FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

Software pro demonstrační laboratoř techniky

Alex König

PLZEŇ

2023

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

Software pro demonstrační laboratoř techniky

Bc. Alex König

Vedoucí práce
Doc. Ing. Libor Váša, Ph.D.

PLZEŇ **2023**

Citace v seznamu literatury:
KÖNIG, Alex. *Software pro demonstrační laboratoř techniky*. Plzeň, 2023. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Doc. Ing. Libor Váša, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Alex KÖNIG**
Osobní číslo:	**A20N0060P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Počítačová grafika**
Téma práce:	**Software pro demonstrační laboratoř techniky**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se se softwarovým řešením pro demonstrační laboratoř techniky vyvíjeným na KIV ZČU pro potřeby Gymnázia Sokolov. Zejména se seznamte s funkcí a rozhraním serverové části řešení.
2. Zvolte dvě konkrétní zařízení (popř. kombinace zařízení) a vytvořte pro každé z nich zadání pro sadu úloh směřující k realizaci komplexní aplikace. Sada se bude skládat ze základní úlohy, realizovatelné bez dohledu podle návodu, která návštěvníkovi laboratoře bude demonstrovat smysl zařízení, navazující úlohy, při jejímž řešení návštěvník získá kontrolu nad konkrétním zařízením, a pokročilé úlohy na úrovni zadání Středoškolské odborné činnosti (SOČ).
3. Vytvořte referenční řešení navržených základních a navazujících úloh a jedné pokročilé úlohy.
4. Ověřte funkčnost celého systému (server, klienti pro jednotlivá zařízení, jejich správná spolupráce, sladění filozofie jednotlivých komponent systému) v demonstrační učebně techniky, identifikované problémy odstraňte.
5. Popište provedené experimenty a jejich výsledky a identifikujte možnosti pro další rozvoj systému.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Doc. Ing. Libor Váša, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **9. září 2022**
Termín odevzdání diplomové práce: **18. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2022

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Plzni dne 18. května 2023

. . . . . .
Alex König

Abstrakt

Tato práce se zaměřuje na zpřístupnění digitálních zařízení studentům. Cílem je vytvořit takové úlohy, které jim umožní osvojit si jejich ovládání. Pro laboratoř techniky Gymnázia Sokolov vznikl systém propojující několik různých zařízení. Tento systém umožňuje zařízením interagovat s jedním virtuálním světem. V rámci této práce vznikly úlohy pro studenty, které byly rozděleny do tří stupňů. První stupeň značí úlohu, která představí technologii začátečníkům, úloha druhého stupně naučí studenta složitější manipulaci a požaduje psaní vlastního kódu, který manipuluje s daty pro daný přístroj. Třetí stupeň je úloha, která složitostí odpovídá středoškolské odborné práci. Pro plnění vzniklých úloh byly vytvořeny aplikace využívající digitální zařízení laboratoře techniky.

Abstract

This work focuses on making specific hardware accessible to students and enable them to learn to develop applications for it. A system linking multiple different devices was developed for a technology laboratory at Sokolov Gymnasium. This system enables all connected devices to operate in one virtual world. This work describes the created tasks for students, that were separated into three classes. The purpose of tasks belonging to the first class is to introduce the device to beginners. Tasks belonging to the second class teach the student more complicated manipulation with the device and require writing and executing user code. Into the third class belong tasks whose difficulty corresponds to high school vocational works. Applications that work with the hardware in the technology laboratory were developed, enabling students to fulfill created tasks.

Klíčová slova

Unity • virtuální realita • VR • hloubková kamera • software pro výuku • RealSense • HTC Vive • herní engine

Poděkování

Na tomto místě bych rád poděkoval panu Doc. Liborovi Vášovi, Ph.D. za odborné vedení práce, připomínky, trpělivost a vstřícný přístup. Dále děkuji pánům Bc. Dominiku Pochovi a Matěji Černému za spolupráci při vývoji aplikací, a panu Bc. Dominiku Frolíkovi za pomoc v tématu digitálních technologiích ve výuce.

V neposlední řadě si poděkování zaslouží také učitelský sbor Gymnázia Sokolov, jmenovitě paní Mgr. Zdeňka Bábíčková, a pánové Mgr. Petr Chlebek a Mgr. Pavel Švácha, a studenti, kteří vyvíjené aplikace testovali.

Obsah

Úvod 1

V posledních několika desetiletích došlo v rychlému, a stále se zrychlujícímu, vývoji technologie. Tím pádem roste i její popularita v důsledku větší cenové dostupnosti pro širší veřejnost. Nejedná se jenom o počítače, notebooky, fotoaparáty, telefony a tablety, ale i o specifičtější zařízení, jako jsou například kamery pro snímání infračerveného záření či hloubky, nebo 3D tiskárny.

Některé z těchto technologií byly ještě v devadesátých letech minulého století považovány spíše za science-fiction, jako například virtuální realita, která se už považuje za běžnou herní platformu, a mnoho lidí s ní už má osobní zkušenost nebo ji má dokonce doma. Jiná zařízení jsou nyní využívána v přístrojích každodenního použití, jako například hloubkové kamery, které našly využití v autech při asistenci parkování. Další jsou samy o sobě stále považována spíše za kuriozity, protože pro ně zatím není využití, které by je vneslo do povědomí laické veřejnosti.

Ale jak tato zařízení vlastně fungují? Jak zpřístupnit tyto technologie studentům zábavnou formou? Jak umožnit se zařízeními manipulovat a případně pro ně psát kód, tak aby začátečníci byli odstíněni od věcí, které by je mohly rychle odradit?

Právě díky myšlence přiblížit moderní digitální technologie studentům, vznikl projekt ve spolupráci Fakulty aplikovaných věd Západočeské univerzity a Gymnázia Sokolov. Jeho cílem je zařídit demonstrační laboratoř techniky na Gymnáziu Sokolov a vybavit ji hardwarem a softwarem, který umožní studentům seznámit se s funkcionalitou zařízení a motivuje je k dalšímu studiu v oboru technologie a programování. Práce byla podpořena projektem IKAP CZ.02.3.68/0.0/0.0/19_078/0017823.

V této práci je nejprve detailněji popsána vize celého projektu a návrh řešení, které bylo implementováno. Dále je zde popsáno teoretické seznámení se zařízeními, pro které byl v rámci této diplomové práce vyvíjen software. Poté je zde popis vypracovaného software a popis dosažených výsledků. V přílohách se nachází uživatelská dokumentace a další doplňující informace, které byly dodány spolu se softwarem.

Tato práce vznikla ve spolupráci s učiteli z Gymnázia Sokolov - Mgr. Zdeňkou Bábíčkovou, Mgr. Petrem Chlebkem a Mgr. Pavlem Šváchou. Dále se na vývoji celkového produktu podílel Matěj Černý a Bc. Dominik Poch[Poc23] z Fakulty aplikovaných věd a Bc. Dominik Frolík z Pedagogické fakulty Západočeské univerzity.

Vize projektu 2

Cílem této práce je vytvořit systém, který umožní studentům získat přehled o funkcích jednotlivých zařízení a propojí laboratoř techniky do jednoho celku. Dále vytvořit úlohy nebo aktivity pro studenty, k jejichž úspěšnému splnění potřebují použít vybavení laboratoře. Aby studenti mohli s tímto systémem dále pracovat a doplňovat jej o další aplikace, je třeba vymyslet takovou sadu úloh, která jim umožní osvojit si práci s jednotlivými zařízeními a umožní jim pro ně dále vyvíjet vlastní software v rámci středoškolských odborných prací (SŠOP). Proto budou studentům poskytnuty vzorové aplikace a jejich zdrojové kódy, které budou moci použít a libovolně modifikovat.

Tato kapitola se zabývá teoretickým úvodem o smyslu využití digitální technologie ve výuce, popsáním hardware, který se nachází v laboratoři, a popisem navrženého řešení software laboratoře.

2.1 Digitální technologie ve výuce

Digitální technologie jsou ve školách využívány běžně. O většině jejich použití už ani nepřemýšlíme. Ve mnoha třídách se nachází stolní počítač, často přímo napojený na dataprojektor, a učitelé tak mohou promítat připravené prezentace, pouštět výuková videa, nebo promítnout QR kód, aby studenti dostali přístup k předpřipraveným otázkám na svém mobilu a mohli ze svých mobilních telefonů vyplňovat rychlé pětiminutové kvízy.

Během pandemie COVID-19 bylo možné pokračovat s výukou distančně po internetu. Efektivita distanční výuky je předmětem diskuzí, ale je jisté, že bez použití digitálních technologií by s výukou nebylo možné pokračovat vůbec.

V roce 2014 byla Ministerstvem školství, mládeže a tělovýchovy připravena Strategie digitálního vzdělávání do roku 2020[MŠM14]. Reaguje na vývoj digitálních technologií a počítá s postupným zapojením digitálních technologií do výuky na základních a středních školách.

Tato strategie zdůrazňuje důležitost výuky znalostí a dovedností z oblasti informatiky, které žákům umožní stát se tvůrci technologií a posílí rozvoj IT sektoru

u nás. Toto je důležité pro udržení a rozvíjení konkurenceschopnosti v Evropě i ve světě[MŠM14]. Dále tato strategie poukazuje na důležitost informatického myšlení (anglicky „computational thinking"). Jedná se o způsob uvažování, který používá informatické metody řešení problémů. Toto myšlení rozvíjí schopnost analyzovat a syntetizovat, zevšeobecňovat, hledat vhodné strategie řešení problémů a ověřovat je v praxi. Studenti se naučí přesně vyjadřovat myšlenky a postupy a zaznamenávat je ve formálních zápisech. Zabývá se věcmi jako jsou algoritmy, struktury, reprezentace informací, efektivita, modelování, informační systémy a principy fungování digitálních technologií[MŠM14].

Mezi cíle vzniklé Strategie digitálního vzdělávání patří:

- zajistit podmínky pro rozvoj digitální gramotnosti a informatického myšlení učitelů a žáků,
- zajistit budování a obnovu vzdělávací infrastruktury,
- zvýšit porozumění veřejnosti cílům a procesům integrace digitálních technologií do vzdělávání,
- zajistit systém podporující rozvoj škol v oblasti integrace digitálních technologií do výuky a do života školy,
- podpořit inovační postupy a sledování a vyhodnocování jejich výsledků.

Vyhodnocení této strategie bylo zveřejněno v roce 2021.[MŠM21a] Na jeho základě vzinkl návrh revizí rámcových vzdělávacích programů (RVP) v oblasti informatiky a informačních a komunikačních technologií.

Cílem nově vzniklé vzdělávací oblasti Informatiky, má být rozvíjet informatické myšlení žáků, s primárním zaměřením na:[MŠM21a]

- data, informace a modelování,
- algoritmizaci a programování,
- informační systémy,
- digitální technologie.

Podle vzniklého RVP je důležité využívat digitální technologie ve všech oblastech vzdělávání tak, aby se staly smysluplnou součástí výuky a podporovaly jak informatické myšlení, tak digitální gramotnost žáků.

V Rámcovém vzdělávacím programu pro gymnázia platném od 1. 9. 2025[MŠM21b] jsou popsány změny v rámci výuky vzdělávací oblasti Informatika. Je zde popsána výuka v rámci jednotlivých výše zmíněných okruhů. Důraz je

kladen na to, aby se žáci učili rozpoznávat situace, kdy je k řešení problému výhodné uplatnit algoritmický přístup, jak shromažďovat relevantní informace a vytvářet, zkoušet a porovnávat různá řešení. Žáci se učí plánovat vývoj řešení v jednotlivých krocích, a své řešení průběžně testují a postupně vylepšují.

Žáci se učí zobecnit řešení konkrétního problému i pro řešení podobných problémů. Pochopení principů fungování digitálních technologií žákům pomáhá lépe porozumět světu kolem nich, rozpoznávat problémy, předcházet jim a nalézat jejich řešení. Je kladen důraz na aktivní přístup žáků k řešení praktických problémů. Obtížnost, rozsah a složitost řešebých problémů postupně roste a žáci se postupně setkávají s větším množstvím úloh s nejasným zadáním a s úlohami s více možnými postupy řešení. Dále je cílem žáky naučit týmové spolupráci na vývoji řešení, adaptaci na nové nástroje, experimentování, přizpůsobení postupů zvolenému nástroji a hledání prostoru pro inovace. [MŠM21b]

Na tyto myšlenky navazuje navržený systém pro laboratoř techniky. Je třeba vymyslet takové úlohy, aby se nároky na studenty postupně stupňovaly, od úloh, které je seznámí s funkcí jednotlivých zařízením až po úroveň zadání středoškolské odborné práce (SŠOP). K vypracování SŠOP musí žáci prokázat více znalostí a schopnost dohledávat si informace. Jednotlivé dodané vzorové aplikace spolu mohou spolupracovat a mohou podporovat spolupráci studentů mezi sebou, rozvíjet jejich komunikační schopnosti a schopnost práce v týmu.

2.2 Laboratoř techniky Gymnázia Sokolov

Pro laboratoř techniky Gymnázia Sokolov byla zakoupena následující zařízení

- interaktivní stůl,
- hloubkové kamery Intel RealSense D415,
- dva headsety pro virtuální realitu HTC Vive Pro 2
- trackery VIVE tracker,
- haptické pero 3D Systems Touch,
- kamery GoPro Hero 7 White,
- několik stolních počítačů,
- vyšívací stroj značky Brother.

2.3 Návrh řešení

Dodaný systém bude vytvořen tak, aby využíval laboratoř jako celek a umožňoval vzájemné propojení a spolupráci co nejvíce zařízení. Zařízení by měla být schopna spolu komunikovat a spolupracovat na úpravách objektů. Zároveň je třeba, aby software umožňoval i úplnému začátečníkovi seznámit se s funkcionalitou přítomných zařízení, a využíval jejich specifické vlastnosti. Student by měl být schopen se znalostmi, které získal díky tomuto systému, a na základě zdrojových kódů dodaných vzorových aplikací, samostatně vyvíjet vlastní aplikace pro různé přístroje laboratoře techniky, tak aby vzniklý software bylo možné propojit se zbytkem systému.

2.3.1 Virtuální svět

Vznikl návrh virtuálního světa, který je společný pro všechna zařízení. Tento virtuální svět je uchováván na serveru, který běží na jednom z počítačů. K serveru se připojují klienti - aplikace, kde každá z nich využívá jedno ze zařízení laboratoře. Klienti mohou do virtuálního světa přidávat objekty, mazat z něj existující objekty, nebo s objekty ve virtuálním světě manipulovat. Například by pak bylo možné, aby se s pomocí hloubkové kamery vytvořil 3D model povrchu určitého objektu, na počítači byl tento model dále upraven a poté by bylo možné si jej ve virtuální realitě ze všech stran prohlédnout, nebo s ním pomocí haptického pera manipulovat.

2.3.2 Stupně složitosti software

Celý systém musí být navrhnut tak, aby byl rozšiřitelný, a bylo možné do něj libovoně přidávat nové klienty pro jakákoli zařízení. Dále je třeba, aby vyvinutý software byl přístupný i začátečníkum. Proto byli klienti pro jednotlivá zařízení rozděleni do tří stupňů podle obtížnosti. Do prvního stupně patří aplikace, které představují uživateli základní funkcionalitu zařízení a zprostředkují mu seznámení se zařízením a jeho vlastnostmi.

Druhý stupeň je určen pro zkušenější uživatele. Umožňuje složitější manipulaci se zařízením a přímo v aplikaci poskytuje uživateli prostor pro psaní vlastního kódu, který je možné vykonat a s jeho pomocí ovlivnit výstup aplikace. Na Gymnáziu Sokolov probíhá výuka programování v jazyce Python, a proto je třeba umožnit psaní kódu v tomto jazyce.

Třetí stupeň jsou aplikace, které mají takovou složitost aby je mohli sami studenti vypracovat v rámci středoškolské odborné práce. K tomu studenti budou potřebovat hlubší porozumění zařízení a musí být schopni pro něj vytvářet aplikace. Proto musí být schopni samostatné práce pod dozorem konzultanta, ať už z Fakulty aplikovaných věd nebo z Gymnázia Sokolov.

2.3.3 Implementovaný software

Pro laboratoř techniky by měly být v rámci této práce a práce D. Pocha[Poc23] vypracovány následující vzorové aplikace:

1. server, uchovávající virtuální svět s objekty, zprostředkovávající komunikaci s klienty,
2. klienti stupně jedna a dva využívající hloubkovou kameru,
3. klienti stupně jedna, dva a tři využívající virtuální realitu,
4. klient pro počítač, umožňující správu objektů na serveru,
5. klient stupně jedna a dva využívající haptické pero.

Tato práce se zabývá hloubkovými klienty a klienty pro virtuální realitu. Dále bude v rámci této práce dodáno zadání pro možnou aplikaci třetího stupně pro hloubkovou kameru, na kterém by mohl samostatně pracovat středoškolský student. Ke všem aplikacím budou dodány i jejich zdrojové kódy, instrukční videa jak aplikace používat a vzorové skripty pro klienty stupně dva pro virtuální realitu a hloubkovou kameru.

Po krátké diskuzi bylo rozhodnuto, že vyšívací stroj nebude pravděpodobně možné ani v budoucnosti do tohoto systému zahrnout. Na vyšívací stroj se dají importovat vlastní designy výšivek ve formátu PES z flash disku. Soubory tohoto typu obsahují příkazy pro stroj - kdy začít vyšívat, kdy skončit, kterým směrem se pohybovat a jak dlouhý má být steh. Neexistuje jednoduchý přímočarý postup, jak převést grafický formát (jako je png nebo jpeg) na formát výšivkový tak, aby vznikl použitelný design. Existuje komerční software, který je tohoto schopný, ovšem jeho cena je značně vysoká. Podařilo se ovšem nalézt plugin do editoru vektorové grafiky InkScape, který umožňuje převádět vektorové obrázky na výšivky. Postup vyšívání pak odpovídá tomu, v jakém pořadí jsou seřazeny vrstvy vektorového obrázku.

Další komplikací bylo, že zakoupené GoPro kamery mají tu nevýhodu, že k jednomu počítači lze v jednu chvíli připojit a z kódu ovládat jen jednu z nich a ne více najednou. Mnoho původně zamýšlených aplikací těchto kamer tedy nebylo možné realizovat.

Zařízení 3

V této kapitole budou popsány technologie, pro které byl v rámci této práce vyvinut software. Bude zde zmíněn obecnější popis daných technologií, a popis konkrétních zařízení tak, aby si čtenář mohl udělat obraz o problematice.

Budou popsány následující technologie:

- virtuální realita - konkrétně HTC Vive Pro 2,
- hloubkové kamery - konkrétně Intel RealSense D415.

3.1 Virtuální realita

Zde je neprve vysvětlen pojem virtuální realita, dále se zde nachází stručný přehled její historie a informace o konkrétním použitém zařízení.

3.1.1 Co je virtuální realita

Virtuální realita (VR) využívá počítač pro vytváření simulovaného virtuálního světa, do kterého může uživatel vstupovat a interagovat s ním. Snaha virtuální reality je uživatele izolovat od reálného světa a poskytnout mu virtuální simulaci jiného světa, který budou jeho smysly považovat za uvěřitelný, i tehdy, když bude v zásadě nerealistický či fiktivní.

VR je někdy ještě dělena na tři typy, podle toho jaký stupeň ponoření do virtuálního světa nabízí[She22]. Jedná se o dělení na:

- Neimerzivní VR - 3D simulované prostředí je vizualizováno pomocí počítačové obrazovky, často je doplněno zvukovými efekty. Uživatel nad ním má kontrolu prostřednictvím počítačové myši a klávesnice (popř. jiného kontroleru). Tato simulace obvykle nemá za cíl uživatele přesvědčit, že se jedná o reálný svět. Příkladem neimerzivní VR jsou běžné počítačové hry.
- Částečně imerzivní VR - k virtuálnímu světu se přistupuje pomocí počítačové obrazovky, nebo častěji brýlí či headsetu. Tento typ VR se zaměřuje na

vizuální aspekt virtuálního světa a neobsahuje reakci na reálný fyzický pohyb uživatele. Narozdíl od neimerzivní VR se snaží vyvolat dojem, že se uživatel ve virtuálním světě přímo nachází. Příkladem jsou letecké simulátory, které se používají pro tréning pilotů, nebo virtuální prohlídky památek.

- Plně imerzivní VR - má za cíl uživatele vtáhnout do přesvědčivého virtuálního světa. Využivá k tomu manipulaci zrakových, sluchových a někdy i hmatových vjemů svého uživatele. Proběhly i pokusy se simulací čichových vjemů. Uživatelé používají speciální headsety nebo brýle, a kontrolery, popřípadě rukavice, a mohou interagovat s objekty virtuálního světa, ve kterém se nacháejí, jako kdyby byly reálné. Dále mohou být využívány i přístroje jako běhací pásy nebo různé trenažery, pro ovládání pohybu ve virtuálním prostředí.

Od zařazení neimerzivní VR pod pojem virtuální reality je nyní spíše upouštěno, a pod pojmem virtuální realita je tedy myšlena plně popřípadě částečně imerzivní virtuální realita.

Další odvětví, která jsou často zařazována pod pojem virtuální realita jsou augmentovaná (anglicky „augmented") a hybridní (anglicky „mixed") realita[She22].

- Augmentovaná realita (AR) - virtuální svět je vytvářen nad reálným světem takovým způsobem, aby jej doplňoval nebo rozšiřoval o další informace. Například se může jednat o aplikaci, která umožňuje lidem pomocí kamery telefonu zobrazit si kus nábytku přímo v jejich pokoji, nebo aplikace, která asistuje chirurgovi před operací a v brýlích pro AR promítne model orgánu přímo na tělo pacienta.

- Hybridní realita - reálný a virtuální svět jsou kombinovány takovým způsobem, který umožňuje interakci mezi reálnými a virtuálními objekty a umožňuje lidem interagovat s virtuálními objekty jako kdyby byly reálné. Jako příklad lze uvést aplikaci, která umí proměnit obrazovku počítače pomocí virutálních tlačítek v dotykovou obrazovku.

Tato odvětví ovšem nemají za cíl izolovat uživatele od reálného světa, ale naopak s reálným světem spolupracují a vytvářejí nad ním jakousi nadstavbu. Nyní jsou klasifikována jako samostatné obory a dohromady s VR se spojují do pojmu rozšířená realita (anglicky „extended reality")[She22]. Když tedy bude v této práci zmiňována virtuální realita, bude tím myšlena částečná, popřípadě plně imerzivní virtuální realita zprostředkovaná pomocí speciálních brýlí či headsetu.

3.1.2 Historie virtuální reality

Historie virtuální reality je delší, než by mnoho z nás čekalo. Samotný pojem virtuální realita vznikl sice až v roce 1987[Ber18], ale první přístroj, který lze pova-

žovat za headset, vznikl už v roce 1960[Soc20]. Nesledoval pohyb uživatele, ale byl schopný zobrazit třírozměrný virtuální svět. Princip zobrazování byl založený na steroskopii, headset měl široké zorné pole a prostorový zvuk.

O rok později vznikl headset s názvem The Headsight[Ber18], který byl schopen sledovat pohyb jeho nositele. Nebyl vyvinut přímo pro VR, ale dálkově ovládal rotaci kamery. Byl používán armádou pro monitoring nebezpečných situací.

První headset připojený k počítači vznikl až v roce 1968[Ber18]. Jeho název byl Damoklův meč (anglicky „Sword of Damocles"). Byl schopen promítat virtuální prostředí generované počítačem, s proměnlivou perspektivou v závislosti na pohybu uživatele. Kvůli velké výpočetní složitosti renderování objektů v reálném čase zobrazoval jen jednoduché wireframe modely. Jeho další nevýhodou byla jeho mohutnost, podle které byl i pojmenován, a pro uživatele byl nepohodlný.

V devadesátých letech byla virtuální realita spíše prostředkem k výcviku armády či astronautů. Pokusy o vývoj komerčních headsetů se nesetkaly s velkým úspěchem u spotřebitelů. Širší popularizace se dočkala až v letech 2010-2020. V těchto deseti letech došlo k rychlému rozvoji VR technologie a její rozšíření mezi širší veřejnost[Ber18].

V roce 2012 byl představen první prototyp headsetu Oculus Rift[Soc20], o dva roky později, v roce 2014, společnost Google vydala produkt Google cardboard[Soc20] - headset, který lze sestavit z kusu kartonu a mobilního telefonu. Vzniklý produkt disponoval snadnou dostupností a měl za cíl dále popularizovat VR mezi širší veřejností a podpořit vývoj amatérských aplikací pro VR. V roce 2016 vyšel headset HTC Vive Pro[Bar22], který měl za cíl konkurovat headsetu Oculus.

Dalším průlomem byl vývoj samostatných VR headsetů, které nepotřebují být připojeny k počítači či konzoli, jako například Oculus Quest, který vyšel v roce 2019. Jeho operačním systémem je systém Android.

3.1.3 Jak VR funguje

Vizuální vjem je nejdůležitější součástí virtuální reality. Pokud se snažíme o co nejuvěřitelnější svět, můžeme se snažit přiblížit vzhledu reálného světa co se týče barev a tvarů. Mnohem důležitější než stylizace je ovšem způsob, jakým je tento svět vnímán.

V klasickém headsetu se nacházejí dvě optické čočky a dva displeje, vždy jedna čočka a jeden displej pro každé oko. Displeje se nacházejí blízko očím, a proto je třeba, aby měly velkou hustotu pixelů.

Pro vyvolání dojmu 3D obrazu je využit princip stereoskopie. Při pozorování světa okolo nás lidský mozek využívá mimo jiné zdánlivý rozdíl polohy předmětu vzhledem k pozadí při pozorování ze dvou míst - dvou lidských očí. Toto je hlavní způsob, jakým vnímáme hloubku zhruba do 5m vzdálenosti, tedy u objektů, které

jsou k nám blízko[SC19]. Proto je v headsetu na každém z displejů zobrazován pohled na virtuální svět z mírně odlišných úhlů pohledu, tak aby se napodobil pohled očima člověka. Mozek si tyto dva obrázky složí do 3D představy o virtuálním světě.

Dalším důležitým faktorem je zorné pole. Lidské oči mají zorné pole okolo 200 stupňů, a z toho na 120 stupních se překrývá obraz z obou očí, což umožňuje hloubkové vidění. Headsety musí nalézt kompromis mezi hustotou pixelů a šířkou zorného pole. Pokud bude zorné pole příliš malé, bude mít uživatel pocit omezeného pohledu na svět. A pokud je překrývající se část zorného pole příliš malá, není možné vnímat virtuální svět stereoskopicky[SC19].

Zpoždění mezi pohybem uživatele a obnovením obrazu zobrazovaném na displejích se nazývá latence. Pokud je jeho hodnota moc velká, uživatel se viditelně pohne ve virtuálním světě se zpožděním oproti jeho reálnému pohybu. Latence je jeden z jevů, které mohou způsobovat nevolnost.

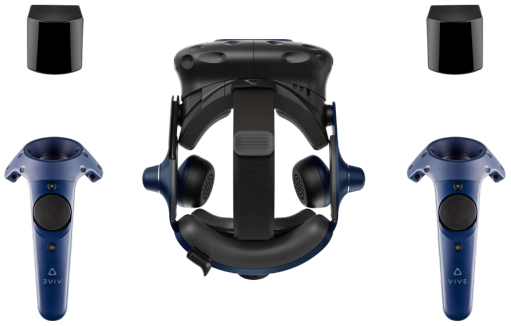
V neposlední řadě kvalitu vnímání virtuálního světa ovlivňuje počet zobrazovaných snímků za vteřinu. Pro vnímání plynulého pohybu na obrazovce počítače stačí zobrazovat 24 snímků za vteřinu, a za standard je považováno 60 snímků. Pro virtuální realitu je ovšem potřeba minimálně 90 snímků za vteřinu, menší hodnoty opět mohou vyvolávat nevolnost[SC19].

Dále moderní VR technologie umožňuje uživateli se ve virtuálním světě pohybovat. Poskytuje tři (3DoF systém) nebo šest (6DoF systém) stupňů volnosti. Systém 3DoF reaguje na změnu orientace uživatele - tedy na rotaci ve 3 osách. Systém 6DoF je schopen reagovat i na změnu pozice člověka, a umožnit mu přirozenější pohyb po virtuálním světě.

Pokud je uživateli umožněno plynule se pohybovat virtuálním světem aniž by se pohyboval ve světě reálném, dochází ke střetu vizuálních vjemů a vjemů z centra rovnováhy ve vnitřním uchu. Tento kontrast opět může některým uživatelům způsobovat nevolnost. Tento nežádoucí efekt může být minimalizován na úkor realističnosti nahrazením plynulého pohybu přemisťováním pomocí teleportace.

3.1.4 HTC Vive Pro 2

Headsety Vive vznikaly za spolupráce společností HTC a Valve. První headset byl představen v roce 2016. Vive využívá techlogie SteamVR, která za pomocí dvou základnových stanic (anglicky „base stations") umožňuje přesnější sledování headsetu a kontrolerů a umožňuje uživateli volně se pohybovat v definovaném prostoru. K plné funkčnosti je tedy třeba headset, dva kontrolery a dvě základnové stanice. Vše je vidět na obrázku 3.1. Dané komponenty si dále detailněji popíšeme.



Obrázek 3.1: Základní součásti balení HTC Vive Pro 2 - headset, kontrolery a dvě základnové stanice[VIV]

.

3.1.4.1 Headset

Headset obsahuje dva displeje, každý s rozlišením 2448x2448 pixelů, poskytuje 120 stupňů široké zorné pole s obnovovací frekvencí 120 snímků za vteřinu[VIV]. Musí být připojen k počítači pomocí kabelu. Umožňuje pohled na své reálné okolí pomocí dvou kamer, což umožňuje zkontrolování toho, kde se člověk momentálně nachází v prostoru bez nutnosti sundavání headsetu. Také má zabudovaná sluchátka s podporou prostorového zvuku. Na jeho přední části se nacházejí senzory, které slouží pro sledování pozice headsetu pomocí základnových stanic. Uvnitř brýlí headsetu se nachází senzor přiblížení (anglicky „proximity sensor"), který slouží k detekci zda má headset někdo na hlavě nebo ne.

3.1.4.2 Kontrolery

Každý kontroler poskytuje pro snímání vstupů od uživatele kruhový touchpad, jedno tlačítko na spoušti a dvě tlačítka po stranách. Horní kruhová část obsahuje senzory sloužící pro sledování pozice pomocí základnových stanic. Kontrolery obsahují znovunabíjecí baterii.

3.1.4.3 Základnové stanice

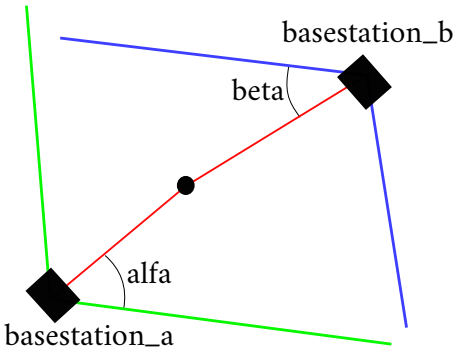
HTC Vive používá inerciální měřící jednotky (IMU) jako primární sledovací systém. IMU je zařízení, které podává informace o zrychlení a orientaci v prostoru. Z IMU lze získat informace o pozici v prostoru s frekvencí 500 Hz - tedy 500 snímků za vteřinu. Tato pozice je značně nepřesná z toho důvodu, že je získávána pomocí dvojité integrace zrychlení detekované pomocí IMU[Xin17]. Pro opravu takto vzniklých nepřenosností používá Vive technologii Lighthouse, která určuje pozici v prostoru za pomocí vysílání světelných signálů. Toto sledování probíhá s nižší frekvencí (60 Hz)[Xin17].

Použité základnové stanice, také označované jako majáky, obsahují pole LED světel a dva infračervené lasery, jeden na horizontálním a druhý na vertikálním rotoru. Majáky jsou malé krabičky, které je třeba umístit do rohů prostoru, ve kterém se bude headset s kontrolery pohybovat. Protože ke sledování používají světlo a potřebují nepřerušenou linii pohledu na sledované objekty, je detekce pozic pomocí základnových stanic citlivá na zakrytí. Použitím dvou majáků na opačných stranách sledovaného prostoru je tento problém minimalizován, popřípadě je možné počet použitých základnových stanic dále zvyšovat[Xin17].

Majáky nepotřebují být připojeny k počítači a slouží jen k produkování světelných signálů, které zpracovávají a získaná data odesílají na počítač headset s kontrolery. Všechna sledovaná zařízení jsou pokryta senzory, které jsou schopny světelné signály detekovat. Postup sledování pozic pomocí technologie Lighthouse je pro jeden maják následovný[Yat16]:

1. maják vyšle synchronizační pulz pomocí LED diod,
2. zařízení pulz zachytí pomocí svých senzorů a spustí časovač,
3. maják vyšle vějíř laserového světla z jednoho z rotorů,
4. laserový paprsek zasáhne jeden ze senzorů na zařízení, zařízení zastaví časovač,
5. maják vyšle synchronizační pulz pomocí LED diod,
6. zařízení pulz zachytí pomocí svých senzorů a spustí časovač,
7. maják vyšle vějíř laserového světla z druhého z rotorů,
8. laserový paprsek zasáhne jeden ze senzorů na zařízení, zařízení zastaví časovač,
9. zařízení odešle získaná data na počítač k dalšímu zpracování.

Aktivní je vždy více než jedna základnová stanice a díky tomu můžeme zjistit pozici objektu ve 3D prostoru. Pokud zjistíme, kdy byl senzor zasažen, a pokud víme, že rotory se otáčejí konstantní rychlostí, známe zorný úhel majáku a přesnou polohu senzorů na konkrétním zařízení, dále pokud je ve vysílaném světelném signálu zakódováno, ze kterého majáku a ze kterého z rotorů byl daný světelný signál vyslán, dokážeme z těchto informací dopočítat pozici a orientaci zařízení. Princip snímání je načrtnut na obrázku 3.2.



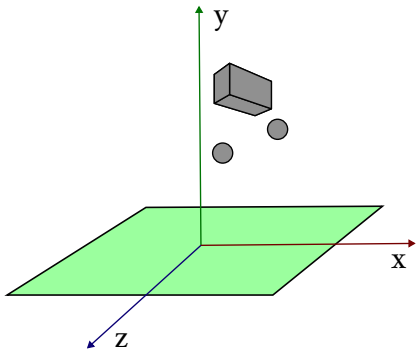
Obrázek 3.2: Schéma znázorňující snímání objektu v horizontální rovině pomocí dvou základnových stanic *basestation_a* a *basestation_b*. Zorné pole základnové stanice je reprezentováno zelenými resp. modrými paprsky. Základnová stanice snímá objekt pod úhlem *alfa* respektive *beta*

.

3.1.4.4 Souřadný systém

Při prvním spuštění zařízení je třeba stanovit prostor, ve kterém se bude uživatel pohybovat. Je možné zvolit mezi dvěmi módy, módu ve kterém je umožněn pohyb v prostoru, a módu určeném jen pro sedícího uživatele. Pokud je zvolen první zmíněný mód, uživatel musí určit, v jak velkém prostoru se bude pohybovat. Toto je možné nastavit kalibrací pomocí kontrolerů. Tento prostor má tvar obecného čtyřúhelníku. Pokud se uživatel během používání VR dostane k pomyslným hranicím vytyčeného prostoru, zobrazí se před ním virtuální stěna.

Počátek souřadného systému virtuálního světa je po spuštění VR aplikace umístěn do středu prostoru určeného k pohybu. Souřadný systém je ilustrován na obrázku 3.3.



Obrázek 3.3: Headset a kontrolery v soustavě souřadnic. Zelený čtverec reprezentuje prostor, ve kterém se uživatel může bezpečně pohybovat.

3.2 Hloubková kamera

Tato sekce je zaměřena na hloubkové kamery. Nejprve budou představeny různé typy hloubkových kamer, poté stručný přehled jejich historie a nakonec bude popsáno konkrétní použité zařízení.

3.2.1 Hloubkový obraz

Nejprve je třeba definovat pojem hloubkový obraz. Hloubkový obraz si lze představit jako matici, ve které je na jednotlivých pozicích uchovávána vzdálenost dané části scény od kamery, která ji snímá. Pokud si představíme běžnou kameru, jejím výstupem je obraz, neboli bitmapa, která obsahuje pixely uchovávající barvu snímané scény. Pro hloubkovou kameru je výstupem hloubkový obraz, tedy bitmapa, ve které hodnota jednotlivých pixelů bude odpovídat vzdálenosti objektů od kamery.

3.2.2 Druhy hloubkových senzorů[Aiv21]

Hloubkové kamery snímají hloubku jednotlivých bodů ve scéně, kterou pozorují, pomocí různých druhů hloubkových senzorů. Hloubkové senzory jsou, jak jejich název napovídá, senzory, které jsou schopny hloubku snímat. Různé typy se od sebe liší způsobem získávání dat, dosahem (tedy tím, jaké minimální a maximální vzdálenosti dokáží snímat) i svým maximálním rozlišením.

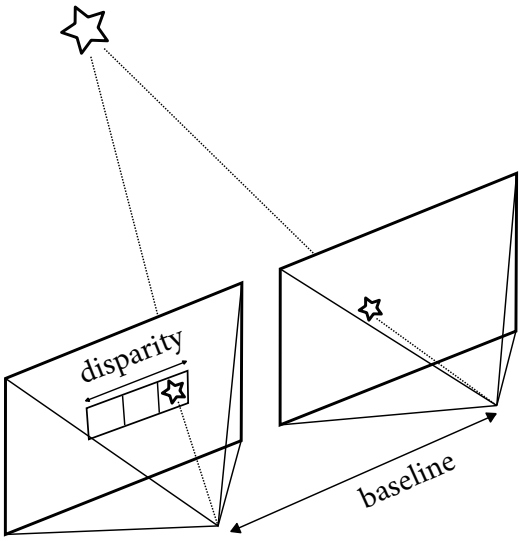
3.2.2.1 Stereo senzory

Stereo senzory fungují na podobném principu jako lidské oči. Mají dvě paralelní kamery, které jsou s od sebe vzdáleny určitý odstup *baseline* a paralelně snímají jednu scénu. Odpovídající části scény jsou namapovány na sebe a do matice zapsán rozdíl mezi jejich pozicemi v jednotlivých obrazech. Tomuto rozdílu se říká disparita. Matici vzniklé výpočtem hodnot disparity pro všechny body ve scéně se říká obraz scény v rozdílové doméně nebo obraz disparity (anglicky „disparity map"). Tento princip je zobrazen na obrázku 3.4.

