbor s dátami, ktorý obsahuje informácie o jednotlivých bodoch pohľadu.

2.3 Spracovanie dát

V tejto časti sa spracovávajú dáta získané z eye trackera. Video aj súbor s dátami sa načítavajú do Python aplikácie, ktorá vo videu znovu zdeteguje značku, čím vypočíta polohu a orientáciu 3D objektu zobrazeného na videu, a spolu s dátami o sledovanom bode vypočíta súradnice bodu objektu, na ktorý sa pozorovateľ pozeral.

2.4 Vizualizácia dát

Po zozbieraní a spracovaní dát vieme tieto dáta vizualizovať. Vizualizácia je vytvorená v Pythone s využitím knižnice OpenGL a má podobu tepelnej mapy zobrazenej na 3D objekte. Na tejto mape sú farebne odlíšené zaujímavé oblasti, ktoré pútali najviac pozornosti pozorovateľa. Aby bolo možné objekt prehliadať z rôznych strán, vizualizácia je vytvorená v interaktívnej podobe, čo umožňuje objekt otáčať okolo súradnicových osí, a tiež obsahuje grafické používateľské rozhranie na výber nastavení vizualizácie.

Kapitola 3

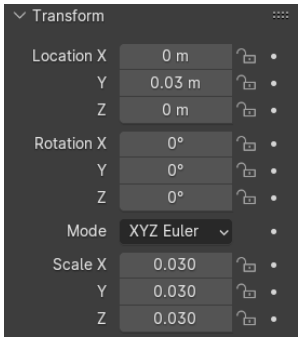
Implementácia

Táto kapitola obsahuje detailné informácie o postupe implementácie a funkcionalite aplikácie spolu so všetkými implementačnými detailami.

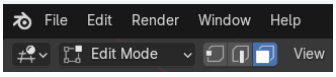
3.1 Vytvorenie 3D modelu

Na vytvorenie 3D modelu sme použili Blender, čo je aplikácia na 3D modelovanie. Na začiatok sme ako podobu sledovaného objektu zvolili obyčajnú kocku. Aby sa kocka správne zobrazovala, je potrebné upraviť jej veľkosť a pozíciu. Veľkosť kocky sme upravili škálovaním v každom smere (x, y, z) hodnotou 0,030. Kocku je tiež treba posunúť v smere osi y o hodnotu 0,03, aby bol bod (0,0,0), teda začiatok globálneho súradnicového systému, v strede podstavy kocky. To je dôležité pre správne vykreslenie objektu na značke v implementácii rozšírenej reality. Na obrázku 3.1 sú znázornené požadované nastavenia objektu v Blenderi.

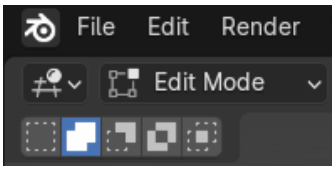
Kocka sa skladá z 8 rohov a 6 plôch (strán). Pre vytvorenie hustejšej siete objektu *(mesh)* je potrebné použiť metódu *subdivide*. Najprv je potrebné zvoliť režim úprav *(Edit mode)* a režim pre výber plôch *(Select Mode - Face select)* obr. 3.2 a následne označiť všetky plochy objektu. To sa dá dosiahnuť viacerými spôsobmi. Jedným z možných postupov je postupne označiť všetky plochy objektu pomocou klávesy Ctrl



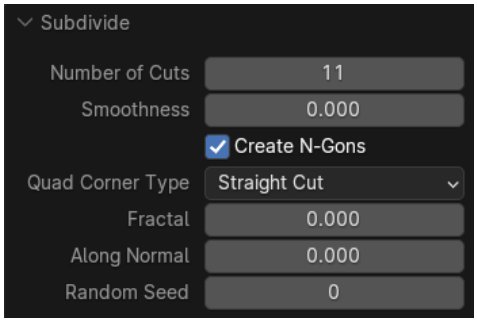
Obr. 3.1: Nastavenie škálovania a polohy 3D objektu v Blenderi



Obr. 3.2: Nastavenie režimu úprav a výber plôch pre vytvorenie hustejšej siete objektu



Obr. 3.3: Nastavenie režimu na rozšírenie existujúceho výberu plôch objektu



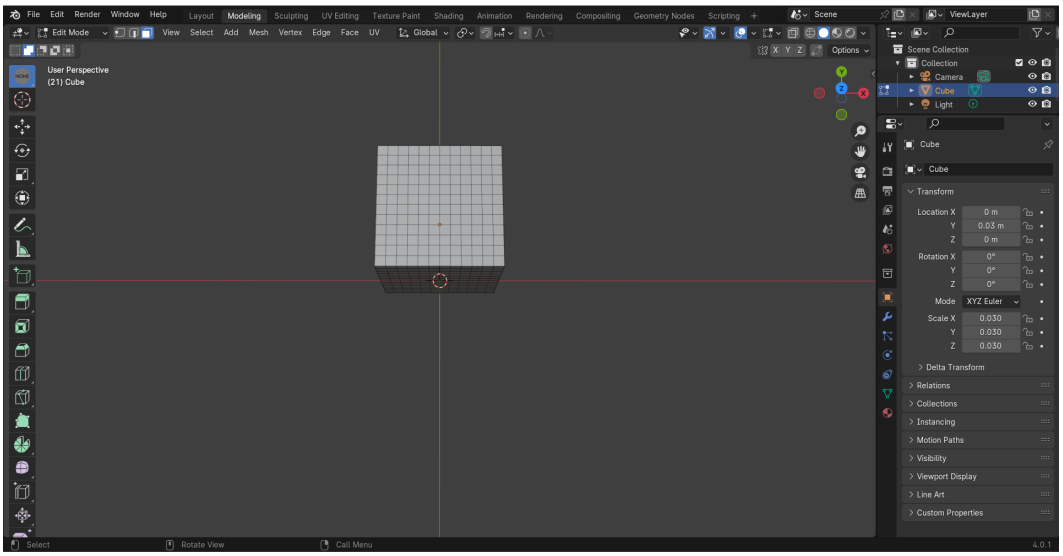
Obr. 3.4: Možnosti metódy subdivide a nastavenie počtu rezov pre vytvorenie hustejšej siete objektu

a súčasným kliknutím na danú plochu, čo môže byť prácne, najmä pri objektoch s veľkým počtom plôch. Ďalší spôsob umožňuje výber viacerých plôch naraz, čo je časovo efektívnejšie. Základ tvorí nastavenie režimu na rozšírenie existujúceho výberu *(Mode - Extend existing selection)* obr. 3.3. Po zvolení tohto režimu je možné kliknutím a ťahaním označiť viaceré plochy súčasne.

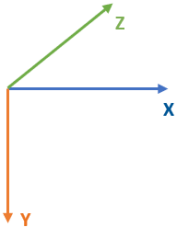
Po označení všetkých plôch objektu vieme pravým kliknutím na objekt zvoliť metódu *subdivide*. Táto metóda poskytuje viaceré možnosti, pre potreby v našej práci sme nastavovali iba počet rezov *(Number of Cuts)* obr. 3.4. Na obrázku 3.5 je znázornená výsledná podoba objektu spolu so všetkými nastaveniami v Blenderi použitá pre ďalšiu prácu.

Po vytvorení modelu je potrebné tento model uložiť do súboru pre ďalšie použitie. Zvolili sme export do súboru s príponou .obj, ktorý obsahuje informácie o 3D geometrii modelu. Tento súbor je potrebný pre neskoršie načítanie a vykreslenie 3D modelu v rozšírenej realite.

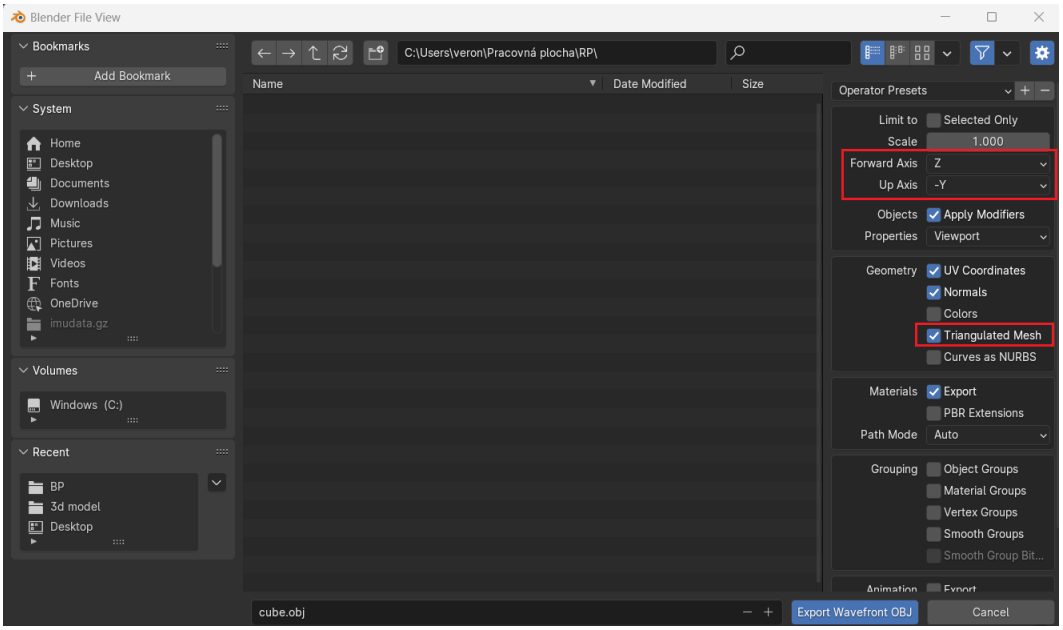
Základom pre správne vykresľovanie objektu v rozšírenej realite je správne určenie orientácie súradnicových osí. Pri implementácii rozšírenej reality v OpenCV je nevyhnutné pri exporte objektu nastaviť orientáciu osí zodpovedajúcu orientácii osí v OpenCV obr. 3.6. Pre vytvorenie trojuholníkovej siete je potrebné vo fáze exportu zvoliť možnosť *Triangulated Mesh*, čím sa vytvorí sieť pozostávajúca zo sady trojuholníkov. Nastavenia exportu objektu sú znázornené na obrázku 3.7.



