Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Herného jadro Unreal so systémom nDisplay v LIRKIS CAVE

Diplomová práca

2020 **Bc. Dominik Tóth**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Herného jadro Unreal so systémom nDisplay v LIRKIS CAVE

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko, PhD.
Konzultant:	

Košice 2020 **Bc. Dominik Tóth**

Názov práce:	Herného jadro Unreal so systémom nDisplay v LIRKIS CAVE
Pracovisko:	Katedra počítačov a informatiky, Technická univerzita v Košiciach
Autor:	Bc. Dominik Tóth
Školiteľ:	Ing. Štefan Korečko, PhD.
Konzultant:	
Dátum:	4. 5. 2020
Kľúčové slová:	Unreal Engine, Virtuálna realita

Abstrakt: Diplomová práca sa zaoberá využitím herného jadra Unreal so systémom nDisplay v LIRKIS CAVE. V prvej časti analyzuje systém nDisplay a prípady použitia herných enginov. Na základe analýzy sa navrhuje a implementuje riešenie vizualizačného systému pre LIRKIS CAVE, ktoré využíva systém nDisplay. Riešenie implementuje konfiguráciu, ktorá je založená na definovaní hierarchie obrazoviek, ktorá zodpovedá skutočnému svetu. V ďalšej časti sa navrhuje a implementuje testovacia scéna. V závere sa vytvorená scéna využíva na vyhodnotenie systému.

Thesis title:	Unreal Game Engine with nDisplay in LIRKIS CAVE
Department:	Department of Computers and Informatics, Technical University of Košice
Author:	Bc. Dominik Tóth
Supervisor:	Ing. Štefan Korečko, PhD.
Tutor:	
Date:	4. 5. 2020
Keywords:	Unreal Engine, Virtual reality

Abstract: This master's thesis deals with the utilization of Unreal Game Engine with nDisplay in LIRKIS CAVE. The first part analyzes nDisplay and use cases of game engines. Based on the analysis, the solution of visualisation system for LIRKIS CAVE using nDisplay is designed and implemented. The solution implements configuration that is based on defining the screen hierarchy that reflects real world. In the next part, the test scene is designed and implemented. In conclusion, the created scene is used for evaluation of the system.

57815

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

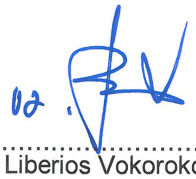
Herného jadro Unreal so systémom nDisplay v LIRKIS CAVE

Unreal Game Engine with nDisplay in LIRKIS CAVE

Študent: **Bc. Dominik Tóth**
Školiteľ: **Ing. Štefan Korečko, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:
1. Oboznámiť sa so systémom nDisplay herného jadra Unreal Engine a virtuálno-realitnou jaskyňou LIRKIS CAVE.
2. Analyzovať existujúce prípady použitia herného jadra Unreal Engine pre virtuálno-realitné jaskyne.
3. Navrhnúť a implementovať softvérové riešenie vizualizácie na báze Unreal Engine a nDisplay pre LIRKIS CAVE.
4. Implementované riešenie nech vykresľuje obraz stereoskopicky a prispôsobuje ho aktuálnej pozícii používateľa, snímanej systémom OptiTrack. Tiež nech umožňuje centralizované ovládanie z jedného počítača.
5. Implementované riešenie overiť a vyhodnotiť s použitím vhodných scén.
6. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 04.05.2020
Dátum zadania diplomovej práce: 31.10.2019



prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 4.5.2020 .....

Vlastnoručný podpis

Poďakovanie

Chcel by som sa poďakovať vedúcemu mojej diplomovej práce, Ing. Štefanovi Korečkovi, PhD. za odbornú pomoc pri písaní záverečnej práce. Ďalej by som sa chcel poďakovať mojim rodičom za podporu počas štúdia.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Diplomová práca sa zaoberá návrhom a implementáciou vizualizačného jadra pre CAVE systém, ktorý sa nachádza v laboratóriu LIRKIS TUKE. Súčasne používané jadro pre LIRKIS CAVE je SuperEngine, avšak pomaly zastaráva, preto je snaha o jeho nahradenie moderným herným jadrom.

V priebehu minulých rokov bolo vytvorených niekoľko prác, ktoré sa zaoberali využitím Unreal Engine 4 v LIRKIS CAVE. Prvým prototypom, ktorý slúžil ako prípadová štúdia bol *SyncMulti3* [1]. Na základe tejto práce vypracoval M. Gabriška riešenie [2], ktoré obsahovalo moduly pre synchronizáciu, konfiguráciu a spúšťanie pri štarte. Funkcionalitou bolo podobné SuperEnginu, avšak neobsahovalo mimo-osové zobrazenie. S novými verziami jadra prišla funkcionalita *nDisplay*, ktorá umožňuje synchronizované zobrazovanie aplikácie na viacerých zariadeniach. O integráciu tejto verzie jadra do CAVE systému sa postarala P. Romanová vo svojej bakalárskej práci [3]. Jej riešenie má však nedostatok v tom, že nezohľadňuje všetky funkcionality, ktoré by mal systém obsahovať, preto je nutné pokračovať v riešení tohoto problému, aby sme dosiahli stav, ktorý dokáže nahradiť súčasne jadro SuperEngine.

Softvérové vybavenie CAVE systému má niekoľko dôležitých súčasti, medzi ktoré patrí synchronizácia a zarovnanie obrazoviek, zmena obrazu podľa pozície hráča a stereoskopické zobrazenie.

Je potrebné získať rozmery CAVE systému v reálnom svete a na základe nich vytvoriť konfiguračný súbor pre nDisplay zásuvný modul [4], ktorý by sa mal pridávať do každej aplikácie postavenej na Unreal Engine 4, ktorú chceme spúšťať v CAVE systéme. Na základe reálnych rozmerov z konfiguračného súboru sa na každej obrazovke vykreslí obraz zodpovedajúci jej polohe, čo vytvára pre používateľa jaskyne pocit, že sa nachádza vo virtuálnom svete. Rozmery neboli v práci P. Romanovej namerané dostatočne presne, čo spôsobuje. že obraz na susedných

monitoroch nie je správne zarovnaný.

Pre zisťovanie polohy používateľa sa v CAVE systéme využíva OptiTrack systém. OptiTrack sa dá využiť spolu s nDisplay modulom pre naplnenie jednej z požiadaviek na CAVE systém.

SuperEngine poskytuje spúšťanie celého systému zo vzdialeného stroja. Táto funkcionalita v súčasnej verzii systému chyba, preto je potrebné ju v tejto práci zohľadniť a doplniť.

Pre súčasnú verziu integrácie Unreal Enginu do CAVE systému je vytvorený iba jeden projekt, ktorý neukazuje dostatočne možnosti, ktoré nám Unreal Engine poskytuje. Preto je potrebné vytvoriť nový projekt, ktorý bude demonštrovať možnosti Unreal Enginu a bude dokazovať, že integrácia nového jadra systému je dostatočná na nahradenie súčasného jadra SuperEngine.

1 Formulácia úlohy

Cieľom tejto diplomovej práce je zavŕšiť integráciu herného jadra Unreal do virtuálno realitnej jaskyne LIRKIS CAVE. Prvým bodom zadania je oboznámiť sa s virtuálno realitnou jaskyňou LIRKIS CAVE. A podrobne si predstaviť funkcionalitu nDisplay zásuvného modulu pre Unreal Engine 4. Druhým bodom zadania je analyzovať dostupné prípady použitia herného jadra Unreal Engine pre virtuálno realitné jaskyne.