Informace o disparitě nepřímo odpovídá hloubce daného bodu ve scéně, a proto ji můžeme použít pro extrakci hloubkové mapy. Pro ideálně paralelní kamery lze použít vzorec 3.1[OK93], kde *baseline* je vzdálenost kamer, $disparity$ je hodnota disparity daného bodu a $focal$ je ohnisková vzdálenost kamer.

$$z = (baseline * focal)/disparity \tag{3.1}$$

Nevýhodou stereo senzorů je, že fungují správně jen pro scény, kde se výrazně mění hloubka a kde je měnící se textura. Mají problémy se snímáním „plochých" scén (nebo plochých částí scény), jako například pokud je třeba snímat bílou zeď.



Obrázek 3.4: Princip stereo kamery

Dále vracejí přesné informace jen pro objekty relativně blízko k senzoru. Konkrétní hranice vzdálenosti, pro kterou senzor ještě funguje dobře, záleží na vzdálenosti dvou použitých kamer od sebe. Čím dále jsou kamery od sebe, tím vzdálenější objekty jsou schopny snímat, zvětšuje se tedy jejich maximální snímaná vzdálenost. Také se ovšem zvětšuje jejich minimální snímaná vzdálenost.

Stereo senzory jsou vhodné jak pro vnitřní tak vnější scény, ovšem jen pro ty ve kterých dochází ke změnám v hloubce. Kvalitu hloubkového obrazu, který je produkován stereo senzorem, je možné vylepšit promítáním strukturovaného osvětlení do snímané scény.

3.2.2.2 Time-of-flight senzory

Time-of-flight (ToF) senzory jsou založny na principu vyslání světelného pulzu a vypočtení hloubky podle toho, jak dlouho trval návrat fotonů odražených od povrchu scény. Tyto senzory lze používat pro vnitřní i vnější scény, jsou schopny zaznamenat data s větší hustotou, za objekty ve scéně vznikají menší stíny a mají jednodušší kalibraci než stereo senzory. Oproti stereo senzorům jsou také více odolné vůči chybám vzniklým špatným osvětlením scény. Jsou ovšem citlivé na rozdílné typy povrchů. Neumějí správně snímat lesklé povrchy nebo naopak povrchy, které jsou hodně tmavé.

3.2.2.3 Structured-light senzory

Tento typ senzorů promítá vzor z infračerveného světla. Poté takto osvícenou scénu snímá a podle toho, jak se vzor zdeformoval, zrekonstruují hloubkovou mapu.

Výhodou structured-light senzorů je, že nepotřebují externí zdroj světla, ale jsou vhodné jen pro použití pro vnitřní scény, protože mají nízkou odolnost vůči slunečnímu světlu, které může způsobovat interferenci s promítaným vzorem.

3.2.2.4 Light-Detection-And-Ranging senzory

Princip Light-Detection-And-Ranging (LiDAR) senzorů je podobný jako ToF senzorů, ale oproti ToF senzorům scénu postupně skenují. Používají více než jeden infračervený pulz pro naskenování celé scény a mapují prostředí bod po bodu. LiDAR senzory jsou dražší a nejsou vhodné pro dynamické scény, ale fungují pro větší vzdálenosti než ToF nebo stereo senzory a jsou také přesnější.

3.2.3 Historie hloubkových kamer

Historie hloubkových kamer je značně bohatá a překrývá se s historií kamer obecně. Každý z typů senzorů si prošel svým vlastním vývojem, proto bude v této kapitole poskytnut jen velice stručný přehled.

3.2.3.1 Stereo senzory

Stereo kamery byly vynalezeny již v roce 1853. Byly používány pro vytvoření stereo fotografií, tedy fotografií, které vyvolávají dojem, že je obraz prostorový. Na zopůsob jak použít stereo kamery pro výpočet hloubkových dat se přišlo až o mnoho let později. V roce 1979 představili D. Marr a T. Poggio první algoritmus pro zpracování fotografií ze dvou paralelních kamer pomocí stereoskopie, jakou používají lidské oči[MP79].

3.2.3.2 Time-of-flight senzory[He+17]

První ToF senzor byl vyvinut v roce 1977 ve Stanfordově výzkumném institutu (anglicky Stanford research institute) v Kalifornii. Tato technologie pro snímání hloubky nebyla široce využívána až do roku 1997, kdy vznikly metody pro rychlé vzorkování odráženého světla.

3.2.3.3 Structured-light senzory

Technika structured-light senzorů byla popsána v roce 1982. Její objev je připisován J. L. Posdamerovi a M. D. Altschilerovi[PA82]. První structured-light senzory byly drahé a náročné na použití, ale s vývojem digitálních kamer došlo i k rapidnímu vývoji těchto senzorů, a už v devadesátých letech začalo vznikat vybavení, které se podobá tomu modernímu. Nyní tyto senzory používá NASA ve vesmírných lodích

a vozítek pro jejich navigaci v prostoru[Lie+06][Nef+17], a dále jsou využívány pro analýzu historických artefaktů a v zubním lékařství[Emm].

3.2.3.4 Light-Detection-And-Ranging senzory[MN20]

Laser byl vynalezen v roce 1960, a o rok později byl vynalezen princip senzoru LiDAR. Byl zamýšlen pro použití v leteckém průmyslu a nejprve byl využíván hlavně armádou. Samotný termín LiDAR ovšem vznikl až v roce 1963. V roce 1970 začal být využíván NASA pro měření vlastností oceánu a atmosféry.

Od roku 1970 se začal používat pro topografické mapování povrchu země z letícího letadla, pro které je nejčastěji používán dodnes. Ke zpřesnění získaných dat došlo na konci osmdesátých let minulého století, kdy došlo k vývoji GPS a IMU. Nyní je LiDAR také často využíván v samořídících vozidlech k detekci překážek a pomáhá tak v navigaci vozidel v prostředí.

3.2.4 RealSense D415

RealSense D415 (zobrazena na obrázku 3.5) je hloubková kamera využívající pro snímání hloubky stereo senzor v kombinaci s promítáním strukturovaného infračervého světla.

Obrázek 3.5: Kamera RealSense D415[Int22].

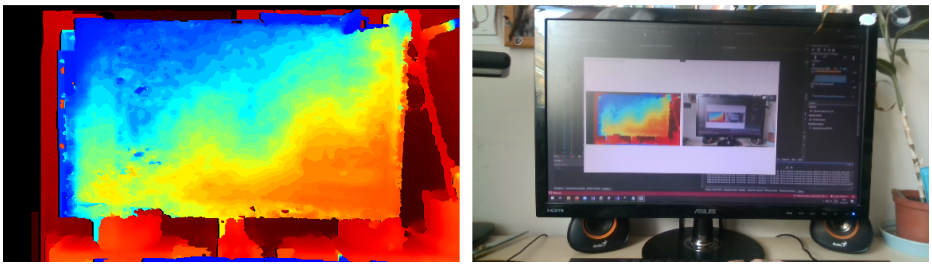
Na kameře se nacházejí dva infračervené senzory, které mezi sebou mají odstup *baseline*, a mezi nimi je umístěn jeden projektor infračerveného světla. Ten promítá infračervené paprsky, které slouží ke zlepšení kvality snímané hloubky. Senzory snímají odražené infračervené světlo a posílají svá data do mikroprocesoru kamery. Veškeré výpočty vedoucí k produkci hloubkového snímku se odehrávají přímo v kameře. Probíhá zde kalkulace hloubkových dat pro jednotlivé pixely hledáním korelace mezi daty z levého a pravého senzoru. Vzniklý hloubkový obraz se posílá na výstup kamery[Tad+22].

Kamera je vhodná pro použití pro vnitřní i vnější scény, vzniklý hloubkový obraz má rozlišení až 1280x720 px a snímková frekvence hloubkového obrazů je 90 snímků za vteřinu. Dále kamera disponuje hloubkovým zorným polem 65°x45° a umí snímat hloubku v rozsahu 45 centimetrů až 3 metry. Pokud se objekty vyskytují blíž, není je možné detekovat, protože pro jejich pixely v obrazu jednoho

senzoru neexistují odpovídající pixely v obrazu druhého senzoru. Hloubkový obraz na výstupu kamery je možné vidět na levé straně obrázku 3.6[Int22].

Dále kamera umí snímat i barevný obrázek scény pomocí jedné obyčejné RGB kamery. Rozlišení tohoto obrazu je až 1920x1080 px, snímková frekvence je 30 snímků za vteřinu a zorné pole je 69°x42°. Barevný obraz na výstupu kamery je možné vidět na pravé straně obrázku 3.6[Int22].

Kameru je možné připojit k počítači pomocí USB kabelu a je k dispozici knihovna *Intel RealSense SDK 2.0*, která zporostředkovává přístup ke kameře. Dále je k dispozici aplikace *Intel RealSense Viewer*, která poskytuje vizualizaci dat z kamery (hloubkový obraz i vytvořené body v prostoru), umožňuje záznam videa a i postprocesing hloubkového obrazu.



Obrázek 3.6: Hloubkový obraz (vlevo) a odpovídající barevný obraz (vpravo) na výstupu kamery.

3.2.4.1 Postprocesing

V přijatém hloubkovém obrazu se mohou nacházet různé nežádoucí artefakty, například se může se jednat o díry, tedy pixely s hodnotou hloubky rovnou nule, nebo o chyby vzniklé nedokonalostí senzoru. Hloubkový obraz je proto vhodné po přijetí z kamery dále upravovat tak, aby byl lépe vizuálně stravitelný pro koncového uživatele, nebo aby byl lépe zpracovatelný koncovou aplikací. Těmto úpravám se říká postprocesing. Úpravy jsou prováděny pomocí filtrů, které jsou na hloubkový obraz aplikovány. Každý filtr reprezentuje jeden algoritmus úprav obrazu. Knihovna *Intel RealSense SDK 2.0* obsahuje implementaci pěti typů filtrů, které si v další části detailněji popíšeme. Knihovna je koncipována tak, aby bylo možné dopisovat vlastní filtry pro zpracování hloubkového obrazu.

Informace o filtrech a o jejich implementaci v knihovně *Intel RealSense SDK 2.0* jsou čerpány z dokumentace Intel RealSense[GT20][Int19].

3.2.4.2 Decimační filtr

Největší kvality hloubkových dat je dosaženo, když je obraz snímán v rozlišení 1280x720 px. Čím větší je rozlišení, tím více je možných úrovní hodnoty disparity,

ale také to znamená tím větší výpočetní složitost pro zpracování obrazu. Většina aplikací, které pracují s hloubkovými daty potřebuje co největší přesnost samotné hloubky, ale většina z nich nepotřebuje tak vysoký počet vzniklých hloubkových bodů. Proto je vhodné obraz podvzorkovat (decimovat) jako první krok postprocesingu.

Princip decimace je takový, že některé z pixelů hloubkového obrazu budou vynechány a tím vznikne obraz s menšími rozměry, například pokud je vynechán každý druhý pixel (vertikálně i horizontálně), vznikne obraz, který má poloviční velikost. Pro větší zmenšení můžeme vzorkovat každý třetí, čtvrtý, pátý nebo i vzdálenější pixel. Takto jednoduchou implementací decimace je ovšem ztrácena část informace, která se v hloubkovém obraze nacházela.

Decimační filtr implementovaný v *Intel RealSense SDK 2.0* má jeden parametr, označme si jej jako *magnitude*. Jeho hodnota je celé číslo, které určuje kolik pixelů má být mezi jednotlivými vzorky vynecháno. Odpovídá tedy tomu, kolikrát bude vstupní hloubkový obraz zmenšen. Dále jako hodnota pixelu, který bude zachován, není použita jen hodnota daného pixelu hloubkového obrazu, ale, aby se omezila ztráta informace, je použit nenulový medián nebo nenulový průměr - tedy medián a průměr počítán jen z pixelů s nenulovou hodnotou hloubky.

Nenulový medián je vhodné použít jen pro malé hodnoty parametru *magnitude*, kvůli jeho výpočetní složitosti, která je závislá na počtu vzorků ze kterého se počítá. Pro větší hodnoty parametru *magnitude* je lepší využít nenulový průměr. Obě tyto metody provádějí základní vyhlazování hloubky. Zároveň díky tomu, že nenulový medián i nenulový průměr je počítán pouze z pixelů, které nemají hodnotu nula (tedy nejedná se o díru), dojde i k základnímu záplatování děr.

Implementace v *Intel RealSense SDK 2.0* umožňuje hodnoty parametru *magnitude* v intervalu celých čísel <2,8>. Pro hodnoty 2 a 3 je používán nenulový medián a pro vyšší je používán nenulový průměr. Z toho vyplývá, že pokud bude například zvolena hodnota $magnitude = 4$ bude vypočten nenulový průměr z pixelů v okolí o rozměrech 4x4.

3.2.4.3 Převod na obraz disparity

Pro hloubkové seznory jako jsou stereo senzory, které počítají hodnoty hloubky pomocí triangulace, se chyba mezi vypočtenou a reálnou hloubkou zvětšuje kvadraticky se vzdáleností objektu od senzoru. Z toho vyplývá, že nastavení některých filtrů, jako jsou filtry vyhlazování, které budou popisovány dále, by mělo být závislé na zpracovávané hloubce. Tím by se mělo předejít přílišnému vyhlazení blízkých a nedostatečnému vyhlazení vzdálených částí scény. Toto chování by ovšem mohlo být pro uživatele matoucí. Aby výsledek vyhlazovacích filtrů nezávisel na rozsahu zpracované hloubky je nutné vyhlazovací filtry neaplikovat na hloubkový

obraz, ale aplikovat je na obraz disparity. Vztah mezi hloubkou a disparitou byl popsán v kapitole 3.2.2.1 vzorcem 3.1.

Vhodný postup je tedy takový, že po aplikaci decimačního filtru, před aplikací vyhlazovacích filtrů, je hloubkový obraz nejprve převeden na obraz disparity, je provedeno jeho vyhlazování a poté je obraz disparity převeden zpět na hloubkový obraz, který může být dále zpracováván.

3.2.4.4 Prostorové vyhlazování

Hloubkový senzor snímá data s určitým šumem. Tento šum může vzniknout například kvůli špatně sejmutým datům z plochých částí scény, jak bylo zmíněno v kapitole 3.2.2.1. Tyto části obrazu je tedy vhodné vyhladit, ovšem v obraze se nacházejí i ostré hrany, které je potřeba zachovat, a které značí rozdíl mezi objektem a pozadím, nebo mezi jednotlivými objekty. Je potřeba využít vyhlazovací filtr, který je schopen tyto hrany zachovávat.

Implementace v *Intel RealSense SDK 2.0* je založena na rekurzivním filtrování popsaném ve článku od E. Gastala [GO11]. Základem tohoto filtru je, že se hloubkový obraz prochází postupně pixel po pixelu v ose x a poté v ose y. Při těchto průchodech se aplikuje 1D filtr, který počítá vážený klouzavý průměr podle vzorce 3.2. Tento vypočtený průměr se zapíše jako hodnota hloubky bodu scény odpovídající danému pixelu. Implementace tohoto filtru v *Intel RealSense SDK 2.0* má ještě parametr *magnitude*, který určuje počet vykonání tohoto algoritmu. Vzorec výpočtu prostorového vyhlazování je následovný

$$S_t = \begin{cases} Y_1 & \text{pokud } t = 1, \\ \alpha Y_t + (1-\alpha) S_{t-1} & \text{pokud } t > 1 \text{ a } \Delta = |S_t - S_{t-1}| < \delta_{thresh}, \\ Y_t & \text{pokud } t > 1 \text{ a } \Delta = |S_t - S_{t-1}| > \delta_{thresh}, \end{cases} \tag{3.2}$$

kde Y_t je hodnota hloubky pixelu t a S_t je nově vypočtená hodnota pixelu t. Parametr $\alpha \in < 0, 1 >$ určuje sílu filtru. Pro hodnotu $\alpha = 1$ platí, že nebude docházet k vyhlazování, a naopak hodnota $\alpha = 0$ způsobuje takové vyhlazování, které kopíruje hodnotu pixelu dokud nenarazí na hranu. Dalším parametrem je δ_{thresh}. Tento parametr určuje minimální velikost hrany, která má být vždy zachována. Jinými slovy všechny hrany, které mají tuto velikost nebo větší budou zachovány a hodnoty jejich pixelů nebudou průměrovány. Velikost hrany Δ je počítána jako rozdíl mezi nově vypočtenou hodnotou pixelu a hodnotou předchozího pixelu.

Ze vzorečku 3.2 je jasné, že odezva filtru není symetrická. Pro utlumení artefaktů, které nesymetričnost způsobí, se ve článku [GO11] používá procházení zpracovávané posloupnosti jedním i druhým směrem. Pro aplikaci na hloubkový obraz to

znamená, že je nutné pixely obrázku projít po řádkách zleva doprava a poté zprava doleva a po sloupcích shora dolů a poté zdola nahoru.

3.2.4.5 Časové vyhlazování

Hloubkový senzor je schopen snímat 90 snímků za vteřinu, ale hodnoty pixelů každého ze snímků jsou počítány zvlášť, bez ohledu na to, jak vypadal snímek předchozí. Pro zvýšení stálosti dat, neboli omezení šumu, který byl způsoben například náhlými změnami v prostředí, jako je změna osvětlení nebo rychlý pohyb objektu, by bylo nutné snížit počet snímků za vteřinu. Toto ovšem negativně ovlivní kvalitu výstupu kamery. Proto je vhodnější využít data z předchozích snímků. K tomu je nutné uchovávat historii snímků v paměti.

Princip filtru časového vyhlazování vychází z toho, že scéna v předchozím snímku bude do velké míry podobná scéně v aktuálním snímku, a je tedy možné aktuální snímek vylepšit pomocí využití dat ze snímku předchozího. V knihovně *Intel RealSense SDK 2.0* je implementován filtr, který opět využívá vážený klouzavý průměr. Pro jeho výpočet podle vzorečku 3.2 využívá aktuální hodnotu pixelu Y_t a S_{t-1} značí hodnotu pixelu na stejné pozici v předchozím snímku. Parametr $\alpha \in < 0, 1 >$ opět ovlivňuje sílu filtru, kde $\alpha = 1$ znamená žádné filtrování a $\alpha = 0$ způsobuje takové vyhlazování, které kopíruje hodnotu pixelu dokud nenarazí na hranu. Parametr δ_{thresh} vypíná filtr pro pixely, u kterých se prudce změnila hloubka - tedy pravděpodobně došlo k pohybu některého z objektů na tomto místě scény.

Pro správné fungování filtru je třeba nezahrnovat do průměrování díry - tedy pixely s nulovou hodnotu hloubky. Naopak lze tento filtr použít tak, aby tyto díry záplatoval. Princip je takový, že každý hloubkový pixel, jehož hodnota je rovna nule, je nahrazen poslední validní, neboli nenulovou, hodnotou pixelu na stejné pozici z uchovávané historie snímků.

Tato implementace filtru má ještě parametr nazvaný *index persistence*, který ovlivňuje, kolik kterých snímků v uložené historii musí mít validní hodnotu na dané pozici aby byla nulová hodnota nahrazena poslední validní. Hloubka historie v implementaci v knihovně *Intel RealSense SDK 2.0* odpovídá osmi snímkům a možné volby podmínky, která určuje kdy bude nulová hodnota pixelu nahrazena poslední hodnotou z historie jsou následující:

- nulovou hodnotu nikdy nenahrazovat,
- nulovou hodnotu nahranit, pokud je v historii na stejné pozici validní hodnota v 8 z 8 uložených snímků,
- pokud je v historii na stejné pozici validní hodnota ve 2 z posledních 3 uložených snímků,

- pokud je v historii na stejné pozici validní hodnota ve 2 z posledních 4 uložených snímků,
- pokud je v historii na stejné pozici validní hodnota ve 2 z 8 uložených snímků,
- pokud je v historii na stejné pozici validní hodnota v 1 z posledních 2 uložených snímků,
- pokud je v historii na stejné pozici validní hodnota v 1 z posledních 5 uložených snímků,
- pokud je v historii na stejné pozici validní hodnota v 1 z 8 uložených snímků,
- nulovou hodnotu vždy nahrazovat.

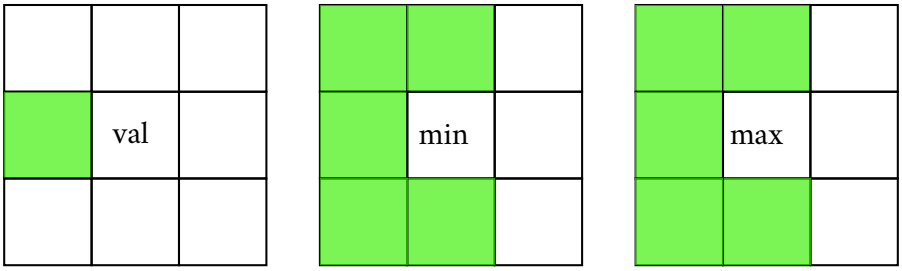
3.2.4.6 Záplatování děr

Jak už bylo zmíněno, v hloubkovém obrazu mohou vznikat díry, neboli pixely jejichž hodnota je rovna nule. Důvodů vzniku děr je několik. Při výpočtu hodnoty pixelu hloubkového obrazu algoritmus stanoví určitou hodnotou jistoty, že spočtená hodnota odpovídá reálné hloubce. Pokud je tato jistota příliš malá, algoritmus si není jistý, jestli vypočtená hodnota odpovídá realitě, a raději zapíše hodnotu pixelu rovnou nule. Dále díry mohou vznikat kvůli chybějícím datům ze senzoru způsobeným zakrytím dané části scény, nebo moc světlou nebo tmavou scénou, nebo tím, že objekt se nacházel příliš blízko k senzorům. Dále po levé straně objektů vznikají díry, které jsou způsobeny tím, že je hloubková mapa počítána relativně k levému senzoru, a pro body scény, které se nacházejí po levém okraji objektů, neexistují odpovídající body v obrazu z pravého senzoru.

V některých případech ovšem může být žádoucí díry ve výsledném hloubkovém obrazu ponechat, například pokud podle hloubkových dat chceme navigovat robota okolo překážek. Člověku je naopak příjemější pozorovat obraz bez děr, takže pokud bude hloubkový obraz vizuálně prezentován přímo uživateli, je lepší díry záplatovat vytvořením nějakého odhadu hodnoty hloubky pixelů. Jednoduché záplatovací metody, které jsou implementovány v knihovně *Intel RealSense SDK 2.0* jsou popsány v této kapitole. První z nich je použití několika levých nebo pravých sousedních validních pixelů k vypočtení hodnoty pro díru - tento přístup je implementován v rámci prostorového vyhlazování. Druhou jednoduchou záplatovací metodou je použití historie snímků, tak jak je popsáno v popisu implementace filtru časového vyhlazování.

Dále lze použít validní hodnoty pixelů z levého okolí. Zde se nabízejí tři metody, které lze vidět ilustrované v obrázeku 3.7. První je použití levého sousedního pixelu, druhou použití pixelu s nejmenší hodnotou z levého okolí a třetí použití pixelu

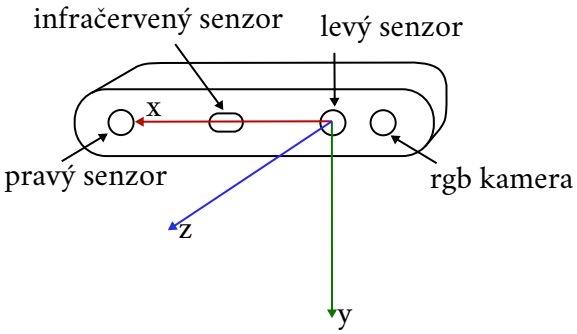
s největší hodnotou z levého okolí. Použití druhé metody je výhodné pro aplikaci filtru na hloubkovou mapu, a třetí metoda je vhodnější pro použití na obraz disparity. Takto popsaný filtr záplatování děr funguje dobře pro opravování děr vzniklých kvůli zakrytí části scény po levé části objektů.



Obrázek 3.7: Možnosti záplatování děr pomocí implementovaného filtru v knihovně *Intel RealSense SDK 2.0*. *Val* značí použití hodnoty zeleného pixelu, *min* použití nenulového minima a *max* použití nenulového maxima.

3.2.4.7 Souřadný systém

Počátek souřadné soustavy, tedy bod $[0, 0, 0]$, odpovídá pozici jejího levého senzoru kamery. Souřadnicový systém je pravotočivý a je vidět na obrázku 3.8.



Obrázek 3.8: Souřadnicový systém kamery RealSense D415. Kamera je načrtnuta zepředu.

Server a komunikace 4

V této kapitole se nachází stručný popis funkcí serveru a knihoven použitých pro komunikaci. Popsány jsou jen ty části, které jsou potřeba k pochopení implementace klientských aplikací. Další detaily jsou dostupné v diplomové práci Dominika Pocha[Poc23], který tuto část systému vyvíjel.

4.1 Server

Všechny klientské aplikace pracují se stejným virtuálním světem. Server slouží jako autorita tohoto celého virtuálního světa a objektů, které se v něm nacházejí. Uchovává informace o všech objektech a zpracovává zprávy od jednotlivých klientů. Komunikace je realizována pomocí knihovny SignalR a pomocí REST api. Oboje může používat jak HTTP tak i HTTPS protokol.

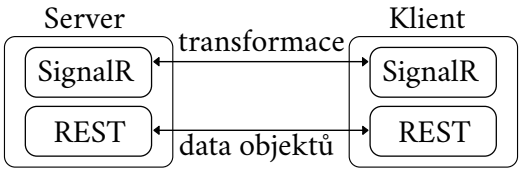
SignalR je knihovna pro .NET, která umožňuje serveru posílat asynchronní notifikace klientům, a klientům umožňuje asynchronně zasílat zprávy serveru. Jedná se o abstraktní vrstvu nad WebSockety. Přes SignalR se klient zaregistruje u serveru, vytvoří se session a je mu přiřazen identifikátor, který umožňuje klienta jednoznačně určit. Klient SignalR je použit k zasílání aktualizací polohy, rotace a škálování objektu.

REST api je rozhraní, které definuje komunikaci pomocí HTTP protokolu, bez potřeby vytváření session. Právě REST api slouží ke komunikaci, ke které je potřeba zasílání většího objemu dat, protože knihovna SignalR má omezenou maximální velikost zpráv. Zde se jedná o manipulaci s daty objektů ve virtuálním světě, jako je zasílání nových objektů na server nebo změny vlastností těchto objektů. Aby bylo umožněna veškerá správa objektů jen přes REST api, odehrává se přes něj i smazání objektů. Tím pádem mohou i klienti, kteří se k serveru nepřipojí přes SignalR, manipulovat s objekty virtuálního světa.

REST api je bezestavové, a samo o sobě neví, od kterého z klientů přišla zpráva o manipulaci s objektem virtuálního světa. Pokud se jedná o zprávu obsahující změnu objektu, kvůli udržení konzistence dat je potřeba, aby se o této změně dozvěděli všichni ostatní klienti. Aby server věděl, od koho zpráva přes REST api přišla,

přidává klient do HTTP headeru informace o svém ID SignalR připojení. Server tak ví, komu nemá zprávu o změně objektu zasílat.

Každý objekt na serveru musí mít přiřazené unikátní jméno, které mu přiřazuje klient při odesílání objektu na server. S pomocí jména je identifikován, když klient zasílá aktualizace jeho transformace nebo vlastností, a také se podle něj určuje, který objekt uložený na serveru bude smazán.



Obrázek 4.1: Schéma komunikace klienta k serveru.

Stručné schéma komunikace serveru a klienta je na obrázku 4.1. Logika připojení klienta a jeho komunikace se serverem je implementována v C# knihovně *Common* a v jejím wrapperu pro Unity *Common.Unity*.

4.2 Knihovna *Common*

V knihovně *Common* je definován formát objektů, které může klient posílat na server, a které z něj umí také přijímat. K definici slouží třída *WorldObjectDto*, kterou je možné vidět v následující ukázce kódu:

```
1  public class WorldObjectDto
2  {
3      public string Name { get; set; }
4      public RemoteVectorDto Position { get; set; }
5      public RemoteVectorDto Rotation { get; set; }
6      public RemoteVectorDto Scale { get; set; }
7      public string Type { get; set; }
8      public Dictionary<string, byte[]> Properties
9              { get; set; }
10 }
11
12 public class RemoteVectorDto
13 {
14     public float X { get; set; }
15     public float Y { get; set; }
16     public float z~{ get; set; }
17 }
```

Zasílané objekty mají své jméno *Name*, které musí být unikátní, a informace o jejich transformaci - vektor s informací o pozici v prostoru *Position*, vektor s informací o rotaci v eulerových úhlech *Rotation* a vektor škálování *Scale*. Ve slovníku

Properties jsou uloženy jednotlivé vlastnosti daného objektu. Klíčem je název dané vlastnosti a data jsou uložena jako pole datového typu byte. Řetězec *Type* definuje o jaký typ objektu se jedná. Od něj se také odvíjí, jaké vlastnosti lze očekávat ve slovníku *Properties*. Ideálně by všechny klientské aplikace zobrazující virtuální svět měly podporovat všechny typy objektů, které se na server posílají. Ve vytvořených klientských aplikacích jsou na server posílány objekty typu Mesh (polygonální síť) a Bitmap (bitmapa), se kterými všechny klientské aplikace zobrazující virtuální svět umí pracovat.

Pro objekt typu Mesh se zasílají následující vlastnosti:

- pozice vertexů,
- typ použitého primitiva,
- indexy vrcholů primitiv,
- texturovací souřadnice,
- šířka a výška textury,
- formát pixelů textury,
- data pixelů textury.

Pro objekt typu Bitmap se zasílají vlastnosti:

- šířka a výška textury,
- formát pixelů textury,
- data pixelů textury.

Dále jsou v této knihovně definovány třídy, které poskytují metody sloužící k připojení k serveru a k zasílání dotazů a objektů na server. Jedná se o třídy *SignalRSession* a *ServerSessionAdapter* pro připojení pomocí SignalR a třídy *RestDataClient* a *ServerDataAdapter* pro připojení přes REST api.

Dále jsou zde třídy, které umožňují serializaci obou typů objektů a jejich vlastností - třídy *RawMeshSerializer* a *RawBitmapSerializer*. Pro serializaci jednotlivých vlastností jsou používány třídy *ArraySerializer*, umožňující serializaci pole, *IntSerializer*, umožňující serializaci celého čísla, a *StringSerializer*, umožňující serializaci řetězce.

4.3 Knihovna *Common.Unity*

Aplikace vyvíjené v herním engine Unity používají Unity wrapper pro knihovnu *Common - Common.Unity*. Zde se nacházejí wrappery pro třídy popsané v kapitole 4.2 - *ServerSessionAdapterWrapper*, *SignalRSessionWrapper*, *ServerDataAdapterWrapper* a *RestDataClientWrapper*. Tyto wrappery jsou potřeba, aby bylo možné přímo v editoru Unity nastavit vše potřebné pro připojení k serveru a pro následnou správu objektů. Dále se zde nachází třída spravující připojení k serveru specifická pro Unity aplikace *TechnologyLabServerConnection*, která umožňuje manipulovat najednou s REST api i připojením pomocí SignalR.

Objekty virtuálního světa, tedy objekty přijaté ze serveru, a objekty, které klient na server zaslal, jsou uloženy ve slovníku ve skriptu *WorldObjectMemoryStorage*. Do tohoto slovníku lze ukládat jen objekty toho typu, jehož prefaby (definice prefabu se nachází v kapitole 5.1.2.4) jsou uloženy ve třídě *PrefabStorage*. Definice použitých prefabů je v kapitole 5.1.4. Zde je tedy třeba mít uložený prefab každého typu objektu, který má aplikace umět zpracovávat.

K manipulaci s objekty slouží skript *WorldObjectManager*. Poskytuje asynchronní metody k načtení všech objektů na serveru (*LoadServerContentsAsync*), přidání objektu na server (*AddObjectAsync*), odstranění objektu ze serveru (*RemoveObjectAsync*) a další. Ke komunikaci používá skript *ServerDataAdapterWrapper*. Pracuje jak s lokální úschovnou objektů *WorldObjectMemoryStorage* a se serverem najednou, tak aby data zůstala konzistentní. Dále se zde nachází skript, který umožňuje reakci na události, které se odehrály na serveru - *ServerEventsHandler*, a skript, který serveru reportuje lokální změny objektů - *WorldObjectEventsHandler*.

Každý prefab uložený v *PrefabStorage* musí obsahovat jako komponentu (definice komponenty se nachází v kapitole 5.1.2.2) skript implementující rozhraní *IPropertiesManager*. V *Common.Unity* jsou připraveny skripty pro objekty typu Mesh a Bitmap - *MeshPropertiesManager* respektive *BitmapPropertiesManager*. Dále prefab může obsahovat jako komponentu skript *OptionalPropertiesManager*, která umožňuje přidávání dalších vlastností objektu do slovníku popsaném v kapitole 4.2.

Vlastnosti je možné přidávat vytvořením skriptu, který dědí od abstraktní třídy *OptionalProperty*. Každá vlastnost musí mít přiřazené unikátní jméno, které se používá jako klíč ve slovníku *Properties* popsaném v kapitole 4.2. Dále musí být implementována metoda *Process*. Tato metoda je volána vždy, když je ze serveru přijat nový objekt. Jako parametr přebírá slovník obsahující všechny serializované vlastnosti objektu. Od implementace této metody se očekává, že hodnotu dané vlastnosti deserializuje a zpracuje. Dále je třeba implementovat metodu *Serialize*. Tato metoda je volána při odesílání objektu na server. Od implementace se očekává, že serializuje hodnotu dané vlastnosti do pole datového typu byte. Pokud prefab objektu má tedy

mít nějakou přidanou vlastnost, je nutné přidat skript popisující tuto vlastnost jako komponentu na prefab objektu, a odkaz na něj přidat do seznamu *_optionalProperties* ve skriptu *OptionalPropertiesManager*.

Klienti 5

Dále budou popsány klientské aplikace implementované v rámci této diplomové práce. Jedná se o másledující aplikace

- Annoying fly - klient prvního stupně pro hloubkovou kameru,
- DepthMesh - klient druhého stupně pro hloubkovou kameru,
- Box City - klient prvního stupně pro virtuální realitu,
- Brush Export - klient druhého stupně pro virtuální realitu,
- PaintVR - klient třetího stupně pro virtuální realitu.

Také zde bude popsané navržené zadání klienta třetího stupně pro hloubkovou kameru a vytvořená knihovna pro vykonávání uživatelského Python kódu v runitime .NET.

5.1 Vývoj pro virtuální realitu

Pro vývoj aplikací pro virtuální realitu je vhodné využít herní engine. Jeho použití usnadní vývoj a umožní soustředit se na specifické funkce vyvíjené aplikace.

5.1.1 Herní engine

Herní engine je software, který je uzpůsoben primárně k vyvíjení počítačových her. Uživateli poskytuje širokou nabídku nástrojů k jejich vývoji. Typicky poskytuje renderer pro 2D a/nebo 3D grafiku, simulaci fyziky, detekci kolizí a reakci na ně, zvukový systém, systém pro přehrávání a popřípadě i vytváření animací, systém, který se stará o detekci a správu vstupu od uživatele a další. V herním enginu je většinou také umožněno uživateli psát vlastní skripty, které se budou za běhu vyvíjené aplikace vykonávat.

V jednom herním enginu lze většinou vyvíjet pro více cílových platforem. Konkrétně pro které, záleží na daném enginu. V jednom enginu je možné vyvíjet široké

spektrum aplikací. Jeho použití je vhodné primárně pro aplikace, které využívají 2D nebo 3D grafiku, a obzvláště výhodné pokud mají tyto aplikace mít i simulaci fyziky.

Pro aplikaci Annoying fly a pro všechny implementované VR klienty byl využit herní engine Unity. Herní engine Unity byl zvolen, protože pro implementaci serveru a komunikace byla zvolena knihovna SignalR. Tato knihovna je v jazyce C# a herní engine Unity umožňuje skriptování a používání pluginů v tomto jazyce. Tento herní engine je přístupný i začátečníkům, hlavně díky obsáhlé dokumentaci, dostupnosti návodů a aktivní komunitě. Toto ocení primárně žáci Gymnázia Sokolov, když budou vzorové aplikace upravovat nebo podle nich vytvářet vlastní klientské aplikace.

5.1.2 Herní engine Unity[Uni23a][Gol10]

Pro pochopení vývoje klientských aplikací si dále definujeme pár pojmů, které jsou třeba k pochopení herního engine Unity.

5.1.2.1 Scéna

Unity scéna je kontejner, který obsahuje herní objekty. Při vytvoření nového projektu se vytvoří defaultní scéna, která obsahuje kameru a směrové světlo. Směrové světlo slouží jako zdroj světla ve scéně. Kamera je ve scéně třeba, aby se při spuštění aplikace scéna vykreslovala na obrazovku. Může být ortografická, pro 2D projekty, a nebo projektivní, pro 3D projekty.

Jednu scénu si lze představit jako jednu úroveň počítačové hry. Scény lze ukládat a libovolně mezi nimi přepínat. V jedné scéně tedy lze vytvořit herní prostředí, uložit ji na disk a poté se vždy, když bude scéna načtena, zobrazí stejné prostředí. Rozdělení aplikace na scény zkracuje čas potřebný k načtení levelu a umožňuje při vyvíjení odděleně testovat různé části aplikace. Pro správnou funkci aplikace musí být vždy aktivní jen jedna scéna.