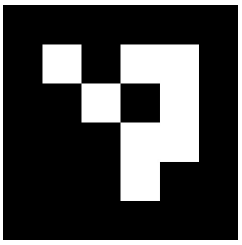
Obr. 3.5: Výsledná podoba objektu so všetkými nastaveniami v Blenderi



Obr. 3.6: Súradnicový systém v Opencv



Obr. 3.7: Nastavenia pri exporte objektu



Obr. 3.8: ArUco značka s identifikátorom 0 použitá v implementácii rozšírenej reality

3.2 Implementácia rozšírenej reality

Rozšírenú realitu sme implementovali v jazyku Python s využitím knižnice OpenCV vo verzii 4.6.0. Pri iných verziách sa môžu podoby funkcií mierne líšiť. Implementácia pozostáva z nasledujúcich hlavných krokov:

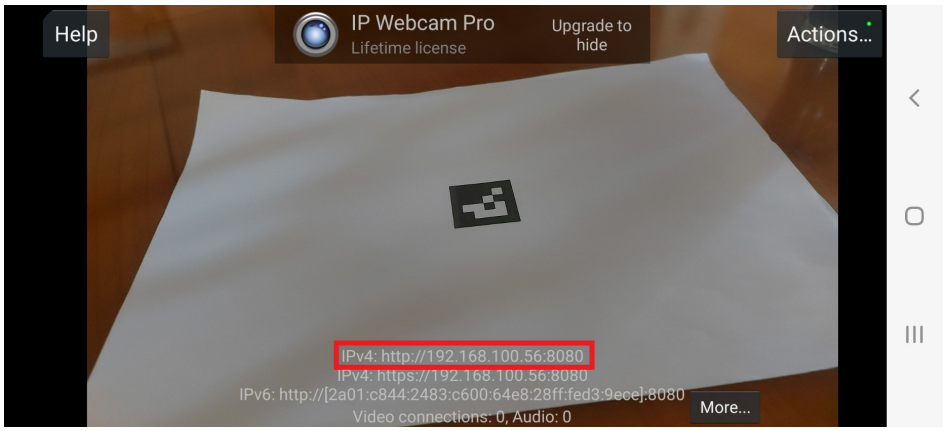
- Vygenerovanie značky
- Získavanie obrazu z kamery
- Detekcia značky
- Výpočet „pose estimation“ (odhad polohy)
- Vykresľovanie objektu

3.2.1 Vygenerovanie ArUco značky

Na generovanie ArUco značiek existuje v OpenCV funkcia *cv2.aruco.drawMarker(dict, id, size)*. Parameter *dict* určuje typ slovníka, *id* určuje identifikátor značky, ktorú chceme vygenerovať, a *size* určuje veľkosť značky v pixeloch. V našej práci používame preddefinovaný slovník *cv2.aruco.DICT_4X4_50*, čo znamená, že vnútorná matica značky je veľkosti 4x4, a pomocou tohto slovníka vieme vygenerovať 50 jedinečných značiek. Z tohto slovníka sme vygenerovali značku s identifikátorom 0 a veľkosťou 500 pixelov obr. 3.8, ktorú sme následne použili pri ďalšej implementácii rozšírenej reality. Z tohto istého slovníka sme tiež vygenerovali ďalšie štyri značky s identifikátorom 1-4, ktoré sme neskôr pridali do rohov obrazu získaného z kamery.

3.2.2 Získavanie obrazu z kamery

Získavanie obrazu z kamery notebooku spôsobovalo, kvôli zlej kvalite videa a nedostatočnému rozlíšeniu kamery, na videu z eye trackera problémy s opätovnou detekciou značky zobrazenej na monitore notebooku. Preto sme sa na získavanie obrazu rozhodli použiť kameru mobilného telefónu. Vyhli sme sa tým aj potenciálnym problémom, kedy by eye tracker mohol zaznamenať tú istú značku dvakrát. Teda vytlačenú značku



Obr. 3.9: Snímka z aplikácie IP Webcam, na ktorej je zobrazený videostream z kamery. Červeným orámovaním je vyznačená URL adresa slúžiaca na získavanie obrazu z kamery

umiestnenú v priestore slúžiacu na implementáciu rozšírenej reality a tú istú značku zachytenú kamerou notebooku a zobrazenú na monitore. V našom prípade bolo rozlíšenie kamery notebooku 1280 x 720px a maximálne rozlíšenie kamery telefónu je 64 Mpx, čo je približne 6936 x 9248px.

Na pripojenie kamery sme použili aplikáciu IP Webcam nainštalovanú v telefóne. Je potrebné, aby telefón aj notebook, na ktorom sa obraz z kamery zobrazuje, boli pripojené k rovnakej sieti. Videostream sa v aplikácii spustí po zvolení možnosti *Start server*. Na obrazovke sa zobrazí URL adresa obr. 3.9, ktorú je potrebné vložiť do programu v Pythone. Okrem toho je potrebné importovať knižnicu *requests* a na získanie dát z URL adresy použiť funkciu *requests.get(url)*. Parameter *url* je reťazec obsahujúci URL adresu doplnený o reťazec */shot.jpg*.

Po získaní obrazu z kamery telefónu sme do rohov obrazu umiestnili štyri vygenerované značky s identifikátormi 1 - 4, ktoré slúžia v ďalšej implementácii na nájdenie okna s rozšírenou realitou vo videu z eye trackera.

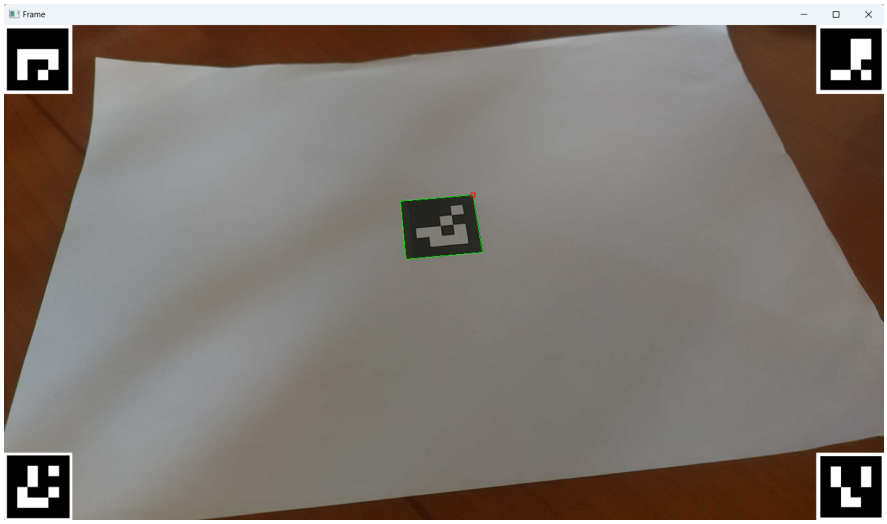
Týmto spôsobom sme získali videostream z kamery telefónu s pridanou značkou v každom rohu a zobrazili ho na monitore notebooku obr. 3.10.

3.2.3 Detekcia značky

Ďalším krokom pri implementácii rozšírenej reality je detekcia značky v obraze. V tomto kroku nás zaujíma iba značka slúžiaca na implementáciu rozšírenej reality, teda detegujeme značku s identifikátorom 0. Využívame na to funkcie z knižnice OpenCV. Funkcia *cv2.aruco.detectMarkers(frame, dict, parameters = arucoParams)* na detekciu značky vracia tri hodnoty: *corners* - zoznam súradníc rohov detegovaných značiek, *ids* - identifikátory zistených značiek, *rejected* - zoznam zamietnutých značiek.

Ako parametre sa tejto funkcii posielajú tri hodnoty: *frame* - snímka, na ktorej

Obr. 3.10: Obraz z kamery telefónu doplnený o značku v každom rohu a zobrazený na monitore notebooku



Obr. 3.11: Kontrola detekcie značky s identifikátorom 0

chceme detegovať značku, *dict* - slovník, ktorý používame a získame ho pomocou funkcie *cv2.aruco.Dictionary_get(cv2.aruco.DICT_4X4_50)*, a *arucoParams* - parametre používané na detekciu, zvyčajne sa využívajú predvolené, získané pomocou funkcie *cv2.aruco.DetectorParameters_create()*.

Detekciu značky skontrolujeme pomocou funkcie *cv2.aruco.drawDetectedMarkers (frame, corners)* obr. 3.11, pričom *frame* definuje snímku, na ktorej budú značky vykreslené, a *corners* definuje zoznam súradníc rohov značky získaný z predošlej funkcie na detekciu značky.

3.2.4 Odhad polohy

Po detekcii značky vieme použiť funkciu *cv2.aruco.estimatePoseSingleMarkers(corners, size, calibration_matrix, distortion_coefficients)* na odhad polohy kamery. Parameter *corners* definuje súradnice rohov značky, ktoré sme získali pri detekcii značky, a *size* určuje veľkosť značky v metroch. Zvyšné parametre *calibration_matrix* a *distortion_coefficients* sme získali z kalibrácie kamery. Funkcia vracia rotačné a translačné vektory, ktoré sú neskôr použité na správne vykreslenie objektov. Aby bola značka stále viditeľná a objekt sa nevykresľoval priamo na značku, posunuli sme objekt v kladnom smere osi x. Teda hodnotu x translačného vektora sme zväčšili, konkrétne o hodnotu 0,1.