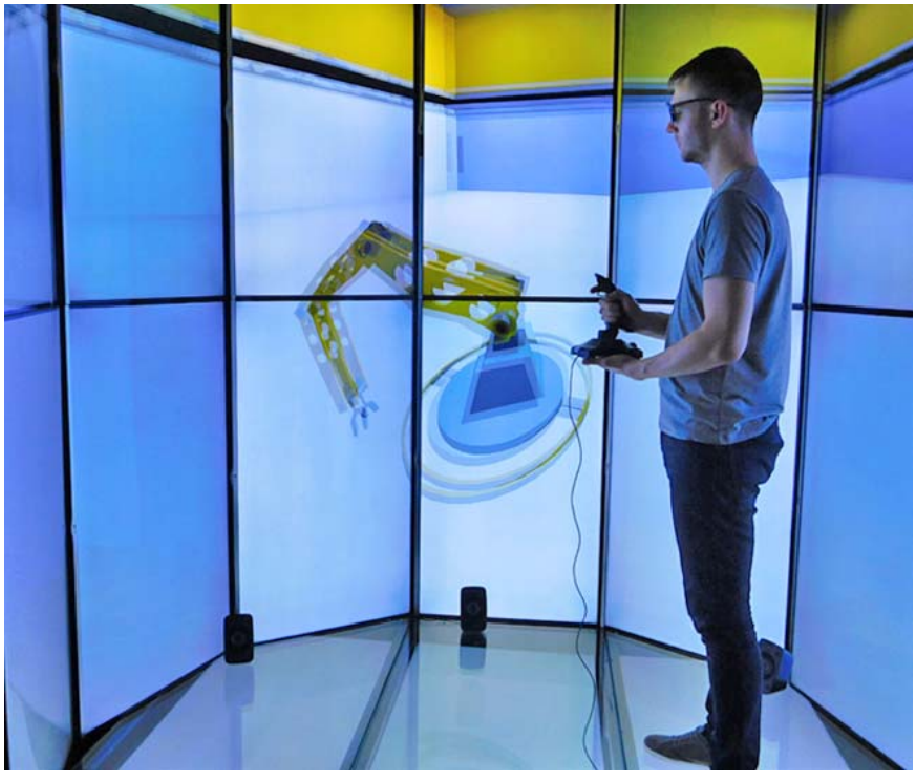
Treťou úlohou je navrhnúť a implementovať softvérové riešenie vizualizácie na báze Unreal Engine a nDisplay pre LIRKIS CAVE.

Štvrtým bodom zadania je vykresľovanie obrazu stereoskopicky a prispôsobenie obrazu aktuálnej pozícii používateľa, snímanej systémom OptiTrack. Spúšťať systém zo vzdialeného počítača, ktorý je pripojený na rovnakej sieti, ako počítače slúžiace na vykresľovanie obrazu, musí byť tiež možné.

Posledným bodom zadania je vytvorenie projektu, ktorý demonštruje funkcionalitu systému.

2 LIRKIS CAVE

Laboratórium LIRKIS na Technickej Univerzite v Košiciach je vybavené virtuálno-realitnou jaskyňou LIRKIS CAVE [5]. Pozostáva zo siedmich vzájomne prepojených počítačov, ktoré umožňujú vykresľovať stereoskopický obraz spolu na dvadsiatich 55 ″ obrazoviek, ktoré sú umiestnené do polkruhu a umožňujú stereoskopický Full HD výstup. Tri z týchto obrazoviek sú na hornej a tri na dolnej stene jaskyne a štrnásť je v polkruhu, v dvoch radoch po sedem (Obr. 2.1). Také usporiadanie umožňuje pohyb po jaskyni a človek vidí obraz aj periférne.



Obr. 2.1: CAVE systém v LIRKIS laboratóriu [5].

Jaskyňa obsahuje aj systém OptiTrack, ktorý sa skladá zo siedmich kamier a

reflexných sledovacích značiek. Tie sú umiestnené na okuliaroch a človeka, ktorý ich má nasadené na hlave systém OptiTrack sleduje. Softvérové vybavenie jaskyne sa skladá aj z aplikácie Motive (verzia 1.9.0), ktorá prijme údaje z kamier a vypočítava polohu hráča vo svete.

Druhá časť softvérového vybavenia jaskyne sa skladá zo systému SuperEngine, založeného na OpenSG, ktorý slúži ako hlavné jadro pre vykresľovanie scén pre jaskyňu. SuperEngine obsahuje konzolu, pomocou ktorej, sa na diaľku ovládajú počítače klastra. Umožňuje tak zapínať a vypínať celý systém jaskyne, ako aj meniť scény, ktoré sú spustené. SuperEngine pomaly zastaráva, a preto je snaha o jeho nahradenie Unreal Engine-om 4, čomu sa venuje aj táto práca.

3 Unreal Engine 4

Unreal Engine 4 je herné jadro vyvíjané spoločnosťou Epic Games. Slúži pre vytváranie hier a aplikácií, ktoré siahajú od AAA konzolových hier až po nezávisle mobilné hry. Unreal Engine beží na Windows a Mac operačných systémoch a umožňuje publikovať pre viaceré platformy, ako napríklad Windows, Mac, Playstation 4, Xbox One, iOS, Android, HTML5 a Linux [6].

Funkcionality Unreal Enginu sú rozsiahle. Patrí medzi ne napríklad vykresľovanie a grafika, editor pre dizajn používateľského rozhrania, animačný systém, simulácia fyziky, systém pre audio, editor levelov a mnoho ďalších. Okrem týchto funkcionalít je možné nainštalovať moduly tretích strán a moduly vytvorené rozsiahlou komunitou, ktorá je okolo tohoto jadra vytvorená. Keďže je Unreal Engine open-source, tak na ňom táto komunita môže taktiež spolupracovať a podieľať sa na jeho ďalšom vývoji.

Aplikácie v Unreal Engine 4 je možné vyvíjať dvoma spôsobmi. Prvým je vizuálne skriptovanie pomocou blueprintov, kde sa logika tvorí pomocou spájania uzlov. Uzly predstavujú procedúry, podmienky, premenné, udalosti a podobne. Avšak pre skriptovanie v blueprintoch nie je dostupná celá funkcionalita Unreal Engine API, preto tvorba logiky iba za pomoci blueprintov nemusí byť vždy dostačujúca. Druhým spôsobom je programovanie v C++, pri ktorom máme k dispozícii celé API Unreal Enginu a môžeme si pridať akúkoľvek inú existujúcu knižnicu. Tieto spôsoby môžeme kombinovať a časť logiky, ktorá je komplexná implementovať v C++, a zvyšnú časť implementovať pomocou blueprintov. Unreal Engine nám taktiež umožňuje sprístupniť C++ kód na použitie z blueprintov.

3.1 Inštalácia Unreal Enginu 4

Inštalácia Unreal Enginu je možná dvoma spôsobmi, podľa potreby používateľa.

Prvým spôsobom je nainštalovanie Epic games launchera , pomocou ktorého môžeme vybrať verziu enginu a nainštalovať ju. Tento spôsob je vo všeobecnosti jednoduchší a pokiaľ nemáme v úmysle zasahovať do zdrojových kódov Unreal Enginu, tak je to preferovaný spôsob inštalácie. Kroky pre inštaláciu programu sú nasledovne:

1. Vytvorenie Epic Games účtu (pokiaľ ho ešte nemáte).
2. Stiahnutie a spustenie inštalačného súbora pre Epic Games Launcher
3. Prihlásenie do Epic Games Launchera
4. Nainštalovanie Unreal Enginu

Tieto kroky sú podrobnejšie opísané na oficiálnej stránke Unreal Enginu [7]

Pokiaľ potrebujeme pre našu aplikáciu robiť zmeny v zdrojovom kóde, tak je nutné zvoliť druhý spôsob. Tým je naklonovanie GitHub repozitára a kompilácia zdrojových kódov. Viac o tom ako získať prístup a naklonovať repozitár sa dozviete na oficiálnej stránke [8].

3.2 nDisplay

Systémy, ako napríklad Powerwall displej alebo CAVE využívajú viaceré zariadenia pre zobrazovanie, aby vtiahli používateľa, čo najefektívnejšie do herného prostredia. NDisplay je modul Unreal Enginu, ktorý slúži na vykresľovanie scény na viaceré synchronizované zobrazovacie zariadenia a podporuje tým tieto systémy. Tento modul adresuje najdôležitejšie výzvy súčasného zobrazovania na viaceré obrazovky [4]:

- Zjednodušuje proces nasadenia a spúšťania viacerých inštancií projektu na rôznych počítačoch po sieti, ktoré zobrazujú obraz na jednej alebo viacerých obrazovkách.
- Stará sa o výpočet zobrazovacích matíc pre každú obrazovku, v každom snímku, podľa rozloženia zariadení v priestore.
- Zaručuje, že obsah zobrazený na rôznych obrazovkách ostane presne synchronizovaný, s deterministickým obsahom naprieč všetkými inštanciami.

- Ponúka pasívne a aktívne stereoskopické zobrazenie.
- Môže využiť vstup zo sledovacích zariadení, aby zmenil bod pohľadu v hre tak, aby nasledoval bod pohľadu používateľa v reálnom svete.
- Podporuje akýkoľvek počet obrazoviek, v akomkoľvek relatívnom usporiadaní a je jednoducho použiteľný naprieč niekoľkými projektmi.

3.2.1 Architektúra

Každé usporiadanie nDisplay musí mať jeden riadiaci (master) počítač a ľubovolný počet ďalších počítačov.

Na každom počítači zadefinovanom v sieti beží jedna alebo viac inštancií aplikácie.

Každá inštancia aplikácie riadi vykresľovanie na jednu alebo viacero obrazoviek alebo projektorov. Pre každé zobrazovacie zariadenie, inštancia aplikácie vykreslí jeden pohľad tej istej 3D scény. Nastavením týchto pohľadov tak, aby ich pozícia v 3D svete zodpovedala fyzickej pozícii obrazoviek v reálnom svete, vytvoríme pre diváka ilúziu, že sa nachádza vo virtuálnom svete.