5.1.2.2 Herní objekt

Herní objekty jsou základním stavebním kamenem každé aplikace v Unity. Je to kontejner pro komponenty, které popisují jeho vlastnosti a chování. Herní objekty se umisťují do scény, a lze z nich vytvářet stromovou strukturu. Každý herní objekt má vždy maximálně jednoho rodiče, a může mít více potomků.

Každý herní objekt má minimálně jednu komponentu - komponentu *Transform*. Ta obsahuje informace o transformaci objektu, tedy jeho golbální pozici ve scéně, lokální pozici relativní vůči jeho rodiči, globální a lokální rotaci, a globální a lokální škálování. Všechny tyto vlastnosti jsou instance Unity třídy *Vector3*, která uchovává

jednotlivé souřadnice na osách x, y, a z. Udávají tedy pozici na jednotlivých souřadných osách, rotaci okolo dané osy nebo zvětšení či zmenšení ve směru dané osy. Dalšími komponentami mohou být kolidery, které slouží k detekcím kolizí, komponenta *Rigidbody*, která umožní fyzikální simulaci pohybu daného objektu nebo komponenty *MeshFilter* a *MeshRenderer*, které jsou třeba pro vykreslení trojúhelníkových sítí.

Herní objekt také může být označen tagem. Stejný tag může být přiřazen jednomu nebo více objektům. Jedná se o slovo, které slouží k identifikaci daného herního objektu.

5.1.2.3 Skritpování

Na herní objekty lze také umisťovat skripty. Tyto skripty umožňují vyvíjení nových vlastností a specifických reakcí na situace, které během běhu aplikace mohou nastat. Skripty v Unity lze psát v jazyce C#. Pokud má být skript umístěn jako komponenta na objekt, musí dědit od třídy *MonoBehaviour*. Tato třída poskytuje metody typické pro Unity komponenty - nejčastěji využívané metody jsou:

- *Start* - tato metoda je volána jednou při spuštění scény,
- *Update* - tato metoda je volána vždy při renderování nového snímku,
- *OnCollisionEnter* - tato metoda je volána pokud dojde ke kolizi s koliderem herního objektu,
- *OnCollisionExit* - tato metoda je volána pokud objekt, který kolidoval s daným herním objektem tuto kolizi opustí a jejich kolidery se přestanou dotýkat.

5.1.2.4 Prefab

Prefab reprezentuje herní objekt spolu s jeho komponentami a potomky uložený pro další použití. Uchovává se v něm také nastavení parametrů jednotlivých komponent. Prefaby je možné přenášet mezi scénami bez toho, aby jejich vlastnosti bylo nutné znovu nastavovat, nebo je možné je exportovat do jiných Unity projektů. Díky prefabům je možné herní objekty klonovat a vytvářet ve scéně kdykoli za běhu aplikace.

5.1.2.5 Aset

Aset je reprezentace jakéhokoli objektu, který může být použit v Unity projektu. Aset může vzniknout ze souboru vytvořeného mimo herní engine Unity - například 3D model, audio soubor nebo obrázek. Aset lze také vytvořit přímo v Unity - například prefab nebo uložená scéna. Asety jsou uchovávány ve složce „Assets" Unity projektu.

5.1.2.6 Pluginy a Unity balíčky

Pro rozšíření funkcionality lze psát vlastní kód nebo importovat kód, napsaný někým jiným. K importování lze použít pluginy nebo balíčky. Rozdíl mezi nimi je, že plugin je kompilovaná knihovna vyvíjená mimo Unity, a balíček je skupina asetů a kódu exportovaných z Unity.

Balíčky lze získat a spravovat v okně Package Manager. Zde je možné balíčky instalovat nebo odstraňovat a aktualizovat. Do Unity projektu lze přidávat kompilované .NET .dll soubory. Stačí je umístit do složky „Assets". Tyto .dll soubory musí být kompatibilní s .NET 2.0.

5.1.2.7 Vstup od uživatele

V herním engine Unity verze 2019.1 a novějším se pro správu vstupu od uživatele doporučuje používat Unity balíček *Input System*. Byl vyvinut, aby řešil problémy s kompatibilitou mezi různými platformami a aby zjednodušil případné přemapování ovládacích prvků ve hře.

Šest hlavních částí balíčku *Input System* jsou

- aset *Input Actions*,
- ovládací schémata (anglicky „control schemes"),
- mapy akcí (anglicky action maps),
- propojení na vstupy z periferie počítače (anglicky „bindings"),
- komponenta *Player Input*.

Aset *Input Actions* slouží k uchování ovládacích schémat, map akcí a jejich propojení na vstupy z periferie počítače. Uvnitř tohoto asetu lze vytvářet ovládací schémata. Ovládací schémata se používají pro organizaci vstupů z odlišných periferií, kterými hráč může aplikaci ovládat. Pro větší přehlednost je vhodné jednotlivá zařízení od sebe oddělit.

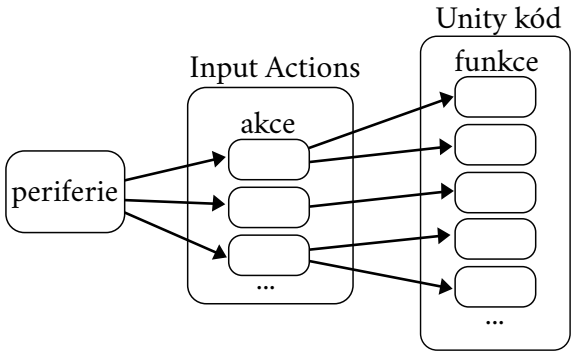
Uvnitř jednoho ovládacího schématu lze vytvářet mapy akcí. Mapy akcí se používají pro oddělení ovládání různých typů aktivit, které může hráč v aplikaci provádět, například pro ovládání herní postavy a pro ovládání menu aplikace je vhodné vytvořit oddělené mapy akcí. Uvnitř jedné mapy akcí se nachází seznam akcí, které se používají pro danou aktivitu, například pro ovládání herní postavy by zde mohla být akce chůze, akce pro otáčení a akce pro skok.

Akce jsou jednotlivé události, které spuštějí funkce Unity skriptů umístěných ve scéně. Propojují vstup z periferie s reakcí, která je provedena ve hře. K jakému vstupu jsou akce navázány je určeno pomocí takzvaných bindings. Bindings definují,

jaká akce bude spuštěna, když uživatel například stiskne mezerník nebo levé tlačítko myši.

Aby šlo takto vytvořený aset *Input Actions* použít, je třeba jej vložit do scény. Pro propojení s herním objektem lze použít komponentu *Player Input*. Tomuto skriptu se předá vytvořený *Input Actions* aset. Tento skript může zasílat zprávy jednotlivým objektům aktivním ve scéně, a tím spouštět metody jejich komponent, které svým jménem odpovídají názvům daných akcí. Další možností je, že se v Unity editoru u komponenty *Player Input* k jednotlivým akcím nastaví odkazy na odpovídající metody, které mají být volány jako reakce.

Jednotlivým akcím je možné pomocí skriptu přiřadit funkce, které budou volány, bez použití skriptu *Player Input*. Ve skriptu se nastaví odkazy na dané akce, na jejichž události poté budou navázány jednotlivé funkce, které mají být volány. Pro interakci s uživatelským rozhraní poskytovaným v Unity objektem typu Canvas se používá komponenta *Input System UI Module*. Zjednodušený náčrt principu zpracování vstupů od uživatele je na obrázku 5.1.



Obrázek 5.1: Ilustrace principu zpracování vstupu od uživatele.

5.1.3 Standard OpenXR

Headset HTC Vive Pro 2.0 používá pro připojení k počítači a výpočet pozic headsetu a kontrolerů aplikaci SteamVR. Tato aplikace mimo to také umožňuje přístup k platformě Steam ve virtuální realitě - platformě umožňující publikaci nebo zakoupení počítačových her a aplikací.

Pro komunikaci se SteamVR doporučuje Vive pro vývoj aplikací pro jejich headsety implementovat standard OpenXR[Khr23], vytvořený organizací Khronos. Jeho cílem je zjednodušit vývoj aplikací pro virtuální a augmentovanou realitu tím, že umožňuje napsat kód tak, aby byl přenositelný na více platforem. Odstraňuje tedy nutnost portování a přepisování aplikace při sestavení pro jinou VR nebo AR platformu. Pro herní engine Unity existuje stejnojmenný balíček *OpenXR*[Uni22] implementující tento standard. Dále je rozšířený balíčkem *XRInteractionTookit*[Uni23b]

o skripty spravující vstup z kontrolerů, skripty zprostředkovávající pohyb v prostoru, skripty umožňující interakci s objekty a skripty zprostředkující haptické reakce.

V nastavení Unity projektu lze vybrat aktivní profil interakce z nabídky podporovaných profilů. Tento profil značí, pro jaké zařízení se bude projekt spouštět nebo sestavovat, a tedy jak budou namapovány jednotlivé vstupy popisované standardem *OpenXR* na konkrétní signály ze zařízení. Je důležité projekt spouštět vždy s odpovídajícím profilem v závislosti na použitém VR zařízení. Hlavní výhodou tohoto přístupu je, že projekt není závislý na konkrétním zařízení, a lze tedy vyvíjet a testovat na jiném zařízení než na jakém bude aplikace finálně nasazena, popřípadě vyvíjet pro více zařízení najednou.

Balíček *XRInteractionToolkit* dodává aset typu *InputActions* obsahující přednastavené mapování vstupů na jednotlivé akce. Díky tomu se nemusí při vyvíjení aplikace nejprve nastavovat ovládací schémata a mapy akcí, ale stačí jen použít tento aset. Tento balíček poskytuje řadu skriptů, které usnadňují vývoj aplikací pro VR. Lze je rozdělit do čtyř hlavních skupin:

- skripty ovládající kontrolery,
- interaktory, neboli skripty umožňující interakci s virtuálním světem,
- skripty umožňující objektům reagovat na interakci,
- skripty umožňující pohyb ve virtuálním světě.

Mezi skripty ovládající kontrolery patří skript *XRController*, který prováže akce z mapování uchovaném v asetu *InputActions* s konkrétními funkcemi, které se mají vykonat.

Interaktory, jako *XRRayInteractor* nebo *XRDirectInteractor*, se umisťují na kontrolery. Definují jakého typu interakce je daný kontroler schopen. Použití *XRDirectInteractor* znamená, že daný kontroler může interagovat s objektem jen pokud je s ním v kolizi. Naopak *XRRayInteractor* vysílá z kontroleru paprsek po lokální ose z, tedy dopředu, a daný kontroler pak může interagovat s prvním objektem, který je tímto paprskem protínán.

Mezi skripty umožňující objektům reagovat na interakci patří například *XRGrabInteractable*. Tento skript umožňuje objektu být uchopen kontrolerem a umožňuje uživateli s tímto objektem pohybovat. Dále se jedná například o skript *TeleportationArea*, který umožňuje teleportaci na libovolné místo objektu, na kterém je umístěný.

Mezi skripty umožňující pohyb patří skripty *ConinuousTurnProvider* a *SnapTurnProvider*, které poskytují plynulou rotaci respektive skokovou rotaci o zadaný úhel a skript *ContinuousMoveProvider*, který umožňuje plynulý pohyb v prostoru.

Balíček *XRInteractionToolkit* poskytuje také prefab pro rig, neboli headset s kontrolery, a asety se skripty ovládající kontrolery a asety se skripty umožňující pohyb, které mají předvyplněné mapování na jednotlivé akce tak, aby programátor při vytváření nové VR aplikace nemusel psát vlastaní kód pro zprovoznění vstupu z VR headsetu, a mohl se soustředit se na specifika vyvíjené aplikace. Při vytváření nového projektu tedy stačí pro nastavení vstupu ze zařízení virtuální reality vybrané asety aplikovat na objekty kontrolerů.

5.1.4 Vytvořené prefaby

Všichni Unity klienti, kteří přijímají objekty ze serveru, používají následující prefaby:

- *ServerMeshPrefab* - prefab pro objekty typu Mesh,
- *ServerBitmapPrefab* - prefab pro objekty typu Bitmap,
- *ServerConnectionPrefab* - prefab obsahující všechny skripty potřebné k obsluze připojení k serveru.

Dále bude popsáno, k čemu tyto prefaby slouží, a jaké jsou jejich komponenty.

5.1.4.1 ServerMeshPrefab

Prefab s názvem *ServerMeshPrefab* reprezentuje objekty virtuálního světa typu Mesh. Pro uchování polygonální sítě a její vykreslení ve scéně musí mít tento herní objekt Unity komponenty *MeshFilter* a *MeshRenderer*.

Dalšími komponentami jsou skripty z knihovny *Common.Unity*. Jedná se o skript *MeshPropertiesHandler*, který slouží ke zpracování a nastavení vlastností objektu. Dále skript *OptionalPropertiesManager*, který slouží ke zpracování a nastavení volitelných vlastností objektu, a skript *ReportTransformChange*, který slouží k odesílání změn transformace objektu na server.

Dále se na tomto prefabu nacházejí tři kolidery. Kolider tvaru kvádru *BoxCollider*, kolider *MeshCollider*, jehož tvar závisí na polygonální síti, a kolider ve tvaru koule *SphereCollider*. Na prefabu jsou umístěny tři kolidery proto, že tento prefab reprezentuje obecný objekt virtuálního světa typu Mesh. Podle toho, který konkrétní objekt instance tohoto prefabu reprezentuje, je nastavován aktivní kolider ve scéně.

Pro objekty typu Mesh jsou implementovány následující volitelné vlastnosti:

- *Color*,
- *ColiderType*,
- *ColiderSize*.

Vlastnost *Color* je popsána ve skriptu *ColorProperty*. Jedná se o informace o barvě objektu ve formátu RGB. Tato vlastnost se posílá na server jako pole s daty jednotlivých barevných kanálů

$$array = [r, g, b] \tag{5.1}$$

kde *r* je hodnota červeného kanálu, *g* hodnota zeleného kanálu a *b* hodnota modrého kanálu.

Vlastnost *ColiderType* je popsána ve skriptu *ColiderTypeProperty*. Tato vlastnost ovlivňuje jaký kolider bude aktivní ve scéně pro daný objekt virtuálního světa. Pro *MeshCollider* je hodnota této vlastnosti řetězec *„mesh"*, pro *BoxCollider* řetězec *„box"* a pro *SphereCollider* řetězec *„sphere"*.

Vlastnost *ColiderSize* je popsána ve skriptu *ColliderProperty*. Tato vlastnost udává velikost a pozici aktivního kolideru. Pro kolider typu *„mesh"* je posíláno prázdné pole *array*.

Pro kolider typu „box" je posíláno pole obsahující pozici středu *c* aktivního kolideru BoxCollider a jeho velikost *s* ve tvaru

$$array = [c.x, c.y, c.z, s.x, s.y, s.z] \tag{5.2}$$

Pro kolider typu „sphere" je posíláno pole obsahující pozici středu *c* a poloměr *r* dané koule ve tvaru

$$array = [c.x, c.y, c.z, r] \tag{5.3}$$

Pole *array* a řetězec reprezentující typ kolideru je vytvářen v metodě *Serialize* odpovídajícího skriptu volitelné vlastnosti. Tato metoda vrací bytové pole, do kterého jsou hodnoty serializovány pomocí třídy *ArraySerializer*, respektive *StringSerializer*. Odkaz na třídy volitelných vlastností je vložen do skriptu *OptionalPropertiesManager* a bytové pole je poté uloženo ve slovníku *Properties*, popsaném v kapitole 4.2. Klíčem je název dané vlastnosti.

Další komponentou je skript *ObjectPropertiesHandler*. Tento skript slouží k nastavování materiálu herního objektu podle toho, o jaký objekt virtuálního světa se jedná. Pokud se jedná o typ Mesh, je nutné o použitém materiálu rozhodnout podle klienta, který jej odeslal. Ze kterého klienta daný objekt pochází je indikováno jeho

názvem. Například pro objekty, které reprezentují tah štětcem nakreslený v aplikaci PaintVR, jméno začíná řetězcem "Line"a je třeba nastavit takový materiál, který zobrazí obě strany daného trojúhelníkového pásu. Je tedy třeba takový materiál, u kterého dochází k vykreslování přivrácené i odvracené strany trojúhelníků.

V aplikaci PaintVR skript *ObjectPropertiesHandler* také slouží k nastavení správné interakční vrstvy a tagu pro objekty reprezentující tahy štětcem. V aplikaci Box City tento skript nastavuje objektům, které nemají žádný aktivní kolider, kolider typu *BoxCollider*. Pozice jeho středu a jeho velikost se nastaví podle hranic dané polygoniální sítě. V aplikaci BoxCity se na tomto prefabu nachází ještě jedna komponenta. Jedná se o skript *XRGrabInteractable*, umožňující interakci objektu s kontrolerem.

5.1.4.2 ServerBitmapPrefab

Objekty typu Bitmap reprezentuje prefab *ServerBitmapPrefab*. Ten má také jako své komponenty Unity komponenty *MeshFilter* a *MeshRenderer*. Bitmapa je totiž ve scéně reprezentována jako obdélník, na který je jako textura namapována přijatá bitmapa.

Na tomto prefabu se nacháuejí jako komponenty skripty z knihovny *Common.Unity*. Jedná se o skript *BitmapPropertiesManager*, který slouží ke zpracování a nastavení vlastností objektu, skript *OptionalPropertiesManager* a skript *ReportTransformChange*. Další komponentou tohoto prefabu je skript *ObjectPropertiesHandler*, který slouží k nastavení správného materiálu a velikosti kolideru. Na tomto prefabu se nachází kolider ve tvaru kvádru *BoxCollider*. Velikost tohoto kolideru je nastavována automaticky podle velikosti vytvořeného obdélníku ve scéně. V aplikaci Box City se na něm ještě nachází skript *XRGrabInteractable*, který umožňuje uživateli manipulaci s tímto objektem.

5.1.4.3 ServerConnectionPrefab

Tento prefab obsahuje všechny skripty, které jsou třeba ke komunikaci se serverem. Jednotlivé skripty jsou popsány v kapitole 4.3.

5.2 Vývoj pro RealSense

Intel RealSense SDK podporuje mnoho programovacích jazyků, mezi nimi jsou C++, C#, Python nebo Matlab. Dále poskytuje wrappery pro herní enginy Unreal a Unity[Int].

5.2.1 C++ knihovna

Knihovna *Intel RealSense SDK* je psána primárně pro použití v aplikacích psaných v jazyce C++. Většina dokumentace a ukázek kódu dostupných v dokumentaci k *Intel RealSense SDK* je také v jazyce C++.

Nastavení proudu vstupních dat je reprezentováno třídou *config*. Zde se zvolí zařízení, ze kterého mají být přijímána data. Proudy dat, které budou snímány, se aktivují pomocí metody *enable_stream* - parametr této metody je hodnota výčtového typu *rs2_stream* udávající proud dat. Toto nastavení se předá instanci třídy *pipeline*, která dále zpracovává proud vstupních dat.

Knihovna *Intel RealSense SDK* poskytuje implementaci upravování snímků získaných z kamery pomocí filtrů. Obecný postprocesing filtr je reprezentován třídou *filter*. Od této třídy dědí implementace pro jednotlivé specifické filtry. V knihovně jsou implementovány všechny filtry, které jsou popsané v kapitole 3.2.4. Parametry filtru lze nastavit při vytváření instance třídy, nebo pozdeji pro existující instanci pomocí metody *set_option*. Tato metoda má jako parametr hodnotou z výčtového typu *rs2_option*, stanovující, který parametr filtru se má nastavit, a novou hodnotu daného parametru. Parametry jednotlivých filtrů jsou popsány v dokumentaci RealSense [Int19].

Z pipeline lze získat nový snímek pomocí volání metody *wait_for_frames*. Tato metoda vrací instanci třídy *frameset*, která obsahuje snímky ze všech aktivních proudů dat. Pro získání hloubkového snímku poskytuje tato třída metodu *get_depth_frame* a pro získání barevného snímku metodu *get_color_frame*. Na snímek lze aplikovat filtr pomocí volání metody daného filtru *process*, které se jako parametr předává snímek, který se má upravit. Metoda vrací nový, upravený snímek.

5.2.2 .NET wrapper

Pro vývoj v prostředí .NET lze využít NuGet balíček *Intel.RealSenseWithNativeDll*, který automaticky zkopíruje potřebné *realsense2.dll* z *Intel RealSense SDK 2.0*. Pro .NET wrapper Intel neposkytuje dokumentaci, jen github se vzorovými projekty. Princip vývoje je ovšem velmi podobný jako pro C++ knihovnu.

5.2.3 Unity wrapper

Intel poskytuje vzorové scény pro Unity projekty, které umožňují rychlejší orientaci ve skriptech Unity wrapperu. Pomocí Unity wrapperu je možné konfigurovat zařízení, určit filtry aplikované během postprocesingu a nastavit, kam se má výsledný proud dat renderovat. Vše je možné nastavit přímo z Unity editoru bez psaní dalších skriptů.

Mezi dodanými scénami se mimo jiné nachází scény, které demonstrují základní renderování snímané barvy a hloubky do textury, příklad aplikace postprocesing filtrů na snímaný hloubkový obraz, nebo scéna, která ukazuje jak vytvořit množinu bodů v prostoru ze snímaného hloubkového obrazu. Wrapper obsahuje prefab *RealSenseDevice*, který zahrnuje skripty, potřebné k nastavení vstupního zařízení, k určení, které filtry jsou aplikovány v rámci postprocesignu, a k nastavení cíle, kam se má renderovat výsledný proud obrázků.

K nastavení zařízení slouží skript *RsDevice*. Při startu scény se tento skript pokusí odstartovat snímání kamerou. Tento skript umožňuje nastavení zda bude snímána scéna pomocí kamery nebo zda bude přehráván předem zaznamenané video. Pro snímání z kamery umožňuje nastavit jaký proud dat bude snímán - zda bude snímána hloubka nebo barva, popřípadě oboje. Dále umožňuje nastavit jaký bude framerate a rozlišení přijímaných snímků.

K nastavení filtrů slouží postprocesing pipeline. V Unity wrapperu je reprezentována skriptem *RsProcessingPipe*. Zde je možné nastavit filtry, které se budou na stream dat aplikovat před renderováním do textury.

Skript *RsStreamTextureRenderer* se stará o vykreslování vstupního proudu dat do přiřazené textury. Zdrojem dat může být přímo kamera (neboli skript *RsDevice*) nebo postprocesing pipeline (neboli skript *RsProcessingPipe*).

5.3 Knihovna *ZCU.PythonExecutionLibrary*

Pro vykonávání Python kódu v runtime .NET byla vyvinuta knihovna *ZCU.PythonExecutionLibrary*. Tato knihovna je používána ve všech aplikacích druhého stupně pro zpracování uživatelského kódu, tedy je použita v aplikacích Depth Mesh a Brush Export.

Pro implementaci byla použita knihovna *Python.NET*, kterou lze do .NET projektu importovat jako balíček NuGet a do Unity přidáním *Python.Runtime.dll* ze zmíněného balíčku mezi asety projektu. Jedná se o knihovnu která umožňuje integraci Pythonu do .NET prostředí. Konkrétní verze Pythonu, která bude použita při vykonávání kódu, je třeba nastavit v kódu před vykonáním vlastního kódu nastavením cesty k instalovanému *python.dll*.

Python kód je třeba uchovávat správně formátovaný v jednom řetězci. Pro jeho vykonání je třeba nejprve inicializovat Python engine pomocí metody *PythonEngine.Initialize*. S Python enginem může vždy pracovat jen jedno vlákno. Přístup k němu lze uzamknout ostatním vláknům pomocí metody *Py.GIL*.

Dále je třeba vytvořit rámec kódu pomocí metody *Py.CreateScope*. Tato metoda vrací instanci třídy *PyModule*. Nad touto instancí se volá metoda *Set*, pokud je třeba před vykonáním Python kódu nastavit hodnotu některé z jeho proměnných. Jako parametry tato metoda přebírá řetězec se jménem proměnné a její hodnotu.

Samotné vykonání je odstartováno voláním metody *Exec* nad vytvořenou instancí *PyModule*. Jejím parametrem je řetězec, který obsahuje kód, který se má vykonat. Po vykonání lze získat konkrétní hodnotu proměnné python kódu pomocí volání metody *Get* nad instancí třídy *PyModule*. Jejím parametrem je jméno proměnné. Po získání proměnné z Python prostředí lze s její hodnotou dále pracovat, a po ukončení práce s Python enginem je třeba jej opět ukončit voláním metody *PythonEngine.Shutdown*.

Například pokud chceme vykonat následující Python kód

```
r = val
g = 1 - val
b = val
res = [r, g, b]
```

bude implementace vykonání kódu v prostředí C# následující

```
PythonEngine.Initialize();
using (Py.GIL())
{
    using (PyModule scope = Py.CreateScope())
    {
        // nastaví proměnnou val na 0.1
        scope.Set("val", 0.1);
        // vykoná kód
        scope.Exec(code);
        // získá hodnotu proměnné val po vykonání kódu
        PyObject result = scope.Get("res");

        // zpracovat objekt result
    }
}
PythonEngine.Shutdown();
```

V knihovně *ZCU.PythonExecutionLibrary* se nachází třída *PythonExecutor*, která se stará o vytváření a vlastní vykonání kódu. Pomocí metody *SetPython* lze nastavit cestu k požadované verzi Pythonu. Tato cesta musí být nastavena před první inicializací Python prostředí.

Metoda *CreateCode* slouží k vytvoření kódu, který se má vykonat. Jejímy parametry jsou kód funkce *userCode*, který se má vykonat, názvy parametrů této funkce *params* a názvy proměnných se kterými bude funkce volána *paramVals*.

Tvar vytvořeného kódu bude následující

```
    def func(params):
        userCode

    res = func(paramVals)
```

kde *params* jsou parametry funkce, *paramVals* jsou konkrétní proměnné, se kterými je funkce volána, *userCode* je uživatelský kód psaný v aplikaci druhého stupně a význam proměnné *res* bude záviset na konkrétním vykonávaném kódu.

K vykonání kódu slouží metoda *RunCode*, které se jako parametry předávají připravený kód ve výše definovaném tvaru, dalším parametrem je slovník jmen proměnných se kterými je funkce volána a jejich hodnot a instance třídy implementující rozhraní *IReturnable*. Tato třída bude naplněna návratovými hodnotami z proměnné *res* v Python kódu.

IReturnable je rozhraní, které definuje metodu *SetParameters*, kterou je potřeba implementovat k tomu, aby mohl být daný objekt naplněn daty z návratové proměnné uživatelské funkce. Její parametr je proměnná typu *PyObject*. *PyObject* je třída z knihovny *Python.NET*, jedná se o C# wrapper pro jakoukoli Python proměnou. Pokud byl vrácen Python seznam, lze k jednotlivým prvkům seznamu přistupovat přes indexy, jako pro typický C# seznam. Prvky seznamu budou opět instance třídy *PyObject*. Dále třída *PyObject* poskytuje metody pro převod na .NET datové typy.

Pokud například návratovou hodnotou metody func bude Python seznam, který obsahuje RGB barvu, která bude uchovávána v C# proměnné *result*, převede se na instanci Unity třídy *Color* následovně

```
CultureInfo baseCulture = new CultureInfo("en-US");
float r = (float)result[0].ToDouble(baseCulture);
float g = (float)result[1].ToDouble(baseCulture);
float b = (float)result[2].ToDouble(baseCulture);
Color c = new Color(r,g,b);
```

5.4 Annoying fly

5.4.1 Vzorová aplikace

Aplikace demonstruje základní schopnosti hloubkového senzoru RealSense. Aplikace je promítána na plátno umístěné na zdi a hloubková kamera je umístěna tak, že snímá celou plochu tohoto plátna. Jedná se o hru, ve které uživatel za pomocí svého „stínu" snímaného senzorem zabíjí/odhání promítanou mouchu pohybující se po plátně.

Aplikace umožňuje nastavit rozsah hloubky, která je snímána, a nastavit posunutí a přiblížení snímaného obrazu tak, aby uživatel mohl nastavit takové zobrazení, které mu pro hraní vyhovuje. Dále aplikace umožňuje zapnutí „barevného módu", ve kterém jsou různé snímané hloubky ilustrovány barevným gradientem.

5.4.2 Implementace

Pro implementaci byl využit herní engine Unity a Unity wrapper knihovny *Intel RealSense SDK 2.0*. Implementace v tomto herním enginu byla zvolena kvůli tomu, že se pro komunikaci se serverem používá C# knihovna SignalR.

RealSense kamera se nachází v Unity scéně jako herní objekt, který má na sobě jako komponentu připojený skript *RsDevice*, který umožňuje nastavení toho, jaký proud dat bude z kamery získáván. Lze nastavit zda bude snímána hloubka nebo barva, jaký bude formát uložení dat, rozlišení a framerate. V aplikaci je nastaven formát snímání Z16 - 16 bitový formát, kde hloubka každého pixelu je uchovávána jako 16 bitové slovo ve formátu little endian.

Dále je třeba skript *RSProcessingPipe*, který aplikuje na proud dat z kamery akce, které se mají vykonat předtím, než se data vyrenderují do textury. Zde se jedná o aplikaci profilu *BackgroudSegmentation*, poskytnutého Unity wrapperem knihovny *Intel RealSense SDK 2.0*. Samotné renderování do textury je vykonáno pomocí skriptu *RsStreamTextureRenderer*, který je umístěn na objektu reprezentujícím RealSense kameru.

Uživatel může zadávat rozsah, ve kterém bude kamera snímat zadáváním minimální a maximální hodnoty hloubky. Informace o hloubce jsou uchovávány v červeném kanálu výstupní textury. Ta tedy musí být dále zpracována tak, aby byly vykreslovány jen ty pixely, které reprezentují objekty nacházející se v hloubce z požadovaného rozsahu. Uživateli je umožněno zobrazovat vzniklý stín snímaných objektů černobíle, nebo barevně, kde barva odpovídá jejich hloubce.

Vzniklý snímek je poté renderován na herní objekt typu Image, neboli obrázek. Tímto obrázkem může uživatel pohybovat pomocí zadávání hodnot horizontálního a vertikálního posunu. Hodnotami mohou být jakákoli celá čísla. Dále ho může

přibližovat a oddalovat zadáváním hodnoty *zoom*, a může si zvolit, zda bude zobrazovaný snímek zrcadlen, nebo ne.

Na obrazovce se nachází obrázek mouchy, který může být umístěn kdekoli v herním prostoru scény. Cílem uživatele je sbírat body tím, že mouchu zakryje svým stínem. Vzhled herní obrazovky je vidět na obrázku 5.3.

Aplikace také komunikuje se serverem. Na něj pravidelně odesílá snímek celé herní obrazovky.

5.4.2.1 Zpracování hloubkového obrazu

V Unity wrapperu knihovny *Intel RealSense SDK 2.0* slouží ke zpracování obrazu na výstupu hloubkového senzoru shader *Depth*. Pro převod hodnoty v červeném kanálu na hodnotu hloubky v metrech používá následující vzorec

$$z = r * \texttt{0xffff} * DepthScale \tag{5.4}$$

kde *r* je hodnota červeného kanálu daného pixelu textury a *DepthScale* je konstanta která je rovna *DepthScale=0.001*. Podle komentářů v shaderu *Depth*, jsou hodnoty v červeném kanálu v rozsahu <0, 1>, přenásobením `0xffff` jsou převedeny na formát ushort a přenásobením *DepthScale* je získána hodnota hloubky dané části scény v metrech. V tomto shaderu lze nastavit proměnné *MinRange* a *MaxRange*, které omezují zabíranou hloubku tak, že všechny fragmenty s hloubkou větší než maximum *MaxRange*, nebo menší než minimum *MinRange* nerenderují.

Shader se používá pro renderování obarveného stínu snímaných objektů. Mapování hloubky na barvy se provádí pomocí textury. Shader má proměnnou *MainTex*, která obsahuje texturu s barevným gradientem. Hodnota texturovací souřadnice *u* odpovídá vypočtené hloubce, a hodnota texturovací souřadnice *v* je stejná pro všechny zobrazované fragmenty. Objekty blíž ke kameře jsou zobrazeny fialovou barvu a objekty dál od ní mají barvu žlutou. Obarvený stín je vidět na obrázku 5.4.

Zpracování hloubkové textury bylo přepsáno do C# kódu ve třídě *DepthProcessing*, aby byla implementace přístupnější studentům. Je předpokládáno, že z větší části nebudou mít zkušenosti s principem shaderů a ty by pro ně mohly být překážkou k pochopení kódu. Pro C# kód platí, že všem pixelům s hloubkou větší než maximum nebo menší než minimum je nastavována hodnota barevného kanálu alfa rovna 1. Tyto pixely textury jsou tedy průhledné. C# kód se používá při vykreslování černého stínu snímaných objektů.

RSStreamTextureRenderer renderuje výstup kamery do dvou přiřazených textur. První z nich je textura nacházející se ve skriptu *DepthProcessing*. V tomto skriptu se data v přijaté textuře převádějí na metry podle vzorce 5.4 a vytváří se textura s černým stínem objektů, které se nacházejí v zadaném rozsahu. Tato textura je poté nastavena hernímu objektu typu Image, tedy obrázek. Druhá přiřazená textura

je textura druhého herního objektu stejného typu. Na tomto objektu je přiřazen materiál, který zajistí, že je obrázek renderován s pomocí shaderu *Depth*.

Ve scéně se tedy nacházejí dva obrázky, které zobrazují snímané objekty, jeden z nich černě a druhý barevně. Když uživatel přepíná mezi barevným a černobílým módem, přepíná vlastně mezi tím, který z obrázků bude ve scéně aktivní.

5.4.2.2 Posun a škálování

Posouvání a přibližování popřípadě oddalování snímku scény je implementováno pomocí upravování pozice obrázku, na kterém je tato textura nastavena. Rozdíl jeho aktuální a počáteční pozice odpovídá uživatelem zadanému posunu a jeho měřítko odpovídá uživatelem zadanému přiblížení nebo oddálení.

Posun je vždy udáván jako celé číslo udávající počet pixelů. Kladné číslo odpovídá posunu doprava v ose x nebo nahoru v ose y. Záporné číslo odpovídá posunu doleva v ose x nebo dolů v ose y. Obrázek je škálován pomocí hodnoty *zoom*. Tato hodnota může být jakékoli kladné desetinné číslo větší než nula. Pokud je menší než jedna, jedná se o zmenšení, pokud je větší než jedna, obrázek je zvětšen.

5.4.2.3 Herní mechanika

Pokud je stínem snímaných objektů zakryto více jak padesát procet obrázku mouchy, je detekován zásah, hráči je přičten bod a moucha se přesouvá na náhodnou, nezakrytou část obrazovky. Pro kontrolu, zda a nakolik stín zakrývá mouchu, potřebujeme přepočítat její pozici v Unity scéně na pozici mouchy v textuře, která obsahuje stín.

Počátek globálních souřadnic scény se nachází v levém dolním rohu obrazovky. Pokud by tedy velikost textury odpovídala veikosti obrazovky a nebyl aplikován posun či přibližování, pozice mouchy v obrázku by přímo odpovídala pozici ve scéně.

Pozici mouchy v obrázku samozřejmě ovlivňuje posun obrázku s renderovaným stínem. Pokud například uživatel posune ve scéně tento obrázek v ose x o 100 pixelů, odpovídá to posunu mouchy v ose x o -100 pixelů vůči tomuto obrázku. Stejný princip platí i pro posun v ose y. Tento přepočet se tedy dá vyjádřit jako

$$posPan = -pan + pos \tag{5.5}$$

kde *pos* je vektor pozice mouchy ve scéně a *pan* je vektor posunu obrázku.

Přibližování obrázku samozřejmě také ovlivňuje výslednou pozici mouchy. Když uživatel obrázek přiblíží, bude se část obrázku nacházet mimo obrazovku. Pokud obrázek oddálí, budou mít okraje obrázku odstup od okrajů obrazovky. Tento přesah respektive odstup můžeme vypočítat jako

$$pad = ((newSize.x - screenW)/2, (newSize.y - screenH)/2) \tag{5.6}$$

kde *pad* je vektor uchovávající rozměry přesahu v osách x a y, *newSize* je vektor uchovávající rozměry obrázku po změně velikosti, *screenW* je šířka obrazovky a *screenH* je výška obrazovky.

Pozice mouchy ve zvětšeném obrázku lze vypočítat jako

$$posZoom = pos + pad \tag{5.7}$$

kde *pos* je vektor pozice mouchy ve scéně.

Na obrázek samozřejmě může být aplikován jak posun tak přiblížení najednou. Přibližování ovlivňuje i posun mouchy vůči obrázku. V přiblíženém obrázku se moucha bude zdánlivě posouvat o menší vzdálenosti, a naopak v obrázku oddáleném o větší.

Pozice mouchy v obrázku, na který působí jak posun tak přiblížení se tedy vypočte podle vzorce

$$pos = -pan * (1/scale) + (pos + pad) \tag{5.8}$$

kde *scale* odpovídá faktoru změny velikosti.

Obrázek má vždy velikost obrazovky, ovšem vlastní textura která je na něj vykreslována může mít rozměry odlišné. Proto je před výpočtem 5.8 nutné přepočítat pozici a přesah na relativní pomocí

$$pos = (pos.x/newSize.x, pos.y/newSize.y) \tag{5.9}$$

$$pad = (pad.x/newSize.x, pad.y/newSize.y) \tag{5.10}$$

kde *newSize* je vektor uchovávající rozměry obrázku po změně velikosti. Posun lze přepočítat na relativní pomocí vzorce

$$pan = (pan.x/screenW, pan.y/screenH) \tag{5.11}$$

kde *screenW* je šířka obrazovky a *screenH* je výška obrazovky.

Souřadnice mouchy v textuře se dopočítá jako

$$posTex = (pos.x * texW, pos.y * texH) \tag{5.12}$$

kde *texW* je šířka textury a *texH* její výška.