Kalibrácia kamery

Kalibráciu kamery telefónu sme vykonali v Pythone a využili sme knižnicu OpenCV. Ako prvé sme vytlačili šachovnicu a nasnímali ju kamerou telefónu z rôznych uhlov a vzdialeností. Obrázky nasnímané kamerou a použité na kalibráciu kamery sú na obrázku 3.12

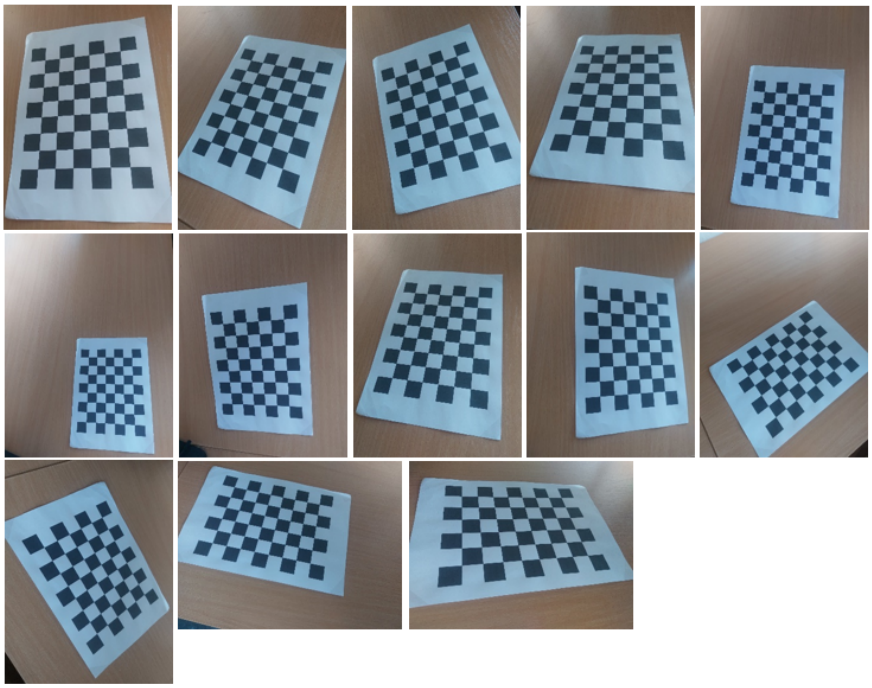
Pri kalibrácii kamery je dôležité načítať obrázky šachovnice a zadefinovať veľkosť šachovnice, v našom prípade je to veľkosť 8x6. Následne je potrebné spustiť kód na kalibráciu kamery. Pre podrobnejšie informácie o kalibrácii odporúčame [8] a pre podrobnosti o kóde na kalibráciu kamery odporúčame [14].

Kalibráciou kamery sme získali maticu kamery (*calibration_matrix*) a koeficienty skreslenia (*distortion_coefficients*), ktoré sme použili vo funkcii na odhad polohy. Získané údaje sme pomocou knižnice NumPy a funkcií *np.save("calibration_matrix", calibration_matrix)* a *np.save("distortion_coefficients",distortion_coefficients)* uložili do súborov vo formáte .npy.

3.2.5 Vykresľovanie objektu

Ďalším krokom je vykreslenie 3D objektu do scény. Objekt je potrebné načítať zo súboru. V súbore typu *obj*, s ktorým pracujeme, sú uložené údaje o 3D geometrii objektu, konkrétne 3D súradnice jednotlivých vrcholov, ktoré tvoria sieť objektu, súradnice textúry, normály vrcholov a informácie o plochách objektu. Pre našu implementáciu je dôležité načítať súradnice vrcholov a plochy objektu, ktoré sú určené indexami vrcholov, ktoré plochu tvoria.

Po načítaní týchto údajov je nutné získať 2D súradnice vrcholov pre vykreslenie plôch objektu do obrazu. Použili sme na to funkciu *cv2.projectPoints(points, rvec, tvec, calibration_matrix, distortion_coefficients)*, ktorá premieta 3D body do roviny obrazu. Paramater *points* určuje 3D súradnice vrcholov, *rvec* a *tvec* sú rotačné a translačné



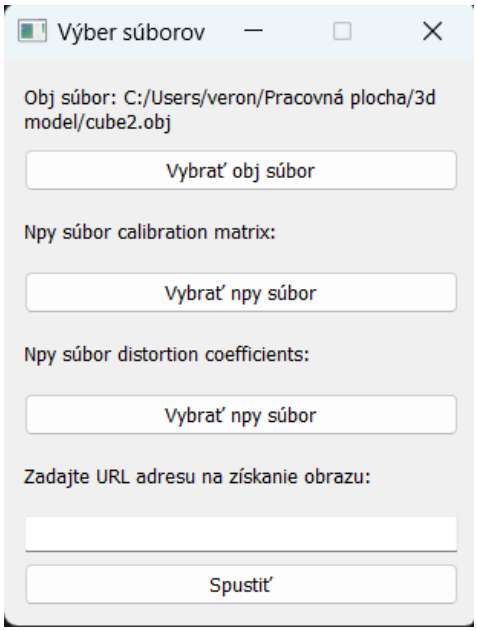
Obr. 3.12: Obrázky zachytené kamerou telefónu použité na kalibráciu kamery

vektory získané pri odhade polohy a *calibration_ matrix* a *distortion_ coefficients* sú parametre načítané zo súborov, ktoré sme získali kalibráciou kamery. Táto funkcia vracia pole prepočítaných bodov obrazu, pričom zachováva poradie jednotlivých vrcholov, čo je dôležité pre ďalšiu implementáciu.

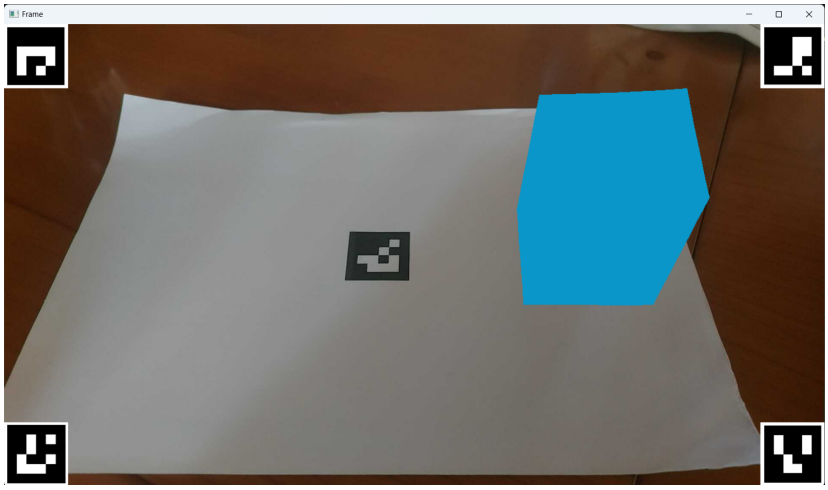
Po získaní 2D súradníc vrcholov je možné každú plochu objektu vykresliť pomocou funkcie *cv2.fillPoly(frame, points, color)*, kde *frame* je snímka, do ktorej sa objekt vykresľuje, *points* je zoznam prepočítaných 2D súradníc bodov, ktoré tvoria danú plochu a *color* je farba vo formáte *(b, g, r)*, ktorou sa plocha zafarbí.

Aby sme sa vyhli priamemu zadávaniu názvov súborov a iných potrebných informácií do kódu, vytvorili sme grafické používateľské rozhranie na ich výber s využitím knižnice PyQT5. Po zvolení súboru sa skontroluje typ súboru a zobrazí sa cesta k súboru. Takéto grafické používateľské rozhranie sme implementovali vo všetkých častiach práce. V každej časti sme zohľadnili všetky súbory a dáta, ktoré sú potrebné načítať na spustenie programu.

V časti implementácie rozšírenej reality je potrebné načítať súbory s uloženým objektom, kalibračnými dátami a zadať URL adresu pre získanie obrazu z kamery telefónu. Po stlačení tlačidla *spustiť* sa skontroluje, či sú zadané všetky údaje a spustí sa program. Grafické používateľské rozhranie implementované v tejto časti práce je znázornené na obrázku 3.13.



Obr. 3.13: Možnosť výberu súborov v implementácii rozšírenej reality

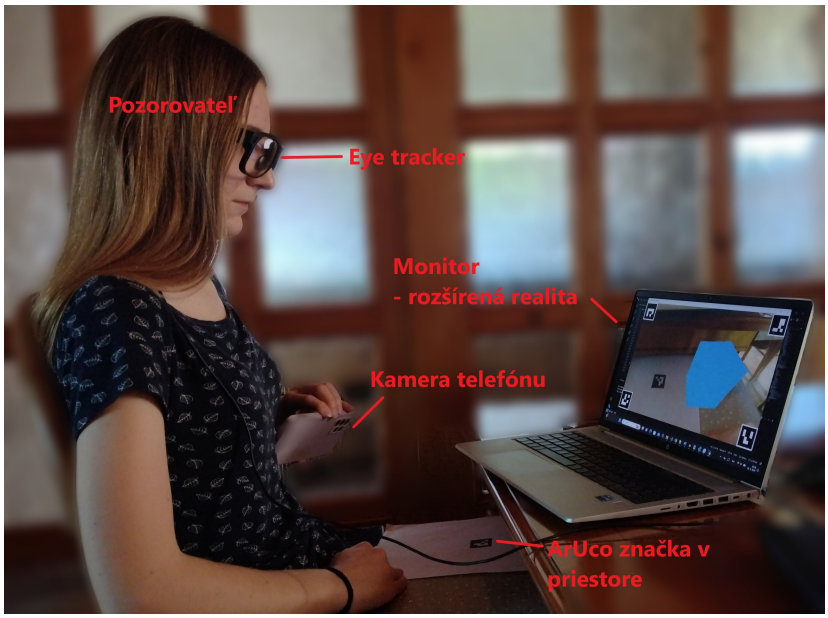


Obr. 3.14: Snímka úplnej implementácie rozšírenej reality

Na obrázku 3.14 je znázornená snímka úplnej implementácie rozšírenej reality vrátane vykresleného 3D objektu. Tento obrázok ilustruje prostredie, v ktorom pozorovateľ skúma 3D objekty.

3.3 Zbieranie dát z eye trackera

Ďalšia časť práce zahŕňa zbieranie dát z eye trackera. Pri sledovaní objektu sedí pozorovateľ pred monitorom notebooku, v priestore hýbe značkou, ktorú sníma kamera telefónu a na monitore sleduje vykreslený objekt. Hýbaním a otáčaním značky vie hýbať objektom a prezerať ho. Počas sledovania objektu má nasadené eye trackingové okuliare, ktoré zaznamenávajú dáta. Celý tento proces je zachytený na obrázku 3.15.



Obr. 3.15: Ilustrácia zberu dát z eye trackera. Na obrázku sú popísané dôležité časti pre porozumenie zbierania dát

Na sledovanie objektu je potrebné pripojiť kameru telefónu cez aplikáciu IP Webcam na získanie obrazu a spustiť Python aplikáciu, ktorá implementuje rozšírenú realitu.

Na zbieranie dát je potrebné spustiť aplikáciu Glasses 3 a pripojiť eye tracker. Eye tracker je možné pripojiť bezdrôtovo prostredníctvom Wi-Fi alebo káblom. V našom prípade nebolo možné použiť pripojenie cez Wi-Fi kvôli získavaniu obrazu z kamery telefónu. Z tohto dôvodu musia byť telefón aj notebook pripojené k rovnakej sieti a nie je možné pripojiť notebook k sieti eye trackeru. Preto sme eye tracker pripojili k notebooku pomocou LAN kábla.