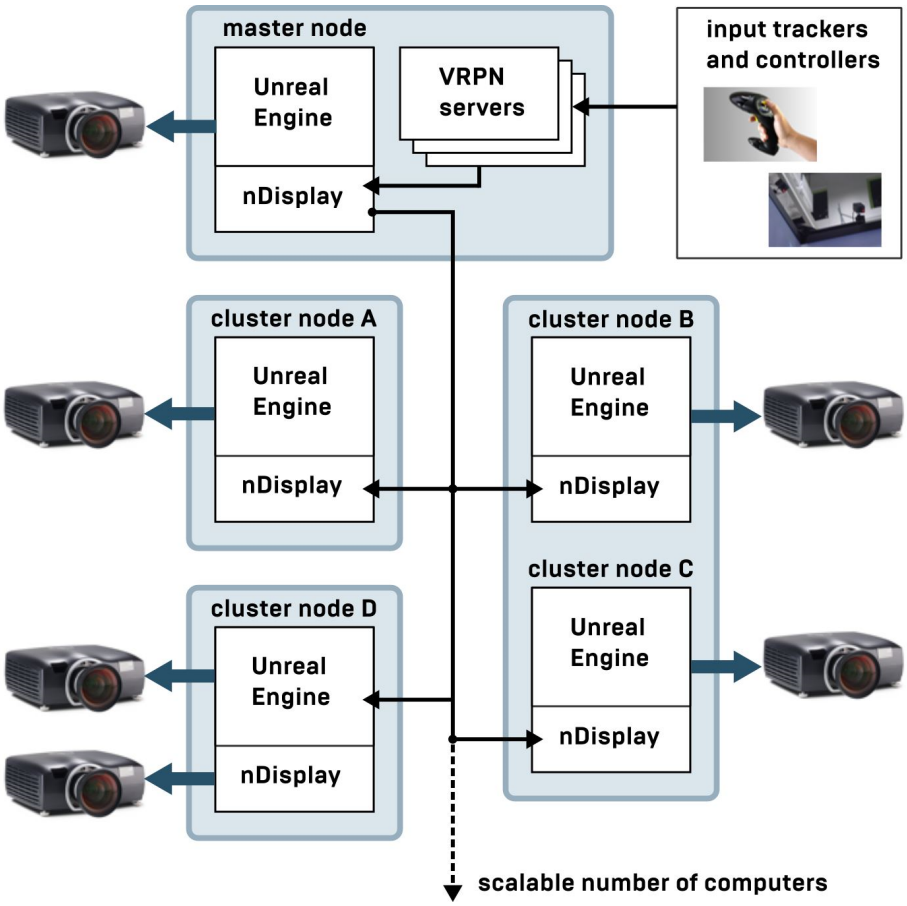
Master je zodpovedný za vstup zo sledovacích zariadení a ovládačov, ktoré sú pripojené cez VRPN a replikovanie vstupov ostatným pripojeným počítačom.

Na obr. 3.1 môžeme vidieť jedno z možných usporiadaní nDisplay siete. Jeden počítač sa správa ako master, ktorý prijíma od VRPN servera vstup zo zariadení. Vstupy následne rozosiela po sieti ostatným uzlom v sieti [9].

3.2.2 Komponenty

NDisplay pridáva niekoľko komponentov do projektu v Unreal Engine:

- Zásuvný modul, ktorý pracuje v Unreal Engine. Komunikuje a synchronizuje informácie medzi všetkými inštanciami aplikácie, ktoré tvoria systém. Zaisťuje, že každá inštancia vykresľuje rovnaký snímok v rovnakom čase, a že každá obrazovka vykresľuje správny pohľad virtuálneho sveta.
- Aplikácia *nDisplayLauncher*, ktorá slúži na jednotné zapínanie a vypínanie inštancií na všetkých počítačoch. *nDisplayLauncher* (obr. 3.2) sa spúšťa na jednom počítači, jeho výhodou je, že môže byť spustený na počítači, ktorý nie



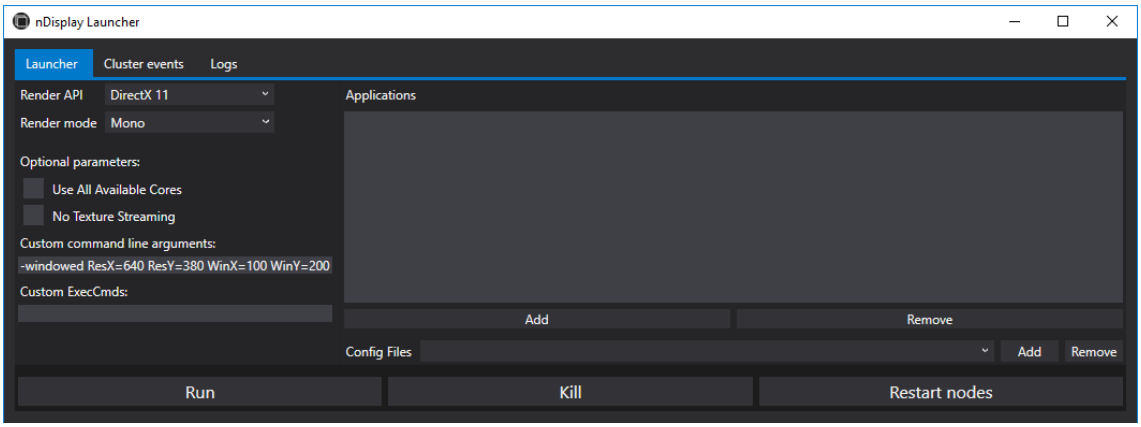
Obr. 3.1: Architektúra nDisplay [9].

je súčasťou systému pre vykresľovanie. Jedinou podmienkou je aby bol na rovnakej sieti ako zvyšné počítače.

- *nDisplayListener* je aplikácia, ktorá počúva prichádzajúce požiadavky od *nDisplayLauncher*-a a spracováva ich na lokálnom počítači.
- Zdielaný konfiguračný súbor, ktorý obsahuje nastavenia, ktoré nDisplay potrebuje pre spustenie správnych inštancií na správnych počítačoch, z ktorého každý zobrazuje správny pohľad virtuálneho 3D sveta [9].

3.2.3 Konfigurácia

Pred spustením akejkoľvek aplikácie na systéme je nutné, najprv si vytvoriť konfiguračný súbor. Tento súbor obsahuje počítače, ktoré tvoria sieť, charakteristiky



Obr. 3.2: Používateľske rozhranie pre *nDisplayLauncher* [10].

okien, ktoré chceme vykresliť na počítačoch, časti virtuálneho sveta, ktoré by mali byť vykreslené, vstupy, ktoré chceme akceptovať pre našu aplikáciu a iné.

Každý komponent, ktorý je možné nakonfigurovať v súbore je zapísaný na vlastnom riadku a je identifikovaný reťazcom ID, ktorý mu pridelíme. Identifikátory sa používajú, keď je nutne odkazovať v jednom komponente na iný.

Väčšina komponentov, ktoré sa konfigurujú majú zadefinovanú pozíciu a rotáciu vo virtuálnom priestore. Pozícia a rotácia každého objektu je relatívna k jeho rodičovi. Štandardne je rodič všetkých objektov počiatočný bod, avšak je možne si zadefinovať vlastne body, na ktoré môžeme odkazovať [11].

Všetky parametre, ktoré odkazujú na veľkosť vo virtuálnom priestore alebo reálnom fyzickom priestore sú zadefinované v metroch a stupňoch. Všetky parametre, ktoré odkazujú na veľkosť obrazovky očakávajú hodnoty v pixloch.

Uzol klastra

Každá inštancia, ktorá sa má zobrazovať v systéme, musí byť zadefinovaná ako uzol klastra. Takýto uzol si zadefinujeme pomocou [*cluster_node*] označenia. Uzlu potrebujeme ešte zadefinovať niekoľko povinných parametrov, identifikátor, IP adresu počítača, na ktorom spúšťame inštanciu a referenciu na konfiguráciu okna, ktoré ma používať.

Konfigurácia okna

Definujeme tu vlastnosti pre hlavné okno aplikácie pomocou komponentu [*window*]. Môžeme tu nastaviť veľkosť a pozíciu okna na obrazovke a to, či sa má spustiť

na celú obrazovku, alebo nie. Taktiež je tu nutné uviesť *viewport*, ktorý určuje, na akej pozícii v okne sa ma scéna vykresliť.