Poslední komplikací je, že souřadný systém hloubkové kamery je pravotočivý, a herní engine Unity používá levotočivý souřadný systém. Proto je textura z hloubkového senzoru zobrazována vertikálně překlopená. Obrázek ve scéně, na který se

tato textura vykresluje, je proto otočen o 180° okolo osy x tak, aby promítaný stín odpovídal snímané scéně.

Proto je třeba vypočtené souřadnice upravit jako

$$posTex = (posTex.x, texH - posTex.y) \tag{5.13}$$

kde *texH* je výška textury.

Protože uživatel může zvolit, zda se obrázek zobrazuje zrcadlově převrácený, může být třeba upravit i souřadnici v ose x. Pro zrcadlově převrácený obrázek platí následující vzorec

$$posTex = (texW - posTex.x, texH - posTex.y) \tag{5.14}$$

kde *texW* je šířka textury a *texH* je výška textury.

Nakonec je potřeba vypočítat velikost mouchy v textuře. Ta opět závisí na přiblížení a na poměru velikosti textury a obrazovky. Moucha se ve zmenšeném obrázku jeví větší, a naopak ve zvětšeném menší. Vztah mezi velikostí a škálováním obrázku je tedy následující

$$size = (1/scale) * size \tag{5.15}$$

kde *size* je vektor velikosti mouchy a *scale* faktor škálování.

Dále je opět třeba převést velikost na relativní pomocí následujícího vzorce

$$sizeRel = (scale.x/screenW, scale.y/screenH) \tag{5.16}$$

kde *screenW* je šířka obrazovky a *screenH* výška obrazovky.

A velikost mouchy v textuře se tedy vypočte jako

$$sizeTex = (sizeRel.x * texW, sizeRel.y * texH) \tag{5.17}$$

kde *texW* je šířka textury a *texH* výška textury.

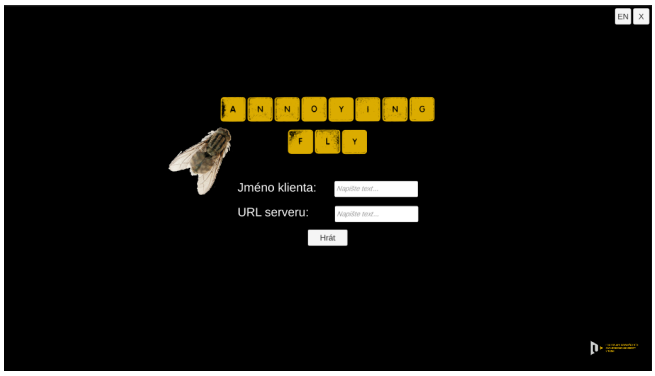
Pro známou pozici mouchy v textuře *posTex* a známou velikost obrázku mouchy *sizeTex* stačí už jen spočítat počet neprůhledných pixelů v daném výřezu textury. Pokud počet neprůhledných pixelů přesahuje 50% výřezu textury, došlo k zásahu mouchy.

5.4.2.4 Komunikace se serverem

Po spuštění aplikace se zobrazí obrazovka z obrázku 5.2, kde uživatel musí zadat URL serveru a jméno klienta. Zadané jméno musí být unikátní pro všechny klienty typu Annoying Fly. Jméno i URL lze nastavit také v konfiguračním souboru aplikace.

Klient se po spuštění automaticky pokusí připojit k serveru. Pokud je připojení na server úspěšné, v levém dolním rohu obrazovky se zobrazuje zelená hláška *„Připojeno"*. Pokud ne, zobrazuje se červená hláška *„Odpojeno"* a klient se o připojení pokouší znovu, o pět vteřin později. Pokud během běhu aplikace dojde k nečekanému dočasnému přerušení připojení, knihovna SignalR se pokouší znovu připojit a v levém dolním rohu obrazovky se zobrazuje žlutá hláška „Připojování". Pokud se klientu nepodaří obnovit připojení, zobrazuje se červená hláška *„Odpojeno"*, a klient se automaticky pokouší o opětovné připojení každých pět vteřin.

Tento klient odesílá na server objekt typu Bitmap. Jedná se o snímek celé obrazovky, který se snímá a aktualizuje opakovaně s intervalem jedna vteřina. Pozice tohoto objektu ve virtuálním světě je nastavena na hodnotu [0, 0, 0].



Obrázek 5.2: Intro obrazovka aplikace Annoying fly.

Obrázek 5.3: Herní obrazovka aplikace Annoying fly, černobílý režim zobrazení scény.

Obrázek 5.4: Barevný režim aplikace Annoying fly.

5.5 DepthMesh

5.5.1 Vzorová aplikace

Aplikace umožňuje snímání scény pomocí hloubkového senzoru RealSense. Kamera vytváří hloubkový obraz, ve kterém je uložena hloubka jednotlivých bodů scény. Kromě pozic jednotlivých bodů v prostoru kamera zaznamenává i jejich reálnou barvu, a umožňuje tedy zobrazit barevný snímek scény.

Aplikace vytváří trojrozměrný model scény z pixelů hloubkového obrazu. Na tento model je jako textura aplikován barevný snímek scény. Dále aplikace umožňuje úpravu tohoto modelu pomocí uživatelem zadaného kódu. Vzniklý model lze zaslat na server pro další zpracování.

Uživatel díky této aplikaci může získat větší kontrolu nad hloubkovou kamerou. Student by měl zkusit snímat různé povrchy a objekty, nacházející se v různých vzdálenostech od kamery, upravovat snímanou vzdálenost, a zkusit zvýšit kvalitu snímaného hloubkového obrazu, potažmo vytvářeného modelu scény, pomocí filtrů postprocesingu.

Pomocí uživatelského kódu může uživatel manipulovat se souřadnicemi vrcholů v prostoru, s uv souřadnicemi vrcholů a s trojúhelníky vytvářeného modelu scény. K aplikaci budou dodány vzorové skripty, které demonstrují příklady úloh, které může student pomocí uživatelského kódu řešit.

5.5.2 Implementace

K implementaci byl použit jazyk C# a C++. Uživatelské rozhraní bylo vytvořeno v .NET pomocí grafického subsystému WPF. Aplikace byla vytvořena rozšířením aplikace vyvíjené Matějem Černým. Okno aplikace je vidět na obrázku 5.6.

V jazyce C++ je napsána knihovna *DepthClientLib*, která poskytuje metody pro manipulaci s hloubkovým senzorem. Nachází se zde metoda na spuštění a zastavení snímání, nastavení používaných filtrů a hodnot jejich parametrů a získání vytvořeného hloubkového snímku a z něj vytvořené trojúhelníkové sítě. Tato knihovna využívá *RealSense SDK 2.0*. Vytvořená C++ knihovna je připojena k .NET projektu a přístup jejím metodám je zprostředkován pomocí C# wrapperu.

RealSense kamera snímá hloubkový snímek scény, na tento snímek jsou poté aplikovány filtry postprocesingu. Výsledkem postprocesingu je upravený hloubkový snímek, který je poté převeden na trojrozměrnou trojúhelníkovou síť, kterou lze v aplikaci dále upravovat pomocí uživatelského Python kódu. Aplikace vytvořenou trojúhelníkovou síť zasílá na server.

5.5.2.1 Knihovna DepthClientLib

Princip snímání hloubkového obrazu pomocí C++ knihovny *Intel RealSense SDK 2.0* je popsán v kapitole 5.2.1. Zde bude popsáno propojení s C# aplikací.

Pokud je aplikace Depth Mesh spuštěna po připojení kamery RealSense k počítači, je aplikace automaticky propojena s výstupem z této kamery. Pokud je kamera připojena až po spuštění aplikace, je nutné připojit kameru manuálně přes položku menu Soubor-Připojit kameru. Pro propojení kamery a aplikace je volána metoda *Start* knihovny *DepthClientLib*.

V metodě *Start* se vytvoří jednotlivé filtry postprocesingu s defaultním nastavením parametrů a spustí se snímací smyčka. Ve snímací smyčce se pomocí metody *wait_for_frame*s snímací pipeline při každé iteraci získá nový set snímků. Na hloubkový obraz se poté aplikují filtry postprocesingu.

Pro získání hloubkového obrazu volá C# aplikace metodu *GetFrameOnce*. Tato metoda je také volána ve smyčce, která běží po celou dobu běhu aplikace a zajistí, že je zobrazený hloubkový obraz vždy aktuální. Když je v aplikaci třeba z hloubkového snímku vytvořit trojúhelníkovou síť, je volána metoda *GetFrame*.

Uživatel může v aplikaci Depth Mesh upravovat jednotlivé parametry postprocesing filtrů. Pro aplikování nových parametrů je volána metoda *UpdateFilters*, vždy když uživatel provede nějakou změnu nastavení filtrů.

5.5.2.2 Postprocesing

Aplikace umožňuje zapínání, vypínání a nastavování parametrů všech filtrů, popsaných v kapitole 3.2.4.

Filtry jsou na hloubkový obraz aplikovány v následujícím pořadí:

- decimační filtr,
- převod na obraz disparity,
- prostorové vyhlazování,
- časové vyhlazování
- převod zpět na hloubkový obraz,
- záplatování děr.

Na obrázku 5.7 je okno sloužící k pokročilému nastavení parametrů jednotlivých filtrů aplikovaných na hloubkový obraz.

5.5.2.3 Trojúhelníková síť

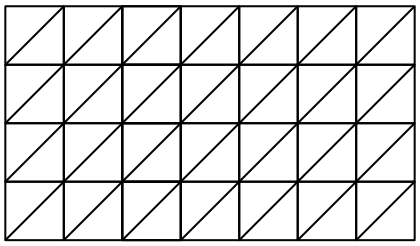
Vytvoření trojúhelníkové sítě z hloubkového snímku umožní větší přenositelnost dat mezi aplikacemi a lepší vizuální vlastnosti, zvláště pro nižší počet hloubkových bodů. Z každého pixelu hloubkového obrazu je vytvořen jeden bod v prostoru. Souřadnice *z* odpovídá zaznamenané hloubce dané části snímané scény. Souřadnice x a y jsou vypočteny z pozice pixelu v obraze a z hloubky[Int18]. Pozice pixelu se značí v horizontálním směru jako souřadnice *i* a ve vertikálním jako souřadnice *j*. Bod $[i, j] = [0, 0]$ odpovídá levému hornímu pixelu obrazu. Pro vyjádření vztahu mezi souřadnicemi v hloubkovém obraze a souřadnicemi v prostoru jsou třeba následující parametry kamery:

- Souřadnice prostorového bodu [0,0,0] v souřadnicích obrazu - $[i, j] = [ppx, ppy]$
- Ohnisková vzdálenost kamery udávána jako násobek pixelů - $[i, j] = [fx, fy]$

Vzoreček na výpočet souřadnic x a y je následující:

$$\begin{aligned} x &= depth * ((i - ppx)/fx) \\ y &= depth * ((j - ppy)/fy) \end{aligned} \tag{5.18}$$

Pro převod hloubkového obrazu na množinu bodů slouží třída knihovny *RealSense SDK 2.0 pointcloud*. Body ve vzniklé mřížce jdou spojit do trojúhelníků tak, aby odpovídali obrázku 5.5.



Obrázek 5.5: Struktura trojúhelníkové sítě vytvořené z hloubkového obrazu.

Je ovšem třeba vyhnout se vytváření trojúhelníků s vrcholem, který má hodnotu hloubky 0. Naměřená hodnota hloubky 0 totiž odpovídá díře, neboli místu, kde se nepodařilo získat validní data. Pokud má pixel hodnotu hloubky 0, trojúhelníky, které by jej obsahovaly, nejsou zahrnuty do vznikající trojúhelníkové sítě. Na vytvořenou síť je namapována textura, kterou tvoří barevný snímek scény.

V aplikaci je trojúhelníková síť reprezentována třemi poli. První je pole *points*, které obsahuje souřadniceme vrcholů ve tvaru

$$points = [x_0, y_0, z_0, x_1, y_1, z_1, ...] \tag{5.19}$$

kde x_0, y_0 a z_0 jsou souřadnice vrcholu p_0, x_1, y_1, z_1 jsou souřadnice vrcholu p_1 a tak dále.

Dále *faces* je pole s indexy vrcholů trojúhelníků

$$faces = [i_{00}, i_{01}, i_{02}, i_{10}, i_{11}, i_{12}...] \tag{5.20}$$

kde i_{00}, i_{01}, i_{02} odpovídají indexům vrcholů prvního trojúhelníku, i_{10}, i_{11}, i_{12} jsou indexy vrcholů druhého trojúhelníku a tak dále. Tyto indexy lze přepočítat na indexy do pole *points* následovně. Pro index i souřadnice odpovídajícího vrcholu jsou:

$$\begin{aligned} x &= points[i * 3] \\ y &= points[i * 3 + 1] \\ z &= points[i * 3 + 2] \end{aligned} \tag{5.21}$$

Texturovací souřadnice jsou uchovávány v poli *uvs* ve tvaru

$$uvs = [u_0, v_0, u_1, v_1, ...] \tag{5.22}$$

kde u_0, v_0 jsou UV souřadnice vrcholu p_0, u_1, v_1 jsou UV souřadnice vrcholu p_1 a tak dále.

5.5.2.4 Python kód

V uživatelském Python kódu je možné upravovat pole *points*, *uvs* a *faces* a tímto modifikovat trojúhelníkovou síť. Uživatel píše metodu, která má jako parametry tato pole a také musí pole *points*, *uvs* a *faces* vracet jako návratovou hodnotu. Délka pole *points* (značena jako *lengthP*) musí být beze zbytku dělitelná třemi, délka pole *uvs* (značena jako *lengthUV*) beze zbytku dělitelná dvěmi. Pro délky těchto polí musí platit, že $lengthUV/2$ musí odpovídat hodnotě $lengthP/3$. Délka pole *faces* musí být beze zbytku dělitelná třemi. Pro vykonání kódu je využita knihovna *ZCU.PythonExecution*.

5.5.2.5 Komunikace se serverem

V aplikaci je možné zadat URL serveru a jméno klienta. Zadané jméno musí být unikátní pro všechny klienty typu Depth Mesh. Jméno i URL lze nastavit také v konfiguračním souboru aplikace. Připojení k serveru se provádí manuálně stisknutím tlačítka v menu Server-Připojit. Po úspěšném připojení je možné posílat na server vytvořenou trojúhelníkovou síť reprezentující model scény. Odeslanou trojúhelníkovou síť lze ze serveru i stahovat, aktualizovat ji, nebo ji popřípadě smazat.

Dále je možné zapnout automatické odesílání, které zajistí, že aplikace po uplynutí zadané prodlevy v sekundách vytvoří nový snímek scény a model na serveru se aktualizuje. Na serveru je vždy uchováván jen jeden model reprezentující scénu

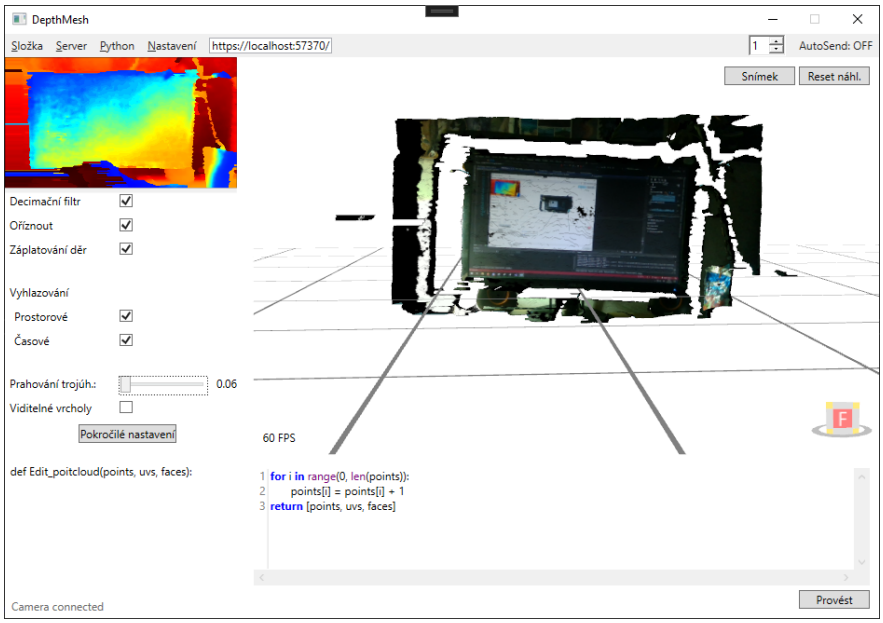
pro každého připojeného klienta Depth Mesh. Pokud za běhu dojde k nečekanému přerušení připojení, knihovna SignalR se pokouší znovu připojit. Pokud se klientu nepodaří obnovit připojení, je nutné, aby uživatel znovu manuálně provedl připojení.

5.6 Aplikace pro hloubkovou kameru stupně tři

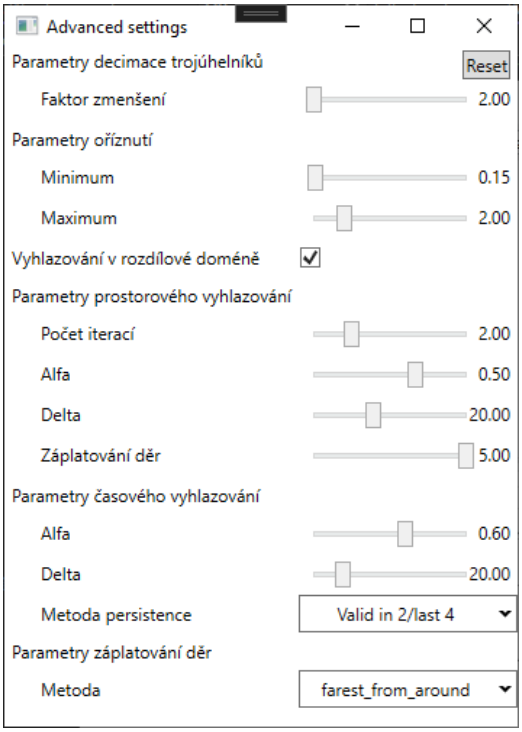
5.6.1 Navržené zadání pro SŠOP

Seznamte se se způsobem jak získat z hloubkové kamery údaje o snímaných bodech. Postavte otáčivý stojan, na který lze umístit libovolný objekt (například za pomocí lega ve vedlejší učebně). Proveďte konfiguraci následujícím způsobem: snímejte jeden snímek každou x-tou vteřinu a zjistěte parametry s jakými se tento stojan otáčí - rychlost otáčení, a úhel pod jakým se nachází jednotlivé snímky vůči výchozí pozici.

Vytvořte aplikaci, která naskenujte objekt, a s pomocí parametrů získaných během konfigurace poskládá jednotlivé množiny bodů do jedné tak, aby se získal sken, který obsahuje data všech stran objektu. Použijte libovolnou kombinaci volně dostupných knihoven, kódu z dodaných aplikací a vlastního kódu.



Obrázek 5.6: Okno aplikace Depth Mesh.



Obrázek 5.7: Pokročilé nastavení filtrů aplikace Depth Mesh.

5.7 Box City

5.7.1 Vzorová aplikace

Aplikace umožňuje pohybovat se po virtuální scéně a manipulovat s objekty, které se v této scéně nacházejí. Omezuje pohyb uživatele po virtuální scéně tak, aby nemohl ve skutečném světe nevědomky odejít z prostoru vyhrazeného pro virtuální realitu. Velikost tohoto prostoru je nastavitelná pomocí konfiguračního souboru. Aplikace komunikuje se serverem a zajišťuje automatické odesílání a přijímání objektů. Také přijímá objekty ze serveru a umožňuje manipulaci s nimi, takže ji lze použít k prohlížení virtuálních objektů z ostatních klientských aplikací.

Tato aplikace slouží k seznámení uživatele s virtuální realitou, jejím cílem je, aby se uživatel naučil pohybovat ve VR a manipulovat s objekty. Poskytuje minihru s virtuálními stavebními kostkami, se kterými má uživatel za úkol pomocí kontrolerů pohybovat.

5.7.2 Implementace

Pro implementaci byl použit herní engine Unity. Byly použity Unity balíčky *OpenXR* a *XRInteractionToolkit* popsané v kapitole 5.1.3.

Tato aplikace slouží k tomu, aby se uživatel seznámil s virtuální realitou. Proto byla vytvořena scéna s městečkem, ve kterém jsou na různých místech schovány krabice. S krabicemi je možné manipulovat a je zde pro uživatele vytvořena minihra. Cílem minihry je nalézt tyto krabice a odnést je na jedno specifické místo. Minihra má sloužit jako motivace pro uživatele osvojit si pohyb ve virtuálním světě a naučit se v něm orientovat a interagovat s objekty. Krabice lze vidět na obrázku 5.13. Snímky hráče, který se právě pohybuje v aplikaci Box City jsou vidět na obrázku 5.8.

Kontrolery a headset virtuální reality jsou ve scéně reprezentovány objekty z obrázku 5.14. Zdrojem modelu kontroleru je SketchFab [Vio19], tento model byl dále upraven. Dále byly při vývoji použity modely z Unity AssetStore [Seb20], [ras19], [Sro22] a textury z AssetStore [Roa22] pro vytvoření herní scény.

Aplikace na server odesílá objekty krabic a headsetu s kontrolery. Ze serveru umí přijímat objekty virtuálního světa, které se zobrazí ve virtuálním světě a uživatel s nimi také může manipulovat.

V aplikaci není simulována gravitace. Tato volba byla učiněna, protože server nemá implementován způsob, který by umožnil uzamykání objektů. Pokud by server tuto funkctionalitu měl, bylo by po celou dobu uzamčení objektu umožněno zasílání aktualizací transformace jen jednomu z připojených klientů, konkrétněji tomu, který jej uzamkl. Bez uzamykání by docházelo k nepředvídatelnému chování objektů, pokud by k serveru byli připojeni dva klienti se simulací gravitace, kteří by

se najednou snažili aktualizovat pozice stejných objektů. Tento problém by byl ještě umocněn, pokud by jeden z klientů s některým z objektů ve scéně pohyboval.

5.7.2.1 Ovládací panel

V Unity scéně se nachází Unity *UI Canvas*, které slouží jako ovládací panel aplikace. Reaguje na *XRRayInteractor* umístěný na levém kontroleru, a umožňuje reset pozic krabic, smazání krabic z lokální scény i ze serveru, uložení aktuální scény na disk nebo načtení scény z disku, přepínání mezi jazyky, které aplikace podporuje a zobrazení návodu k ovládání. Tento ovládací panel je vidět na obrázku 5.12. Na tomto *UI Canvas* jsou také napsány instrukce popisující, jak hrát implementovanou minihru, a také se zde také nachází kolonka, ve které je napsán aktuální stav připojení k serveru.

Resetování pozic krabic je implementováno tak, že krabice, které spravuje tato instance klienta Box City, jsou nejprve smazány ze serveru a poté znovu instancovány na svých počátečních pozicích. Kterému klientu krabice patří je určeno podle jejich jména, ve kterém je zapsáno jméno klienta, který je instancoval.

5.7.2.2 Uložení scény

Všechny objekty virtuálního světa ve scéně, tedy objekty přijaté ze serveru a krabice daného klienta, lze přímo v aplikaci uložit na disk. Na ovládacím panelu se nachází možnost Scény, kde je možné uložit tři různé scény najednou. Okno pro ukládání a načítání scén je vidět na obrázku 5.11. Všechny scény jsou uloženy ve složce „Saves", umístěné na stejné úrovni jako exe soubor spouštějící aplikaci, v podsložkách označených pořadovým číslem.

Při ukládání scény je pro každý objekt vytvořen .txt soubor, ve kterém jsou zapsány informace o daném objektu. Jedná se o vlastnosti popsané v kapitole 4.2 - tedy pozici, rotaci, škálování, typ a další vlastnosti ze slovníku *properties*. Jméno .txt souboru odpovídá jménu daného objektu. Scény lze z disku také načítat. Při načtení scény jsou všechny objekty na serveru smazány, je načten obsah .txt souborů a vytvořené objekty jsou poslány na server.

5.7.2.3 Ovládání aplikace

Aplikace je ovládána pomocí kontrolerů virtuální reality. Na levém kontroleru je umístěn *XRRayInteractor*, který umožňuje uživateli teleportaci po světě. Na pravém kontroleru je umístěn *XRDirectInteractor*, díky kterému je možné pravým kontrolerem zvedat objekty se skripty *XRGrabInteractable* ve scéně. Obrazovka s návodem k ovládání aplikace Box City je vidět na obrázku 5.10. Kontrolery *HTC Vive Pro 2.0*

jsou symetrické, a proto lze používat i levý kontroler v pravé ruce, a aplikace je tak přístupná jak pravákům tak levákům bez dalších úprav.

5.7.2.4 Pohyb v prostoru

Uživatel se po virtuálním světě může pohybovat buďto teleportací pomocí levého kontroleru, nebo pohybem v reálném světě. Při pohybu je ovšem třeba, aby nevyšel z bezpečného prostoru, vyhrazeného pro hraní aplikací ve virtuální realitě. Mimo tento prostor by se uživatel mohl zranit, nebo poškodit reálné objekty kolem sebe. Proto je třeba aby ve virtuálním světě existoval nějaký systém, který uživatele varuje, že tento bezpečný prostor opouští.

V aplikaci Box City je možné nastavit velikost bezpečného prostoru pro pohyb v konfiguračním souboru. Tento prostor je poté vyznačen zeleným obdélníkem umístěným na zemi virtuálního světa. Tento obdélník je vidět na pravém obrázku 5.8. Přiblížení uživatele k hranicím tohoto obdélníku je detekováno pomocí tzv. trigger koliderů. Tyto kolidery umožňují reagovat na kolizi s jiným objektem, ale nijak neomezují pohyb objektů ve scéně skrz ně. Pokud dojde ke kolizi jednoho z těchto kolideru s objektem reprezentujícím headset, zobrazí se ve virtuálním světě poloprůhledná varovná stěna.

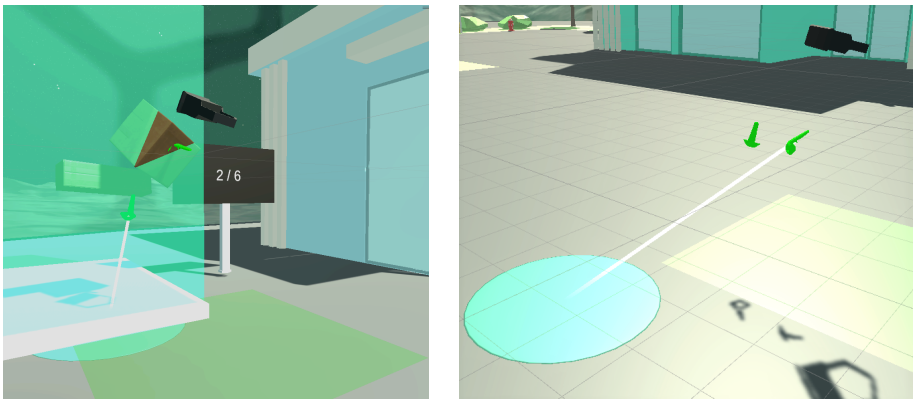
SteamVR, pomocí kterého se spouští headset HTC Vive Pro 2.0, poskytuje vlastní varovný systém pro vystoupení z prostoru určeného pro VR. Tento systém je stejný pro všechny aplikace, které jsou na daném headsetu pouštěny a lze jej nastavit přímo ve virtuální realitě. Varovný systém implemenovaný v této aplikaci (potažmo i v aplikaci PaintVR) poskytuje možnost stanovit bezpečný prostor unikátní pouze pro tuto aplikaci.

5.7.2.5 Komunikace se serverem

Po spuštění aplikace se zobrazí obrazovka z obrázku 5.9. Zde uživatel musí zadat jméno klienta a URL serveru. Zadané jméno musí být unikátní pro všechny VR klienty. Jméno i URL lze také nastavit v konfiguračním souboru aplikace.

Klient se po spuštění automaticky pokusí připojit na server. Pokud připojení není úspěšné, klient se o připojení pokouší znovu o pět vteřin později. Pokud dojde k nečekanému přerušení spojení se serverem, knihovna SignalR se pokouší znovu připojit. Pokud je připojení úspěšné, všechny lokální objekty jsou smazány a znovu přijaty ze serveru. Pokud došlo ke smazání dat na serveru, klient znovu instancuje krabice na jejich počátečních pozicích. Pokud se klientu nepodaří připojení obnovit, dojde ke smazání lokálních objketů a klient se s intervalem pěti vteřin pokouší znovu připojit.

Na server se odesílají krabice, které uživatel nachází ve scéně. Ve scéně se instancují až poté, co proběhne první úspěšné připojení na server. Při pohybu s krabicemi



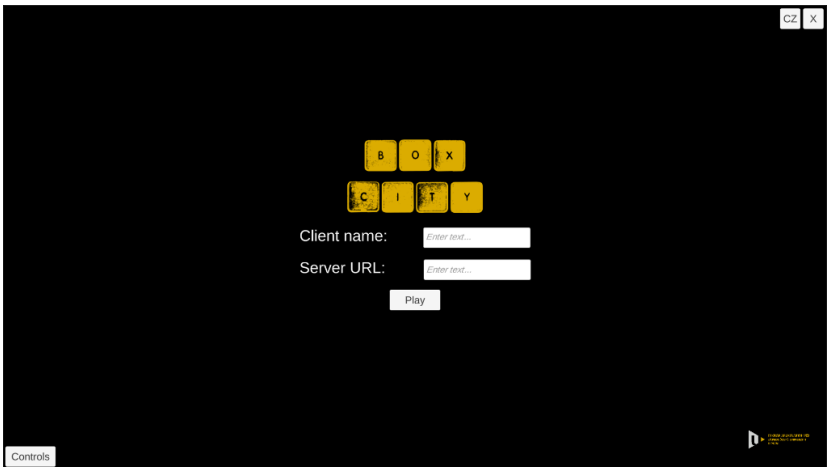
Obrázek 5.8: Obrázky z editoru Unity. Záběr na hráče hrajícího hru Box City. Nalevo je vidět manipulace s objekty, napravo je na zemi vidět zelený čtverec reprezentující bezpečný prostor pro volný pohyb a kruhová nabídka teleportu.

je jejich pozice na serveru aktualizována každých 0.1 vteřiny pomocí skriptu *ReportTransformChange* z knihovny *Common.Unity*. Herní objekty těchto krabic komponentami odpovídají prefabu *ServerMeshPrefab*, popsanému v kapitole 5.1.4, a jde je tedy zařadit mezi objekty virtuálního světa, které jsou uchovávané ve třídě *WorldObjectMemoryStorage*.

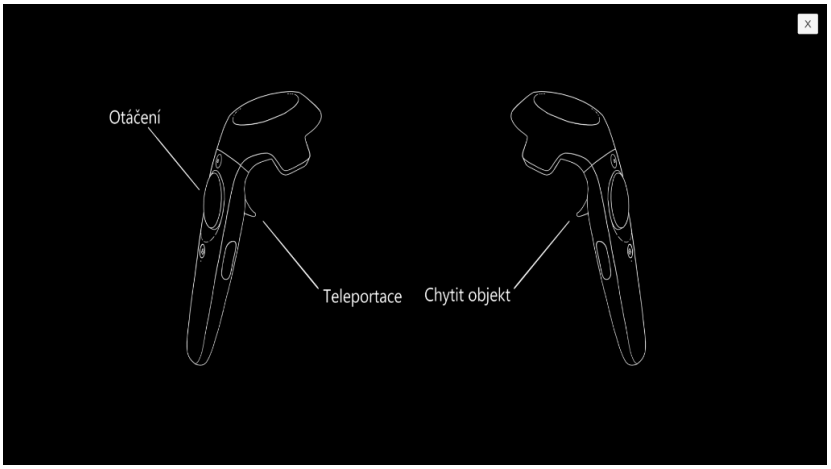
Pro zasílané krabice platí, že hodnota vlastnosti „*ColiderType*" je řetězec s hodnotou „box", serializovaný do bytového pole pomocí třídy *StringSerializer* z knihovny *Common*. Dále hodnota vlastnosti „*ColiderSize*" je pole proměnných typu float obsahující pozici středu kolideru a jeho šířku a výšku.

Aplikace také posílá na server objekty reprezentující headset a kontrolery. Tyto objekty jsou typu Mesh a mění svou pozici v závislosti na pohybu uživatele. Jejich pozice na serveru je také aktualizována každých 0.1 vteřiny pomocí skriptu *ReportTransformChange*. Kontrolery mění barvu v závislosti na stavu připojení k serveru. Instance těchto objektů komponentami odpovídají prefabu *ServerMeshPrefab*. Při korektním ukončení aplikace jsou objekty reprezentující headset a kontrolery ze serveru smazány.

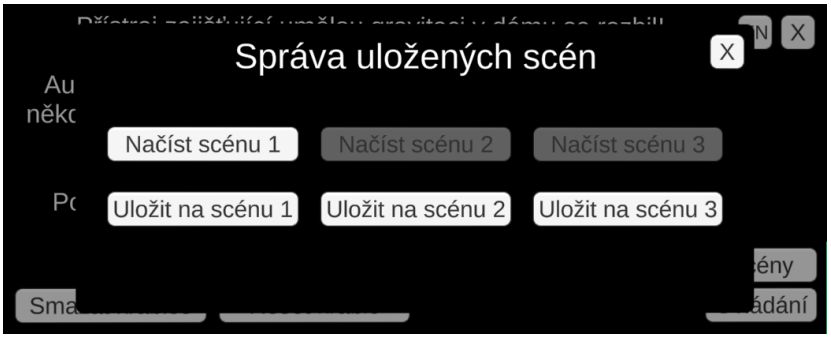
V aplikaci se dále zobrazují objekty přijaté ze serveru, se kterými může uživatel také pohybovat. Pokud objekt přijatý ze serveru nemá nastavené vlastnosti *ColiderType* a *ColiderSize*, nastaví se mu kolider typu *BoxColider* o velikosti jeho trojúhelníkové sítě.



Obrázek 5.9: Intro obrazovka aplikace Box city.

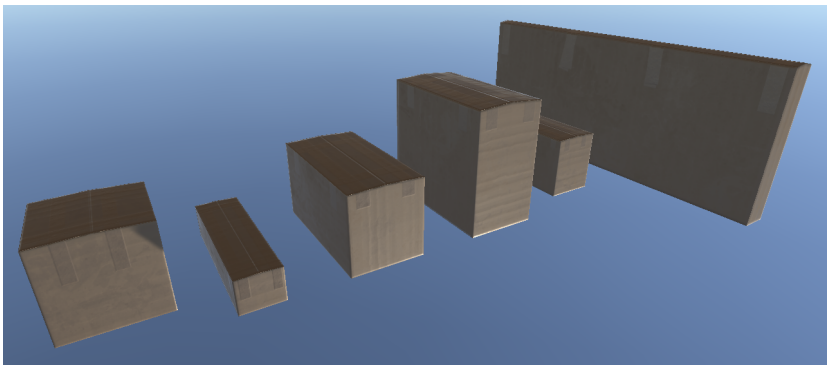


Obrázek 5.10: Obrazovka s ovládáním aplikace Box city.

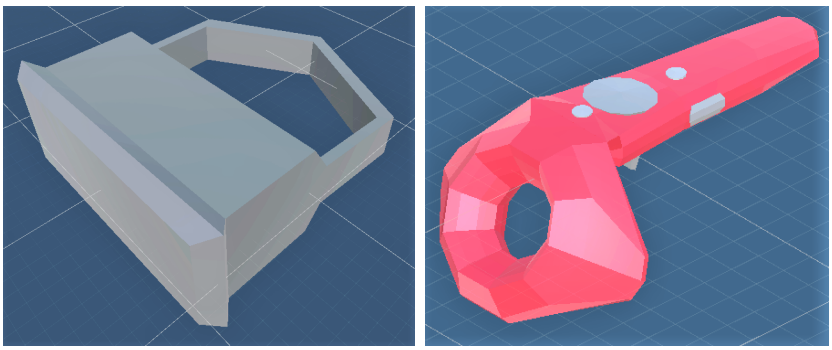


Obrázek 5.11: Obrazovka s možnostmi uložení scény. Tlačítka pro načtení scény 2 a 3 nejsou aktivní, protože se na disku nenachází odpovídající složky.

Obrázek 5.12: Ovládací panel ve scéně v aplikaci Box city.



Obrázek 5.13: Použité objekty krabic v aplikaci Box city.



Obrázek 5.14: Modely použité pro preprezentaci headsetu (vlevo) a kontrolerů (vpravo) v aplikacích pro virtuální realitu

5.8 Brush Export

5.8.1 Vzorová aplikace

Aplikace slouží jako podpůrná aplikace k aplikaci třetího stupně - PaintVR. Umožňuje vytvářet uživatelské štětce, poskytuje uživateli náhled na vytvářený štětec a tento štětec lze exportovat jako XML soubor.

Mezi parametry štětce patří jeho jméno, velikost, barva a jeho textura. Všechny tyto parametry je možné v aplikaci nastavit. Velikost štětce se mění v průběhu času kreslení. V praxi to znamená, že na začátku tahu může mít štětec například poloviční velikost, ta se pak v průběhu tahu začne zvětšovat, a po chvíli zase zmenšovat, dokud nedosáhne počáteční hodnoty. Tento proces změny velikosti se periodicky opakuje po celou dobu kreslení. Uživatel může nastavit průběh změny velikosti a délku periody. Jaké pixely textury jsou na danou část tahu aplikovány, závisí na tom, jak dlouho je se štětcem kresleno. Výsledná barva štětce v náhledu odpovídá barvě textury přenásobené barvou štětce.

Vzhled štětce uživatel ovlivňuje pomocí Python kódu, kde lze přímo zadávat hodnoty jednotlivých parametrů, stanovit průběh změny velikosti štětce a generovat texturu jako pole barev. K aplikaci budou dodány vzorové skripty, které demonstrují základní principy vytváření štětců.