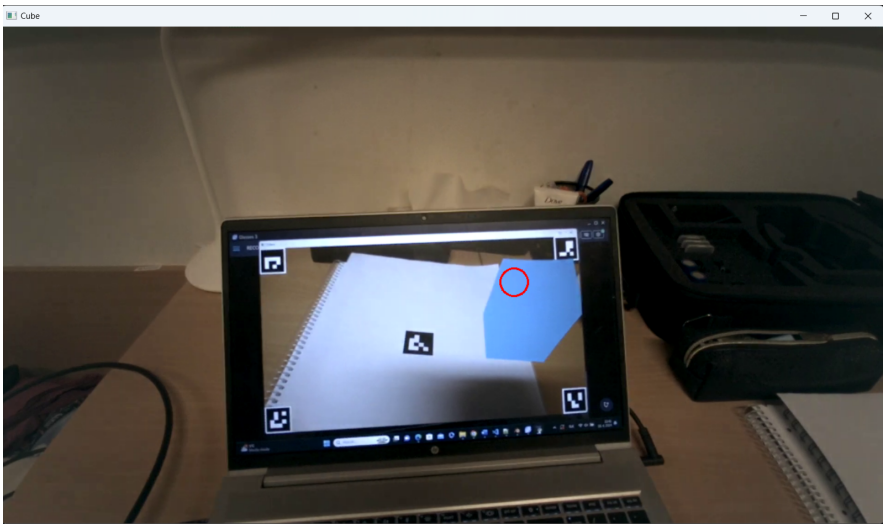
Pred začatím nahrávania je potrebná kalibrácia eye trackera pomocou kalibračnej karty priloženej k eye trackeru. Po kalibrácii je možné spustiť nahrávanie a samotné zbieranie dát, počas ktorého pozorovateľ sleduje 3D objekt v rozšírenej realite.

Po ukončení sledovania objektu nahrávanie a zbieranie dát ukončíme, dáta sa uložia do aplikácie Glasses 3, odkiaľ priečinok s dátami stiahneme a použijeme na spracovanie.

3.4 Spracovanie dát z eye trackera

Po zozbieraní dát je ďalším krokom spracovanie dát. Zo stiahnutého priečinka sme získali video nasnímané kamerou eye trackera *(scenevideo.mp4)* a súbor s podrobnými informáciami o každom bode pohľadu *(gazedata)*, ktoré sme načítali do Python aplikácie na spracovanie.

Ako prvé sme načítali potrebné údaje o bodoch pohľadu. V súbore *gazedata* má každý riadok tvar slovníka, a preto sme použili funkciu *ast.literal_ eval(line)*. Funkcia



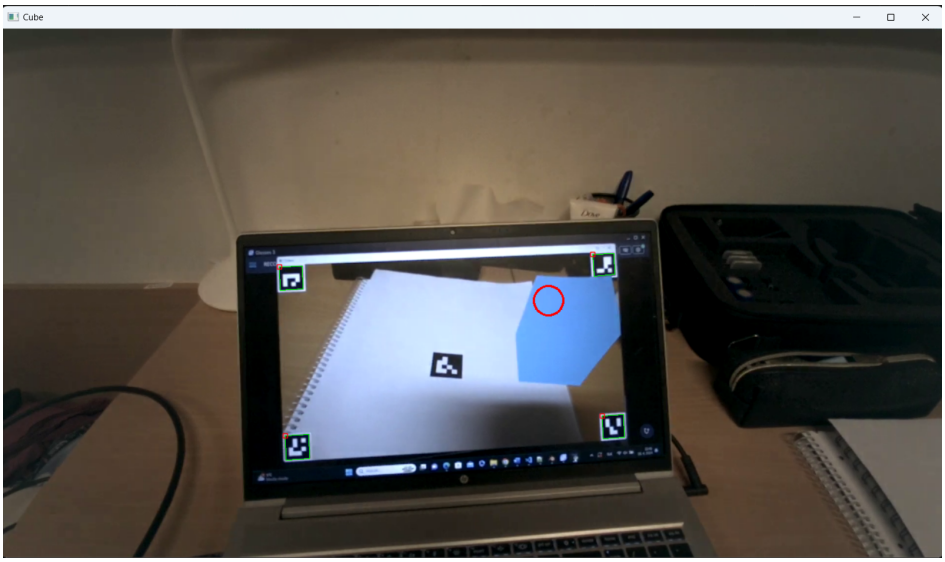
Obr. 3.16: Snímka z videa z eye trackera. Červeným krúžkom je vyznačený bod pohľadu prislúchajúci k tejto snímke

dostane ako parameter reťazec, ktorý rozdelí a uloží do slovníka. V našom prípade je reťazec riadok načítaný zo súboru. Pre použitie tejto funkcie je potrebné importovať knižnicu *ast*. Následne sme zo slovníka jednoducho získali potrebné údaje pre každý bod pohľadu. Pre našu implementáciu sú dôležité tri údaje: časový údaj *(timestamp)* - teda počet milisekúnd od začiatku videa, kedy bol daný bod pohľadu zaznamenaný a 2D súradnice bodu pohľadu *(gaze2D)* - teda súradnice x a y, ktoré určujú pixelové súradnice bodu pohľadu na snímke videa. Vytvorili sme pole, do ktorého sme uložili túto trojicu dát pre každý zaznamenaný bod pohľadu.

Po načítaní potrebných údajov o bodoch pohľadu program spracováva tieto údaje a video z eye trackera snímku po snímke. Z poľa sme postupne načítavali dáta o všetkých bodoch pohľadu. Získali sme snímku z videa zaznamenanú v danom časovom okamihu a k nej prislúchajúce súradnice bodu pohľadu. Na obrázku 3.16 je zobrazená jedna snímka z videa a červeným krúžkom je označený bod pohľadu, ktorý sa vzťahuje k tejto snímke.

Spracovanie snímky pozostáva z niekoľkých krokov. Ako prvé je potrebné nájsť na snímke okno s rozšírenou realitou, ktoré je zobrazené na monitore. Na to slúžia značky umiestnené v rohoch okna. V rohoch okna sú umiestnené značky s rôznymi identifikátormi od 1 do 4, ktoré sú v okne usporiadané nasledovne: ľavý horný roh, pravý horný roh, pravý dolný roh a ľavý dolný roh. Toto známe usporiadanie umožňuje nájsť presné súradnice rohov okna. Na snímke sme tieto štyri značky zdetegovali obr. 3.17, čím sme pre každú značku získali súradnice jej rohov a tiež jej identifikátor. Vďaka tomu, že poznáme usporiadanie značiek v rohoch okna, vieme po detekcii značiek určiť presné súradnice rohov okna.

Získaním štyroch rohov okna sme boli schopní snímku upraviť pre ďalšie spracova-

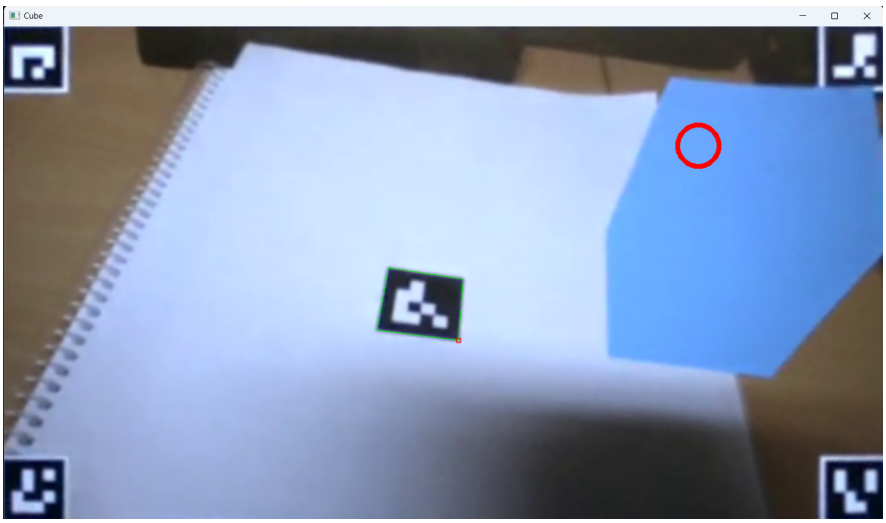


Obr. 3.17: Snímka z videa z eye trackera, na ktorej sú vyznačené zdetegované značky v rohoch okna slúžiace na nájdenie okna na snímke

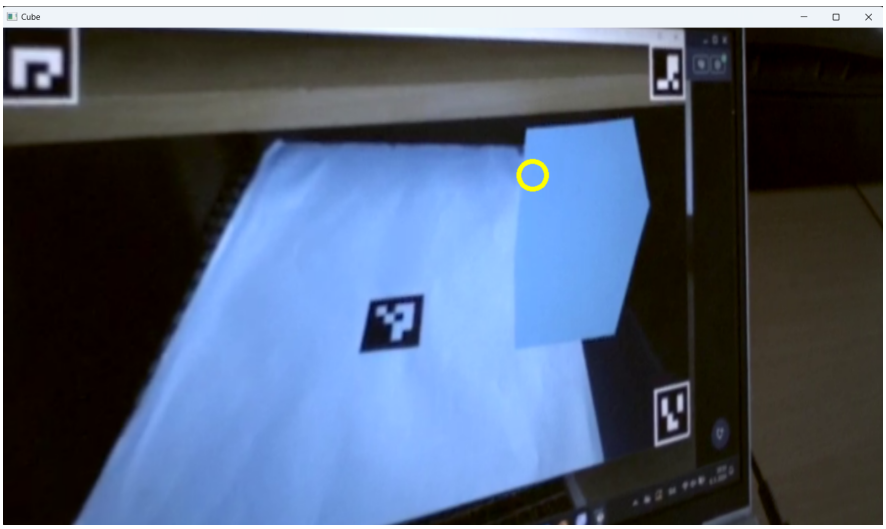
nie. Táto úprava zahŕňala použitie funkcií *cv2.findHomography(src_pts, dst_pts)* a *cv2.warpPerspective(frame, matrix_homography, (width, height))*. Ako prvé sme vypočítali homografiu pomocou funkcie *cv2.findHomography*. Na výpočet homografie sú potrebné štyri páry bodov. V našom prípade *src_pts*, teda štyri body zo zdrojového obrazu, sú zistené štyri rohy okna detekciou značiek a *dst_pts*, teda body z cieľového obrazu, na ktoré chceme tieto body namapovať, sú rohy snímky. Funkcia vracia maticu homografie *(matrix_homography)*. Túto maticu sme poslali ako parameter do funkcie *cv2.warpPerspective*, ktorá aplikuje perspektívnu transformáciu na vstupný obrázok *(frame)* a vracia transformovaný obrázok. Parameter *(width, height)* určuje veľkosť výsledného obrázka. V našom prípade sme veľkosť výsledného obrázka zvolili tak, aby bola rovnaká ako veľkosť okna pri rozšírenej realite. Týmto spôsobom sa nám podarilo snímku orezať tak, aby zobrazovala iba okno s rozšírenou realitou. Bod pohľadu bolo treba tiež prepočítať na nové súradnice pomocou vypočítanej homografie. Upravenú snímku s prepočítanými súradnicami bodu pohľadu zobrazuje obrázok 3.18.