Viewport

Konfigurácia viewport-u je vo všeobecnosti rovnaká ako konfigurácia okna, avšak môže sa líšiť v istých prípadoch, keď je nutné posunúť obraz. Napríklad keď chceme nastaviť 2 projektory, ktorých obraz sa má prekrývať.

Obrazovky

Obrazovky sú v konfiguračnom súbore zapísané pomocou [*screen*]. Ich zadefinovaním vravíme Unreal Enginu, aby prispôsoboval pohľad súčasnej kamery, tak aby zodpovedal uloženiu obrazovky na ktorej sa zobrazuje.

Pre lepšie logické zadefinovanie virtuálneho priestoru, môžeme nastaviť parameter parent, aby odkazoval na uzol scény.

Kamera

Všetky inštancie v nDisplay klastri vykresľujú scénu z rovnakej pozície vo virtuálnom svete. Tieto pozície sa zadefinujú pomocou [*camera*] komponentu.

Počas behu je možne prepínať medzi týmito kamerami z blueprintov, alebo z C++ kódu. Kamery môžu byť navyše poháňané sledovacím zariadením.

Uzly scény

V konfiguračnom súbore môžeme definovať hierarchiu uzlov scény, z ktorých každý predstavuje transformáciu v 3D priestore. Definuju sa pomocou komponentu [*scene_node*]. Čokoľvek, čo v konfiguračnom súbore vyžaduje polohu a rotáciu v 3D priestore, ako napríklad [*camera*] alebo [*screen*], môže použiť jednu z týchto konfigurácií uzlov ako svojho rodiča. To nám môže pomôcť definovať priestorové vzťahy medzi všetkými komponentami vizualizačného systému.

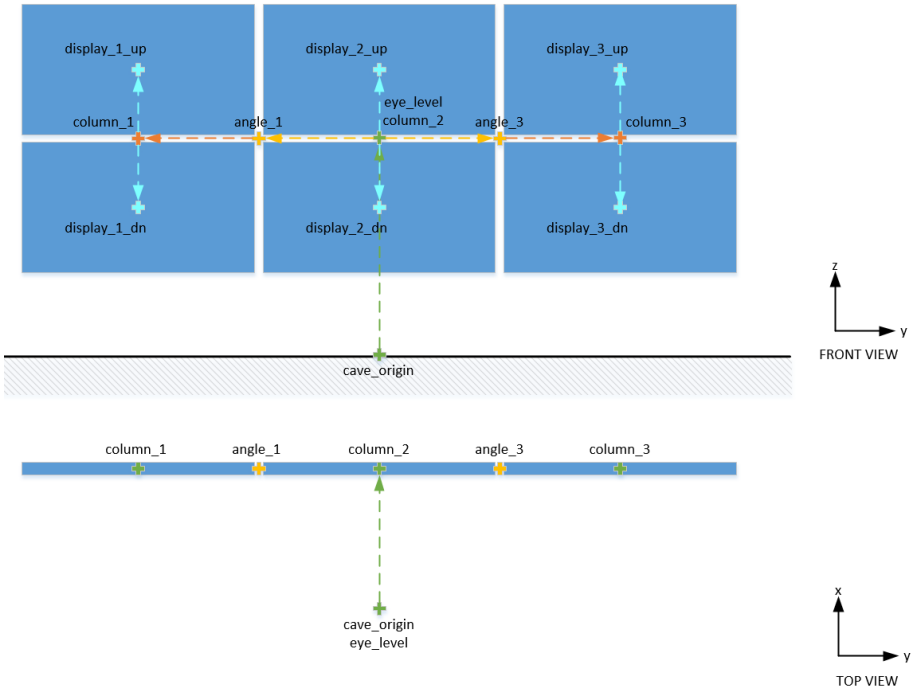
Vstupné zariadenia

Je potrebné zadefinovať [*input*] komponent pre každé zariadenie, ktoré chceme využívať pre vstup do systému nDisplay. Napríklad každá kamera a každý uzol

scény môžu byť riadené sledovacím zariadením VR, ktoré sme nastavili ako input a na ktoré odkazujeme v konfigurácii kamery alebo uzla scény. Alternatívne môžeme nastaviť, aby vstupné zariadenia posielali generické vstupné udalosti do systému Unreal Engine. a vstupné hodnoty naviazali na blueprinty, na ktoré môžeme reagovať v herných skriptoch nášho projektu.

Príklad konfigurácie

Ukážkové konfiguračné súbory môžeme nájsť v priečinku *src/Templates/TP_nDisplay-/Content/ExampleConfigs* na priloženom optickom disku. Jeden z týchto súborov je napríklad *wall_flat_3x2.cfg*, ktorý definuje šesť obrazoviek usporiadaných v troch stĺpcoch (Obr. 3.3).



Obr. 3.3: Vizualizácia konfigurácie uzlov scény [11].

Práca s týmto modulom je jednoducho pochopiteľná a vytvorenie projektu, ktorý ho používa je rovnako jednoduché. Stačí si vybrať šablónu nDisplay pri vytváraní projektu a modul sa nám automatický nastaví. Taktiež sa nám pri vytváraní spustiteľných súborov do výstupného priečinka automaticky nakopírujú aplikácie nDisplayLauncher a nDisplayListener pre spúšťanie.

Pokiaľ chceme pridať nDisplay do projektu, ktorý je už vytvorený, tak je po-

trebne povoliť plugin v Unreal Editore v záložke *Edit -> Plugins*. Následne môžeme v nastaveniach projektu povoliť používanie plugin, a to v záložke *Edit -> Project Settings* z hlavného menu a sekcie *Plugins -> nDisplay*. Následne je ešte potrebne reštartovať Unreal Editor a po tomto kroku sme pripravený vyvíjať aplikáciu pre systém z viacerými obrazovkami.

4 Využitie herných enginov

V tejto kapitole sa zameriame na analýzu využitia herných enginov vo vedeckom prostredí. Najprv si povieme, čo vo všeobecnosti znamená herný engine, a čo by mal obsahovať. Predstavíme si niekolko projektov z rôznych oblastí, ktoré pre implementáciu využívaju rôzne herné enginy a povieme si prečo je ich využitie v konkrétnych prípadoch výhodné.

4.1 Herný engine

Herný engine označuje sadu vývojových nástrojov, ktoré pomáhajú pri tvorbe hier, ale nešpecifikujú priamo hernú logiku, správanie hry, ani jej prostredie. Hlavná funkcionalita spočíva v spracovaní herných vstupov, vykresľovaní 2D aj 3D grafiky, prehrávaní zvuku, sieťovej komunikácii a simulácii fyziky a kolízii [12].

Moderné herné enginy túto funkcionalitu rozširujú o ďalšie veci, ako napríklad podpora virtuálnej reality, čím tvoria vysoko funkčné nástroje, ktoré je možné využiť na tvorbu aplikácií s rôznym využitím, ako napríklad využitie pri tvorbe filmov.

Modulárna štruktúra herných enginov (Obr. 4.1) pozostáva z niekoľkých prepojených časti. Úplne na vrchu sú virtuálne svety, s ktorými môže používateľ interagovať. Virtuálne svety sú v širokej škále vzhľadu a pravidiel interakcie, môžu napríklad obsahovať rôzne typy simulovanej fyziky.

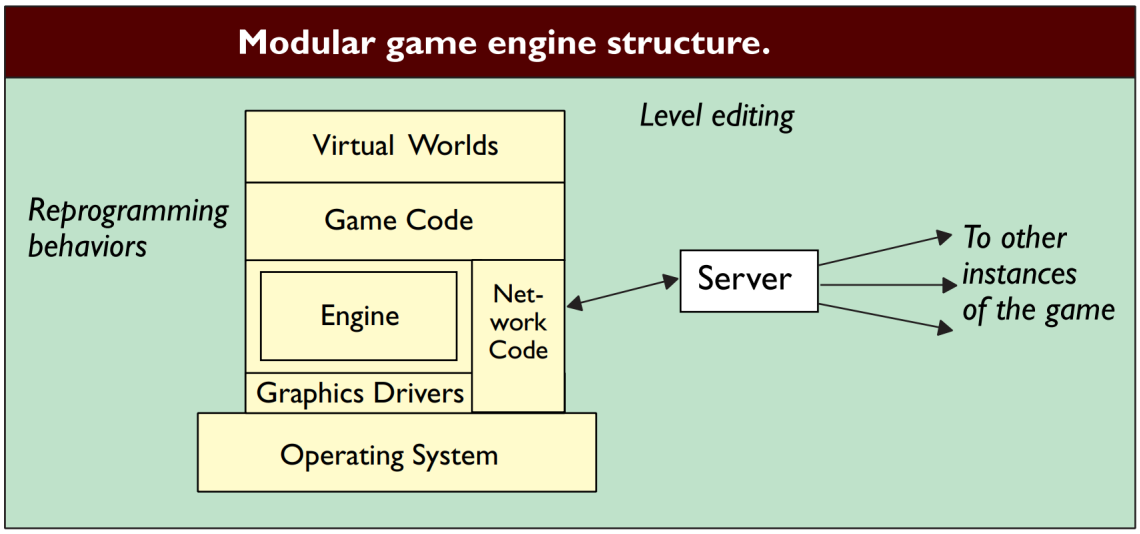
O stupeň nižšie sa nachádza herný kód, ktorý sa stará o základne herné mechaniky, ako napríklad fyzika, parametre zobrazovania, sieťovú komunikáciu a základne akcie a animácie. Tento stupeň sa vo všeobecnosti môže meniť za pomoci špecifického skriptovacieho jazyka.