5.8.2 Implementace

Aplikace je implementována v herním engine Unity. Díky tomu bude možné vytvořit náhled na vytvářený štětec tak, aby co nejvíce odpovídal tahu štětcem v aplikaci PaintVR. Uživatelské rozhraní je implementováno pomocí Unity *UI Canvas*. Okno aplikace je vidět na obrázku 5.15 a okno se zobrazeným návodem v aplikaci na obrázku 5.16.

Štětec je reprezentován třídou *Brush*. Třída implementuje rozhraní *IDrawInstrument*, které definuje atributy, které mají oba kreslící nástroje implementované v PaintVR - štětec a guma. Třída a rozhraní jsou vidět v následující ukázce kódu.

```
1 public class Brush : IDrawInstrument
2 {
3     Color color;
4     Texture2D texture;
5     float[] widthModifier;
6     float timePerIter;
7 }
```

```
public class IDrawInstrument
{
    string name;
    float width;
}
```

Atribut *color* reprezentuje defaultní barvu štětce, *texture* je textura, která je na tah štětcem mapována. Je vytvářena z pole barev, které uživatel zadá v Python kódu a jedná se o instanci Unity třídy *Texture2D*. Atribut *width* je defaultní velikost štětce - povolené hodnoty čísla v rozsahu <0.001, 0.1>. Tyto hodnoty byly stanoveny kvůli lepšímu zobrazení ve scéně. Uživatel tyto hodnoty zadává v procentech. Povolené hodnoty vstupu v kódu tedy jsou čísla v rozsahu <1, 100>. Atribut *widthModifier* je pole modifikátorů velikosti, které popisují změnu velikosti v čase, a kterými se při kreslení násobí velikost štětce. Povolené hodnoty jsou čísla v rozsahu (0,1>. Atribut *timePerIter* je proměnná, která stanoví čas ve vteřinách, za který se projde celé pole *widthModifier*. Atribut *name* je jméno štětce, které lze upravovat přímo v uživatelském rozhraní aplikace. Zbytek atributů lze nastavit pomocí uživatelského Python kódu.

5.8.2.1 Výpočet aktuální velikosti štětce

Jaká konkrétní hodnota modifikátoru velikosti štětce se bude aplikovat v daném okamžiku závisí na tom, kolik času uběhlo od začátku tahu a na hodnotě *timePerIter*. Pro vytvoření efektu plynulé změny velikosti se využívá lineární interpolace. Nejprve je třeba stanovit proměnnou *timestep*, která odpovídá časovému odstupu jednotlivých hodnot pole *widthModifier* od sebe.

$$timestep = timePerIter/(modLen - 1) \tag{5.23}$$

kde *modLen* je délka pole *widthModifier*.

Platí, že pokud se začne kreslit v čase $t = 0$, bude se na velikost štětce aplikovat modifikátor na indexu 0, tedy $widthModifier[0]$. V čase $t = timestep$ se bude aplikovat modifikátor na indexu 1, a obecně v čase $t = n * timestep$ se bude aplikovat modifikátor na indexu n. Toto platí pro všechna $n < modLen$.

Proměnná *paintTime* se vypočte jako čas od začátku tahu štětcem *currPaintTime*, který je modulován časem na jednu iteraci polem *widthModifier*. Čas *currPaintTime* je modulován, protože pole *widthModifier* je procházeno cyklicky, dokud není tah štětcem přerušen. Tento výpočet je zapsán vzorcem 5.24.

$$paintTime = currPaintTime\%timePerIter \tag{5.24}$$

Dále je třeba vypočítat indexy hodnot, mezi kterými bude interpolováno pomocí vzorců 5.25 a 5.26, kde *paintTime* se vypočte podle vzorce 5.24 a *timestep* se vypočte podle vzorce 5.23.

$$startIn = (int)(paintTime/timestep) \tag{5.25}$$

$$endIn = (startIn + 1)\%modLen \tag{5.26}$$

Pokud index *startIn* není poslední index v poli, tak je třeba vypočítat parametr *t*, $t \in < 0, 1 >$, který určuje váhu interpolovaných hodnot podle vzorce

$$t = (paintTime\%timestep)/timestep \tag{5.27}$$

Hodnota použitého modifikátoru velikosti je vypočtena jako

$$widthMod = Lerp(widthModifier[startIn], widthModifier[endIn], t) \tag{5.28}$$

Kde *Lerp* je metoda pro lineární interpolaci.

Aktuální velikost štětce se vypočte jako

$$currWidth = b.width * widthMod \tag{5.29}$$

5.8.2.2 Texturovací souřadnice

Jeden průchod polem *widthModifier* během tahu štětcem odpovídá jednomu průchodu celou texturou štětce. Jaká část textury je mapována na aktuální část tahu, neboli jaké jsou aktuální souřadnice do textury, se vypočte pomocí lineární interpolace mezi hodnotami *uSt* a *uEn*. Tyto hodnoty jsou vypočteny podle vzorců 5.30 a 5.31, kde *startIn* a *endIn* jsou indexy vypočtené pomocí 5.25 a 5.26 a *modLen* je délka pole *widthModifier*. Pokud je délka pole *widthModifier* rovna 1, oba indexy se nastavují na hodnotu 0.

$$uSt = startIn/(float)(modLen - 1) \tag{5.30}$$

$$uEn = endIn/(float)(modLen - 1) \tag{5.31}$$

Výpočet aktuální hodnoty souřadnice *u* tedy odpovídá vzorci

$$uVal = Lerp(uSt, uEn, t) \tag{5.32}$$

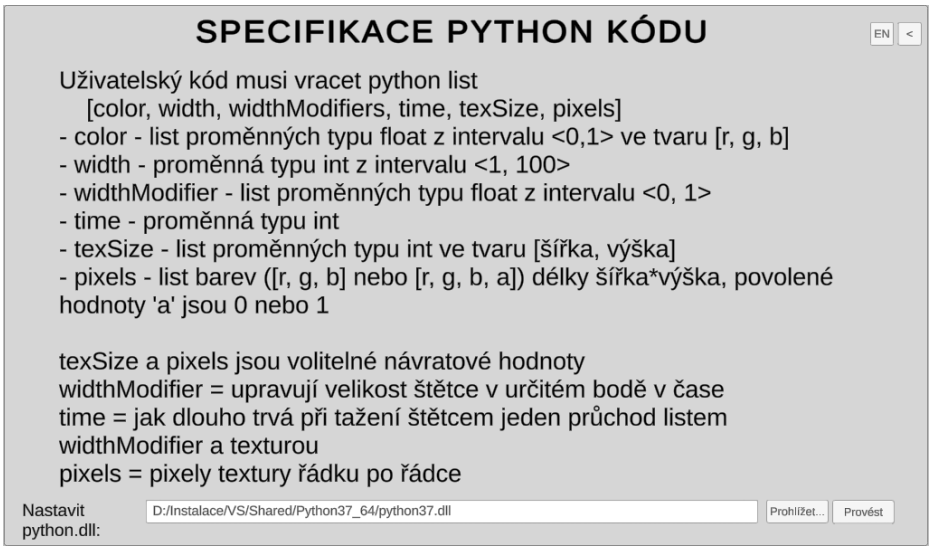
kde *t* odpovídá parametru, který byl vypočten při výpočtu *widthMod* pomocí vzorce 5.27 a *Lerp* je funkce pro lineární interpolaci.

5.8.2.3 Export XML souboru

K exportu štětce slouží třída *LineExporter*. Štětec je exportován do dvou souborů - XML soubru a png souboru. Oba soubory mají stejný název, který odpovídá jménu štětce. XML soubor je vytvářen pomocí instance .NET třídy *XmlSerializer*.

Exportovaný XML soubor obsahuje hodnoty všech atributů třídy *Brush*, popsané v kapitole 5.8.2, kromě atributu *Texture*. Textura je uchovávána jako výše zmíněný png soubor. Tvar XML souboru je popsán v příloze D. Exportované štětce se poté umisťují do složky *„Brushes"* na úrovni PaintVR.exe - souboru, který spouští aplikaci malování ve virtuální realitě.

Obrázek 5.15: Okno aplikace Brush Export.



Obrázek 5.16: Okno návodu v aplikaci Brush Export.

5.9 PaintVR

5.9.1 Navržené zadání pro SŠOP

Vytvořte aplikaci pro malování ve 3D. Aplikace musí fungovat ve virtuální realitě, a uživatel musí být schopen kreslit pomocí VR kontrolerů a měl umožněn volný pohyb po vytvořeném virtuálním světě.

Umožněte importování štětců ze souborů tak, aby byla aplikace kompatibilní s aplikací Brush Export. Je tedy třeba, aby uměla pracovat s XML soubory ve stejném tvaru. Z XML souboru jsou načítány parametry štětce, a je třeba aby aplikace uměla ke každému štětci načítat i odpovídající png soubor, který obsahuje texturu štětce, pokud takový soubor existuje. Je třeba, aby aplikace uměla zobrazit základní informace o štětci, jako je jeho jméno, barva a velikost. Popřípadě aby bylo umožněno uživateli některé parametry i měnit za běhu aplikace. Dále musí být umožněno do aplikace importovat více jak jeden štětec a uživateli umožnit si vybrat jakým štětcem si přeje kreslit.

Při vývoji je možné použít libovolné volně dostupné knihovny, vlastní kód, kód dodaných vzorových aplikací a dodaných knihoven. Aplikaci vyvíjejte tak, aby byla dále rozšiřitelná - dodržujte zásady slušného kódování, pište komentáře do kódu, dodejte dokumentaci, která mimo jiné zmiňuje, které knihovny byly použity. Dále by mělo být možné vytvořenou aplikaci začlenit do existujícího systému laboratoře techniky.

5.9.2 Vzorové řešení

Aplikace umožňuje kreslení ve virtuální realitě a změnu velikosti a barvy štětců za běhu. Štětce jsou načítány z XML souborů (definice tvaru XML bude dodána spolu s aplikací) umístěných ve složce „Brushes" na úrovni exe souboru spouštějícího celou aplikaci. Uživatel může mezi jednotlivými importovanými štětci přepínat, a vybrat si, kterým bude kreslit. Informace o momentálně aktivním štětci jsou zobrazovány na paletě na jednom z kontrolerů. Tahy štětcem vytvářené v prostoru uživatel může i mazat.

Aplikace omezuje pohyb uživatele po virtuální scéně tak, aby nemohl ve skutečném světe nevědomky odejít z prostoru vyhrazeného pro virtuální realitu. Velikost tohoto prostoru je nastavitelná pomocí konfiguračního souboru.

5.9.3 Implementace

Pro implementaci byl použit herní engine Unity. Byly použity Unity balíčky *OpenXR* a *XRInteractionToolkit* popsané v kapitole 5.1.3. Kontrolery a headset virtuální reality jsou ve scéně reprezentovány objekty z obrázku 5.14. Zdrojem modelu kontroleru je

SketchFab [Vio19], tento model byl dále upraven. Dále byly použity textury z Unity AssetStore [rpg21].

Aplikace přijímá ze serveru objekty virtuálního světa a odesílá tahy štětcem a objekty headsetu s kontrolery. Kromě toho, že aplikace slouží k malování ve virtuální realitě, a uživatel může vytvářet vlastní umělecká díla, může uživatel také použít tahy štětcem ke zvýraznění objektů virtuálního světa nebo jejich částí. Stejně jako v aplikaci Box City, není v PaintVR simulována gravitace.

5.9.3.1 Ovládání aplikace

Na levém kontroleru je umístěn *XRRayInteractor*, který umožňuje uživateli pohyb po světě pomocí teleportace. Na levém kontroleru je také umístěn herní objekt typu *UI Canvas*, který slouží jako paleta, která slouží k ovládání aplikace. Na pravém kontroleru je také umístěn *XRRayInteractor*, díky kterému je možné pravým kontrolerem interagovat s tlačítky palety. Pomocí tlačítek pravého kontroleru lze kreslit a přepínat mezi štetci. Pomocí tlačítek levého kontroleru lze přepínat mezi módem kreslení a gumování. Návod na ovládání v aplikaci PaintVR je vidět na obrázku 5.19.

Kontrolery *HTC Vive Pro 2.0* jsou symetrické, a proto lze používat i levý kontroler v pravé ruce a naopak. Aplikace je tak přístupná jak pravákům, tak levákům, bez dalších úprav.

5.9.3.2 Pohyb v prostoru

Pohyb v prostoru v aplikaci PaintVR je implementován stejným způsobem jako v aplikaci Box City. Bezpečný prostor pro pohyb uživatele je označen pomocí zeleného obdélníku na zemi virtuálního světa. Jeho velikost je opět možné nastavit v konfiguračním souboru aplikace. Další detaily jsou popsány v kapitole 5.7.2.4.

5.9.3.3 Gumování tahů štetcem

Pravý kontroler má kolider typu *SphereCollider*. Při gumování se detekuje kolize tohoto kontroleru s objekty, které mají jako tag nastavenou hodnotu „line". Herní objekty reprezentující tahy štětcem mají kolider typu *MeshCollider*, aby detekce kolizí byla co nejpřesnější. Implementace detekce kolize je taková, že při stisknutí tlačítka pro gumování je vykonán následující kód

$$delLines = Physics.OverlapSphere(controllerPos, eraserW/2) \tag{5.33}$$

kde *controllerPos* je pozice pravého kontroleru, a *eraserW* je velikost gumy. Metoda *Physics.OverlapSphere* je Unity metoda, která vrací seznam se všemi objekty, se kterými koliduje pomyslná koule se zadaným středem a poloměrem. Tedy vrací všechny

objekty, které jsou pozici *controllerPos* blíže, než zadaný poloměr *eraserW/2*. Vrácený seznam *delLines* poté stačí projít a smazat všechny herní objekty, které mají tag nastaven na hodnotu „line".

5.9.3.4 Trojúhelníkový pás

Tah štětcem je generován jako trojúhelníkový pás umístěný v prostoru. Tento trojúhelníkový pás je reprezentován třídou *TriangleStrip* a k jeho úpravám slouží třída *TriangleStripGenerator*.

Při vytvoření nového trojúhelníkového pásu se instancuje trojúhelníková síť v Unity scéně, která slouží k zobrazení tahu štětcem. Trojúhelníková síť v Unity vždy musí mít alespoň tři vrcholy a jeden trojúhelník. Proto se nově vytvořený trojúhelníkový pás zobrazí ve scéně až po přidání druhé validní pozice kontroleru. Při vytvoření se do vznikající trojúhelníkové sítě přidají dva body, jejichž pozice se vypočtou podle vzorců 5.34 a 5.35. Dále po celou dobu kreslení tahu při každém volání metody Unity *Update* dochází k vykonání metody *AddPointLine* třídy *TriangleStripGenerator*, která přidává další body do trojúhelníkového pásu.

Trojúhelníkový pás je uchováván jako instance Unity třídy *Mesh*. Tato třída slouží k reprezentaci libovolné trojúhelníkové sítě. Uchovává souřadnice vrcholů jako pole třídy *Vector3*, které je pojmenované *vertices*, uv souřadnice vrcholů jsou uchovávány jako pole třídy *Vector2*, které má jméno *uv*, a v poli celých čísel *triangles* jsou ukládány indexy vrcholů, které vždy po trojicích tvoří trojúhelníky zobrazované sítě.

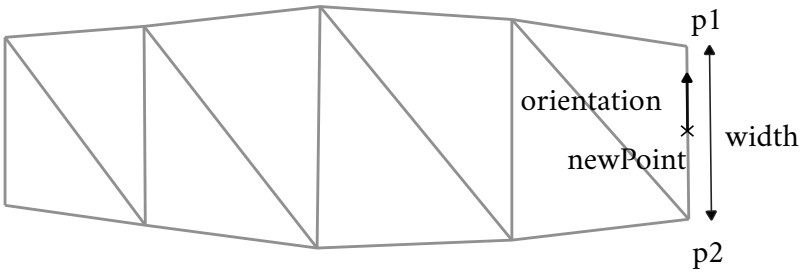
Když je do této trojúhelníkové sítě přidán nový bod, musí se vytvořit nová pole, která budou delší o data přidávaného bodu. Nejprve je třeba upravit pole *vertices*. Je třeba vytvořit pole o jeden index delší, a uložit do něj instanci třídy *Vector3* obsahující x, y a z souřadnice nového vrcholu. Poté lze upravit pole *uv*, a prodloužit jej o instanci třídy *Vector2*, obsahující u a v souřadnice bodu v textuře. Pokud je přidáván i trojúhelník, tak je pole *triangles* prodlouženo o tři indexy vrcholů, které nový trojúhelník tvoří. Vrcholy se vždy indexují od nuly, a index udává pořadí v jakém byl daný vrchol do trojúhelníkové sítě vložen.

Při každém přidání nové pozice kontroleru do trojúhelníkového pásu reprezentujícím tah, je třeba znát pozici kontroleru *newPoint* a směrový vektor jeho lokální osy y *orientation* a aktuální šířku štětce *width*. Aktuální šířka štětce se vypočte podle postupu popsaném v kapitole 5.8.2. Pozice nově přidávaných vrcholů do trojúhelníkového pásu se vypočtou podle vzorců 5.34 a 5.35.

$$p_1 = newPoint + (width/2) * orientation \tag{5.34}$$

$$p_2 = newPoint - (width/2) * orientation \tag{5.35}$$

Dva nové trojúhelníky se vytvoří tak, aby odpovídaly nákresu 5.17.



Obrázek 5.17: Trojúhelníkový pás reprezentující tah štětcem. Body *p1* a *p2* odpovídají bodům nově přidávaným do pásu vypočteným podle vzorce 5.34 respektive 5.35.

Jeden průchod polem *widthModifier* během tahu odpovídá jednomu průchodu celou texturou. Oba nově vzniklé body p_1 a p_2 mají stejnou souřadnici u, která je vypočtena pomocí vzorce 5.32. Vrchol p_1 má UV souřadnice rovny $(u,v) = (uVal, 1)$ a vrchol p_2 má UV souřadnice rovny $(u,v) = (uVal, 0)$.

Vzniklá trojúhelníková síť je renderována pomocí oboustranného shaderu, ve kterém nedochází k výpočtu osvětlení. Vykreslují se tedy jak přivrácené, tak odvrácené strany trojúhelníků. Tah štětcem v prostoru je vidět na obrázku 5.21.

5.9.3.5 Paleta

Na levém kontroleru se nachází paleta, která slouží k ovládání aplikace. Zobrazují se na ní informace o aktivním štětci, s její pomocí lze měnit barvu a šířku štětce, a pomocí tlačítek, které jsou na ní umístěné, lze ovládat i ostatní funkce aplikace. Tyto funkce jsou import štětců, ukončení aplikace, zobrazení ovládání, přepínání mezi jazyky, které aplikace podporuje, a uložení scény. Paleta je vidět na obrázku 5.20.

5.9.3.6 Import štětců

Štětce, které mají být importovány do aplikace se musí nacházet ve složce *„Brushes"*. Tato složka musí být umístěna na stejné úrovni jako exe soubor, spouštějící celou aplikaci. Při spuštění aplikace se štětce automaticky importují a zobrazují se v nabídce štětců na paletě. Pokud je obsah složky *„Brushes"* modifikován za běhu aplikace, je nutné na paletě stisknout tlačítko *„Importovat štětce"* aby došlo k propagaci této změny a zobrazení aktuální nabídky štětců.

5.9.3.7 Uložení scény

Princip uložení scén v aplikaci PaintVR je stejný jako v aplikaci Box City. Na paletě se nachází tlačítko „Scény", po jeho stisknutí se zobrazí okno, ve kterém lze ukládat

a načítat scény. Opět je možné uložit tři rozdílné scény. Paleta s otevřeným oknem pro správu scén je vidět na obrázku 5.20. Objekty virtuálního světa jsou ukládány do podsložek složky „Saves", která se nachází na stejné úrovni jako exe soubor, spouštějící celou aplikaci. Další detaily jsou popsány v kapitole 5.7.2.2.

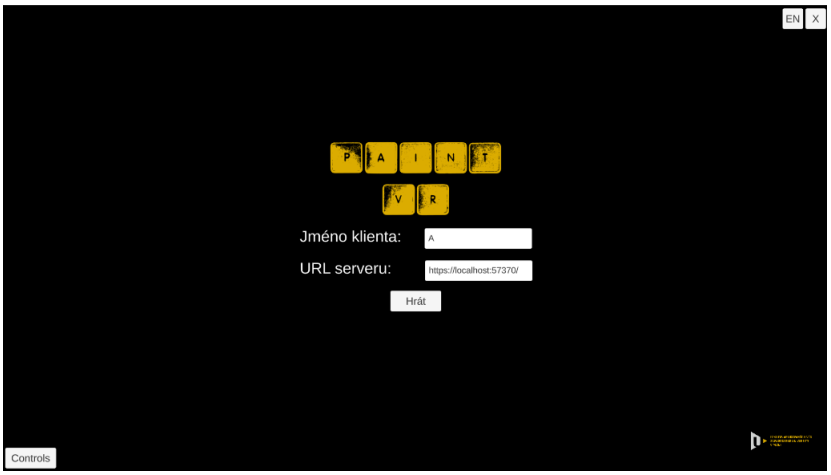
5.9.3.8 Komunikace se serverem

Po spuštění aplikace se zobrazí obrazovka z obrázku 5.18. Uživatele musí zadat jméno klienta a URL serveru. Zadané jméno opět musí být unikátní pro všechny VR klienty. Jméno i URL lze také nastavit v konfiguračním souboru aplikace.

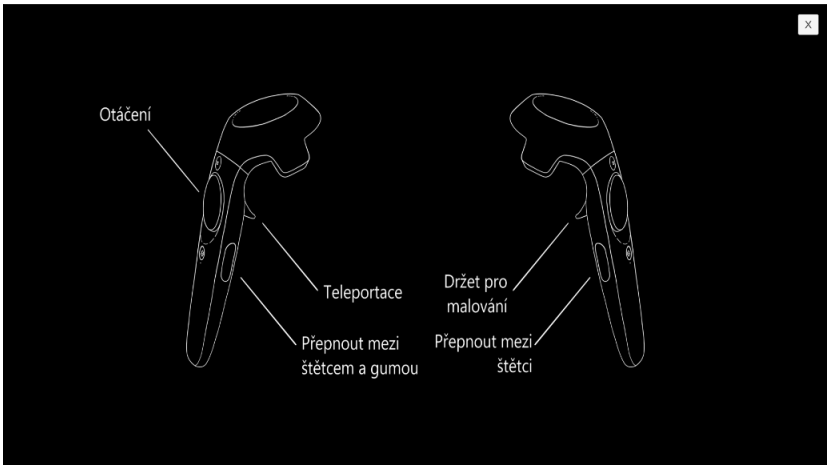
Klient se po spuštění automaticky pokusí připojit na server. Pokud připojení není úspěšné, klient se o připojení pokouší o pět vteřin později znovu. Pokud dojde k nečekanému přerušení spojení se serverem, knihovna SignalR se pokouší znovu připojit. Pokud je připojení úspěšné, všechny lokální objekty jsou smazány a znovu přijaty ze serveru. Pokud se klientu nepodaří připojení obnovit, dojde ke smazání lokálních objketů a klient se s intervalem pěti vteřin pokouší znovu připojit.

Aplikace posílá na server objekty reprezentující headset a kontrolery, jejich modely jsou na obrázku 5.14. Jedná se o objekty typu Mesh, které mění svou pozici v závislosti na pohybu uživatele. Tyto objekty komponentami odpovídají prefabu *ServerMeshPrefab*. K aktualizaci transformace objektů zaslaných na server používají třídu *ReportTransformChange* z knihovny *Common.Unity*. Kontrolery mění barvu v závislosti na stavu připojení k serveru. Při korektním ukončení aplikace jsou objekty reprezentující headset a kontrolery ze serveru smazány.

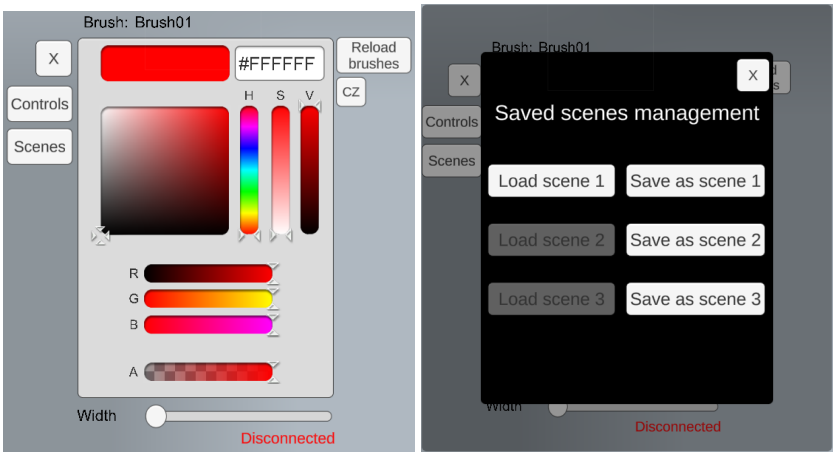
Dále aplikace odesílá na server vytvořené tahy štětcem jako objekty typu Mesh. Tahy jsou instancovány jako herní objekty, které komponentami odpovídají prefabu *ServerMeshPrefab* popsaném v kapitole 5.1.4. Hodnota vlastnosti *ColiderType* je řetězec s hodnotou „mesh", serializované do bytového pole pomocí třídy *StringSerializer* z knihovny *Common*. Hodnota vlastnosti *ColiderSize* je prázdné bytové pole. Dále tento klient přijímá objekty, které se na serveru nacházejí. Uživatel s nimi nemůže manipulovat, a na objekty nepůsobí gravitace.



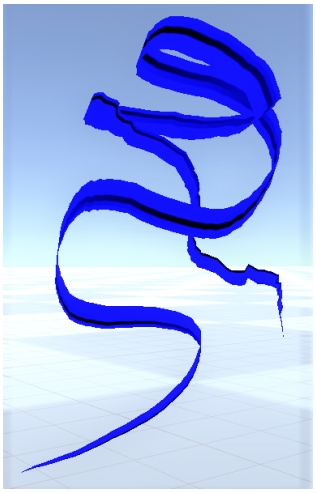
Obrázek 5.18: Intro obrazovka aplikace PaintVR.



Obrázek 5.19: Obrazovka s ovládáním aplikace PaintVR.



Obrázek 5.20: Paleta v aplikaci PaintVR. Vpravo s otevřeným oknem s možnostmi uložení scény. Tlačítka pro načtení scény 2 a 3 nejsou aktivní, protože se na disku nenachází odpovídající složky.



Obrázek 5.21: Tah štětcem v aplikaci PaintVR.

Výsledky 6

V této kapitole se nachází přehled dosažených výsledků a výsledky testování.

6.1 Spolupráce aplikací

Cílem systému bylo, aby spolu jednotlivé aplikace spolupracovaly. Server slouží k uchovávání virtuálního světa a jeho objektů, které přijímá od klientů a které se posílají do těch klientů, kteří je umí zobrazit.

Klient Annoying Fly na server odesílá objekt typu Bitmap, tedy obrázek, který obsahuje screenshot herní obrazovky. Klient Depth Mesh na server odesílá objekt typu Mesh, jedná se o trojúhelníkovou síť vytvořenou ze snímaného hloubkového obrazu. Tento objekt umí také přijímat. V aplikaci PaintVR může uživatel ve scéně kreslit, a tyto tahy štětcem se také odesílají na server jako objekt typu Mesh. V aplikaci Box City může uživatel manipulovat s krabicemi v herní scéně, tyto krabice se posílají na server jako objekty typu Mesh. V obou klientech pro virtuální realitu jsou objekty virtuálního světa typu Mesh a Bitmap zobrazovány a v aplikaci Box City s nimi může uživatel pohybovat, a oba VR klienti posílají na server rig (tedy objekty reprezentující headset s kontrolery). Diagram komunikace klientů a serveru je vidět na obrázku 6.1.

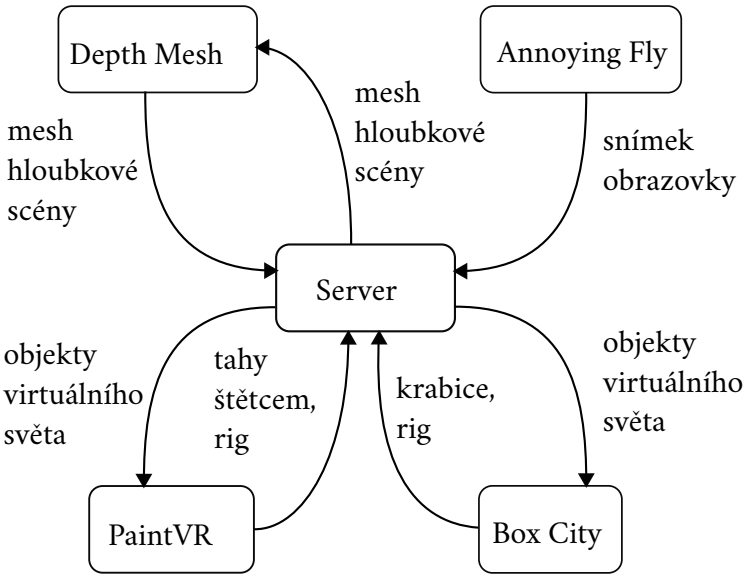
Na serveru jsou jednotlivé objekty identifikovány pomocí jména, a proto jsou jména jednotlivých objektů tvořena tak, aby k serveru mohlo být připojeno více klientů stejného typu zároveň. Jedinou podmínkou je, aby dva klienti stejného typu - tedy dva klienti Annoying Fly, dva klienti Depth Mesh, nebo dva klienti pro virtuální realitu - neměli stejné jméno. Tvar jmen je volen následovně:

- Screenshot herní obrazovky, zasílán z klienta Annoying Fly, má přiřazené jméno ve tvaru „FlyKiller_<jmeno>", kde „<jmeno>" je jméno klienta, který jej odesílá.

- Jméno trojúhelníkové sítě, zasílané z klienta Depth Mesh, má tvar „DepthMesh_<jmeno>", kde „<jmeno>" je jméno klienta, který jej odesílá.

- Krabice zasílané z klienta Box City, mají přiřazována jména ve tvaru „Cardboardbox_<jmeno>_<cislo>", kde „<cislo>" označuje pořadí v jakém byla krabice vytvořena v rámci daného klienta a „<jmeno>" je jméno daného klienta.
- Tahy štětcem z klienta PaintVR mají jména ve tvaru „Line_<jmeno>_<cislo>", kde „<cislo>" označuje pořadí v jakém byl tah štětcem vytvořen v daném klientovi, a „<jmeno>" je jméno klienta. Před začátkem kreslení se vždy nejprve projodu všechny objekty na serveru, a určí se počet tahů, které byly vytvořeny tímto klientem. Při vytváření nových tahů se začne počítat od tohoto počtu.

Na obrázcích 6.2, 6.3, 6.4 a 6.5 je tato spolupráce aplikací názorně ilustrována. Na obrázku 6.2 je vidět obrazovka klienta Annoying Fly a na obrázku 6.3 trojúhelníková síť vytvořená z hloubkového obrazu v aplikaci Depth Mesh. Na obrázku 6.4 je zachycen hráč při zvýrazňování jedné oblasti v trojúhelníkové síti. Dále na obrázku 6.5 je vidět pohled na tyto vytvořené objekty v aplikaci Box City.



Obrázek 6.1: Diagram znázorňující spolupráci klientských aplikací se serverem.

6.2 Vzniklé úkoly pro středoškoláky

Aby studenti mohli s tímto systémem dále pracovat a doplňovat jej o další aplikace, je třeba, aby jim vzniklá sada úloh umožnila osvojit si práci s jednotlivými zařízeními a pomohla jim vyvíjet vlastní software v rámci středoškolských odborných prací (SŠOP). Ke všem vytvořeným vzorovým aplikacím jsou dodány zdrojové kódy, které mohou studenti libovolně modifikovat, a využít je pro vytváření vlastních aplikací. Všechny vytvořené aplikace podporují jak český tak anglický jazyk. V této kapitole budou shrnuty funkce jednotlivých vytvořených aplikací a zadání.

6.2.1 Hloubková kamera RealSense

Pro hloubkovou kameru RealSense D415 byly dodány vzorové aplikace Annoying Fly a Depth Mesh. V těchto dvou aplikacích by měl student získat dostatečný přehled o funkcionalitě hloubkové kamery RealSense D415 na to, aby s pomocí jejich zdrojových kódů a volně dostupných knihoven byl schopen vypracovat zadání SŠOP popsané v kapitole 5.6.

Aplikace Annoying Fly slouží k seznámení se základní funkcí hloubkových kamer. Uživatel má za úkol odhánět promítanou mouchu z obrazovky, k tomu musí nastavit snímanou hloubku tak, aby kamera zabírala pouze jeho, a popřípadě posunout obrázek se snímanou scénou. Snímaná hloubka se zobrazuje jako jednobarevný nebo jako obarvený stín, kde barva odpovídá hloubce dané části scény.

Aplikace Depth Mesh slouží k rozvinutí získaného povědomí o snímaném hloubkovém obrazu. V aplikaci je možné nejenom nastavit snímanou hloubku, ale i ovlivňovat kvalitu vznikajícího hloubkového obrazu pomocí aplikování filtru postprocesingu a nastavování jejich parametrů. Funkce filtrů a jejich parametrů je v aplikaci uživateli přiblížena pomocí informačních bublin, jejich detailnější popis se nachází v kapitole 3.2.4.1, popřípadě v dokumentaci Intel RealSense. Pro pochopení jejich funkce uživatel může snímat různé povrchy a objekty v různých vzdálenostech od kamery a experimentovat s nastavením parametrů filtrů. Ze získaného hloubkového obrazu lze v aplikaci vytvořit trojrozměrný model snímané scény. Tento model je reprezentován jako trojúhelníková síť. Uživatel může s touto sítí dále manipulovat a přímo v aplikaci ji upravovat pomocí Python kódu. Příklady kódu, který může vykonat, jsou v příloze C - Ukázkové Python skripty.

6.2.2 Virtuální realita

Pro virtuální realitu Vive HTC Pro 2.0 byly dodány vzorové aplikace Box City, PaintVR a k ní podpůrná aplikace Brush Export. S pomocí zdrojových kódu těchto aplikací by měl být student schopen vytvořit v herním enginu Unity vlastní aplikace využívající virtuální realitu.

V aplikaci Box City se uživatel může naučit orientovat ve virtuální realitě, naučit se v ní pohybovat a seznámit se s jejím ovládáním. Úkol uživatele je, sbírat ve virtuálním městečku krabice a odnést je na jedno určité místo. Díky tomu se naučí manipulovat s virtuálními objekty. Aplikace PaintVR umožňuje uživateli kerslit v prostoru pomocí kontrolerů virtuální reality. Uživatel může ovlivňovat barvu a velikost štětce, kterým kreslí, a přepínat mezi štětci. Štětce lze vytvořit pomocí Python kódu v aplikaci Brush Export. PaintVR je vzorovým vypracováním zadání pro SŠOP, a měla by refletkovat úroveň aplikací, jaké by měl být student schopen vytvořit s použitím dodaných zdrojových kódů vzorových aplikací a volně dostupných knihoven.

Aplikace Box City a PaintVR mohou vhodně doplňovat i aplikace pro hloubkovou kameru, obzvlášťe aplikaci Depth Mesh, popřípadě i jiné aplikace pro jiná zařízení, které implementují studenti. Obě tyto vzorové aplikace přijímají objekty ze serveru, a uživatel si je může detailněji prohlížet ve virtuální realitě. V aplikaci Box City s nimi může manipulovat a prohlédnout si je ze všech stran, a v aplikaci PaintVR může okolo nich kreslit, například proto, aby pro jiné uživatele, kteří svět také pozorují, zvýraznil nějakou jejich část.

Aplikace Brush Export je podpůrná aplikace k aplikaci PaintVR, slouží k vytváření štětců pomocí uživatelského Python kódu. Uživatel může ovlivňovat parametry štětce, tedy jeho jméno, velikost, průběh veikosti v čase a jeho texturu. Vytvořený štětec se poté exportuje jako XML soubor, který lze načíst v aplikaci PaintVR. Příklady štětců, které lze vytvářet a vzorové kódy, které tyto štětce vytvoří jsou v příloze C.

6.3 Uživatelské testování

Aplikace byla testována pomocí uživatelských testů. Tento styl testování byl zvolen, protože lépe otestuje reálné použití klientských aplikací v laboratoři techniky.

Před nasazením v laboratoři techniky Gymnázia Sokolov byly provedeny testy podle scénářů, které se nacházejí v příloze B. Nasazené aplikace byly poté testovány studenty a vyučujícími gymnázia. Výsledky testování byly předávány prostřednictvím uživatelských dotazníků a emailové komunikace.

Dotazníky obsahovaly otázky na intuitivnost ovládání aplikací, komunikaci se serverem a intuitivnost a názornost ovládání daného zařízení. Součástí byl i prostor pro studenty navrhovat nové funkcionality aplikací. Tyto navrhované funkce mohou být obsahem další práce, nebo mohou středoškolští studenti tyto funkce do aplikací doplnit v rámci SŠOP. Z výsledků dotazníků vyplývá, že žádná z aplikací neměla problém s připojením k serveru. Některé aplikace byly hodnoceny jako méně intuitivní, konkrétně se jedná o aplikace Depth Mesh a PaintVR. Z toho důvodu jsou dodána instrukční videa, která obsahují popis aplikací a návod, jak s nimi pracovat. Ke každé klientské aplikaci vypracované v rámci této práce bylo vytvořeno jedno instrukční video, a bylo vytvořeno také instrukční video, které obsahuje základní znalosti potřebné pro práci s trojúhelníkovými sítěmi. Toto video je koncipované tak, aby pomohlo studentům pochopit princip upravování trojúhelníkové sítě v aplikaci Depth Mesh. Odpovědi na dotazníky jsou k nahlédnutí v příloze E, ve zbytku této kaptioly jsou odpovědi dále detajněji analyzovány.