Pre výpočet homografie je potrebné poznať štyri body. Detekciou jednej značky získame práve štyri body jej rohov, takže by bolo postačujúce pracovať iba s jednou značkou umiestnenou v rohu okna. Avšak pri tomto spôsobe je výpočet homografie a orezanie snímky veľmi nepresné, ako to ilustruje aj obrázok 3.19. Pre presnejší výpočet homografie sme zvolili prístup cez štyri značky. Potrebujeme poznať veľkosť okna na aplikáciu perspektívnej transformácie, aby sa snímka orezala správne a získali sme snímku totožnú snímke, na ktorej bola zobrazená rozšírená realita.

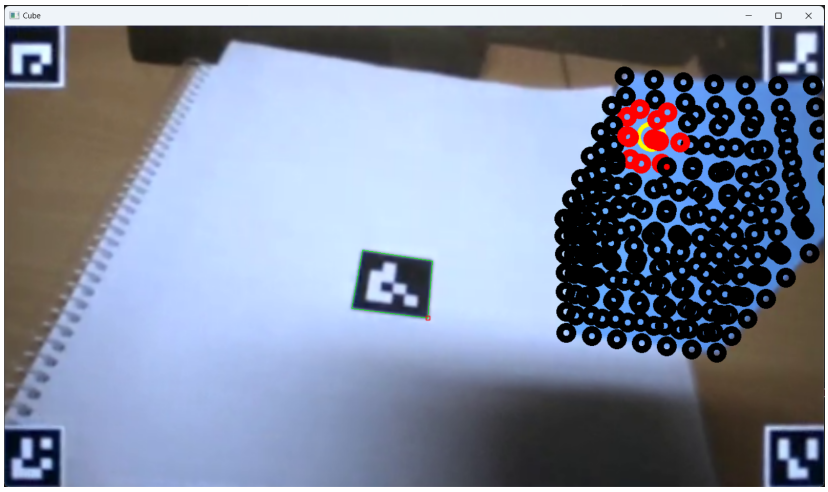
Na tejto upravenej snímke sme zdetegovali značku určenú na zobrazenie rozšírenej reality. Aby bola táto detekcia možná, bolo potrebné, aby bol objekt posunutý a nezob-



Obr. 3.18: Orezaná snímka s prepočítaným bodom pohľadu vyznačeným červeným krúžkom



Obr. 3.19: Nepresne orezaná snímka pri použití iba jednej značky na výpočet homografie



Obr. 3.20: Snímka, na ktorej sú čiernou farbou vyznačené všetky vrcholy, žltým krúžkom je znázornený bod pohľadu a vrcholy v blízkosti bodu pohľadu sú vyznačené červenou farbou

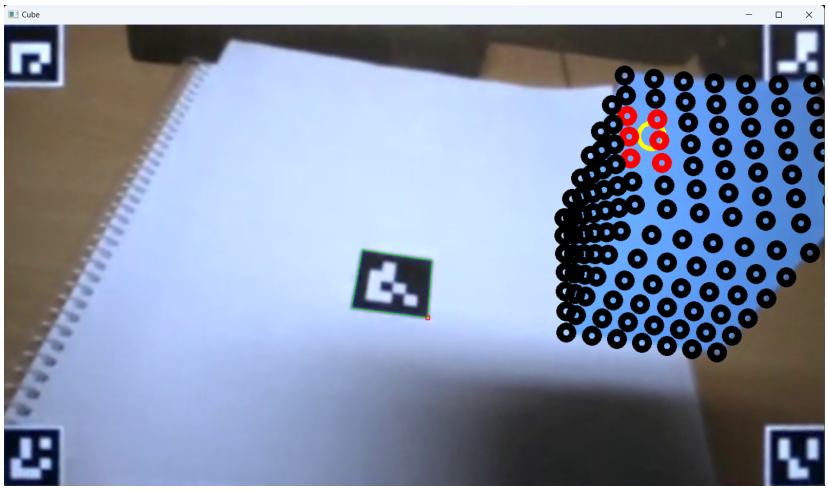
razoval sa priamo na značke. Po detekcii značky sme vypočítali odhad polohy rovnako ako pri implementácii rozšírenej reality. Vďaka úprave snímky sme získali rovnaké hodnoty ako pri vykresľovaní objektu, čo nám umožnilo určiť presnú pozíciu a orientáciu objektu na snímke.

Následne sme použili už spomenutú funkciu *cv2.projectPoints(points, rvec, tvec, calibration_matrix, distortion_coefficients)*, čím sme získali z 3D súradníc každého vrcholu objektu jeho 2D súradnice. Bod pohľadu je tiež určený 2D súradnicami, a teda je ľahké zistiť, na ktorý vrchol objektu sa pozorovateľ pozeral. Pri hľadaní týchto vrcholov neberieme do úvahy iba samotný bod pohľadu, ale aj okolie tohto bodu do určitej vzdialenosti.

Na obrázku 3.20 sú čiernou farbou znázornené všetky vrcholy objektu a červenou farbou sú zobrazené vrcholy v blízkosti bodu pohľadu, ktoré boli označené ako pozreté. Bod pohľadu je vyznačený žltým krúžkom. Problém, ktorý sa tu vyskytuje, je ten, že funkcia *cv2.projectPoints* vracia 2D súradnice každého bodu a nevie automaticky odfiltrovať body, ktoré v skutočnosti nie sú viditeľné.

Krok, ktorý ešte bolo potrebné urobiť, bol odstránenie vrcholov, ktoré nie sú viditeľné, teda sú zakryté inými vrcholmi alebo stenami objektu. Pracovali sme s knižnicou *trimesh*, ktorá slúži na prácu s objektami s trojuholníkovou sieťou. Pri práci s touto knižnicou je potrebné načítať objekt, na čo slúži funkcia *trimesh.load_mesh(file)*, kde *file* je názov súboru, v našom prípade typu *obj*. Po načítaní objektu je možné využívať ďalšie funkcie z tejto knižnice, ktoré sa vykonávajú nad načítaným objektom.

Na zisťovanie viditeľnosti vrcholov sme využili metódu ray castingu. Základ tvoril zostrojenie lúča z pozície kamery do 3D priestoru smerujúceho k danému vrcholu. Z knižnice *trimesh* sme na to použili funkciu *intersects_location(origin, direction, mul-*



Obr. 3.21: Snímka, na ktorej sú čiernou farbou vyznačené iba viditeľné vrcholy, žltým krúžkom je znázornený bod pohľadu a viditeľné vrcholy v blízkosti bodu pohľadu sú vyznačené červenou farbou

tiple_hits=False), ktorá hľadá priesečníky zostrojeného lúča s objektom. Paramater *origin* je bod, z ktorého sme zostrojili lúč. V našom prípade to bola pozícia kamery, teda bod (0,0,0). Parameter *direction* určuje smer lúča, ktorý sme vypočítali ako rozdiel 3D bodu, do ktorého mieri lúč a pozície kamery. Parametrom *multiple_hits* sme určili, že hľadáme iba prvý prienik. Keďže pozícia kamery je bod (0,0,0), potrebovali sme zistiť umiestnenie objektu vzhľadom ku kamere. Po detekcii značky na snímke a odhade polohy sme získali rotačné a translačné vektory. Tieto vektory sme použili na zostrojenie transformačnej matice, ktorú sme následne aplikovali na každý 3D bod objektu, čím sme objekt presunuli do súradnicového systému kamery. Funkcia *intersects_location* vrátila pozíciu prvého nájdeného priesečníka. Ak sa táto pozícia zhodovala s pozíciou bodu s malou odchýlkou, vyhodnotili sme tento bod ako viditeľný.

Po odstránení všetkých neviditeľných vrcholov sme prechádzali cez všetky zvyšné vrcholy a počítali vzdialenosť vrcholu od bodu pohľadu. Ak bola táto vzdialenosť menšia než stanovená hodnota, vyhodnotili sme tento vrchol ako pozretý pozorovateľom. Táto hodnota závisí od hustoty siete objektu. Čím je sieť objektu hustejšia, tým je potrebná menšia hodnota. Pri spustení programu si používateľ vie nastaviť vlastnú hodnotu, pričom defaultne nastavená hodnota je 20 pixelov.

Na obrázku 3.21 sú čiernou farbou znázornené už iba viditeľné vrcholy, červenou farbou sú znázornené viditeľné vrcholy v blízkosti bodu pohľadu, ktorý je znázornený žltým krúžkom.

Tento spôsob vyhodnocovania sa ukázal ako veľmi neefektívny, najmä kvôli neustálemu prechádzaniu cez všetky vrcholy a zisťovaniu všetkých viditeľných vrcholov. Pri objekte s hustou sieťou trvalo spracovanie jednej snímky niekoľko sekúnd. Optimalizovali sme spôsob vyhodnocovania tým, že sme najprv hľadali vrcholy v blízkosti bodu

Obr. 3.22: Možnosť výberu súborov pri spracovaní dát

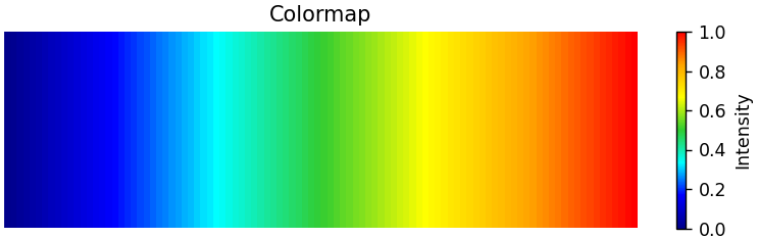
pohľadu a iba pre tieto vrcholy sme zisťovali, či sú viditeľné. Tento spôsob zrýchlil spracovanie snímky niekoľkonásobne v porovnaní s predchádzajúcim spôsobom.