Zobrazovací engine (Rendering engine) je základ herného enginu, ktorý sa stará o zobrazovanie pohľadu hráča, na základe 3D modelu prostredia. V niekto-

rých herných enginoch je možné meniť aj túto časť (napríklad Unreal Engine 4), avšak vyžaduje si to detailnú znalosť technológie enginu.

Sieťová komunikácia umožňuje viacerým používateľom interakciu s tým istým virtuálnym prostredím. Jeden počítač v tomto prípade slúži ako server, ku ktorému sa pripájajú používatelia (architektúra klient-server [13]). Server je samostatný proces, ktorý uchováva informácie o virtuálnom svete. Komunikuje s hernými klientmi, aby zachytil globálne informácie o prostredí, interakcii medzi hráčmi a synchronizoval. Každý hráč má vlastný pohľad na zdieľaný virtuálny svet. Ak niektorý hráč zmení stav, niektorého objektu, tak sa tento stav musí zmeniť rovnako pre všetkých hráčov

Grafické ovládače prekladajú všeobecné pokyny od zobrazovacieho enginu do nižšej grafickej knižnici. Využívajú rozhranie pre programovanie aplikácií (API), ako napríklad DirectX, OpenGl a iné. Vďaka nim môžeme dosiahnuť rôzne typy zobrazení, ktoré su špecifické pre rôzne použitia, ako napríklad CAVE a Head-mounted display (HMD) [12].



Obr. 4.1: Modulárna štruktúra herných enginov [12].

4.2 GEARS

GEARS(Game-Engine-Assisted Research platform for Scientific computing) je vizualizačný framework, ktorý slúži na simulovanie a skúmanie údajov v prostredí virtuálnej reality. Pomocou 3D modelov tvorí interaktívne zobrazovanie vopred

vypočítaných výsledkov. Používateľovi to umožňuje priamu a rýchlu vizualizáciu stavu dát o simulácii materiálov alebo molekulárnych štruktúr.

GEARS využíva Unreal Engine 4 a Unity, ktoré im ponúkajú zjednodušený vývoj pre VR headset. Jedná z aplikácii, ktorú vytvorili pre VR headset je interaktívne zobrazovanie predpočítaných výsledkov, ktoré slúži na skúmanie dát. 3D objekty sú vytvorené pomocou aplikácii VMD a Blender. Objekty môžu následne byť pridané do scén, pomocou príslušných skriptov.

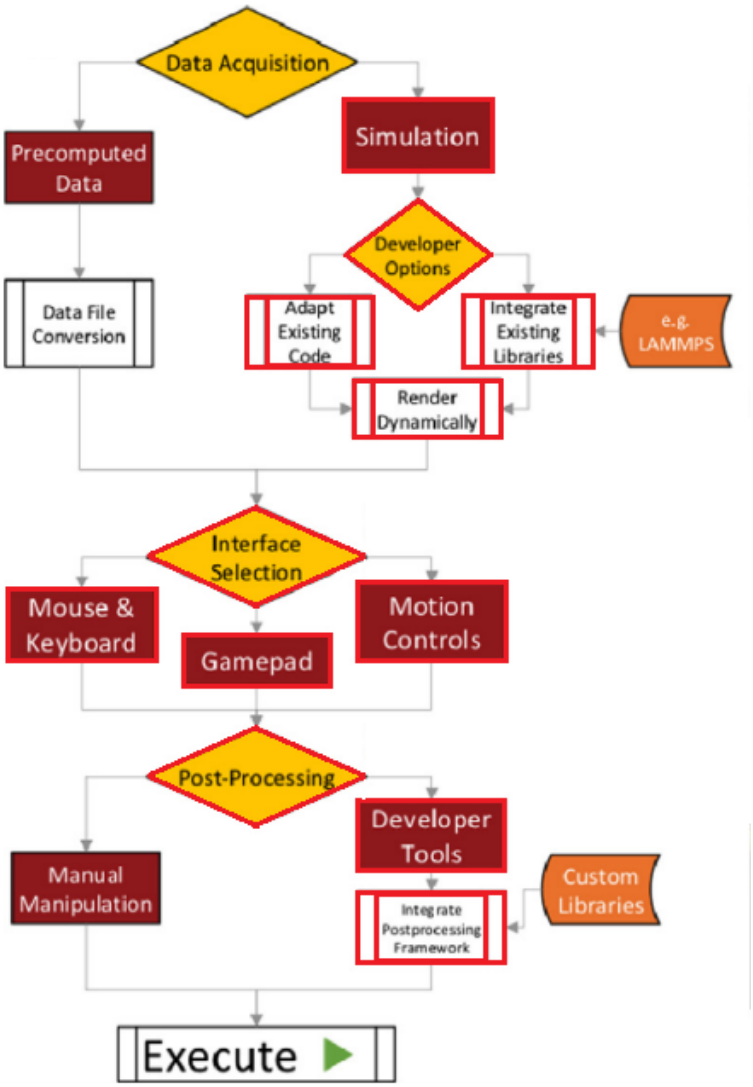
GEARS tiež využíva možnosti programovania, ktoré sú poskytnuté hernými enginmi, ako napríklad podpora pre *C#* a *JavaScript* v Unity, alebo *C++* v Unreal Engine, aby uľahčili použitie existujúcich simulačných kódov. Vytvára sa tým ďalšia funkcionalita GEARS, ktorou je simulácia v reálnom čase. V závislosti od veľkosti údajov a zložitosti simulácie, je možne simuláciu vykonať v dvoch modoch:

- *Run-when-ready*, ktorý volá simulačné jadro pri každom snímku obrazovky, aby posunul stav simulácie.
- *Render-when-ready*, ktorý využíva viac vlákien, kde jedno je zodpovedné za simulačné jadro a beh simulácie, zatiaľ čo druhé má na starosti iba zobrazovanie údajov.

Vizualizačný postup pre GEARS je rozdelený do troch hlavných krokov (Obr. 4.2):

- Zber dát, kde sa buď dodajú výsledky v určitom formáte, alebo vypočítajú pomocou simulácie na mieste.
- Výber rozhrania, ktoré umožňuje interakciu s dátami, alebo simuláciou.
- Následne spracovanie (Post-processing), v ktorom si môže používateľ vybrať jednu z vopred dostupných techník, alebo využiť vlastný kód.

V každom z týchto krokov sa aspoň čiastočne využíva herné jadro, pričom vďaka hernému jadru je možné adaptovať a rozširovať aplikáciu o nové zobrazenie, vďaka čomu môže mať všeobecnejšie použitie [14].



Obr. 4.2: Tri hlavne kroky vizualizačného postupu GEARS. Kroky, ktoré su implementované za pomoci herného jadra sú vyznačené červenou [14].

4.3 Unrealty

Unrealty je aplikácia postavená na Unreal Engine 4, ktorá sa zameriava na dizajn, vizualizáciu a prezentáciu komerčných nehnuteľnosti. Využitie virtuálnej reality pre komerčné nehnuteľnosti malo v minulosti mnoho nedostatkov, ktoré ich robili príliš drahé a časovo náročne pre komerčné využitie [15]. V koncepte Unrealty je ale popísane, ako by využitie Unreal Enginu ako nástroja pre dizajn umožnilo využitie virtuálnej reality v sfére predaja nehnuteľnosti.

Autor postupne adresuje pripomienky a ponúka riešenia, v ktorých využíva Unreal Engine.

Nedostatočné prijatie

Tradične chýba modelový balík, ktorý je k dispozícii dizajnérom od fázy koncepcie. Toto je jedna zo silných stránok Unreal Enginu.

Unreal engine poskytuje kompletné riešenie pre vývoj hry. Editor úrovní UnrealEd je úplne integrovaný so zobrazovacím nástrojom a spolu s rozšíriteľným *C++* jadrom, vysoko-úrovňovým skriptovacím rozhraním UnrealScript, vizuálnou editáciou objektov a povrchov vo virtuálnom svete, poskytuje okamžité, priame a intuitívnejšie ovládanie v trojrozmernej, nástrojovej súprave pre tvorbu svetov, ktorá zodpovedá zložitosti dnešných CAD softvérov.