Z některých odpovědí v dotaznících vyplývá, že pravděpodobně nebyl správně nastaven obsah konfiguračního souboru, který klientovi umožňuje zapamatovat si URL serveru a jméno klienta. Formát konfiguračního souboru byl proto zdůrazněn v dodané uživatelské dokumentaci.

6.3.1 Aplikace pro RealSense

Ovládání aplikace Annoying Fly bylo hodnoceno ve většině odpovědí jako velmi jednoduché. Uživatelé navrhovali jako další rozšíření především propracovanější pohyb mouchy. Z šesti obdržených odpovědí se tento návrh objevil dvakrát. Jeden uživatel navrhoval herní režim, při kterém se na obrazovce zobrazí větší počet much, a hráč je časován jak dlouho mu trvá všechny odehnat. Dva uživatelé nenašli počítadlo kolikrát hráč mouchu zasáhl, proto bylo toto počítadlo v další verzi aplikace zvýrazněno a zvětšeno.

Na dotazník k aplikaci Depth Mesh přišlo pět odpovědí. Všichni uživatelé uměli v aplikaci vytvořit trojrozměrný model scény a uměli s náhledem na něj bez větších obtíží manipulovat. Jeden z pěti uživatelů se neuměl připojit k serveru a tímpádem ani odeslat nebo přijmout trojúhelníkovou síť. Dva uživatelé nevěděli jak vykonat Python kód. Jeden z nich ke konci dotazníku podle odpovědí Python kódu rozuměl a zmiňoval, že se „prvních pár minut nebyl schopen v aplikaci zorientovat". Více korespondentů zmiňovalo, že aplkiace je obtížnější na pochopení, hlavně kvůli velkému počtu možností, které poskytuje. Z tohoto důvodu je k aplikaci dodáváno instrukční video, které uživatelům pomůže seznámit se s funkcionalitami aplikace. Dále změny způsobené některými z filtrů postprocesingu byly jedním až dvěmi uživateli hodnoceny jako nepostřehnutelné. Jednalo se hlavně o časové vyhlazování, u kterého záleží síla jeho efektu i na parametrech ostatních filtrů, a o záplatování děr, kde záleží hlavně na snímané scéně a nastavené snímané hloubce.

6.3.2 Aplikace pro virutální realitu

Častým problémem VR aplikací bylo částečné nebo úplné zasekávání. Experimentálně bylo zjištěno, že k tomu dochází, protože se headset nesprávně detekuje jako „nenasazený", a proto se nepropagují pohyby ovladačů do Unity aplikace. Pro detekci nasazení a sundání headsetu slouží senzor přiblížení, který se nachází v brýlích. Potom záleží na tom, jak má uživatel headset nasazen, a může se stát, že hlava uživatele nebude detekována. Tato chyba je efektivně odstraněna větším utažením headsetu, nebo lze v nastavení SteamVR vypnout pozastavování aplikace když si uživatel headset sundá z hlavy. Poté stačí aby senzor v brýlích jednou detekoval, že si jej někdo nasadil, a pak si uživatel může headset opět povolit tak, aby mu byl pohodlný. Tento problém se odrazil hlavně na hodnocení aplikace PaintVR.

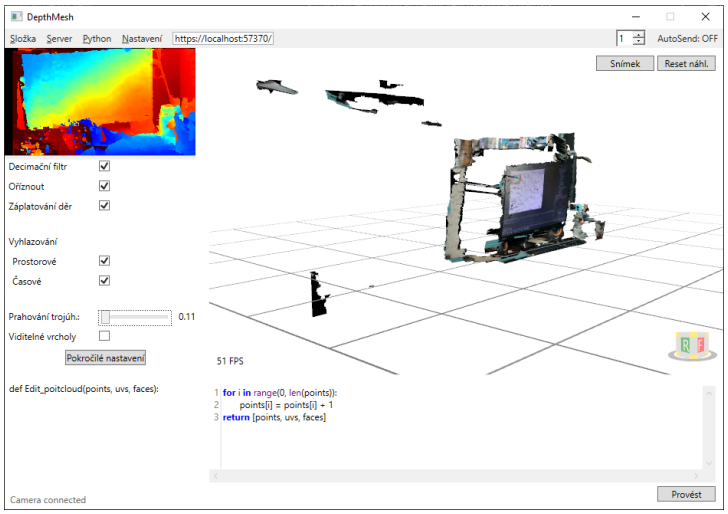
Dotazník pro aplikaci Box City vyplnili čtyři lidé. Pohyb v prostoru byl hodnocen jako intuitivní, na hodnocení manipulace s krabicemi se odrazily výše popsané potíže s headsetem. Aplikace Brush Export byla testována dvěmi uživateli, oba ji hodnotili jako intuitivní, a rozuměli očekávanému tvaru Python kódu.

Na dotazník k aplikaci PatinVR přišlo pět odpovědí. Podle statistik je vidět, že tato aplikace byla znatelně hůře hodnocená. Tři obdržené odpovědi jsou ovšem od

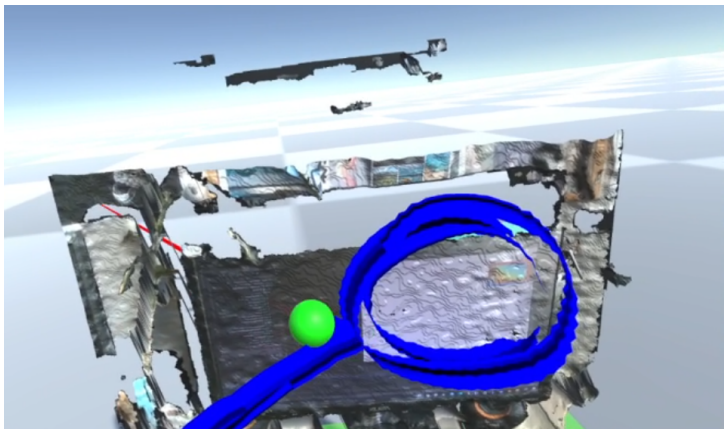
uživatelů, kteří aplikace testovali před vyřešením problému se zasekáváním. Proto měli větší problémy s ovládáním aplikace, nebo ji nemohli používat vůbec, v závislosti na tom, jak si nasadili headset. Dva uživatelé, kteří dotazník vyplňovali po opravení zasekávání VR, hodnotili ovládání aplikace jako intuitivní.

Pro aplikaci PaintVR bylo uživateli navrhováno nejvíce vylepšení. Jedním z nich byl návrh sjednotit všechny tahy štětcem od jednoho klienta do jednoho objektu, aby jiní uživatelé nemohli přesunem jednotlivých tahů rozbít vytvořené umělecké dílo. Dále byl vznesen návrh „tvůrce štětců" přímo v aplikaci PaintVR a ne v oddělené aplikaci Brush Export. Toto by ovšem omezilo schopnost uživatele štětec upravovat, protože ve VR by nebyl schopný efektivně psát Python kód. Dále jeden z uživatelů postrádal možnost volit mezi prostředími, ve kterém se ve virtuální realitě pohybuje. Toto bylo doplněno do nové verze aplikace, nyní může uživatel stisknutím mezerníku na klávesnici volit mezi několika verzemi zobrazovaného skyboxu a barvy podlahy, po které chodí.

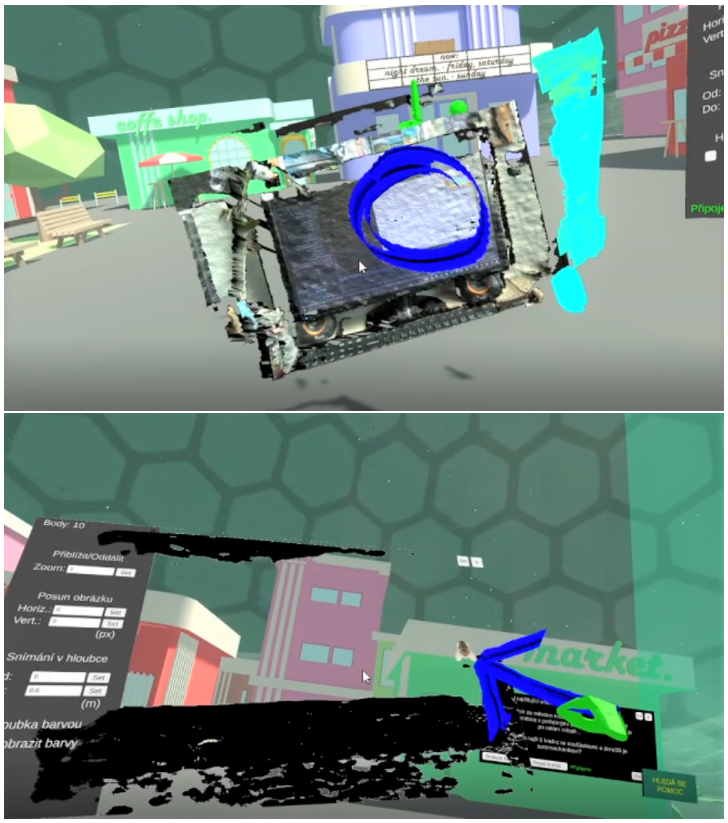
Obrázek 6.2: Záznam obrazovky aplikace Annoying fly.



Obrázek 6.3: Záznam okna aplikace Depth Mesh. V náhledu se nachází trojúhelníková síť získaná z hloubkového obrazu.



Obrázek 6.4: Záznam obrazovky aplikace PaintVR. Ve scéně je vidět trojúhelníková síť z aplikace Depth Mesh, ve které uživatel upozorňuje na zobrazený náhled.



Obrázek 6.5: Záznam obrazovky aplikace Box City. Na levém obrázku je vidět trojúhelníková síť z aplikace Depth Mesh. Na pravém obrázku je vidět bitmapa z aplikace Annoying Fly a hráč přesouvá zvýraznění vytvořené v aplikaci PaintVR.

Závěr 7

Tato diplomová práce je součástí projektu, zabývajícího se vývojem systému pro demonstrační laboratoř techniky v Gymnáziu Sokolov, skládajícího se ze serveru a klientských aplikací. Klientské aplikace byly rozděleny do tří úrovní, podle toho, jakou úroveň pochopení problematiky požadují od uživatele. První úroveň slouží k seznámení uživatele s daným zařízením, druhá umožňuje uživateli získat větší kontrolu nad zařízením a manipulovat s daty pro zařízení nebo s daty ze zařízení pomocí uživatelského kódu. Třetí úroveň je koncipována jako zadání pro středoškolský odborný projekt, které by student mohl vypracovat pod dozorem konzultanta. Uživatelského rozhraní všech aplikací, vytvořených v rámci této práce, je možné přepínat mezi češtinou a angličtinou.

V rámci této práce byly vytvořeny následující klientské aplikace

- Annoying Fly - aplikace prvního stupně pro hloubkovou kameru RealSense,
- Depth Mesh - aplikace druhého stupně pro hloubkovou kameru RealSense,
- Box City - aplikace prvního stupně pro virtuální realitu,
- Brush Export - aplikace druhého stupně pro virtuální realitu,
- PaintVR - aplikace třetího stupně pro virtuální realitu.

Server dodaný do laboratoře byl vyvinut Dominikem Pochem v rámci diplomové práce Demonstrační aplikace pro laboratoř techniky[Poc23]. Celý systém je koncipován tak, aby byl rozšiřitelný o nové klienty a aby bylo možné implementovat nové typy objektů, které jsou na server posílány, bez nutnosti změny existujících aplikací.

Systém byl úspěšně nasazen v Gymnáziu Sokolov a učitelský sbor byl instruován, jak s jednotlivými komponentami zacházet. K aplikacím byla dodána stručná uživatelská dokumentace a instrukční videa, detailněji popisující práci s nimi. Všechny aplikace byly otestovány pomocí uživatelských testů a nalezené chyby byly opraveny. Vyvinuté aplikace a jejich zdrojové kódy jsou k dispozici z online úložiště GitHub[KP23].

Bibliografie

[Aiv21] AIVERO. *Overview of Depth Cameras*. Aivero, 2021-11. Dostupné také z: https://aivero.com/overview-of-depth-cameras/.

[Bar22] BARNARD, Dom. *History of VR - timeline of events and Tech Development*. VirtualSpeech, 2022-10. Dostupné také z: https://virtualspeech.com/blog/history-of-vr.

[Ber18] BERKMAN, Mehmet Ilker. History of virtual reality. *Encyclopedia of Computer Graphics and Games*. 2018, s. 1–9. Dostupné z DOI: 10.1007/978-3-319-08234-9_169-1.

[Emm] EMMRICH, Jean-Francois. *What is structured light scanning?* Dostupné také z: https://blog.medit.com/medit/what-is-structured-light-scanning.

[GO11] GASTAL, Eduardo S.; OLIVEIRA, Manuel M. Domain transform for edge-aware image and video processing. *ACM SIGGRAPH 2011 papers*. 2011. Dostupné z DOI: 10.1145/1964921.1964964.

[Gol10] GOLDSTONE, Will. *Unity Game Development Essentials*. Packt Publ., 2010.

[GT20] GRUNNET-JEPSEN, Anders; TONG, Dave. *Depth Post-Processing for Intel® RealSense™ Depth Camera D400 Series*. Intel, 2020. Dostupné také z: https://dev.intelrealsense.com/docs/depth-post-processing.

[He+17] HE, Ying; LIANG, Bin; ZOU, Yu; HE, Jin; YANG, Jun. Depth errors analysis and correction for time-of-flight (TOF) cameras. *Sensors*. 2017, roč. 17. Dostupné z DOI: 10.3390/s17010092.

[Int18] INTEL. *Projection in Realsense SDK 2.0*. 2018. Dostupné také z: https://github.com/IntelRealSense/librealsense/wiki/Projection-in-RealSense-SDK-2.0.

[Int19] INTEL. *Post-processing filters*. 2019. Dostupné také z: https://dev.intelrealsense.com/docs/post-processing-filters.

[Int22] INTEL. *Depth camera D415*. 2022-10. Dostupné také z: https://www.intelrealsense.com/depth-camera-d415/.

[Int] INTEL. *Intel® RealSense™ Developer Documentation*. Intel Corporation. Dostupné také z: https://dev.intelrealsense.com/docs/supported-platforms-and-languages.

[Khr23] KHRONOS. *OpenXR*. Khronos, 2023. Dostupné také z: https://www.khronos.org/openxr/.

[KP23] KÖNIG, A.; POCH, D. *Projekt IT Laboratoře Sokolov*. 2023. Dostupné také z: https://github.com/ITLaboratorySokolov.

[Lie+06] LIEBE, C.C. et al. Spacecraft Hazard avoidance utilizing Structured Light. *2006 IEEE Aerospace Conference*. 2006. Dostupné z DOI: 10.1109/aero.2006.1655898.

[MP79] MARR, D.; POGGIO, Tomaso A. A computational theory of Human Stereo Vision. *Proceedings of the Royal Society of London. Series B. Biological Sciences*. 1979, roč. 204, č. 1156, s. 301–328. Dostupné z DOI: 10.1098/rspb.1979.0029.

[MN20] MEHENDALE, Ninad; NEOGE, Srushti. Review on lidar technology. *SSRN Electronic Journal*. 2020. Dostupné z DOI: 10.2139/ssrn.3604309.

[MŠM14] MŠMT, ČR. *Strategie digitálního vzdělávání do roku 2020*. 2014. Dostupné také z: https://www.msmt.cz/vzdelavani/skolstvi-v-cr/strategie-digitalniho-vzdelavani-do-roku-2020.

[MŠM21a] MŠMT ČR. *Celkové vyhodnocení Strategie digitálního vzdělávání do roku 2020*. 2021. Dostupné také z: https://www.msmt.cz/vzdelavani/skolstvi-v-cr/celkove-vyhodnoceni-strategie-digitalniho-vzdelavani-do-roku.

[MŠM21b] MŠMT ČR. *Rámcový vzdělávací program pro gymnázia, RVP G*. 2021. Dostupné také z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcove-vzdelavaci-programy-pro-gymnazia-rvp-g.

[Nef+17] NEFIAN, Ara et al. Structured light-based hazard detection for planetary surface navigation. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017. Dostupné z DOI: 10.1109/IROS.2017.8206090.

[OK93] OKUTOMI, M.; KANADE, T. A multiple-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1993, roč. 15, č. 4, s. 353–363. Dostupné z DOI: 10.1109/34.206955.

[Poc23] POCH, Dominik. *Demonstrační aplikace pro laboratoř techniky*. Diplomová práce. Fakulta aplikovaných věd, Západočeská univerzita, 2023.

[PA82] POSDAMER, J. L.; ALTSCHULER, M. D. Surface measurement by space-encoded projected beam systems. *Computer Graphics and Image Processing*. 1982, roč. 18, s. 1–17. Dostupné z DOI: 10.1016/0146-664x(82)90096-x.

[ras19] RASTLEKS ASSETS. *Realistic Cardboard Boxes (PBR, HQ)*. 2019. Dostupné také z: https://assetstore.unity.com/packages/3d/realistic-cardboard-boxes-pbr-hq-58749.

[Roa22] ROAD TURTLE GAMES. *Nebula Skyboxes*. 2022. Dostupné také z: https://assetstore.unity.com/packages/2d/textures-materials/sky/nebula-skyboxes-219924.

[rpg21] RPGWHITELOCK. *AllSky Free - 10 Sky / Skybox Set*. 2021. Dostupné také z: https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014.

[Seb20] SEBASTIANSOSNOWSKI. *Moon - Giordano Bruno Crater*. 2020. Dostupné také z: https://skfb.ly/6RrLu.

[She22] SHELDON, Robert. *What is virtual reality?* TechTarget, 2022-08. Dostupné také z: https://www.techtarget.com/whatis/definition/virtual-reality.

[SC19] SHERMAN, William R.; CRAIG, Alan B. *Understanding virtual reality: Interface, application, and design*. Cambridge, MA, USA: Morgan Kaufmann, 2019. ISBN 978-0-12-800965-9.

[Soc20] SOCIETY, Virtual Reality. *History Of Virtual Reality*. 2020-01. Dostupné také z: https://www.vrs.org.uk/virtual-reality/history.html.

[Sro22] SROKA, Paulina. *Low Poly City Assets*. 2022. Dostupné také z: https://skfb.ly/ozpFz.

[Tad+22] TADIC, Vladimir et al. Perspectives of Realsense and Zed depth sensors for Robotic Vision Applications. *Machines*. 2022, roč. 10. Dostupné z DOI: 10.3390/machines10030183.

[Uni22] UNITY. *OpenXR plugin*. Unity Technologies, 2022-12. Dostupné také z: https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.6/manual/index.html.

[Uni23a] UNITY. *Unity Documentation*. Unity Technologies, 2023-04. Dostupné také z: https://docs.unity3d.com/Manual/index.html.

[Uni23b] UNITY. *XR Interaction Toolkit plugin*. 2023-03. Dostupné také z: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/index.html.

[Vio19] VIOLET-BOOM & NOMNOMKING. *VR Controller LP*. 2019. Dostupné také z: `https://skfb.ly/6RPAQ`.

[VIV] VIVE. *Vive Pro 2 full kit specs*. Dostupné také z: `https://www.vive.com/us/product/vive-pro2-full-kit/specs/`.

[Xin17] XINREALITY. *Lighthouse, XinReality: Virtual Reality and Augmented Reality Wiki*. 2017-07. Dostupné také z: `https://xinreality.com/wiki/Lighthouse`.

[Yat16] YATES, Alan. *Alan Yates on the impossible task of making Valve's VR work*. HACKADAY, 2016-12. Dostupné také z: `https://www.youtube.com/watch?v=75ZytcYANTA`.

Seznam obrázků

Příloha A: Obsah ZIP souboru

Adresářová struktura:

- Text_prace – text práce ve formátu .pdf a .tex. Obsahuje podsložku Tex s LaTex projektem textu práce.
- Poster – poster k práci ve formátu .pub a .pdf.
- Aplikace_a_knihovny – vytvořené aplikace a jejich zdrojové kódy.
  - Build - vytvořené aplikace s konfiguračními soubory, soubor s návodem pro spuštění jednotlivých aplikací. Obsahuje aplikace Annoying Fly (ve složce Annoying_Fly), Box City (složka Box_City), Brush Export (složka Brush_Export), Depth Mesh (složka Depth_Mesh) a PaintVR (složka PaintVR).
  - Zdrojove_kody - zdrojové kódy vytvořených aplikací, vytvořené a použité knihovny, soubory s návodem pro sestavení jednotlivých aplikací. Obsahuje zdrojové kódy aplikací Annoying Fly (složka Annoying_Fly), Box City (složka Box_City), Brush Export (složka Brush_Export), Depth Mesh (složka Depth_Mesh), a PaintVR (složka PaintVR), dále zdrojové kódy knihoven Common (složka Common), Common.Unity (složka Common.Unity), a ZCU.PythonExecutionLibrary (složka PythonExecution) a zdrojové kódy serveru (složka Server).
- Vysledky
  - Dotazniky - uživatelské dotazníky s odpověďmi od studentů Gymnázia Sokolov.
  - Videa - instrukční videa k aplikacím.

Příloha B: Scénáře testů

Zde se nachází popis scénářů uživatelských testů, pomocí kterých byla ověřována funkctionalita aplikací.

Struktura zápisu testů je následující:

- Předpoklady - kroky, které musí být provedeny před tímto testem, nebo co musí být splněno, před tímto testem.
- Postup - popis postupu, jak test provést.
- Očekávaný výsledek - jaký je očekávaný výsledek daného testu.

Testy připojení k serveru

Testy, které se zaměřují na spolupráci daného klienta se serverem. Nezabývají se herní funkcionalitou.

Annoying Fly

Předpoklady:

Kamera RealSense je připojena k počítači.

Připojení k serveru

Postup:

1. Spustím aplikaci Annoying Fly.
2. Do kolonky *„URL serveru"* zadám URL serveru, ke kterému se chci připojit a do kolonky *„Jméno klienta"* jméno, pod kterým se připojím.
3. Stisknu tlačítko *„Hrát"*.
4. Počkám až se zobrazí herní obrazovka.

5. Spustím server.

Očekávaný výsledek:

Po kroku 4 vidím, se v levém dolním rohu herní obrazovky zobrazuje červená hláška „Odpojeno".

Po kroku 5 se místo ní zobrazí zelená hláška „Připojeno". Na server se po připojení odesílá objekt s bitmapou, obsahující snímek obrazovky. Tento objekt se obnovuje každou vteřinu. Jeho jméno odpovídá jménu klienta a má tvar „FlyKiller_<jmenoklienta>".

Připojení k serveru verze 2

Postup:

1. Spustím server.
2. Spustím aplikaci Annoying Fly
3. Do kolonky *„URL serveru"* zadám URL serveru, ke kterému se chci připojit a do kolonky *„Jméno klienta"* jméno, pod kterým se připojím.
4. Stisknu tlačítko *„Hrát"*.
5. Počkám až se zobrazí herní obrazovka.

Očekávaný výsledek:

V levém dolním rohu herní obrazovky se zobrazuje zelená hláška „Připojeno". Na server se po připojení odesílá objekt s bitmapou, obsahující snímek obrazovky. Tento objekt se obnovuje každou vteřinu. Jeho jméno odpovídá jménu klienta a má tvar „FlyKiller_<jmenoklienta>".

Obnovení připojení - dočasná nedostupnost

Předpoklady:

Server je zapnutý a aplikace Annoying Fly je spuštěná a k serveru je připojena.

Postup:

1. Vypnu server.
2. Chvíli počkám (několik vteřin), a sleduji, jak aplikace reaguje.
3. Zapnu server.
4. Chvíli počkám (několik vteřin), a sleduji, jak aplikace reaguje.

Očekávaný výsledek:

Po kroku 1 se zelená hláška „Připojeno" změní na žlutou hlášku „Připojování".

Po kroku 3 se hláška opět změní na zelenou hlášku „Připojeno". Na server se po připojení odesílá objekt s bitmapou, obsahující snímek obrazovky. Tento objekt se obnovuje každou vteřinu. Jeho jméno odpovídá jménu klienta a má tvar „FlyKiller_<jmenoklienta>".

Obnovení připojení - odpojení

Předpoklad:

Server je zapnutý a aplikace Annoying Fly je spuštěná a k serveru je připojena.

Postup:

1. Vypnu server.
2. Počkám delší dobu (cca minutu), než při předchozím testu, a sleduji, jak aplikace reaguje.
3. Zapnu server.
4. Chvíli počkám (několik vteřin), a sleduji, jak aplikace reaguje.

Očekávaný výsledek:

Po kroku 1 se zelená hláška „Připojeno" změní na žlutou hlášku „Připojování". Později se tato hláška změní na červenou hlášku „Odpojeno".

Po kroku 3 se hláška opět změní na zelenou hlášku „Připojeno". Na server se po připojení odesílá objekt s bitmapou, obsahující snímek obrazovky. Tento objekt se obnovuje každou vteřinu. Jeho jméno odpovídá jménu klienta a má tvar „FlyKiller_<jmenoklienta>".

Depth Mesh

Předpoklady všech testů:

Zapnutá aplikace Depth Mesh, spuštěný server.

Změna jména klienta

Postup:

1. Otevřít dialog pro změnu jména pomocí menu „Nastavení" -> „Jméno klienta".
2. Do otevřeného dialogového okna zadat nové jméno.

Očekávaný výsledek:

V levém dolním rohu se zobrazí hláška o úspěchu operace.

Připojení k serveru

Postup:

1. Připojit se k serveru pomocí menu „Server" -> „Připojit"

Očekávaný výsledek:

V levém dolním rohu se zobrazí hláška o úspěchu operace. Menu tlačítko pro změnu jména klienta nelze stisknout. Nelze změnit URL serveru. Menu „Připojit" se změní na „Odpojit" a je možné kliknout na menu „Poslat mesh", „Stáhnout mesh" a „Smazat mesh".

K serveru se připojil nový klient.

Poslání meshe

Předpoklady:

Úspěšně povedený test „Změna jména klienta", aplikace je připojena k serveru. Kamera RealSense je připojena a aktivní.

Postup:

1. Vytvořím mesh reprezetnující scénu pomocí tlačítka „Snapshot".
2. Mesh odešlu na server pomocí menu „Server" -> „Poslat mesh".

Očekávaný výsledek:

V levém dolním rohu aplikace Depth Mesh se zobrazí hláška o úspěchu operace. Na server je přijata mesh reprezentující scénu, její jméno odpovídá jménu klienta a je ve tvaru „DepthMesh_<jmenoklienta>".

Update meshe

Předpoklady:

Na serveru se nachází mesh zaslána tímto klientem. Aplikace je k serveru připojena.

Postup:

1. Změním snímanou scénu.
2. Vytvořím novou mesh reprezetnující scénu pomocí tlačítka „Snapshot".
3. Mesh odešlu na server pomocí menu „Server" -> „Odeslat mesh".

Očekávaný výsledek:

V levém dolním rohu aplikace Depth Mesh se zobrazí hláška o úspěchu operace. Na serveru je updatována mesh reprezentující scénu.

Přijmutí meshe

Předpoklady:

Na serveru se nachází mesh zaslána tímto klientem. Aplikace je k serveru připojena.

Postup:

1. Pomocí menu tlačítka „Server" -> „Stáhnout mesh" stáhnu mesh ze serveru.

Očekávaný výsledek:

V levém dolním rohu aplikace Depth Mesh se zobrazí hláška o úspěchu operace. V náhledu se zobrazí nově stažená mesh.

Smazání meshe

Předpoklady:

Na serveru se nachází mesh zaslána tímto klientem. Aplikace je k serveru připojena.

Postup:

1. Pomocí menu tlačítka „Server" -> „Smazat mesh" smažu mesh ze serveru.

Očekávaný výsledek:

V levém dolním rohu aplikace Depth Mesh se zobrazí hláška o úspěchu operace. Ze serveru je smazána mesh s názvem ve tvaru „DepthMesh_<jmenoklienta>".

Dočasná nedostupnost serveru

Předpoklady:

Aplikace je připojena k serveru.

Postup:

1. Vypnu server.
2. Chvíli počkám (pár vteřin), a sleduji, jak aplikace reaguje.
3. Zapnu server.
4. Chvíli počkám (pár vteřin), a sleduji, jak aplikace reaguje.

Očekávaný výsledek:

Po vypnutí serveru se v menu Server místo „Odpojit" zobrazí „Připojování". Nelze odeslat, přijmout nebo smazat mesh ze serveru.

Po opětovném zapnutí serveru se klient automaticky připojí. Místo „Připojování" se opět zobrazuje „Odpojit". Lze odeslat, přijmout nebo smazat mesh ze serveru.

Odpojení

Předpoklady:

Aplikace je připojena k serveru.

Postup:

1. Vypnu server.
2. Počkám (cca minutu), a sleduji, jak aplikace reaguje.

Očekávaný výsledek:

V levém dolním rohu se objeví hláška o odpojení. Po odpojení od serveru se změní v menu „Server" kolonka „Odpojit" na „Připojování" a později na „Připojit". URL serveru a jméno klienta bude opět možné upravit. Nebude možné odeslat, stáhnout nebo updatovat mesh.

Manuální odpojení

Předpoklady:

Aplikace je připojena k serveru.

Postup:

1. Pomocí menu „Server" -> „Odpojit" se odpojím od serveru.

Očekávaný výsledek:

V levém dolním rohu se zobrazí hláška o úspěchu operace. Od serveru se odpojil klient. URL serveru a jméno klienta bude opět možné upravit. Nebude možné odeslat, stáhnout nebo updatovat mesh.

Automatické zasílání updatů

Předpoklady:

Aplikace je připojena k serveru.

Postup:

1. Nastavím časový interval v comboboxu v pravém horním rohu aplikace.
2. Kliknutím na „AutoSend: OFF" v pravém horním rohu zapnu automatické odesílání meshe na server.
3. Počkám, a sleduji, jak aplikace reaguje.

Očekávaný výsledek:

„AutoSend: OFF" se změní na „AutoSend: ON". Podle nastavené hodnoty v comboboxu se jednou za zadaný počet vteřin vytvoří nový snapshot scény a odešle se na server.

Box City

Připojení k serveru

Postup:

1. Spustím aplikaci Box City.
2. Do kolonky „URL Serveru" zadám URL serveru, ke kterému se chci připojit, do kolonky „Jméno klienta" zadám jméno pod kterým se připojím.
3. Stisknu tlačítko „Hrát".
4. Nasadám si headset a rozhlédnu se po virtuálním světě.
5. Pustím server.
6. Nasadím si headset a rozhlédnu se po virtálním světě.

Očekávaný výsledek:

Po kroku 3 jsou kontrolery zobrazené ve virtuálním světě červené. Kontrolery a headset se pohybují s reálnými kontrolery a headsetem. Na instrukční obrazovce umístěné ve scéně se v dolní části zobrazuje červený nápis „Odpojeno". Je možné pohybovat se po scéně pomocí teleportu.

Po kroku 5 se dole na instrukční obrazovce změní nápis na „Připojeno". Kontrolery zobrazené ve scéně jsou zelené. Je možné pohybovat se po scéně pomocí teleportu. Ve scéně se objeví krabice. K serveru se připojil nový klient a z aplikace Box City byly přijaty krabice, kontrolery a headset. Server začne přijímat aktualizace pozic kontrolerů a headsetu.

Připojení k serveru verze 2

Postup:

1. Pustím server.
2. Spustím aplikaci Box City.
3. Do kolonky „URL Serveru" zadám URL serveru, ke kterému se chci připojit, do kolonky „Jméno klienta" zadám jméno pod kterým se připojím.
4. Stisknu tlačítko „Hrát".
5. Nasadám si headset a rozhlédnu se po virtuálním světě.

Očekávaný výsledek:

Dole na instrukční obrazovce se změní nápis na „Připojeno". Kontrolery zobrazené ve scéně jsou zelené. Je možné pohybovat se po scéně pomocí teleportu. Ve scéně se objeví krabice. K serveru se připojil nový klient a z aplikace Box City byly přijaty krabice, kontrolery a headset. Server začne přijímat aktualizace pozic kontrolerů a headsetu.

Reconnect

Předpoklady:

Server je spuštěný a aplikace je k němu připojena.

Postup:

1. Vypnu server.
2. Chvíli počkám (pár vteřin) a sleduji jak reaguje klient.
3. Zapnu server.
4. Chvíli počkám (pár vteřin) a sleduji jak reaguje klient.

Očekávaný výsledek:

Po vypnutí serveru kontrolery změní barvu na červenou. Na instrukční obrazovce se zobrazí hláška „Připojování".

Po zapnutí serveru kontrolery změní barvu na zelenou. Na instrukční obrazovce se opět zobrazí hláška „Připojeno". Krabice ve scéně se vrátí do své defaultní polohy. Je možné pohybovat se po scéně pomocí teleportu. K serveru se připojil nový klient a z aplikace Box City byly přijaty krabice, kontrolery a headset. Server začne přijímat aktualizace pozic kontrolerů a headsetu.

Disconnect

Předpoklady: server běží, klient je připojen

Postup:

1. Vypnu server.
2. Počkám delší dobu než při předchozím testu (cca minutu), a sleduji jak reaguje klient.
3. Zapnu server.
4. Chvíli počkám (pár vteřin) a sleduji jak reaguje klient.

Očekávaný výsledek:

Po vypnutí serveru kontrolery změní barvu na červenou. Na instrukční obrazovce se zobrazí hláška „Připojování" a poté na „Odpojeno". Při změně na „Odpojeno" ze scény zmizí krabice.

Po zapnutí serveru kontrolery změní barvu na zelenou. Na instrukční obrazovce se opět zobrazí hláška „Připojeno". Krabice ve scéně se znovu objeví na své defaultní poloze. Je možné pohybovat se po scéně pomocí teleportu. K serveru se připojil nový klient a z aplikace Box City byly přijaty krabice, kontrolery a headset. Server začne přijímat aktualizace pozic kontrolerů a headsetu.

Přijímání objektů ze serveru

Předpoklady:

Aplikace je připojena k serveru. K serveru jsou připojeny i aplikace PaintVR a Annoying Fly.

Postup:

1. Aplikace PaintVR a Annoying fly zašlou na server objekty:
    a) Annoying Fly - posílá bitmapu, pravidelně zasílá její aktualizace,
    b) PaintVR - posílá objekty typu Mesh reprezentující tah štětcem.

Očekávaný výsledek:

Pokud hráč v PaintVR nakreslí nový tah štětcem, tento tah se zobrazí i ve scéně Box City. Ve scéně je dále vidět bitmapa z aplikace Annoying Fly a pravidelně se updatuje v závislosti na tom, co použitá hloubková kamera snímá. Dále se ve scéně se pohybují kontrolery a headset druhého VR hráče.

Reset lokálních objektů po reconnect

Předpoklady:

Byl úspěně proveden test „Přijímání objektů ze serveru". Aplikace je stále připojena k serveru.

Postup:

1. Pohnu krabicemi ve scéně.
2. Vypnu server.
3. Pozoruji jak bude klient reagovat.
4. Zapnou server.

5. Pozoruji jak bude klient reagovat.

Očekávaný výsledek:

Po kroku 2 se zobrazí hláška „Připojování" na informační tabuli ve scéně.

Po kroku 4 se zobrazí hláška „Připojeno" na informační tabuli ve scéně. Po opětovném připojení k serveru byly smazány objekty z ostatních klientů, které byly ve scéně. Dále se krabice vrátily do své defaultní polohy, a znovu se zaslaly na server. Ve scéně se po celou dobu nachází objekty reprezentující kontrolery a headset.

Reset lokálních objektů po disconnect

Byl úspěně proveden test „Přijímání objektů ze serveru". Aplikace je stále připojena k serveru.

Postup:

1. Pohnu krabicemi ve scéně.
2. Vypnu server.
3. Chvíli počkám (cca minutu) a pozoruji jak bude klient reagovat.
4. Zapnou server.
5. Pozoruji jak bude klient reagovat.

Očekávaný výsledek:

Po kroku 2 se zobrazí hláška „Připojování", a poté „Odpojeno" na informační tabuli ve scéně. Ze scény zmizely všechny objekty, které sou uchovávány na serveru.

Po kroku 4 se zobrazí hláška „Připojeno" na informační tabuli ve scéně. Krabice se znovu objeví na své defaultní poloze, a znovu se zašlou na server. Ve scéně se po celou dobu nachází objekty reprezentující kontrolery a headset.

PaintVR

Připojení k serveru

Postup:

1. Spustím aplikaci PaintVR.
2. Do kolonky „URL Serveru" zadám URL serveru, ke kterému se chci připojit, do kolonky „Jméno klienta" zadám jméno pod kterým se připojím.
3. Stisknu tlačítko „Hrát".

4. Nasadám si headset a rozhlédnu se po virtuálním světě.
5. Pustím server.
6. Nasadím si headset a rozhlédnu se po virtálním světě.

Očekávaný výsledek:

Po kroku 3 jsou kontrolery zobrazené ve virtuálním světě červené. Kontrolery a headset se pohybují s reálnými kontrolery a headsetem. Na paletě na levém kontroleru se dole zobrazuje červený nápis „Odpojeno". Je možné pohybovat se po scéně pomocí teleportu. V aplikaci nyní nelze kreslit.

Po kroku 5 se dole na paletě změní nápis na „Připojeno". Kontrolery zobrazené ve scéně jsou zelené. Je možné pohybovat se po scéně pomocí teleportu. K serveru se připojil nový klient a z aplikace PaintVR byly přijaty kontrolery a headset. Jména těchto objektů odpovídají jménu klienta. Server začne přijímat aktualizace pozic kontrolerů a headsetu. V aplikaci lze kreslit.

Připojení k serveru verze 2

Postup:

1. Pustím server.
2. Spustím aplikaci PaintVR.
3. Do kolonky „URL Serveru" zadám URL serveru, ke kterému se chci připojit, do kolonky „Jméno klienta" zadám jméno pod kterým se připojím.
4. Stisknu tlačítko „Hrát".
5. Nasadím si headset a rozhlédnu se po virtuálním světě.