Dáta, ktoré sme potrebovali zistiť spracovaním snímok z videa, určovali, koľkokrát sa pozorovateľ pozrel na každý vrchol objektu. Vytvorili sme pole, v ktorom sme pre každý vrchol počítali a uchovávali početnosť pozretí. Po spracovaní všetkých snímok, teda celého videa, sme početnosti zapísali do textového súboru, ktorý je potrebný v ďalšom kroku pri vizualizácii dát.

Grafické používateľské rozhranie implementované v tejto časti práce je znázornené na obrázku 3.22. Na spustenie programu je potrebné načítať sledovaný objekt, dáta z eye trackera, kalibračné dáta, zadať vzdialenosť od bodu pohľadu, pre ktorú sa bude vrchol vyhodnocovať ako pozretý a zadať názov textového súboru, do ktorého sa uložia dáta po spracovaní.

3.5 Vizualizácia dát z eye trackera

Poslednou časťou je vizualizácia získaných dát v podobe tepelnej mapy. Vizualizáciu dát sme vytvorili v Pythone s využitím knižnice OpenGL. Základ tvorí načítanie objektu z obj súboru a načítanie súboru s početnosťami, ktorý sme získali v predošlom kroku. Pri načítavaní súboru s početnosťami zisťujeme hodnoty maximálnej a minimálnej



Obr. 3.23: Paleta farieb použitá na vytvorenie tepelnej mapy pre účel vizualizácie

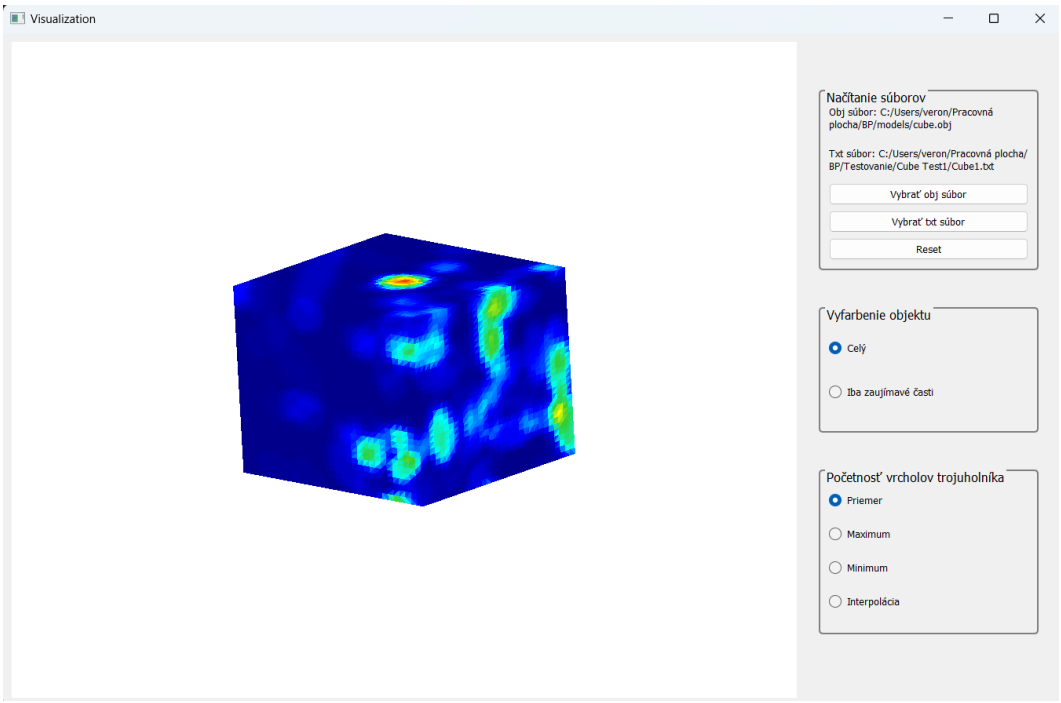
početnosti pre celý objekt, s ktorými pracujeme pri vizualizácii objektu. Pre minimálnu početnosť zisťujeme dve hodnoty: celkovú minimálnu početnosť, ktorá je vo väčšine prípadov nula a minimálnu početnosť väčšiu ako nula.

Vizualizácia objektu je realizovaná zafarbením každého trojuholníka podľa počtu pozretí vrcholov, ktoré tvoria daný trojuholník. Vytvorili sme vlastnú paletu farieb obr. 3.23, podľa ktorej sme jednotlivé trojuholníky objektu zafarbovali. Maximálnej početnosti sme priradili hodnotu 1 z palety farieb, teda červenú farbu, a minimálnej početnosti sme priradili hodnotu 0, teda tmavomodrú farbu. Pre ostatné početnosti určujeme farbu normalizovaním hodnoty početnosti a zistením farby z palety farieb.

Každý vrchol v trojuholníku môže mať inú početnosť pozretí, a preto sme museli nájsť spôsob, ako určiť hodnotu výslednej početnosti pre celý trojuholník a na základe toho zistiť farbu trojuholníka. Výsledná farba môže byť vypočítaná podľa priemeru početností v trojuholníku, podľa maximálnej početnosti alebo podľa minimálnej početnosti. Najprv je teda dôležité z početností jeho vrcholov určiť hodnotu celého trojuholníka a následne zistiť farbu, ktorou sa zafarbí. Ďalšou možnosťou je každý vrchol trojuholníka zafarbiť inou farbou a stred trojuholníka sa vyplní interpoláciou. Tiež je dôležité vybrať, či chceme zafarbiť celý objekt alebo iba tie časti, na ktoré sa pozorovateľ aspoň raz pozrel.

Voľbu spôsobu vizualizácie sme umožnili používateľovi podľa jeho súčasných potrieb a preferencií, čím sme prispôsobili vizualizáciu jeho špecifickým požiadavkám. Použili sme knižnicu PyQt5 na implementáciu grafického používateľského rozhrania. Toto rozhranie slúži na výber súborov aj na výber spôsobu vizualizácie z viacerých nastavení a je zobrazené na obrázkoch 3.24 a 3.25 v pravej časti. Vizualizácia sa automaticky spustí po načítaní potrebných súborov.

Prvé z nastavení určuje, či bude celý objekt zafarbený farbami z danej palety farieb. Možnosť „Celý“ znamená, že ako minimálnu početnosť berieme do úvahy aj nulu, hodnote nula sa priradí tmavomodrá farba a celý objekt je zafarbený farbami z palety farieb. Možnosť „Iba zaujímavé časti“ znamená, že tmavomodrá farba sa priradí minimálnej početnosti väčšej než nula. Trojuholníky, ktorých hodnota je menšia než toto minimum, sú zafarbené šedou farbou. Pre ostatné trojuholníky sa vypočíta farba z palety farieb. Týmto spôsobom zafarbíme iba tie oblasti, ktoré zaznamenali aspoň



Obr. 3.24: Vizualizácia objektu s defaultnými nastaveniami

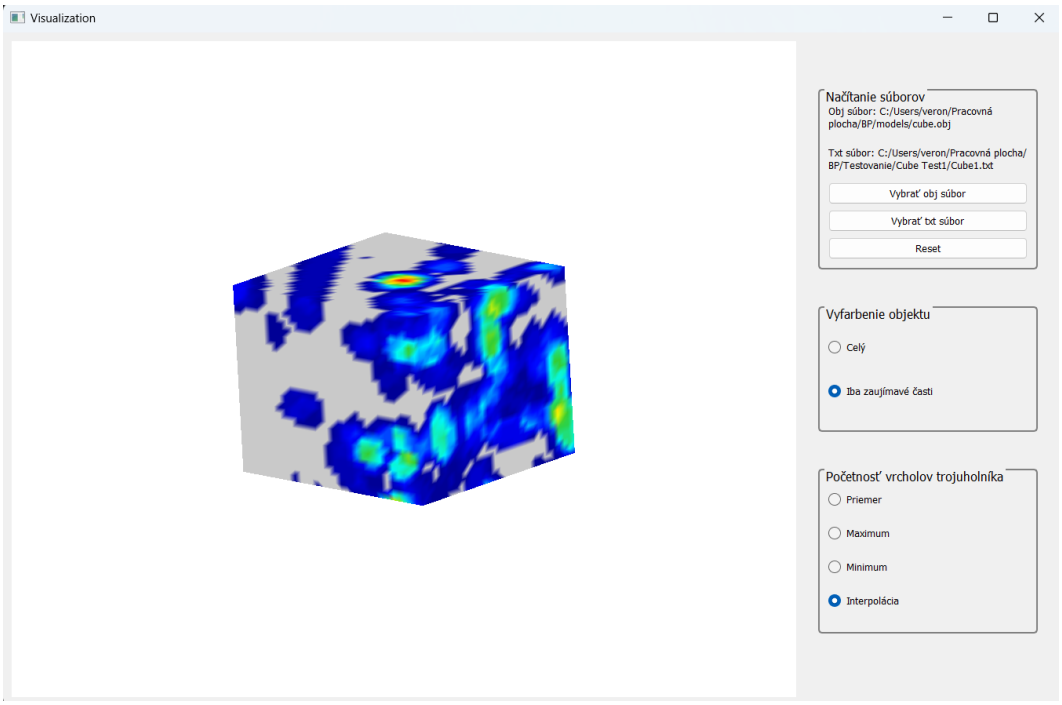
nejaký bod pohľadu, teda boli pre pozorovateľa aspoň trochu zaujímavé a pritiahli aspoň minimum jeho pozornosti.

Druhé z nastavení určuje spôsob výpočtu výslednej hodnoty početnosti trojuholníka. Na výber sú štyri možnosti, pričom:

- priemer znamená, že hodnota trojuholníka sa vypočíta ako priemer početností jeho vrcholov
- maximum znamená, že hodnota sa rovná maximálnej početnosti jeho vrcholov
- minimum znamená, že hodnota sa rovná minimálnej početnosti jeho vrcholov
- interpolácia znamená, že sa nezisťuje hodnota celého trojuholníka, ale každý vrchol sa zafarbí inou farbou na základe početnosti vrcholu a vnútro trojuholníka sa vyplní interpoláciou farieb vo vrcholoch

Tieto dve nastavenia sa dajú kombinovať a každý trojuholník objektu je zafarbený podľa zvolených nastavení. Defaultné nastavenia sú „Celý“ a „Priemer“. Príklady vizualizácií so zvolenými nastaveniami pre objekt s hustejšou sieťou sú zobrazené na obrázkoch 3.24 a 3.25. Pri vizualizácii zohráva dôležitú úlohu hustota siete objektu. Čím je sieť objektu hustejšia, tým je vizualizácia atraktívnejšia a presnejšie zobrazuje zaujímavé oblasti objektu.