Nedostatočná odozva

Odozva medzi konceptom a vizualizáciou dizajnov neumožňovala priamu spätnú väzbu v procese tvorby dizajnu, čo tvorilo VR neužitočné v tomto aspekte.

Jednou z najsilnejších stránok Unreal enginu ako nástroja pre návrh je integrácia s editorom levelov UnrealEd. UnrealEd je návrhový nástroj, ktorý je optimalizovaný na vytváranie 3D prostredí v reálnom čase. Je integrovaný so zobrazovacím modulom Unreal Enginu, čím ponúka pohľad kamery a okamžité zobrazenie všetkých operácií osvetlenia, umiestnenia textúry a geometrie. UnrealEd ponúka tiež možnosť uprostred procesu návrhu spustiť hru. Dizajnér tak môže spustiť Unrealty a prechádzať sa po jeho budove v reálnom čase.

Problémy s úrovňou detailov

Campbell a Wells zistili, že pri vývoji modelu,počas toho ako sa zložitosť zvyšuje so zvyšovaním úrovne realizmu (textúry, osvetlenie, abstraktné prvky atď.), predlžuje sa čas vykresľovania. To je pre prezentačné účely neprijateľné [15].

Unreal Engine netrpí týmito problémami, pretože dokáže zvládnuť viac ako 60 000 polygónov v jednej úrovni aj s textúrami a osvetlením. Unreal dokáže zvládnuť obrovské priestory. Unreal Engine má tiež natívnu podporu pre API DirectX, OpenGL, Glide a PowerVR.

Problémy s prezentáciou

Použitie 3D technológie na prezentáciu virtuálnych budov bolo tradične brzdené dlhou dobou vykresľovania a neinteraktivitou vopred vykreslenej trasy. Vykreslenie týchto preddefinovaných tras niekedy trvalo niekoľko dní až týždeň, v závislosti od zložitosti sekvencie. V tom čase to nebolo možné kvôli nákladom a zložitosti prepravy VR vybavenia, ktoré používali [15]. V dnešnej dobe by to tak ale nemalo byť.

Unreal Engine bol navrhnutý tak, aby bežal na dnešných domácich počítačoch, s hardvérovou akceleráciou OpenGL alebo DirectX. Vzhľadom na nízke náklady na systém, ktorý dokáže spustiť Unreal Engine by mala byť možnosť prehliadky virtuálnej budovy v reálnom čase bežná udalosť.

Schopnosť vytvárať ultra-realistické virtuálne svety, spojená s technológiou pre prezentovanie a spolupracovanie na rovnakých svetoch odkiaľkoľvek dáva konceptu Unrealty veľkú výhodu oproti samostatným nástrojom na dizajn [16].

4.4 Využitie herného enginu pre plánovanie evakuácie

Simulácie založené na virtuálnej realite si vyžadujú dobrú grafickú vizualizáciu v 3D. Na dosiahnutie realistických výsledkov musia brať do úvahy aj účinky fyzikálnych zákonov, ako sú gravitácia a kolízie. Herné enginy spĺňajú požiadavky na simuláciu vo VR.

Herný engine sa v tomto prípade využíva pre simuláciu evakuácie v núdzových situáciach. Pre účely simulácie bol vytvorený 3D model reálnej budovy inštitútu jadrového inžinierstva v Brazílii [17]. Textúry boli pridané v UnrealEd nástroji na základe fotiek reálneho prostredia budovy.

Výkon aplikácie je vylepšený vďaka analýze 3D scény s podporou z-bufferov. Na základe polohy kamery sa identifikujú povrchy a objekty v blízkosti kamery a priradí sa im vyššia kvalita vykresľovania a súčasne sa zníži kvalita povrchov a objektov vzdialených od kamery.

Unreal Engine bol rozšírený o funkcionality, ktoré boli špecifické pre túto aplikáciu, ako napríklad meranie času evakuácie, zníženie rýchlosti hráča, aby zodpovedala rýchlosti v reálnom svete a pridania používateľského rozhrania.

Výsledkom tohoto výskumu bolo aj porovnanie virtuálnych simulácii evakuácie oproti reálnym simuláciám. Časy pre evakuáciu boli veľmi podobné v prospech virtuálnych simulácii.

4.5 CaveUT

CaveUT je softvér pre CAVE systémy, založený na Unreal Tournament hernom engine. CaveUT modifikuje Unreal Tournament, tak aby bol schopný zobrazovať aplikácie na viacerých obrazovkách. Aplikácie vyvinuté pomocou CaveUT obsahujú všetky možnosti Unreal Enginu [18].

Obsahuje implementáciu stereoskopie v reálnom čase, sledovanie rúk a hlavy používateľa a synchronizáciu vykresľovania. Implementácia CaveUT ukazuje, že pokročilé funkcie herných jadier, pre vizualizáciu a interakciu, umožňujú adaptáciu pre špecificke požiadavky virtuálnych prostredí.

4.5.1 Funkcionalita riešenia

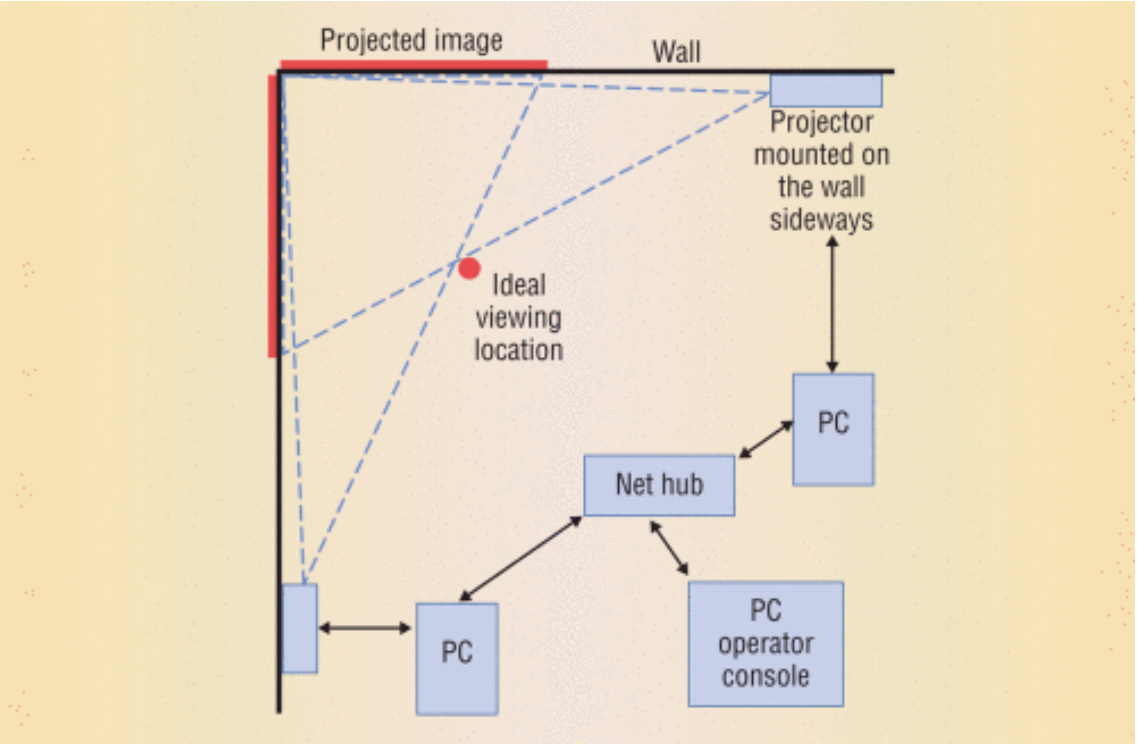
Riešenie CaveUT je navrhnuté tak, aby využilo možnosti Unreal Tournament, ktorý je čiastočne open-source. Unreal Tournament využíva jadro Unreal, ktoré zodpovedné za vykresľovanie grafiky, animácie, fyziku, sieťové prepojenie a interpreter, ktorý podporuje Unreal Script programovací jazyk.

Systém si vyžaduje jeden serverový počítač, ktorý je cez LAN sieť pripojený, ku viacerým klientským počítačom. Každý klient zobrazuje na jednu obrazovku.

Aplikácia sa spúšťa ako hra viacerých hráčov, kde je server normálny hráč, ktorého môžeme ovládať. Každý klient je pripojený ako divák, pričom duplikuje hráčov pohľad a otáča ho tak, aby zobrazoval pohľad zodpovedajúci uloženiu projekčných obrazoviek. Obr. 4.3 ukazuje otočenie jedného obrazu o 45 stupňov vľavo a druhého o 45 stupňov vpravo.