Očekávaný výsledek:

Dole na paletě na levém kontroleru se změní nápis na zelené „Připojeno". Kontrolery zobrazené ve scéně jsou také zelené. Je možné pohybovat se po scéně pomocí teleportu. V aplikaci lze kreslit. K serveru se připojil nový klient a z aplikace PaintVR byly přijaty kontrolery a headset. Server začne přijímat aktualizace pozic kontrolerů a headsetu.

Reconnect

Předpoklady:

Server je spuštěný a aplikace je k němu připojena.

Postup:

1. Vypnu server.
2. Chvíli počkám (pár vteřin) a sleduji jak reaguje klient.
3. Zapnu server.
4. Chvíli počkám (pár vteřin) a sleduji jak reaguje klient.

Očekávaný výsledek:

Po vypnutí serveru kontrolery změní barvu na červenou. Na paletě na levém kontroleru se zobrazí hláška „Připojování". Nyní v aplikaci nelze kreslit.

Po zapnutí serveru kontrolery změní barvu na zelenou. Na instrukční obrazovce se opět zobrazí hláška „Připojeno". Pokud se ve scéně nacházely nějaké tahy štětcem, byly smazány. Je možné pohybovat se po scéně pomocí teleportu. V aplikaci lze kreslit. K serveru se připojil nový klient a z aplikace PaintVR byly přijaty kontrolery a headset. Server začne přijímat aktualizace pozic kontrolerů a headsetu.

Disconnect

Předpoklady: server běží, klient je připojen
Postup:

1. Vypnu server.
2. Počkám delší dobu než při předchozím testu (cca minutu), a sleduji jak reaguje klient.
3. Zapnu server.
4. Chvíli počkám (pár vteřin) a sleduji jak reaguje klient.

Očekávaný výsledek:

Po vypnutí serveru kontrolery změní barvu na červenou. Na paletě na levém kontroleru se zobrazí hláška „Připojování" a poté na „Odpojeno". V aplikaci nyní nelze kreslit. Pokud se ve scéně nacházely nějaké tahy štětcem, byly smazány.

Po zapnutí serveru kontrolery změní barvu na zelenou. Na instrukční obrazovce se opět zobrazí zelená hláška „Připojeno". Je možné pohybovat se po scéně pomocí teleportu. V aplikaci lze kreslit K serveru se připojil nový klient a z aplikace PaintVR byly přijaty kontrolery a headset. Server začne přijímat aktualizace pozic kontrolerů a headsetu.

Přijímání objektů ze serveru

Předpoklady:

Aplikace je připojena k serveru. K serveru jsou připojeny i aplikace Box City a Annoying Fly.

Postup:

1. Aplikace PaintVR a Annoying fly zašlou na server objekty -
    a) Annoying Fly - posílá bitmapu, pravidelně zasílá její aktualizace,
    b) Box City - posílá objekty typu Mesh reprezentující krabice, jejich pozice se mohou aktualizovat.

Očekávaný výsledek:

Pokud hráč v Box City pohne s krabicí, je tento pohyb sledovatelný v reálném čase i v aplikaci PaintVR. Ve scéně je dále vidět bitmapa z aplikace Annoying Fly, která se pravidelně updatuje v závislosti na tom, co použitá hloubková kamera snímá. Dále se ve scéně se pohybují kontrolery a headset druhého VR hráče.

Testy funkctionalit aplikací

Testy, které se zaměřují na herní funkcionalitu aplikací.

Předpoklad pro všechny následující testy:

Server je spuštěný a jsou provedeny testy připojení k serveru.

Annoying Fly

Snímání hloubky

Postup:

1. Připojím kameru k počítači.
2. Spustím aplikaci, zadám jméno klienta a URL serveru a stisknu tlačítko „Hrát".
3. Nastavím minimální a maximální vzdálenost tak, aby kamera zabírala jen mě.
4. Pokusím se zabít mouchu zakrytím jejího obrázku pomocí zobrazovaného stínu.

Očekávaný výsledek:

Zobrazí se herní obrazovka (viz. obrázek 5.3), v pozadí se zobrazuje „stín" snímané scény. Při každém zakrytí mouchy se zvýší počítadlo bodů v levém horním rohu o jedna a moucha změní svou pozici v obrázku.

Připojení kamery

Postup:

1. Spustím aplikaci, zadám jméno klienta a URL serveru a stisknu tlačítko „Hrát".
2. Počkám dokud se nenačte herní obrazovka.
3. Připojím kameru k počítači.
4. Dvakrát stisknu tlačítko na resetování kamery (v pravém horním rohu ovládacího panelu po levé straně obrazovky).
5. Pokusím se zabít mouchu zakrytím jejího obrázku pomocí zobrazovaného stínu.

Očekávaný výsledek:

Zobrazí se herní obrazovka (viz. obrázek 5.3), v pozadí se zobrazuje „stín" snímané scény. Načtení herní scény po spuštění trvá delší dobu, protože se knihovna *RealSense SDK 2.0* snaží nalézt aktivní kameru. Při každém zakrytí mouchy se zvýší počítadlo bodů v levém horním rohu o jedna a moucha změní svou pozici v obrázku.

Posun a přiblížení obrázku

Postup:

1. Měním parametry horizontálního a vertiákálního posunu a přiblížení. Zkouším
   - desetinná čísla,
   - záporná desetinná čísla,
   - celá čísla,
   - záporná celá čísla
2. Po každé změně parametru vždy stisknu odpovídající tlačítko „Set".

Očekávaný výsledek:

Kladné číslo v horizontálním posunu obrázku odpovídá posunu doprava na obrazovce. Záporné posunu doleva. Kladné číslo vertikálního posunu odpovídá posunu nahoru a záporné posunu dolů. Desetinná čísla nejsou akceptována, a po stisknutí odpovídajícího tlačítka „Set", se obsah kolonky mění na poslední validní hodnotu.

Pro přiblížení platí, že desetinné číslo větší než jedna odpovídá zvětšení obrázku, menší než jedna odpovídá zmenšení. Záporné číslo není akceptováno, a po stisknutí odpovídajícího tlačítka „Set", se obsah kolonky mění na poslední validní hodnotu.

Hra při transformovaném obrázku

Předpoklady:

Úspěšně provedený test „Posun a přiblížení obrázku".

Postup:

1. Měním parametry horizontálního a vertiákálního posunu a přiblížení.
2. Pokusím se zabít mouchu zakrytím jejího obrázku pomocí zobrazovaného stínu.

Očekávaný výsledek:

Při každém zakrytí mouchy se zvýší počítadlo bodů v levém horním rohu o jedna a moucha změní svou pozici v obrázku. Toto funguje vždy, nehledě na posun nebo přiblížení obrázku.

Změna barevného módu

Postup:

1. Zaškrtnu „Zobrazit barvy".
2. Pokusím se zabít mouchu zakrytím jejího obrázku pomocí zobrazovaného stínu.

Očekávaný výsledek:

Stín změní barvu, nyní je obarvený barevným gradientem. Pixely mění barvu v závislosti na tom, jak daleko od kamery se daná část scény nachází.

Při každém zakrytí mouchy se zvýší počítadlo bodů v levém horním rohu o jedna a moucha změní svou pozici v obrázku.

Zrcadlení obrázku

Postup:

1. Zaškrtnu „Zrcadlit".
2. Pokusím se zabít mouchu zakrytím jejího obrázku pomocí zobrazovaného stínu.

Očekávaný výsledek:

Stín je zobrazován zrcadlově obrácený. Při každém zakrytí mouchy se zvýší počítadlo bodů v levém horním rohu o jedna a moucha změní svou pozici v obrázku.

Depth Mesh

Automatické připojení kamery

Postup:

1. Připojím kameru RealSense k počítači.
2. Spustím aplikaci Depth Mesh.

Očekávaný výsledek:

V náhledu hloubkového obrazu v levé horní části okna aplikace se zobrazí barevný náhled hloubkového obrazu. Barvy pixelů odpovídají vzdálenosti dané části scény od kamery.

Manuální připojení kamery

Postup:

1. Spustím aplikaci Depth Mesh.
2. Připojím kameru RealSense k počítači.
3. Pomocí menu „Složka" -> „Připojit kameru" v aplikaci aktivuji kameru.

Očekávaný výsledek:

V náhledu hloubkového obrazu v levé horní části okna aplikace se zobrazí barevný náhled hloubkového obrazu. Barvy pixelů odpovídají vzdálenosti dané části scény od kamery.

Vytvoření snapshotu scény

Předpoklady:

Kamera je připojena k aplikaci.

Postup:

1. Stisknu tlačítko „Snapshot".

Očekávaný výsledek:

Ve 3D náhledu v pravé části aplikace se zobrazí vytvořená mesh, reprezentující snímek scény. Souřadnice z odpovídá hloubce snímané části scény.

Manipulace s náhledem

Předpoklady:

Ve 3D náhledu se nachází mesh reprezentující snímek scény.

Postup:

1. Pomocí myši zkusím hýbat s 3D náhledem meshe.
2. Po levé straně aplikace změním toggle „Zobrazit vrcholy".

Očekávaný výsledek:

Pomocí myši dokážu rotovat s modelem a přibližovat/oddalovat jej. Pokud je aktivní toggle „Zobrazit vrcholy" jsou vrcholy v moedelu zvýrazněny jako červené tečky.

Filtrování trojúhelníků

Předpoklady:

Ve 3D náhledu se nachází mesh reprezentující snímek scény.

Postup:

1. Změním hodnotu slideru „Prahování trojúhelníků".
2. Sleduji jak se mění mesh v náhledu v závislosti na měnící se hodnotě.

Očekávaný výsledek:

Slider ovlivňuje odfiltrování trojúhelníků, které mají některou ze stran delší než zadaná hodnota. Při zvyšující se hodnotě se tedy v meshi objevují nové trojúhelníky a při snižování hodnoty dlouhé trojúhelníky mizí.

Filtry hloubkového obrazu

Předpoklady:

Kamera je připojena k aplikaci.

Postup:

1. Po levé straně aplikace se nacházejí filtry aplikované na hloubkový obraz. Postupně zkouším vypínat a zapínat jednotlivé filtry. Jedná se o:
    - Decimační filtr
    - Oříznout
    - Záplatování děr
    - Prostorové vyhlazování

- Časové vyhlazování

2. Pozoruji jak se mění náhled hloubkového obrazu.

Očekávaný výsledek:

Náhled na hloubkový obraz se mění když se změní nastavení filtrů. Může se stát, že některé změny nebudou rovnou viditelné, proto je nutné provést ještě test pokročilých možností filtrů.

Decimační filtr ovlivňuje počet bodů ve vzniklém snapshotu scény, oříznutí ořezává části scény mimo zadaný rozsah. Záplatování děr se snaží „zašít" díry ve hloubkovém snímku. Časové a prostorové vyhlazování se snaží odstranit šum z hloubkového obrazu.

Pokročilé možnosti filtrů hloubkového obrazu

Předpoklady:

Kamera je připojena k aplikaci.

Postup:

1. Stisknutím tlačítka „Pokročilé nastavení" otevřu okno s pokročilým nastavením filtrů.

2. Postupně zkouším měnit jednotlivé parametry filtrů. Jedná se o filtry:

   - Decimační filtr
   - Oříznout
   - Vyhlazování v rozdílové doméně
   - Prostorové vyhlazování
   - Časové vyhlazování
   - Záplatování děr

3. Pozoruji, jak se mění náhled hloubkového obrazu.

Očekávaný výsledek:

Náhled na hloubkový obraz se mění když se změní nastavení parametrů jednotlivých filtrů.

Nastavení cesty k Python.dll

Postup:

1. Stisknutím menu „Python" -> „Cesta k dll" otevřu dialogové okno pro výběr dll souboru.
2. Vyberu nainstalované python.dll.

Očekávaný výsledek:
Vlevo dole se oběví hláška o úspěchu operace.

Uživatelský Python kód

Předpoklady:
Úspěšně provedený test „Nastavení cesty k python.dll".
Postup:

1. V kódu vytvořím dva trojúhelníky:
   - Souřadnice bodů - $points = [-0.2, -0.2, -0.2, 0.2, -0.2, -0.2, -0.2, 0.2, -0.2, 0.2, 0.2, -0.2]$
   - Trojúhelníky - $faces = [0, 3, 1, 0, 2, 3]$
   - UV souřadnice - $uvs = [0, 0, 1, 0, 0, 1, 1, 1]$
2. Tyto nově vytvořené seznamy vracím jako návratovou hodnotu vytvářené funkce.
3. Vykonám kód pomocí tlačítka „Provést".
4. Nastavím hodnotu slideru „Prahování trojúhelníků" na nejvyšší možnou.

Očekávaný výsledek:
Ve scéně se vytvoří čtverec, který má na sobě jako texturu namapovaný snímek scény.

Export .ply souboru

Předpoklady:
Ve 3D náhledu se nachází mesh reprezentující snímek scény.
Postup:

1. Pomocí menu „Složka" -> „Uložit PLY" otevřu dialog pro uložení .ply souboru.
2. Uložím .ply na disk.

Očekávaný výsledek:

Ve vybrané složce se nachází vytvořený .ply soubor. Po otevření vidím, že jeho obsah vizuálně odpovídá náhledu meshe v aplikaci Depth Mesh.

Box City

Posun krabice

Postup:

1. Ve virtuální realitě pomocí spouště pravého kontroleru uchopím krabici.
2. Pohnu s kontrolerem a přesunu krabici na jinou část scény.
3. Během pohybu se teleportuji na jiné místo scény.

Očekávaný výsledek:

Krabice se pohybuje spolu s kontrolerem, dokud je držena spoušť. Při teleportaci se krabice teleportuje spolu s kontrolerem.

Minihra

Postup:

1. Postupně nanosím krabice automechanikovi.
2. Krabice umisťuji jedna po druhé do vyhrazeného prostoru.
3. Krabice zkusím z prostoru i vyndavat a vracet.
4. Postupně nanosím do vyhrazeného prostoru požadovaný počet krabic.

Očekávaný výsledek:

Při umístění krabice do vyhrazeného prostoru se zvyšuje počítadlo umístěné vedle něj. Pokud je krabice z prostoru vyjmuta, počitadlo se sníží. Při dosažení cílového počtu krabic se zobrazí vítězná hláška.

Test uložení scény

Postup:

1. Spustím aplikaci Annoying Fly a počkám, až se připojí k serveru.
2. V aplikaci Box City pohnu krabicemi a umístím je na takové místo, abych snadno poznal zda se pohybují nebo ne.

3. Na ovládacím panelu pomocí tlačítka „Scény” otevřu okno pro uložení scén.
4. Uložím scénu na libovolnou pozici.
5. Pomocí tlačítka „Resetovat krabice” resetuji polohu krabic.
6. Načtu scénu, kterou jsem uložil v kroku 4.

Očekávaný výsledek:
Po kroku 6 se krabice a bitmapa herního okna aplikace Annoying Fly pohnou zpátky na pozice v jakých jsem scénu ve kroku 4 uložil.

Brush Export

Pro všechny následující testy platí předpoklady:

- spuštěná aplikace Brush Export,
- na počítači je nainstalován Python.

Změna jména

Postup:

1. Do kolonky "Jméno štětce"zadám následující řetězce
    a) NášŠtětec?
    b) NasStetec1

Očekávaný výsledek:
Pro vstup a) se filtrují nepovolené charaktery, v kolonce zůstane napsáno "Nttec". Pro vstup b) zůstane v kolonce "NasStetec1".

Nastavení cesty k Python.dll

Předpoklady:
Aktuální nastavená cesta k python.dll neodkazuje na žádné python.dll na disku.
Postup:

1. Do pole pro uživatelský kód nakopíruju jeden ze vzorových skriptů.
2. Stisknu tlačítko „Provést”.
3. Stisknu tlačítko „?” v pravém horním rohu k zobrazení návodu.
4. Stisknu tlačítko „Procházet...” v pravém dolním rohu.

5. V otevřeném okně zvolím python.dll.
6. Stisknu tlačítko „Provést".
7. Návod opět zavřu stisknutím tlačítka „x" v pravém horním rohu.
8. Stisknu tlačítko „Provést".

Očekávaný výsledek:

Po kroku 2 se zobrazí chybová hláška, která oznamuje, že není nastavené validní dll. Po kroku 4 se zobrazí okno ve kterém lze navigovat přes soubory na disku a zvolit .dll. Po kroku 8 se změní náhled na štětec v závislosti na tom, jaký kód byl do programu nakopírován. Dále v návodu už nelze zvolit jinou cestu k python.dll.

Uživatelský kód

Předpoklady:

Úspěšně provedený test „Nastavení cesty k Python.dll".

Postup:

1. Zkusím změnit barvu v kódu. Barva je ve tvaru [r, g, b]. Pro jednotlivé kanály zkouším
   - záporné hodnoty,
   - čísla větší než 1.
   - Zkusím [r, g, b] = [1, 0, 0]
2. Po každé změně stisknu tlačítko „Provést".
3. Zkusím změnit velikost štětce v kódu na jinou než aktuální. Zkouším
   - desetinná čísla,
   - kladná celá čísla,
   - kladná záporná čísla.
4. Po každé změně stisknu tlačítko „Provést".
5. Zkusím změnit pole widthModifier v kódu. Zkouším
   - Záporné hodnoty,
   - hodnoty větší než 1,
   - kladná desetinná čísla.
6. Po každé změně stisknu tlačítko „Provést".

Očekávaný výsledek:

Hodnoty barevných kanálů mohou být z intervalu <0, 1>. Pro jiné hodnoty se provede oříznutí. Po provedení kroku 2 se barva štětce nakonec změní na červenou.

Zadávaná velikost štětce může mít hodnoty z intervalu <1, 100>. Pro jiné hodnoty se provede oříznutí. Po kroku 4 se změní velikost štětce. Změna odpovídá tomu, jaká byla velikost předtím. Pokud tedy zadáme menší číslo, štětec bude užší a naopak.

Hodnoty v poli widthModifier mohou mít hodnoty z intervalu <0, 1>. Pro jiné hodnoty se provede oříznutí. Po kroku 6 se změní velikost štětce v určité části tahu. Změna odpovídá tomu, jaká byla daná hodnota v poli předtím. Pokud tedy zadáme menší číslo, štětec bude užší a naopak.

Export do složky

Postup:

1. Do pole pro uživatelský kód nakopíruju jeden ze vzorových skriptů.
2. Stisknu tlačítko „Provést".
3. Stisknu tlačítko „Prohlížet" u kolonky „Exportovat štětec".
4. V otevřeném okně zvolím složku, do které se má exportovat vzniklý štětec.
5. Stisknu tlačítko „Exportovat".

Očekávaný výsledek:

Po zvolení složky se ukáže vybraná cesta v kolonce „Exportovat štětec".

Po exportování se ve vybrané složce vytvoří soubor s daty štětcem .xml. Soubor je pojmenován podle jména štětce v kolonce „Jméno štětce". Uvnitř souboru se nachází šířka (vynásobena konstantou 0.01), barva, pole modifikující aktuální šířku, a parametr timestep štětce. Jejich hodnoty odpovídají tomu, co bylo nastaveno v uživatelském kódu.

Dále se v této složce nachází .png soubor s texturou štětce. Soubor je pojmenován podle jména štětce v kolonce „Jméno štětce".

Export bez textury

Postup:

1. Do pole pro uživatelský kód nakopíruju jeden ze vzorových skriptů.
2. Upravím kód tak, aby nevracel texturu a její velikost.

3. Stisknu tlačítko „Provést".
4. Stisknu tlačítko „Prohlížet" u kolonky „Exportovat štětec".
5. V otevřeném okně zvolím složku, do které se má exportovat vzniklý štětec.
6. Stisknu tlačítko „Exportovat".

Očekávaný výsledek:

Po zvolení složky se ukáže vybraná cesta v kolonce „Exportovat štětec".

Po exportování se ve vybrané složce vytvoří soubor s daty štětcem .xml. Soubor je pojmenován podle jména štětce v kolonce „Jméno štětce". Uvnitř souboru se nachází šířka (vynásobena konstantou 0.01), barva, pole modifikující aktuální šířku, a parametr timestep štětce. Jejich hodnoty odpovídají tomu, co bylo nastaveno v uživatelském kódu.

PaintVR

Pro všechny následující testy platí předpoklady:

- spuštěná aplikace PaintVR (kromě testu „Import štětce"),
- spuštěný server,
- aplikace PaintVR je připojena k serveru.

Import štětce

Postup:

1. Do složky „Brushes" nakopíruji soubory štětce, který chci importovat. Vyzkouším
   - štětec s texutrou,
   - štětec bez textury.
2. Spustím aplikaci PaintVR a připojím se k serveru.
3. Zkusím nakreslit tah nově importovaným štětcem.

Očekávaný výsledek:

V nabídce štětců se nachází štětec, který měl být importován. Tah štětcem vzhledově odpovídá náhledu z aplikace Brush Export.

Import štětce za běhu

Postup:

1. Do složky „Brushes" nakopíruji soubory štětce, který chci importovat. Vyzkouším
    - štětec s texutrou,
    - štětec bez textury.
2. Importuji štětce pomocí tlačítka „Importovat štětce" na paletě.
3. Zkusím nakreslit tah nově importovaným štětcem.

Očekávaný výsledek:
V nabídce štětců se nachází štětec, který měl být importován. Tah štětcem vzhledově odpovídá náhledu z aplikace Brush Export.

Tah štětcem

Postup:

1. Stisknutím spouště pravého kontroleru, začnu kreslit.
2. Za kreslení zkusím přepnout mezi štětci, a mezi štětci a gumou.

Očekávaný výsledek:
Tah štětcem začne se stiskem spouště. Po dobu držení dochází ke kreslení, po uvolnění spouště tah štětcem se ukončí. Při kreslení nejde přepínat mezi štětci nebo mezi štětci a gumou.

Změna parametrů štětce

Postup:

1. Na paletě zkusím změnit barvu štětce.
2. Nakreslím tah ve scéně.
3. Na paletě zkusím změnit velikost štětce
4. Nakreslím tah ve scéně.

Očekávaný výsledek:
Po změně barvy se změní barva kresleného tahu. Stejně tak se po změně velikosti změní velikost kresleného tahu.

Mazání tahů štětcem

Postup:

1. Nakreslím tah libovolným štětcem.
2. Přepnu ze štětců na gumu.
3. Stisknutím spouště pravého kontroleru, když objekt reprezentující gumu koliduje s tahem tah smažu.

Očekávaný výsledek:
Po smazání tahu daný objekt zmizí ze scény ve virtuální realitě a je smazán ze serveru.

Test uložení scény

Postup:

1. Spustím aplikaci Box City a počkám, až se připojí k serveru.
2. V aplikaci PaintVR nakreslím několik tahů štětcem.
3. Na paletě pomocí tlačítka „Scény" otevřu okno pro uložení scén.
4. Uložím scénu na libovolnou pozici.
5. Některé tahy ve scéně štětcem smažu a nakreslím nějaké nové.
6. Načtu scénu, kterou jsem uložil v kroku 4.
7. Zkusím nakreslit nový tah štětcem.

Očekávaný výsledek:
Po kroku 6 se scéna obnoví do stavu v jakém byla v kroku 4. Po nakreslení nového tahu štětcem se tento tah odesílá na server a je vidět v aplikaci Box City.

Příloha C: Ukázkové Python skripty

V této příloze se nacházejí ukázkové skripty, které je možné vykonat v aplikacích druhého stupně. Pro virtuální realitu se jedná o aplikaci Brush Export, sloužící pro vytváření uživatelských skriptů. Pro hloubkovou kameru se jedná o aplikaci Depth Mesh, která umožňuje manipulaci s vytvořenou trojúhelníkovou sítí reprezentující hloubkový obraz snímané scény.

Brush Export

Zde bude popsáno několik příkladů skriptů, které mohou být použity pro vygenerování uživatelského štětce v aplikaci Brush Export.

Gradient zhora dolů

Tento skript generuje štětec, který má bílou barvu a jeho textura je zelenomodrý gradient měnící se shora dolů. Šířka štětce je nastavena na hodnotu 100 a z hodnot proměnných *widthmod* a *time* lze odvodit, že když se se štětcem začne kreslit, jeho šířka bude mít hodnotu 0.1×100, tedy 10, v první vteřině jeho šířka bude mít hodnotu 0.3×100, v druhé 1×100, ve třetí 0.7×100 a ve čtvrté opět hodnotu 0.1×100. Vytvořená textura bude mít rozměry 6 pixelů na šířku a 10 na výšku. Náhled na vzniklý štětec je na obrázku 1.

Obrázek 1: Náhled na štětec vytvořený skriptem gradient zhora dolů

```
color = [1, 1, 1]
width = 100
widthmod = [0.1, 0.3, 1, 0.7, 0.1]
time = 4

```

```
texsize = [6, 10]
tex = []

updates = 0
for h in range(0, 10):
  clr = [updates / 255.0, 0.8, 1 - updates / 255.0, 1]
  for w in range(0, 6):
    tex.append(clr)
  updates = updates + 20
  updates = updates % 255

return [color, width, widthmod, time, texsize, tex]
```

Gradient zleva doprava

Tento skript generuje štětec, který má bílou barvu a jeho textura je zelenomodrý gradient měnící se zleva doprava. Šířka štětce je nastavena na hodnotu 100, a z hodnot proměnných *widthmod* a *time* lze odvodit, že když se se štětcem začne kreslit, jeho šířka bude mít hodnotu 0.1×100, tedy 10, v první čtvrtině vteřiny jeho šířka bude mít hodnotu 0.3×100, v druhé čtvrtině vteřiny 1×100, ve třetí čtvrtině 0.7×100 a v jednu vteřinu opět hodnotu 0.1×100. Vytvořená textura bude mít rozměry 6 pixelů na šířku a 10 na výšku. Náhled na vzniklý štětec je na obrázku 2.

Obrázek 2: Náhled na štětec vytvořený skriptem gradient zleva doprava

```
color = [1, 1, 1]
width = 100
widthmod = [0.1, 0.3, 1, 0.7, 0.1]
time = 1

texsize = [6, 10]
tex = []

for h in range(0, 10):
  updates = 0
  for w in range(0, 6):
    clr = [updates / 255.0, 0.8, 1 - updates / 255.0, 1]
    tex.append(clr)
    updates = updates + 40
    updates = updates % 255

return [color, width, widthmod, time, texsize, tex]
```

Čára

Tento skript generuje štětec, který má růžovou barvu a jeho textura je čára vedoucí z levého dolního rohu do pravého horního. Šířka štětce je nastavena na hodnotu 100, a z hodnot proměnných *widthmod* a *time* lze odvodit, že když se se štětcem začne kreslit, jeho šířka bude mít hodnotu 0.1×100, tedy 10, v první vteřině jeho šířka bude mít hodnotu 0.3×100, v druhé 1×100, ve třetí 0.7×100 a ve čtvrté opět hodnotu 0.1×100. Vytvořená textura bude mít rozměry 10 pixelů na šířku a 10 na výšku. Náhled na vzniklý štětec je na obrázku 3.

Obrázek 3: Náhled na štětec vytvořený skriptem čára

```
color = [1, 0.3, 1]
width = 100
widthmod = [0.1, 0.3, 1, 0.7, 0.1]
time = 4

texsize = [10, 10]
tex = []

for i in range(0, 10):
  for j in range (0, 10):
    col = [1, 1, 1]
    if (i==j):
      col = [0, 0, 0]
    tex.append(col)

return [color, width, widthmod, time, texsize, tex]
```

Velké V

Tento skript generuje štětec, který má fialovou barvu a jeho textura je lomená čára připomínající písmeno V. Šířka štětce je nastavena na hodnotu 100, a z hodnot proměnných *widthmod* a *time* lze odvodit, že když se se štětcem začne kreslit, jeho šířka bude mít hodnotu 0.5×100, tedy 50, v druhé vteřině jeho šířka bude mít hodnotu 1×100, a ve třetí opět hodnotu 0.5×100. Vytvořená textura bude mít rozměry 30 pixelů na šířku a 16 na výšku. Náhled na vzniklý štětec je na obrázku 4.

```
color = [0.6, 0.3, 1]
width = 100
widthmod = [0.5, 1, 0.5]
```

Obrázek 4: Náhled na štětec vytvořený skriptem velké V

```
time = 4

texsize = [30, 16]
tex = []

mid = 15
for i in range(0, 16):
  colin1 = mid + i
  colin2 = mid - i
  for j in range (0, 30):
    col = [1, 1, 1]
    if (j==colin1 or j==colin2):
      col = [0, 0, 0]
    tex.append(col)

return [color, width, widthmod, time, texsize, tex]
```

Depth Mesh

Zde bude popsáno několik příkladů skriptů, které upravují trojúhelníkovou síť vzniklou z hloubkového obrazu scény v aplikaci Depth Mesh.

Posunutí doprava

Tento skript posune všechny vrcholy trojúhelníkové sítě o jeden metr doprava - tedy o jeden metr v kladném směru x-ové osy.

```
for i in range(0, len(points), 3):
    points[i] = points[i] + 1
return [points, uvs, faces]
```

Zvětšení

Tento skript zdvojnásobí vzdálenost mezi všemi vrcholy trojúhelníkové sítě. Neboli celou trojúhelníkovou síť zvětší na dvojnásobek jejího aktuálního rozměru.

```
for i in range(0, len(points)):
  points[i] = points[i] * 2
return [points, uvs, faces]
```

Posunutí textury

Tento skript slouží k posunutí textury po modelu. Texturovací souřadnice jsou souřadnice do obrázku, který se má na model namapovat. Jedná se o souřadnici *u* ve vodorovném směru a *v* ve směru svislém. Hodnoty souřadnic jsou normalizované - tedy mohou mít hodnoty jen v intervalu <0, 1>. V tomto skriptu se ke každé texturovací souřadnici přičte 0.1 - výsledkem bude posun obrázku po vytvořeném modelu scény směrem doleva a nahoru.

```
for i in range(0, len(uvs)):
  uvs[i] = (uvs[i] + 0.1)%1
return [points, uvs, faces]
```

Odstranění vrcholu

Tento skript ukazuje, jak je možné odstranit jeden vrchol z trojúhelníkové sítě. Index odstraňovaného vrcholu je daný proměnnou *tIndex*. Pro odstranění vrcholu je třeba odstranit jeho souřadnice z pole *points*. Dále je třeba odstranit jeho uv souřadnice z pole *uvs*. Důležité je korektně odstranit všechny trojúhelníky z pole *faces*, ve kterých se tento vrchol nacházel. Trojúhelník je udán třemi čísly reprezentujícími pořadová čísla vrcholů, které tento trojúhelník tvoří. Při odstranění vrcholu dojde ke zkrácení pole *points*. Všechny vrcholy, které se nacházejí v poli *points* za odstraněným vrcholem, nyní leží na nižším indexu než předtím, a je třeba v poli *faces* o jedna snížit jejich pořadová čísla. Proto je třeba upravit čísla vrcholů uložené v poli *faces*.

```
tIndex = 10
newpoints = []
for i in range(0, len(points), 3):
  x = points[i+0]
  y = points[i+1]
  z = points[i+2]
  if ((i / 3) != tIndex):
    newpoints.append(x)
    newpoints.append(y)
    newpoints.append(z)

newuvs = []
for i in range(0, len(uvs), 2):
  u = uvs[i+0]
  v = uvs[i+1]
  if ((i / 2) != tIndex):
    newuvs.append(u)
    newuvs.append(v)

newF = []
for i in range(0, len(faces), 3):
```

```
  f1 = faces[i+0]
  f2 = faces[i+1]
  f3 = faces[i+2]

  if (f1 != tIndex and f2 != tIndex
        and f3 != tIndex):
    if (f1 > tIndex):
      f1 = f1 - 1
    if (f2 > tIndex):
      f2 = f2 - 1
    if (f3 > tIndex):
      f3 = f3 - 1

    newF.append(f1)
    newF.append(f2)
    newF.append(f3)

return [newpoints, newuvs, newF]
```

Příloha D: Uživatelská dokumentace

Všechny klientské aplikace k plné funkcionalitě potřebují připojení k serveru. Aplikace Brush Export a Depth Mesh ke správné funkcionalitě potřebují nainstalovaný Python. Aplikace Depth Mesh potřebuje runtime prostředí .NET 6.0. Aplikace Annoying Fly a Depth Mesh potřebují ke své funkcionalitě kameru RealSense D415 a aplikace Box City a PaintVR potřebují ke své funkcionalitě headset HTC Vive Pro 2.0.

Annoying Fly

Aplikace demonstruje základní schopnosti hloubkového senzoru RealSense. Aplikace je promítána na plátno umístěné na zdi a hloubková kamera je umístěna tak, že snímá celou plochu tohoto plátna. Jedná se o hru, ve které uživatel za pomocí svého „stínu" snímaného senzorem zabíjí/odhání promítanou mouchu pohybující se po plátně. Screenshot herního okna se posílá na server.

Aplikace umožňuje nastavit rozsah hloubky, která je snímána, a nastavit posunutí a přiblížení snímaného obrazu tak, aby uživatel mohl nastavit takové zobrazení, které mu pro hraní vyhovuje. Dále aplikace umožňuje zapnutí „barevného módu", ve kterém jsou různé snímané hloubky ilustrovány barevným gradientem.

Aplikace se spouští pomocí souboru *AnnoyingFly.exe*. Způsob používání je popsán v dodaném videu *AnnoyingFly.mp4*.

Konfigurační soubor

Konfigurační soubor s názvem *config.txt* musí být umístěn na stejné úrovni jako *AnnoyingFly.exe*. Jeho tvar je následující

```
<minimální hloubka>
<maximální hloubka>
<horizontální posun>
<vertikální posun>
```

```
<přiblížení>
<url serveru>
<jméno klienta>
```

Z konfiguračního souboru se tedy načítají počáteční hodnoty parametrů omezujících hloubku, se kterými kamera snímá scénu a posun a přiblížení zobrazovaného obrázku scény. Zadané url serveru musí končit lomítkem a jméno by mělo být unikátní pro všechny klienty typu Annoying Fly.

Depth Mesh

Aplikace umožňuje snímání scény pomocí hloubkového senzoru RealSense. Kamera vytváří hloubkový obraz, ve kterém je uložena hloubka jednotlivých bodů scény. Kromě pozic jednotlivých bodů v prostoru kamera zaznamenává i jejich reálnou barvu, a umožňuje tedy zobrazit barevný snímek scény.

Aplikace vytváří trojrozměrný model scény z pixelů hloubkového obrazu. Na tomto modelu je jako textura aplikován barevný snímek scény. Dále aplikace umožňuje úpravu tohoto modelu pomocí uživatelem zadaného kódu. Tento model lze poslat na server. Aplikace se spouští dvojklikem na soubor *DepthMesh.bat*. Způsob používání aplikace je popsán v dodaném videu *DepthMesh.mp4*.

Konfigurační soubor

Konfigurační soubor s názvem *config.txt* musí být umístěn ve složce *src*, ve které se mimo jiné nachází exe soubor aplikace. Tvar konfiguračního souboru je následující

```
<cesta k python.dll>
<url serveru>
<jméno klienta>
```

Zadané URL serveru musí končit lomítkem a jméno by mělo být unikátní pro všechny klienty typu Depth Mesh.

Box City

Aplikace umožňuje pohybovat se po virtuální scéně a manipulovat s objekty, které se v této scéně nacházejí. Ve scéně se mimo jiné nacházejí krabice, se kterými si uživatel může vyzkoušet manipulaci s objekty ve virtuálním světě.

Aplikace omezuje pohyb uživatele po virtuální scéně tak, aby nemohl ve skutečném světe nevědomky odejít z prostoru vyhrazeného pro virtuální realitu. Velikost tohoto prostoru je nastavitelná pomocí konfiguračního souboru.

Ve scéně se zobrazují objekty virtuálního světa, které jsou uloženy na serveru, a aplikace na server odesílá krabice. Aplikace se spouští pomocí souboru *BoxCity.exe*. Způsob používání je popsán v dodaném videu *BoxCity.mp4*.

Konfigurační soubor

Konfigurační soubor s názvem *config.txt* musí být umístěn na stejné úrovni jako *BoxCity.exe*. Jeho tvar je následující

```
<jméno klienta>
<url serveru>
<rozměry pokoje, např.: 1.5, 1.5>
```

kde rozměry pokoje udávají šířku a délku bezpečné zóny, ve které se uživatel může pohybovat aniž by se musel obávat, že narazí do objektů v reálném světě. Rozměry se zadávají v metrech, pokud se jedná o desetinné číslo, musí být zapsáno s desetinnou tečkou a mezi jednotlivými hodnotami se nachází čárka. Zadané URL serveru musí končit lomítkem a jméno by mělo být unikátní pro všechny VR klienty.

Brush Export

Aplikace slouží jako podpůrná aplikace k aplikaci PaintVR. Umožňuje vytvářet uživatelské štětce, umožňuje uživateli měnit jejich parametry, a pomocí uživatelem zadaného kódu generuje jednotlivé pixely textury štětce. Textura štětce se mění v závislosti na tom jak dlouho je s ním malováno. Aplikace poskytuje uživateli náhled na vytvořený štětec a umožňuje jej exportovat jako XML soubor a jeho texturu jako png soubor. Aplikace se spouští pomocí souboru *BrushExport.exe*. Způsob používání je popsán v dodaném videu *BrushExport.mp4*.

Konfigurační soubor

Konfigurační soubor s názvem *config.txt* musí být umístěn na stejné úrovni jako soubor *BrushExport.exe*. Tvar konfiguračního souboru je následující

```
<cesta k python.dll>
```

Exportovaný XML soubor

Tvar exportovaného XML souboru s daty štětce je následující.

```
<?xml version="1.0" encoding="utf-8"?>
<Brush xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Name>BrushGrad</Name>
```

```
  <Width>0.05</Width>
  <Color>
    <r>1</r>
    <g>1</g>
    <b>1</b>
    <a>1</a>
  </Color>
  <WidthModifier>
    <float>0.1</float>
    <float>0.5</float>
    <float>1</float>
    <float>0.5</float>
    <float>0.1</float>
  </WidthModifier>
  <TimePerIter>5</TimePerIter>
</Brush>
```

Kořenovým prvkem XML je značka <Brush>. Ve značce <Name> se nachází název štětce. Značka <Color> obsahuje barvu štětce a obsahuje značky <r>, <g>, <b> a <a>, které obsahují jednotlivé barevné kanály. Značka <WidthModifier>, obsahuje hodnoty pole *widthModifier*, každou ve značce <float>. Ve značce <TimePerIter> se nachází parametr času průchodu polem *widthModifier*.