Pre možnosť prezerania objektu sme implementovali otáčanie objektu okolo súradnicových osí pomocou klávesov nasledovne:



Obr. 3.25: Vizualizácia objektu so zvolenými nastaveniami

Obr. 3.26: QR kód na spustenie videa, ktoré ilustruje proces zbierania a vizualizácie dát

- x - otáčanie okolo osi x v kladnom smere
- X - otáčanie okolo osi x v zápornom smere
- y - otáčanie okolo osi y v kladnom smere
- Y - otáčanie okolo osi y v zápornom smere
- z - otáčanie okolo osi z v kladnom smere
- Z - otáčanie okolo osi z v zápornom smere

Pre lepšie porozumenie a ilustráciu zbierania a vizualizácie dát sme natočili video, ktoré sa spustí po naskenovaní QR kódu na obrázku 3.26.

Kapitola 4

Testovanie a výsledky

Pre potreby implementácie bola kocka dostatočným objektom, avšak na testovanie sme vybrali zložitejší objekt, konkrétne zajaca, ktorý sme tiež podľa uvedeného postupu v 3.1 exportovali z Blenderu.

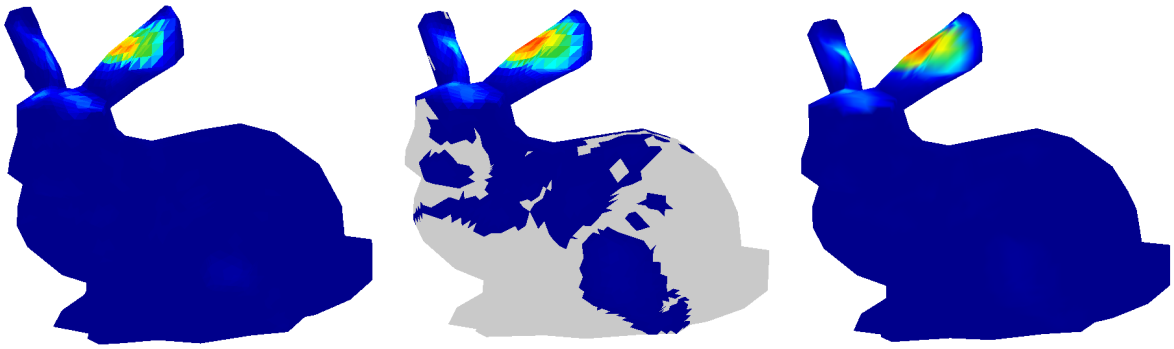
Táto kapitola obsahuje podrobnosti o testovaní našej práce. Testovania sa zúčastnili traja pozorovatelia, ktorí sledovali objekt a mali za úlohu sústrediť sa na konkrétnu oblasť na objekte. Každé pozorovanie trvalo približne jednu minútu. Dĺžka spracovania dát zavisí od trvania pozorovania, od hustoty siete objektu a od nastavenej vzdialenosti, ktorá určuje, do akej vzdialenosti od bodu pohľadu hodnotíme vrcholy ako pozreté. Pri predvolenej vzdialenosti, ktorá je 20, sa približne minútové video spracováva zhruba 20 minút. Zväčšením vzdialenosti na 40 je čas spracovania približne dvojnásobný.

V nasledujúcich podkapitolách podrobne popíšeme každé z týchto troch pozorovaní. Každá podkapitola obsahuje informácie o skúmanej oblasti, o počte bodov pohľadu, ktoré sa zaznamenali a niekoľko vizualizácií spracovaných dát.

4.1 Testovanie 1

Pri skúmaní objektu sa pozorovateľ zameriaval na oblasť uší. Zaznamenaných bolo približne 2800 bodov pohľadu. Vizualizácie spracovaných dát znázorňuje obrázok 4.1. Dáta z tohto pozorovania sme spracovávali s nastavenou vzdialenosťou 20, čo zobrazuje obrázok 4.1a, aj vzdialenosťou 40, čo znázorňujú obrázky 4.1b a 4.1c. Nastavenia vizualizácie pre každý obrázok sú:

- Obrázok 4.1a: Celý, Priemer
- Obrázok 4.1b: Iba zaujímavé časti, Priemer
- Obrázok 4.1c: Celý, Interpolácia

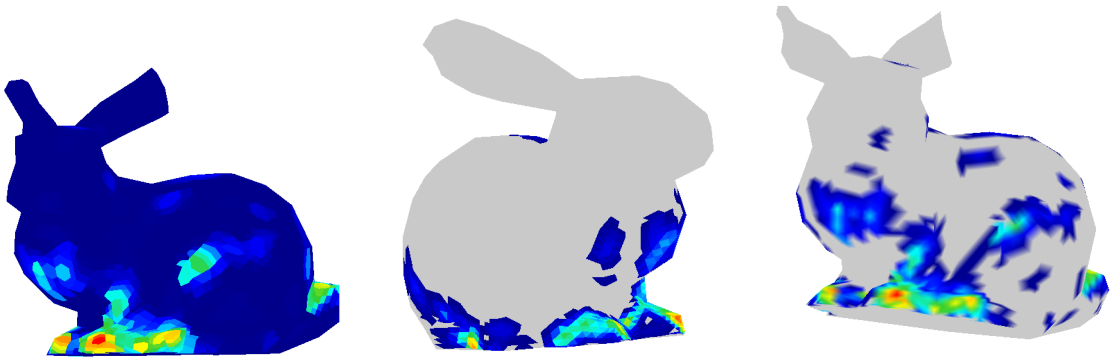


(a) Vzdialenosť: 20
Nastavenie vizualizácie:
Celý, Priemer

(b) Vzdialenosť: 40
Nastavenie vizualizácie:
Iba zaujímavé časti, Priemer

(c) Vzdialenosť: 40
Nastavenie vizualizácie:
Celý, Interpolácia

Obr. 4.1: Vizualizácia dát z prvého testovania



(a) Vzdialenosť: 20
Nastavenie vizualizácie:
Celý, Maximum

(b) Vzdialenosť: 20
Nastavenie vizualizácie:
Iba zaujímavé časti, Priemer

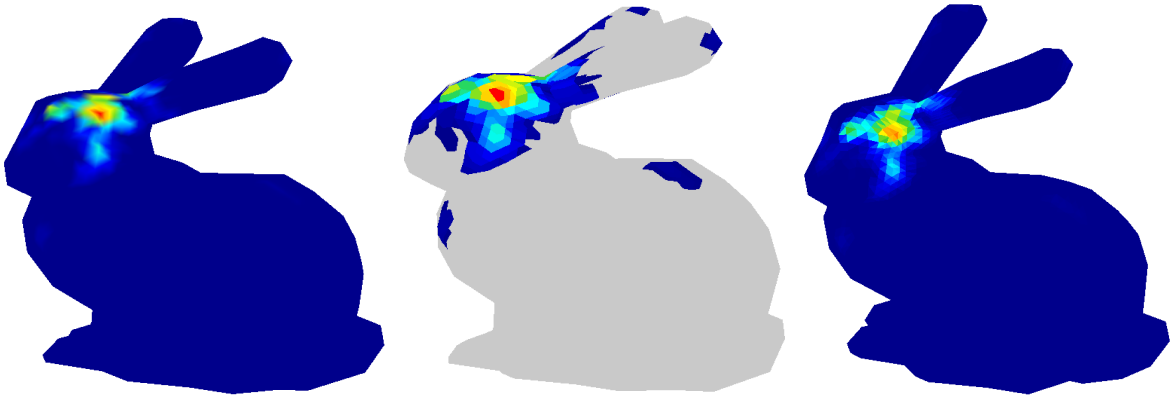
(c) Vzdialenosť: 20
Nastavenie vizualizácie:
Iba zaujímavé časti, Interpolácia

Obr. 4.2: Vizualizácia dát z druhého testovania

4.2 Testovanie 2

V tomto pozorovaní skúmal pozorovateľ oblasť nôh a chvosta, zaznamenalo sa približne 3200 bodov pohľadu. Na obrázku 4.2 sú znázornené vizualizácie dát z tohto pozorovania, pričom vzdialenosť pri spracovaní bola predvolená a jednotlivé nastavenia vizualizácie pre každý obrázok sú:

- Obrázok 4.2a: Celý, Maximum
- Obrázok 4.2b: Iba zaujímavé časti, Priemer
- Obrázok 4.2c: Iba zaujímavé časti, Interpolácia



(a) Vzdialenosť: 20
Nastavenie vizualizácie:
Celý, Interpolácia

(b) Vzdialenosť: 20
Nastavenie vizualizácie:
Iba zaujímavé časti, Maximum

(c) Vzdialenosť: 20
Nastavenie vizualizácie:
Celý, Priemer

Obr. 4.3: Vizualizácia dát z tretieho testovania

4.3 Testovanie 3

Pri poslednom testovaní pozorovateľ skúmal oblasť hlavy a zaznamenalo sa približne 3400 bodov pohľadu. Vizualizácie spracovaných dát ilustruje obrázok 4.3. Aj pri tomto testovaní bola vzdialenosť predvolená, teda 20, a nastavenia vizualizácie pre každý obrázok sú nasledovné:

- Obrázok 4.3a: Celý, Interpolácia
- Obrázok 4.3b: Iba zaujímavé časti, Maximum
- Obrázok 4.3c: Celý, Priemer

Pri testovaní sme narazili na niekoľko nedostatkov, ktoré ovplyvňujú presnosť dát. Prvým z nich je odraz svetla na monitore, ktorý môže skomplikovať detekciu značiek. Detekciu značiek v rohoch okna tiež môže skomplikovať nedostatočne široký biely rám okolo značky. Pre správne spracovanie snímky je nevyhnutné, aby boli zdetegované všetky štyri značky v rohoch okna. Tento problém sme vyriešili rozšírením bieleho okraja okolo značiek a zväčšením značiek pre lepšiu detekciu. Pri pozorovaní sme sa vyhýbali priamemu osvetleniu, aby sme predišli odrazom svetla na monitore. Presnosť testovania tiež závisí od vzdialenosti pozorovateľa od monitora a uhla, pod ktorým naň pozorovateľ pozerá. Pre najpresnejšie výsledky je potrebné, aby bol pozorovateľ čo najbližie k monitoru a aby sa naň pozeral priamo.