4.5.2 Vykresľovanie obrazu

Na opravu perspektívy sa využíva knižnica VRGL od Willema de Jonge, čo je OpenGL modifikované pre aplikácie virtuálnej reality. Pre inštaláciu bez sledovania používateľa sa musí zadefinovať jedna pozícia. Pokiaľ sa hlava hráča nachádza dosť blízko tejto pozície, tak pohľad ostane spojitý a neskreslený. Ak sa využíva



Obr. 4.3: Schéma V-Cave inštalácie. V tejto inštalácii CaveUT sú oba obrazy otočené o 45 stupňov [18].

sledovanie používateľa, tak perspektíva sa opravuje v reálnom čase, tak aby nasledovala hráča.

O dobrý výkon CaveUT sa stará najmä Unreal Tournament, ktorý zaisťuje, ze každý počítač zachováva kompletnú inštanciu virtuálneho sveta a vykonáva vlastne vykresľovanie grafiky, simuláciu fyziky, animácie a súvisiace operácie. Klient-server komunikácia, sa tak skladá z rýchlych a jednoduchých aktualizácii zmeny stavov. Napríklad hráč je na tomto mieste, pohybuje sa týmto smerom a touto rýchlosťou.

Synchronizácia vykresľovania obrazu

Zaťaž jednotlivých klientskych počítačov môže byt odlišná podla toho, ktorú časť scény vykresľujú. To ovplyvňuje kvalitu vykresleného obrazu a aj používateľsky zážitok. CaveUT tento problém rieši pridaním jednoduchého swaplock servera, ktorý beží na serverovom počítači, popri hernom serveri.

Swaplock server začne vyslaním správy *"ready"* klientským počítačom. Signalizuje tým každému klientovi, že majú čakať na správu *"render"* predtým ako

zobrazia súčasný vykreslený snímok. Každý klient pošle swaplock serveru signál *"ready"*. Server počká, kým príjme signál od všetkých počítačov, a následne odošle signál *"render"*.

Sledovanie pozície

CaveUT podporuje sledovanie pozície v reálnom čase. Sledovanie hlavy hráča umožňuje nastavenie stabilného zobrazenia virtuálneho sveta, čím umožňuje hráčovi volný pohyb v priestore jaskyne.

Spracovanie vstupov z periférnych zariadení, joystick, herný ovládač, ale aj sledovací systém, sa využíva Virtual Reality Peripheral Network (VRPN) [19]. VRPN server normalizuje údaje z periférnych zariadení a odosiela ich cez UDP port všetkým klientom, ktorí ho príjmu pomocou VRPN klienta a pošlú ho do VRGL časti, ktorá použije údaje pre úpravu perspektívy v reálnom čase.

4.5.3 Využitie CaveUT

Flexibilnosť a konfiguračné možnosti CaveUT umožňujú využiť tento systém v rôznych usporiadaniach obrazoviek a dokáže stabilne podporiť minimálne 32 rôznych zobrazení pre jednu aplikáciu. Od vydania tohoto systému bolo vytvorených niekoľko projektov, ktoré ho využívajú.

V-Cave

V-Cave je jednoduchý CAVE systém, ktorý má dve steny, medzi ktorými je uhoľ 90 stupňov a na ktoré sa pomocou projektorov premieta obraz [20].

V-Cave bol vytvorený ako demonštrácia využitia CaveUT, ktorý je prenosný a vytvorený pomocou 3 počítačov, 2 projektorov a 2 plátien na zobrazovanie.

ALTERNE

ALTERNE je projekt, ktorý sa zameriava na vývoj platformy pre tvorbu virtuálno-realitných umeleckých inštalácií. Hlavným cieľom je vytvoriť technologickú platformu pre dizajn prostredí alternatívnej reality, v ktorej môžeme napríklad meniť zákony fyziky [21].

Softvérová platforma ALTERNE sa skladá z troch komponentov:

1. Unreal Tournament, ako jadro pre vizualizáciu a interakciu medzi objektami.

2. CaveUT, pre podporu stereoskopickej vizualizácie a sledovanie pozície.

3. Jadro pre alternatívnu realitu, ktoré umožňuje prepísať natívne jadro fyziky, pre špecifické kategórie objektov.

Pre túto platformu je možné vytvoriť rôzne prostredia virtuálnej reality, ktoré využívajú všetky funkcionality CaveUT.

4.6 Projekcia na kupolu pre živý koncert

Festival Pharos od Childish Gambina v roku 2018 bol trojdňovým pohlcujúcim hudobným podujatím, na ktorom sa predstavili vystúpenia pod nafukovacou kupolou s dĺžkou 160 stôp. Počas celého podujatia bol na nafukovaciu kupolu premietaný obraz generovaný v reálnom čase, ktorý súvisel s piesňami účinkujúcich.

Architektúra systému, na ktorej pracovali Weta Digital a 2n Design, bola ovplyvnená najmä požiadavkami na vysokú vizuálnu vierohodnosť premietaného obrazu. Pre pokrytie vnútorného povrchu kupoly bol určený cieľ 60 snímkov za sekundu v 5k rozlíšení. Tento cieľ je ťažko dosiahnuteľný pre väčšinu systémov. Preto sa Weta a 2n spojili s Epic Games, aby sa zaoberali využitím ich nDisplay technológie v rámci Unreal Enginu. Keďže nDisplay bol navrhnutý hlavne pre CAVE systémy a Powerwall displeje, Weta a 2n spolupracovali na zdokonalení systému, aby zvládol požiadavky Pharos festivalu. Toto umožnilo vykresľovanie s uzamknutým počtom snímkov, distribuované na piatich strojoch s *NVIDIA p6000* a *Quadro Sync II* grafickými kartami. Každý stroj bol zodpovedný za vykreslenie jednej časti mapy. Výsledný obraz s vysokým rozlíšením, ktorý bol distribuovaný medzi 12 projektorov v kupole, môžeme vidieť na Obr. 4.4 [22].

Festival Pharos patrí medzi prvé komerčné využitie nDisplay technológie. Dôležitú úlohu zohral nDisplay najmä pri synchronizácii obrazov vykresľovaných na piatich strojoch. Avšak mnohé funkcie nDisplay-u, ktoré sú špecifickejšie pre CAVE systémy, ako napríklad sledovanie polohy a stereoskopické zobrazenie v tomto riešení neboli využité.

Obr. 4.4: Projekcia na kupolu [22].

4.7 Unreal Engine pre LIRKIS CAVE

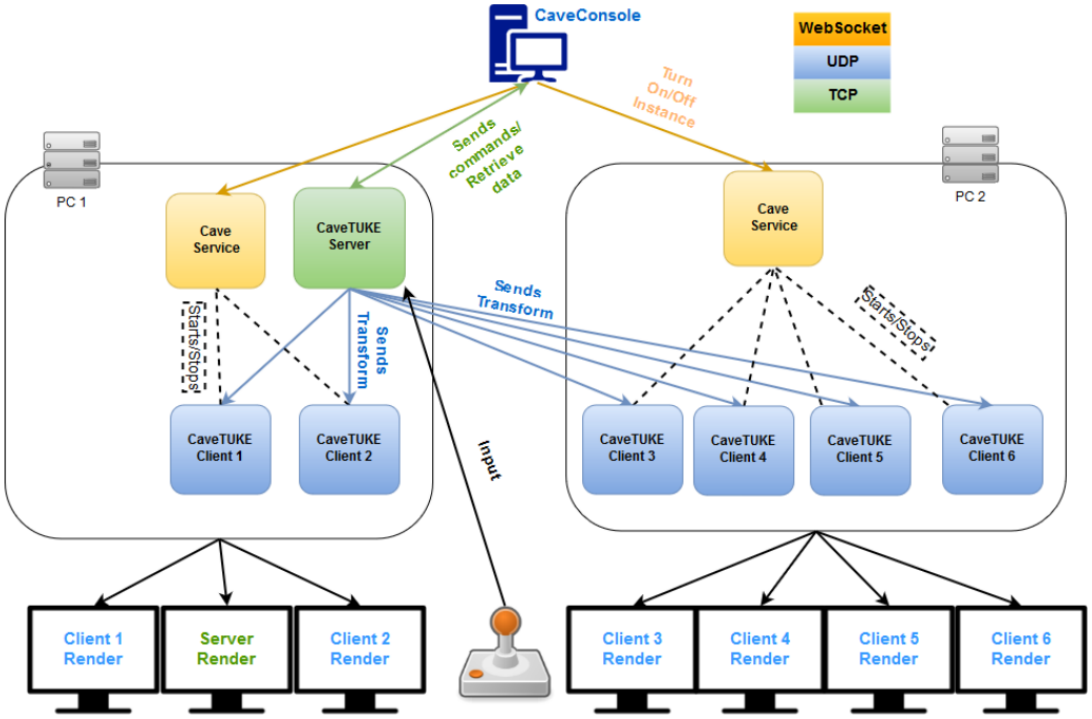
Vizualizačné jadro pre LIRKIS CAVE na báze Unreal Engine 4 vyvinul vo svojej práci [2] M. Gabriška v roku 2018. Systém vychádzal z predchádzajúceho prototypu SyncMulti 3 [1], ktorý bol vyvinutý ako prípadová štúdia, v ktorej sa dokázalo, že Unreal Engine 4 je vhodný pre účely CAVE systému. Oproti SyncMulti 3 adresuje problémy ako sieťovú synchronizáciu, konfiguráciu a administráciu spustenej aplikácie .