PaintVR

Aplikace umožňuje kreslení ve VR a změnu velikosti a barvy štětců za běhu. Štětce jsou načítány z xml souborů umístěných ve složce „Brushes" na úrovni exe souboru spouštějícího celou aplikaci. Aplikace omezuje pohyb uživatele po virtuální scéně tak, aby nemohl ve skutečném světe nevědomky odejít z prostoru vyhrazeného pro virtuální realitu. Velikost tohoto prostoru je nastavitelná pomocí konfiguračního souboru. Uživatel může stisknutím mezerníku na klávesnici volit mezi několika verzemi zobrazovaného skyboxu a barvy podlahy, po které chodí.

Ve scéně se zobrazují objekty virtuálního světa, kterí jsou uložené na serveru. Tahy štětcem jsou automaticky zasílány na server. Aplikace se spouští pomocí souboru *PaintVR.exe*. Způsob používání je popsán v dodaném videu *PaintVR.mp4*.

Konfigurační soubor

Konfigurační soubor s názvem *config.txt* musí být umístěn na stejné úrovni jako *PaintVR.exe*. Jeho tvar je následující

```
<jméno klienta>
<url serveru>
<rozměry pokoje, např.: 1.5, 1.5>
```

kde rozměry pokoje udávají šířku a délku bezpečné zóny v metrech, ve které se uživatel může pohybovat aniž by se musel obávat, že narazí do objektů v reálném světě. Pokud je zadaná velikost udána jako desetinné číslo, musí být zapsáno s desetinnou tečkou a mezi jednotlivými hodnotami se nachází čárka. Zadané URL serveru musí končit lomítkem a jméno by mělo být unikátní pro všechny VR klienty.

Příloha E: Zpětná vazba z uživatelských dotazníků

V této příloze se nacházejí odpovědi na uživatelské dotazníky ve formě statistik, tak jak byly exportovány z Google Forms.

Annoying fly - Realsense aplikace

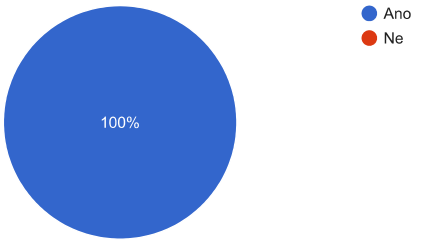
6 responses

Publish analytics

První dojmy

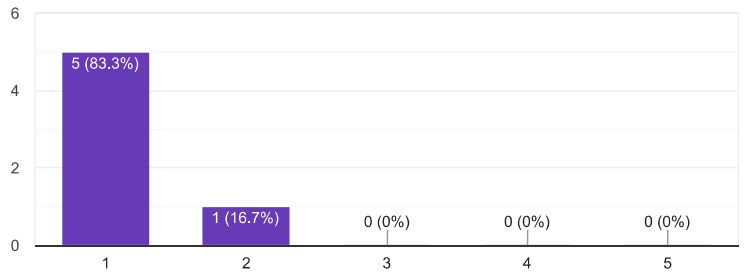
Podařilo se připojit k serveru? Copy

6 responses



Povedlo se nastavit hloubku záběru? Jak těžké to bylo? Copy

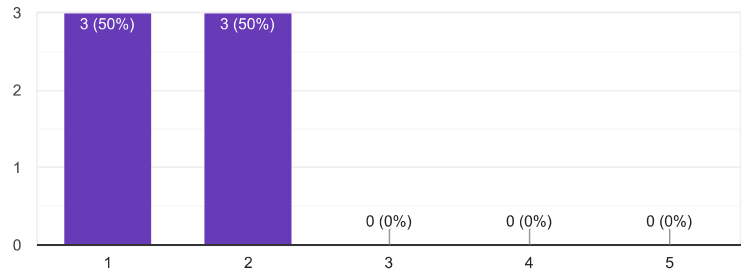
6 responses



Reaguje aplikace na zakrytí mouchy? Jak přesná je detekce zabití mouchy?

Copy

6 responses



Popište prosím případné problémy

0 responses

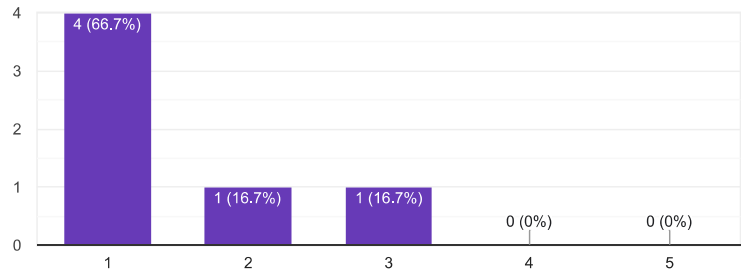
No responses yet for this question.

Ovládání

Chová se přiblížení/oddálení tak jak jste očekávali?

Copy

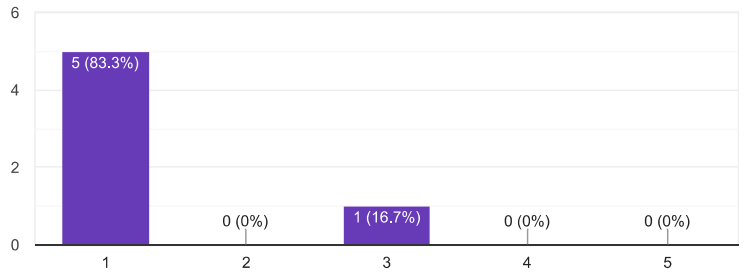
6 responses



Chová se posun tak jak ste očekávali?

Copy

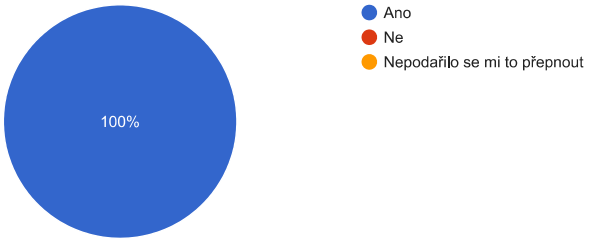
6 responses



Zobrazují se po přepnutí různé barvy pro různou hloubku?

Copy

6 responses



Popište prosím případné problémy

0 responses

No responses yet for this question.

Závěr

Pokud nebylo některé otázce v dotazníku rozumět, dejte nám prosím vědět zde

0 responses

No responses yet for this question.

Jedna věc, která mi na aplikaci opravdu vadila

6 responses

nic

Asi nic

Vzdálenost od kamery byla označena v cm ale pro nastavení se uváděla v m

použití barev a grafického designu. Barvy byly vkusné a sladěné

Chování mouchy by mohlo být více propracované.

Jedna věc, která se mi na aplikaci líbila

6 responses

Nevim

Jednoduchost nastavení všech parametrů

nevim

změna barev ukazující vzdálenost ruky od kamery

Aplikace by si mohla pamatovat adresu serveru např mít ji uloženou někde v databázi pro rychlejší ovládání a pohodlí uživatele.

Obrázek mouchy je adekvátně nechutný.

Co aplikace neumí, ale bylo by super, kdyby to uměla

6 responses

Nevim

Herní režim kdy bude na obrazovce určitý počet much a bude se počítat čas zaplácnutí všech.

nevim

počítání zabitých much

odpověď nad ^^

Vím, že moucha možná nebyla hlavní prioritou této aplikace, ale myslím si, že by mohla být o dost vylepšena. V tuto chvíli se pouze zjeví na náhodné pozici na obrazovce a poté co je zaplácnuta tak zmizí na jinou pozici.

Mám návrh. Poté co bude moucha zaplácnuta, tak místo aby hned zmizela tak postupně prolne s pozadím. Takový přechod nemusí být dlouhý a může trvat jen cca 250ms. Když se mám moucha objevit, tak místo toho aby se jen tak zjevila na obrazovce, tak se objeví z okraje obrazovky a bude se pohybovat po náhodnou krátkou dobu (500 - 1500ms) kroužívými pohyby dokud neusedne a její efekt usednutí bude doprovázen jemným zvětšením jejího obrázku (aby bylo jasnější že si sedla).

Prostor na jiné připomínky/poznatky

0 responses

No responses yet for this question.

Realsense aplikace - DepthMesh klient

5 responses

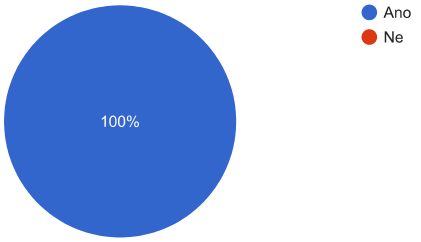
Publish analytics

První dojmy

... umím udělat nový snímek

Copy

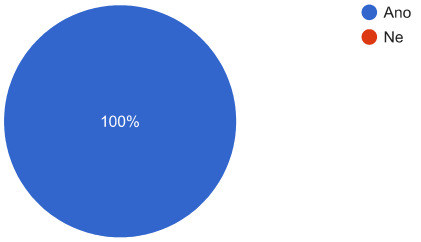
5 responses



... rozumím zobrazenému náhledu

Copy

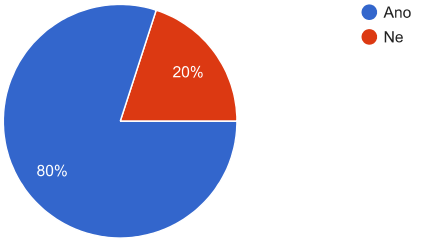
5 responses



... umím se připojit k serveru

Copy

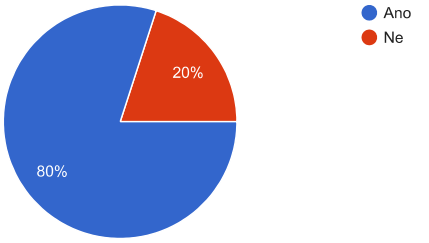
5 responses



... umím odeslat trojúhelníkovou síť na server / stáhnout ji ze serveru

Copy

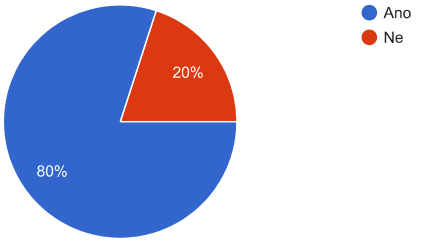
5 responses



... umím uložit snímek na disk

Copy

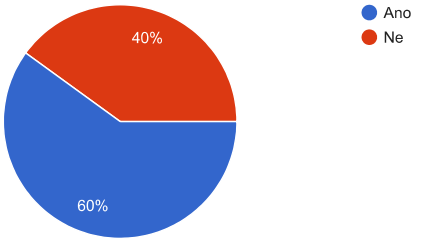
5 responses



... umím vykonat ukázkový python kód upravující snímek

Copy

5 responses



Popište prosím případné problémy

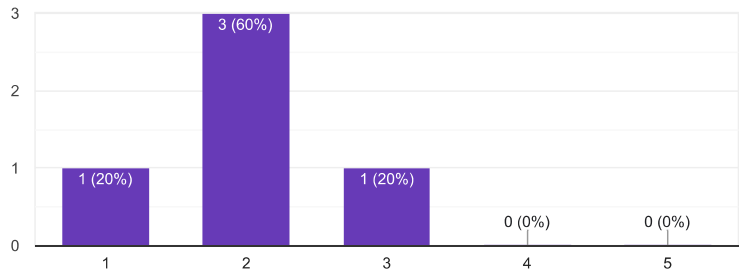
1 response

Nepochopil jsem, co dělá číslo v pravém horním rohu.

Manipulace se snímkem

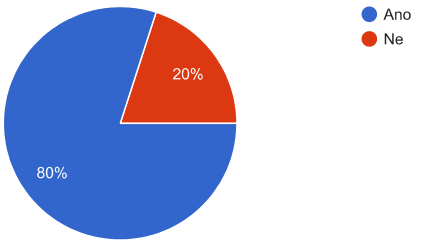
Jak se ovládá náhled na snímek? (rotace, přiblížení, posun...) Copy

5 responses



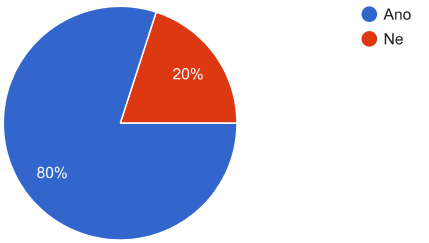
Vidím, jak aplikace reaguje na změnu parametru - Decimace trojúhelníků Copy

5 responses



Vidím, jak aplikace reaguje na změnu parametru - Oříznout Copy

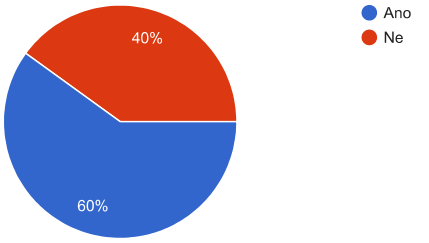
5 responses



Vidím, jak aplikace reaguje na změnu parametru - Záplatování děr

Copy

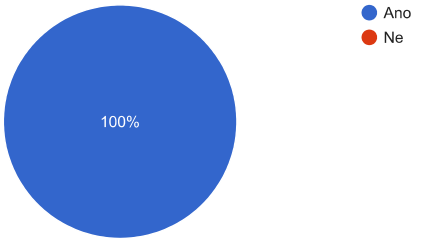
5 responses



Vidím, jak aplikace reaguje na změnu parametru - Prostorové vyhlazování

Copy

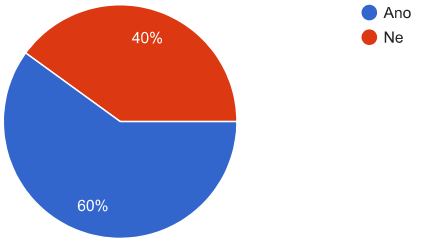
5 responses



Vidím, jak aplikace reaguje na změnu parametru - Časové vyhlazování

Copy

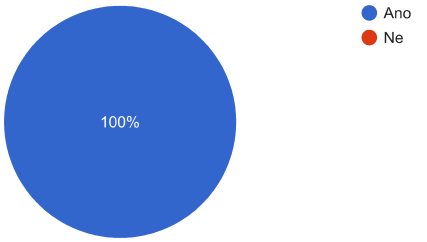
5 responses



Vidím, jak aplikace reaguje na posun slideru ovlivňující maximální velikost trojúhelníků - Prahování trojúhelníků

Copy

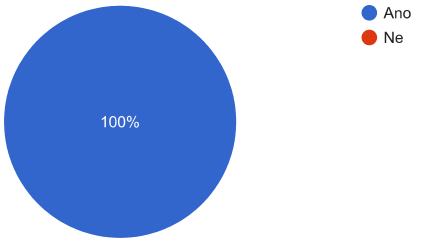
5 responses



Vidím, jak aplikace reaguje na změnu parametru - Viditelné vrcholy

Copy

5 responses



Popište prosím případné problémy

0 responses

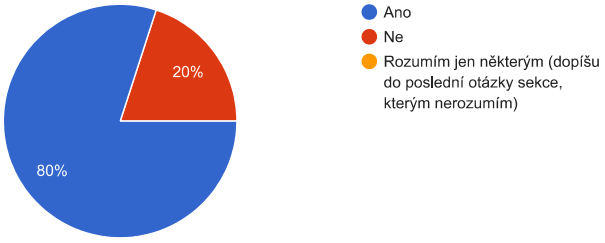
No responses yet for this question.

Python kód

Vím jaké proměnné mám k dispozici a rozumím jejich významu

Copy

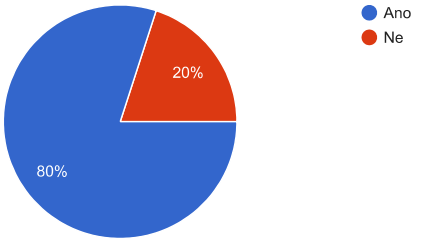
5 responses



Vím jaké proměnné má kód vracet

Copy

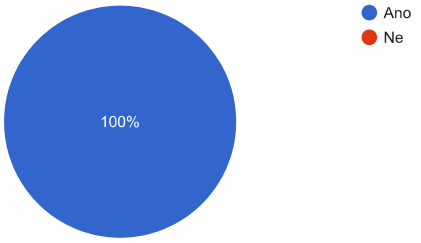
5 responses



Po vykonání kódu vidím změny v náhledu

Copy

5 responses



Pokud udělám chybu v kódu, zobrazí se mi hláška, která mi pomůže chybu opravit

Copy

5 responses

Popište prosím případné problémy - pokud např. nebyl jasný význam některých proměnných python kódu, nebo varovných hlášek, napište to zde

0 responses

No responses yet for this question.

Závěr

Pokud nebylo některé otázce v dotazníku rozumět, dejte nám prosím vědět zde

0 responses

No responses yet for this question.

Jedna věc která mi na aplikaci opravdu vadila

5 responses

Nevěděl jsem, co dělá číslo v pravém horním rohu

nic

Prvních pár minut jsem nebyl schopen se v aplikaci zorientovat, poté vše v pořádku

aplikace je těžší na pochopení

Nic

Jedna věc která se mi na aplikaci líbila

5 responses

Nevím

nevim

zajímavé ovládání, zábavné

Grafický design byl pěkně proveden a pomohl aplikaci vypadat moderně a profesionálně.

Vše bylo ok.

Co aplikace neumí, ale bylo by super, kdyby to uměla

5 responses

Nevím

nevim

vše v pohodě

Personalizovat ovládání na základě uživatelských preferencí.

Nevím.

Prostor na jiné připomínky/poznatky

0 responses

No responses yet for this question.

Box city - VR aplikace

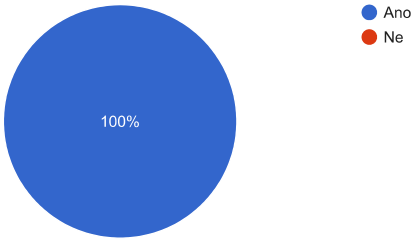
4 responses

Publish analytics

První dojmy

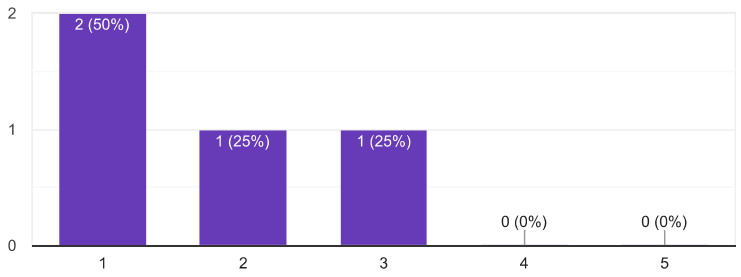
Podařilo se připojit k serveru? Copy

4 responses



Jak snadné je pohybovat se v prostoru Copy

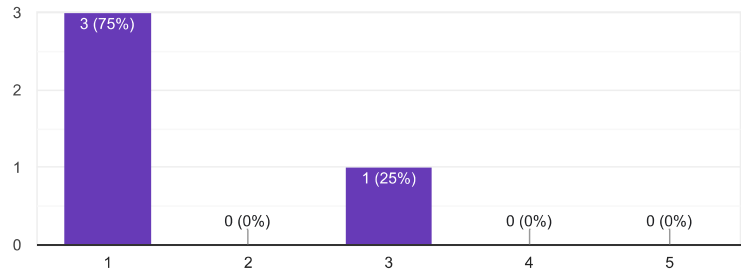
4 responses



Jak snadná je manipulace s objekty

Copy

4 responses



Popište prosím případné problémy

3 responses

Nebyly

nic

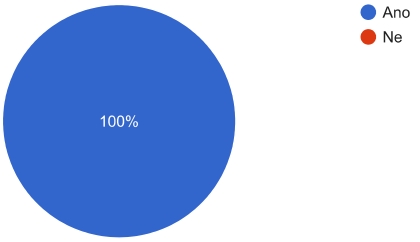
Žádné

Úkoly

Našel/Našla jsem krabice ve městě

Copy

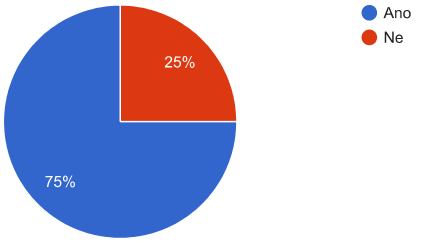
4 responses



S nalezenými krabicemi jsem mohl/a hýbat

Copy

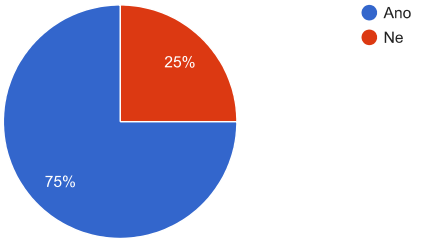
4 responses



Krabice se pohybují s ovladačem

Copy

4 responses



Popište prosím případné problémy

1 response

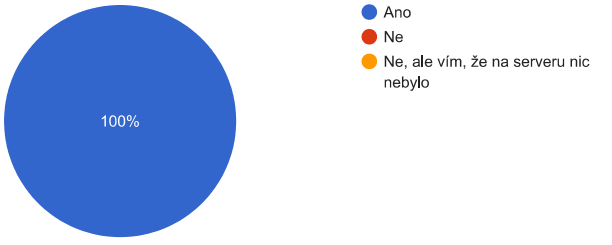
nic

Jiné objekty

Našel/Našla jsem ve městě objekty z jiných aplikací

Copy

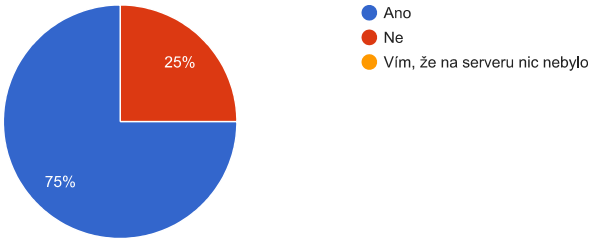
4 responses



S objekty z jiných aplikací můžu hýbat

Copy

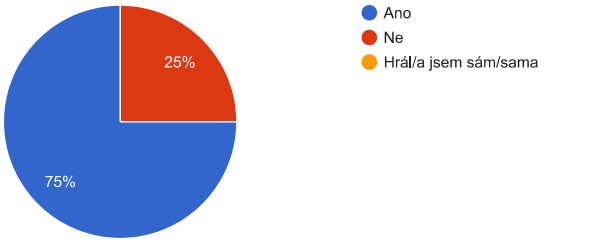
4 responses



Vidím objekty značící ruce a headset jiného hráče, a jak se pohybují po virtuálním světě

Copy

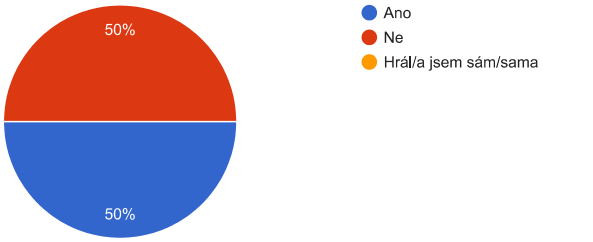
4 responses



Vidím jak jiný hráč manipuluje s objekty nebo kreslí nové tahy štětcem

Copy

4 responses



Popište prosím případné problémy

2 responses

nic

Nebylo vidět nové kreslené tahy štětcem, teprve až je dokreslil

Závěr

Pokud nebylo některé otázce v dotazníku rozumět, dejte nám prosím vědět zde

0 responses

No responses yet for this question.

Pokud nebylo některé otázce v dotazníku rozumět, dejte nám prosím vědět zde

0 responses

No responses yet for this question.

Jedna věc která mi na aplikaci opravdu vadila

4 responses

nic

asi mi nic nevadilo

na aplikaci opravdu vadila, místy pomalá odezva.

Mohl jsem vzít více tahu štětcem od jiného hráče, všechna díla se pak zničila

Jedna věc která se mi na aplikaci líbila

4 responses

nevim

jak to vypadalo reálně

Grafický design

Mohl jsem se dostat do budov

Co aplikace neumí, ale bylo by super, kdyby to uměla

4 responses

nevim

skákat

plnohodnotný multiplayer

Sjednotit tahy štětcem do jednoho objektu

Prostor na jiné připomínky/poznatky

0 responses

No responses yet for this question.

Brush export

2 responses

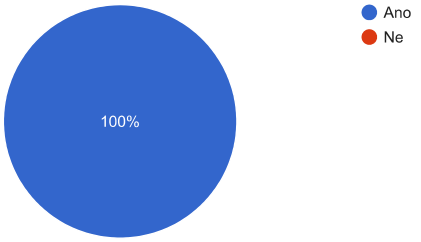
Publish analytics

První dojmy

... umím přejmenovat štětec

Copy

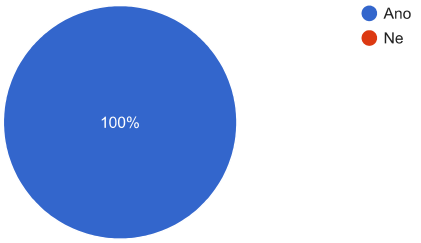
2 responses



... umím vykonat vzorový uživatelský kód

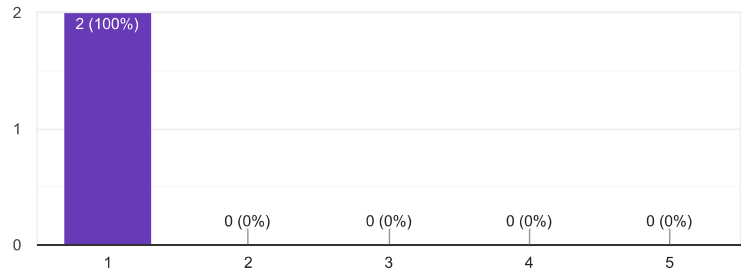
Copy

2 responses



... umím exportovat štetec (popř i změnit kam bude uložen) Copy

2 responses



Popište prosím případné problémy

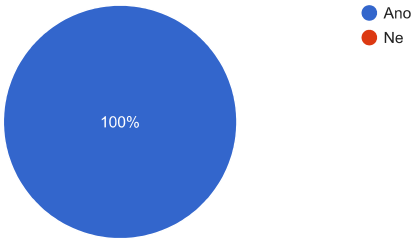
0 responses

No responses yet for this question.

Python kód

Je jasné jaké proměnné má python funkce vracet Copy

2 responses



Rozumím významu proměnných, které má funkce vracet vuživatelském kódu

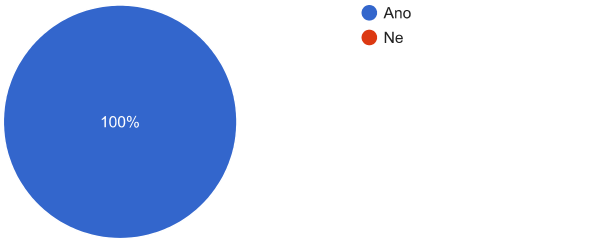
Copy

2 responses

Po vykonání kódu vidím změny v náhledu

Copy

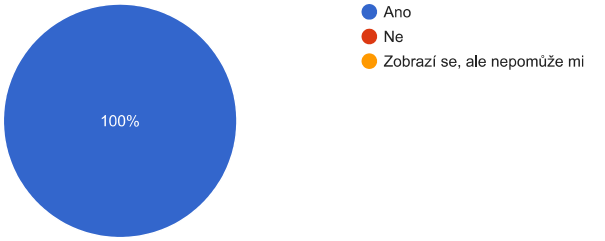
2 responses



Pokud udělám chybu v kódu, zobrazí se mi hláška, která mi pomůže chybu opravit

Copy

2 responses



Popište prosím případné problémy - pokud např. nebyl jasný význam některých proměnných python kódu, nebo varovných hlášek, napište to zde

0 responses

No responses yet for this question.

Závěr

Pokud nebylo některé otázce v dotazníku rozumět, dejte nám prosím vědět zde

0 responses

No responses yet for this question.

Jedna věc která mi na aplikaci opravdu vadila

2 responses

nic

Mohl být dodělán ingame tvůrce štětců

Jedna věc která se mi na aplikaci líbila

2 responses

nevim

autorovo dokumentace

Co aplikace neumí, ale bylo by super, kdyby to uměla

2 responses

nevim

ingame tvůrce štětců

Prostor na jiné připomínky/poznatky

0 responses

No responses yet for this question.

PaintVR - VR aplikace

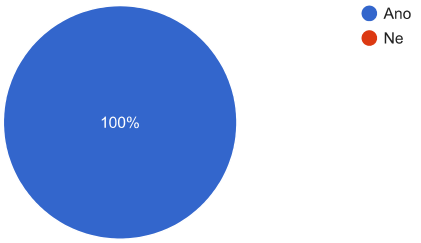
5 responses

Publish analytics

První dojmy

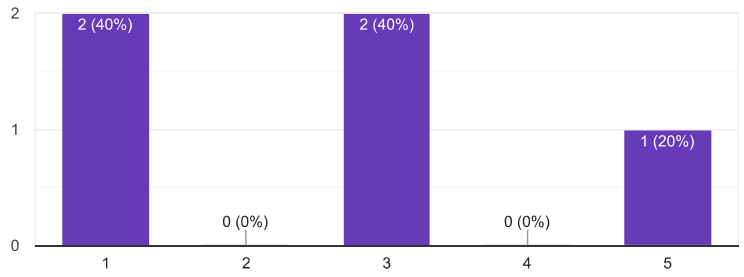
Podařilo se připojit se k serveru? Copy

5 responses



Jak snadné je pohybovat se v prostoru Copy

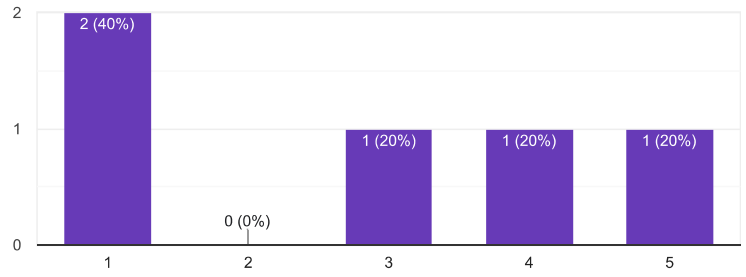
5 responses



Jak snadné je ovládání kreslení

Copy

5 responses



Popište prosím případné problémy

4 responses

když jsem se pokoušela něco nakreslit tak se mi to z ničeho nic seklo ale jinak se mi to líbilo

nebyly

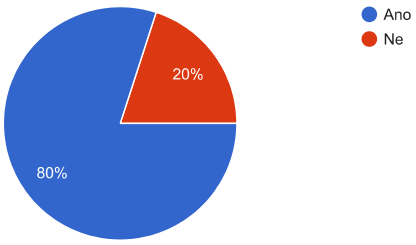
problémy spíše souvisely s použitým typem Vr headsetu

Kreslení

Tah štetcem se zobrazuje tam kde mám/měl/a jsem ovladač

Copy

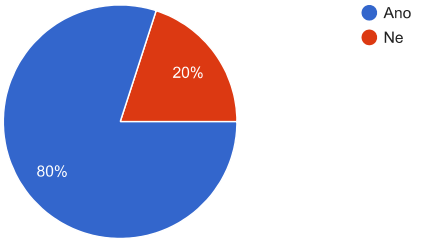
5 responses



Aplikace reaguje na změnu barvy štětce

Copy

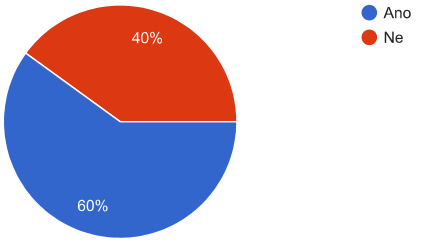
5 responses



Aplikace reaguje na změnu šířky štětce

Copy

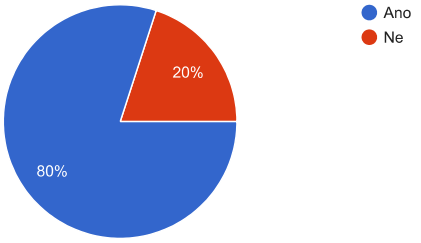
5 responses



Podařilo se mi přepnout ze štětce na gumu (a zpět)

Copy

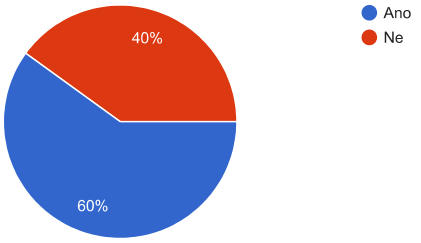
5 responses



Gumování funguje - tah zmizí když na něj kliknu gumou

Copy

5 responses



Popište prosím případné problémy

3 responses

nezkoušela jsem vše

občas ovladač nereagoval jak by měl

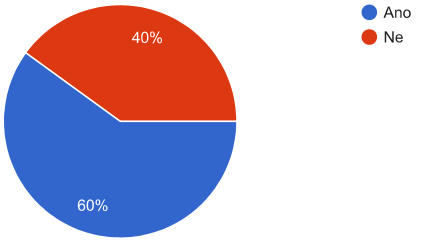
Není hned zcela jasné, jak si na paleťe vybírat barvu, ale to je možná způsobeno moji debilitou.

Štětce

Podařilo se mi přepínat mezi štětci

Copy

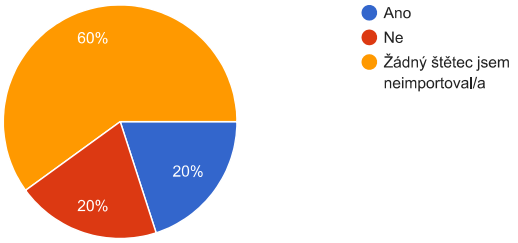
5 responses



V nabídce štětců se zobrazují importované uživatelské štětce

Copy

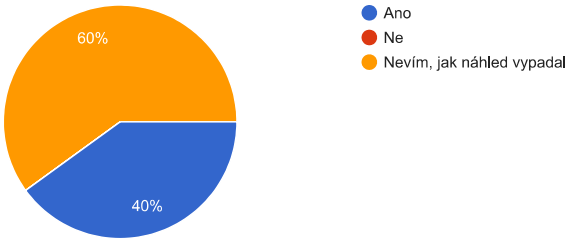
5 responses



Vzhled a chování importovaných štětců odpovídá náhledu v aplikaci Brush export

Copy

5 responses



Popište prosím případné problémy

3 responses

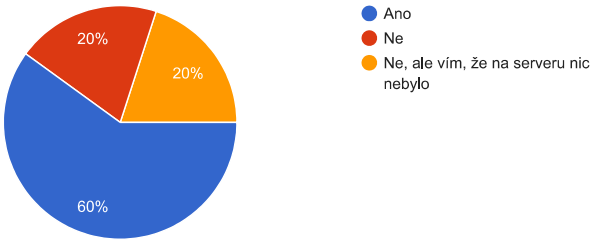
žádné

Jiné objekty

Vidím ve VR i jiné objekty než tahy štětcem

Copy

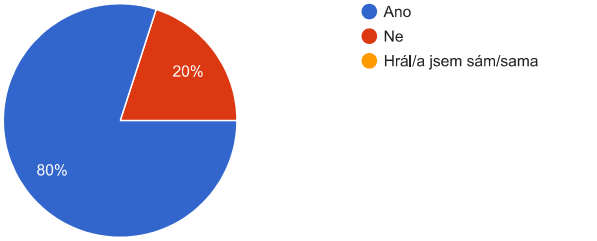
5 responses



Vidím objekty značící ruce a headset jiného hráče, a jak se pohybují po virtuálním světě

Copy

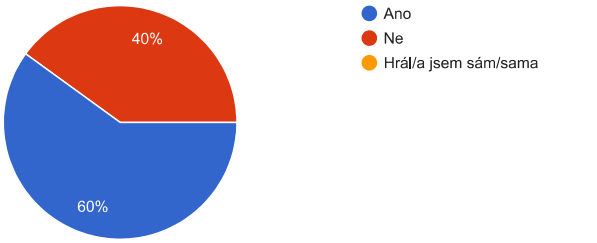
5 responses



Vidím jak jiný hráč manipuluje s objekty nebo kreslí nové tahy štětcem

Copy

5 responses



Popište prosím případné problémy

3 responses

nic

Závěr

Pokud nebylo některé otázce v dotazníku rozumět, dejte nám prosím vědět zde

0 responses

No responses yet for this question.

Jedna věc která mi na aplikaci opravdu vadila

5 responses

trochu se to sekalo když jsem malovala

nešlo vůbec kreslit

nevím

nefunkčnost ovladače

Myslím, že textura oblohy a země (okolního prostředí), by mohla být vybrána lépe a nejlepší by byla možnost si vybrat z více různých pozadí (pokud je to možné nevím o tom). Třeba zcela černé pozadí, by se taky kolikrát hodilo, protože by na něm dobře vynikaly barvy štětce (lepší kontrast).

Jedna věc která se mi na aplikaci líbila

5 responses

všechno až na to sekání

nic

hezky to kreslilo

jednoduché ovládání

Malování štětcem je bezproblémové, netrhá se a funguje pole očekávání.

Co aplikace neumí, ale bylo by super, kdyby to uměla

5 responses

nevím

nevim

nějaké šablony

nvm

Možnost si vybrat z více různých pozadí (pokud je to možné nevím o tom)

Prostor na jiné připomínky/poznatky

0 responses

No responses yet for this question.