Z výsledkov testovaní vidíme, že program dokáže relatívne presne identifikovať a vizualizovať oblasti záujmu na 3D objekte. Aby boli tieto oblasti čo najpresnejšie a vizualizácie čo najatraktívnejšie, je potrebné zvoliť vhodné nastavenie vzdialenosti pre konkrétnu sieť objektu. Pre hustú sieť objektu s adekvátnou vzdialenosťou dokáže program presne určiť oblasti záujmu, avšak čím je sieť hustejšia a vzdialenosť väčšia, tým je spracovanie dát časovo náročnejšie.

Kapitola 5

Ďalšia práca

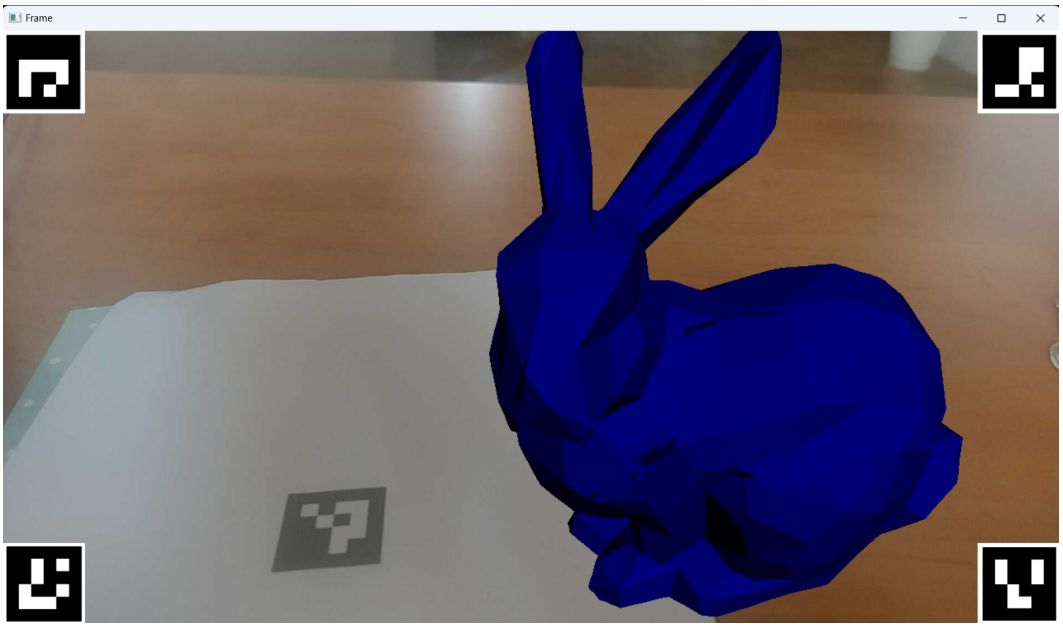
V tejto kapitole rozoberáme možné modifikácie a rozšírenie práce, ako aj jej potenciálne ďalšie využitie.

5.1 Modifikácia práce

Táto práca, hoci spĺňa všetky požiadavky, má niekoľko nedostatkov. Jedným z nich je vykresľovanie 3D objektu v rozšírenej realite. Pri vykresľovaní objektu sa každá stena objektu vykresľuje rovnakou farbou a pri zložitejších objektoch je tak ťažké rozlíšiť štruktúru objektu. Riešením je implementácia osvetlenia objektu, napríklad pomocou Phongovho modelu, pričom výslednú intenzitu svetla sme vypočítali iba pomocou difúznej zložky. Vykresľovanie objektu s pridaným osvetlením je znázornené na obrázku 5.1. Pre pozorovateľa by bol takýto spôsob vykresľovania atraktívnejší, avšak je výpočtovo náročnejší. Rýchlosť v tomto prípade je približne 1,30 FPS a pri vykresľovaní bez osvetlenia je rýchlosť približne 5 FPS. Riešením by mohla byť implementácia efektívnejších algoritmov, avšak túto časť sme do práce nestihli zakomponovať. Spôsob vykresľovania objektu neovplyvňuje presnosť aplikácie, a preto sme so školiteľkou dospeli k záveru, že je výhodnejšie použiť jednoduchšie a rýchlejšie vykresľovanie objektu.

Ďalším riešením by mohlo byť použitie knižnice na vykresľovanie 3D objektov, napríklad OpenGL, pomocou ktorej by sme vedeli vykresľovať 3D objekt aj s jeho textúrou alebo materiálom.

Ďalšou modifikáciou je implementácia efektívnejšej vizualizácie. Pri veľmi hustej sieti objektu je vizualizácia atraktívnejšia a presnejšia, avšak otáčanie objektu v OpenGL spôsobuje sekanie. Možným riešením by bolo použitie inej knižnice na prácu s 3D objektami alebo implementácia iného spôsobu vizualizácie.



Obr. 5.1: Vykresľovanie objektu s osvetlením

5.2 Rozšírenie práce

V rámci rozšírenia práce je možné implementovať sledovanie viacerých objektov súčasne. Stačí pridať do scény viac značiek a pre každú značku určiť objekt, ktorý sa na nej vykreslí.

Táto práca poskytuje základ pre vyhodnocovanie kvality 3D skenov. Po naskenovaní objektu môže byť táto práca použitá na overenie presnosti skenovania objektu, pričom môže slúžiť ako kontrola, či sú zaujímavé oblasti objektu dostatočne presne zachytené.

Ďalej je možné implementovať sledovanie reálnych objektov a zisťovanie ich zaujímavých oblastí a výsledky porovnávať s výsledkami zo sledovania naskenovaných virtuálnych objektov. Rozdiely vo výsledkoch môžu naznačovať chyby v skenovaní.

Záver

Podarilo sa nám vytvoriť nástroj, ktorý dokáže identifikovať a vizualizovať zaujímavé oblasti na 3D objekte. Na hľadanie týchto zaujímavých oblastí sme využili ľudské vnímanie pri sledovaní 3D objektov. Na určenie miesta, kam sa pozorovateľ pozerá, sme využili eye tracker.

3D objekty sme sledovali v rozšírenej realite založenej na značkách, ktorá je implementovaná v Pythone s využitím knižnice OpenCV. Na spustenie rozšírenej reality sme získavali obraz z kamery telefónu kvôli lepšiemu rozlíšeniu kamery.

Spracovanie získaných dát z pozorovania objektu zahŕňalo spracovanie videa a bodov pohľadu zaznamenaných eye trackerom. Z videa sme postupne spracovávali snímku po snímke a k nim prislúchajúci bod pohľadu. Detekciou značky na snímke sme určili pozíciu a orientáciu 3D objektu. Z dát o pozícii objektu a bode pohľadu sme dokázali určiť vrcholy v blízkosti bodu pohľadu, ktoré boli pozreté pozorovateľom. Pre každý vrchol objektu sme počítali početnosť jeho pozretí.

Vo vizualizácii sme farebne odlíšili zaujímavé oblasti na 3D objekte podľa početnosti pozretí vrcholov a využitím vytvorenej palety farieb. Implementovali sme grafické používateľské rozhranie pre zvolenie spôsobu vizualizácie z daných možností. V rámci vizualizácie vie používateľ otáčať objektom, čo mu poskytuje úplný obraz o zaujímavých oblastiach na 3D objekte.

Vytvorený nástroj sme testovali na troch pozorovateľoch, ktorí skúmali konkrétnu oblasť 3D objektu. Zo získaných vizualizácií sme vyhodnotili, že tento nástroj dokáže relatívne presne identifikovať a vizualizovať tieto zaujímavé oblasti. Podrobnosti o testovaní sme popísali v kapitole 4.

Pri testovaní aplikácie sa nám nepodarilo implementovať atraktívnejšie vykresľovanie 3D objektov pre pozorovateľa, avšak navrhli sme niekoľko riešení, ktoré sme rozobrali v kapitole 5. Hoci spôsob vykresľovania objektu nemá vplyv na funkčnosť a presnosť aplikácie, v budúcnosti by sme chceli jedno z týchto riešení do práce implementovať, aby sme dosiahli lepšie zobrazenie objektov a ich geometrie a atraktívnejšie sledovanie pre pozorovateľov.

Táto práca poskytuje základ pre hľadanie chýb na 3D skenoch. V rámci ďalšej práce by bolo potrebné rozlíšiť, ktoré zo zistených zaujímavých oblastí zachytávajú skutočne chyby na skenoch a ktoré oblasti sú zaujímavé iba pre ich zaujímavú geometriu.

Literatúra

[1] Elena Šikudová, Zuzana Černeková, Wanda Benešová, Zuzana Haladová, and Júlia Kučerová. Počítačové videnie detekcia a rozpoznávanie objektov. *Praha: Wikina Praha*, page 397, 2013.

[2] History of Tobii. `https://corporate.tobii.com/about-us/history-of-tobii`.

[3] About Tobii company. `https://corporate.tobii.com/about-us`.

[4] How do Tobii eye-trackers work? `https://connect.tobii.com/s/article/How-do-Tobii-eye-trackers-work?language=en_US`. Prístupné dňa: 2023-05-14.

[5] Jennifer Romano Bergstrom and Andrew Schall. *Eye tracking in user experience design*. Elsevier, 2014.

[6] Sanni Siltanen. Theory and applications of marker-based augmented reality: Licentiate thesis. 2012.

[7] Vladimir Geroimenko. Augmented reality technology and art: The analysis and visualization of evolving conceptual models. In *2012 16th International Conference on Information Visualisation*, pages 445–453. IEEE, 2012.

[8] OpenCV: Camera Calibration. `https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html`.

[9] ArUco detection. `https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html`.

[10] About Opencv. `https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html`.

[11] About Homography. `https://people.scs.carleton.ca/~roth/comp4900d-12/notes/homography.pdf`.

[12] About Ray casting. `https://developer.nvidia.com/discover/ray-tracing`.

[13] Patrik Modrovský. *Vyuzitie eye tracker pre vyhodnocovanie registracie 3D skenov.* Bakalárska práca, 2021.

[14] Camera Calibration Code. `https://www.geeksforgeeks.org/camera-calibration-with-python-opencv/`.