4.7.1 Architektúra riešenia

Riešenie sa skladá z troch modulov:

- CaveTUKE, vizualizačný modul systému.
- CaveConsole, ktorý slúži pre administráciu systému.
- CaveService, ktorý vytvára spojenie s CaveConsole a spúšťa CaveTUKE inštancie s príslušnymi argumentmi.

Prehľad týchto modulov a celej architektúry je zobrazený na diagrame .



Obr. 4.5: Architektúra riešenia [2].

CaveTUKE

Je hlavným modulom celého systému vytvorený za pomoci upraveného jadra Unreal vo verzii 4.19. Pozostáva zo serverovej a klientskej aplikácie, ktoré sú pripojené k tej istej hre viacerých hráčov. Server je implementovaný ako tzv. listen server, ktorý slúži zároveň ako klient, ktorý zobrazuje jednu inštanciu. Kvôli výkonu je server jediná inštancia, ktorá prijíma vstupy zo zariadení. Ostatné inštancie majú vstup zo zariadení explicitne zakázaný. Pre každú obrazovku je jedna inštancia aplikácie, ktorá sa stará o vizualizáciu. Jeden počítač má súčasne spustené tri alebo štyri inštancie, podľa počtu obrazoviek. Alternatívou by bola jedna inštancia pre každý počítač, ktorá by siahala cez niekoľko obrazoviek, čo by umožnilo synchronizáciu FPS, avšak Unreal Engine, nemá pre túto funkcionalitu dostatočnú podporu v API. Pre synchronizáciu FPS sa miesto toho využije nastavenie konštantnej hodnoty frekvencie snímkov.

Pre replikáciu údajov sa využíva komunikácia pomocou protokolu UDP. Po-

mocou neho sa replikujú údaje o pozícii hlavnej kamery. Každá inštancia si následne vypočítava ešte vlastnú pozíciu kamery, na základe ofsetu, ktorý je možné počas behu meniť v CaveConsole module.

Stereoskopické zobrazenie je v tejto verzii dostupné pomocou parametra *-emulatestero*. Kvôli nemožnosti modifikovať projekčnú maticu zobrazenia nebolo možné vytvoriť mimo-osové zobrazenie bez modifikácie jadra. Po modifikácii jadra bolo implementované riešenie podľa článku od Kooimu [23], avšak mimo-osové zobrazenie bolo stále nestabilné, keďže obsahovalo artefakty zo svetlom a miznúcimi objektami. Preto sa miesto neho zvolila alternatíva zobrazenia na osi, kde správna perspektíva bude zobrazená z jedného fixného bodu v LIRKIS CAVE systéme.

CaveConsole

Administrácia systému prebieha pomocou CaveConsole aplikácie s grafickým rozhraním, ktorá je implementovaná pomocou rámca *Qt* [24]. CaveConsole je spustená zo vzdialeného počítača a komunikácia prebieha pomocou TCP. Ako formát správ sa vybral JSON. V rozhraní je možné nakonfigurovať IP adresu a port servera a následne spustiť server. Okrem toho sa v rozhraní nachádza niekoľko záložiek pre konfiguráciu nastavení aplikácie.

Cameras je záložka, ktorá dynamicky mení obsah podľa pripojenia alebo odpojenia inštancií klientov. Po zvolení danej obrazovky je možné zadať transformáciu posunu, ktorá je v tvare *X, Y, Z* a *Pitch, Roll, Yaw*, ďalej je možne meniť FOV danej kamery.

Záložka *Settings* poskytuje nastavenie kvality vizualizácie, medzi ktoré patrí vyhladzovanie hrán, post-processing efekty, vzdialenosť vykresľovania, kvalita textúr, kvalita tieňov, vizuálne efekty a škálovateľnosť. Ďalej je dostupné nastavenie počtu FPS a výber scény, v ktorom sú všetky dostupné scény aplikácie.

Clients je posledná záložka tohoto modulu a sú v nej zobrazene počítače v klastri. Každý klient má zoznam inštancií CaveTUKE, z ich konfiguráciou, ktoré má spúšťať. Každú inštanciu možno zvlášť nakonfigurovať. Medzi parametre konfigurácie patria súradnice obrazovky, kde sa má zobraziť inštancia, či sa ma spúšťať aplikácia na celú obrazovku alebo v okne, či sa má používať stereoskopické zobrazenie a dodatočné argumenty.

Nastavenia sú následne uložené do súboru vo formáte JSON a pokiaľ znova

spustíme aplikáciu, tak sú načítané.

CaveService

Modul CaveService slúži na spúšťanie a vypínanie inštancií CaveTUKE. Aplikácia sa spúšťa pri štarte počítača a počas jeho behu sa pokúša nadviazať spojenie s CaveConsole. Potom ako sa spojenie nadviaže, aplikácia deserializuje JSON správu a na základe údajov zapne alebo vypne inštancie na danom počítači [2].

Riešenie pre LIRKIS CAVE je dostupnou náhradou SuperEnginu, avšak nemá úplnú funkcionalitu, nakoľko nepodporuje mimo-osové zobrazenie. Neoceniteľným pokrokom oproti SuperEnginu je využitie open-source softvéru jadra Unreal.

4.8 nDisplay v LIRKIS CAVE

V bakalárskej práci P. Romanovej [3] bol využitý modul nDisplay pre mimo-osové zobrazenie v LIRKIS CAVE systéme. Počas práce na nasadení nDisplay do LIRKIS CAVE sa autorka potýkala s viacerými problémami. Hlavným z nich bolo nastavenie vstupu zo sledovacieho systému *Optitrack*, na základe ktorého sa zabezpečuje mimo-osové zobrazenie v CAVE systéme. Problém bol vyriešený správnou konfiguráciou *Rigid Body* mena v aplikácii Motive, ktorá slúži na rekonštrukciu dát získavaných z kamier a odosielanie ďalším aplikáciam.

4.8.1 Konfigurovateľný súbor

Pre riešenie LIRKIS CAVE, nevyhovoval žiaden z dostupných konfiguračných súborov, preto bol vytvorený nový konfiguračný súbor. Boli v ňom zadefinované všetky počítače a obrazovky. Pre konfiguráciu obrazoviek (atribút [*screen*]) sa zvolila odporúčaná stratégia tvorby hierarchie scény, kde sú jednotlivé obrazovky zadefinované relatívne, vzhľadom k svojmu rodičovi.

Aplikácie sa spúšťajú a vypínajú centralizovane, z počítača *Console_Cave*, ktorý je na rovnakej sieti ako počítače klastra. Pre prístup počítačov k aplikácii bola vytvorená domáca skupina, ku ktorej má prístup každý počítač v sieti.

4.8.2 Nedostatky riešenia

Medzi nedostatky tohoto riešenia patrí najmä nesprávne zadefinovanie obrazoviek podlahovej a stropovej steny, ktoré nie sú správne zarovnané, čím je vidieť nesprávny prechod objektov na hranách obrazoviek.

Zvyšné obrazovky sa v rámci prechodov zdajú byť zadefinované správne a prechody vyzerajú plynulé. Avšak stereoskopické zobrazenie nedosahuje správny dojem, čo môže byť spôsobené nepresnosťou konfigurácie.

Druhým nedostatkom je potreba spúšťať *nDisplayListener* na každom počítači klastra manuálne. Tieto počítače nemajú pripojené periférne zariadenia, a preto takéto spúšťanie môže trvať aj niekoľko minút, čo je neefektívne. Neefektívnosť sa prehĺbi najmä v prípadoch, kedy nastane neočakávaná chyba a niektorý z počítačov sa vypne,čím je nutné listener spustiť opätovne.

5 Záver analýzy

V prvej časti analýzy sme si v krátkosti opísali herné jadro Unreal, ktoré máme za úlohu využiť pri tvorbe systému pre CAVE. Najdôležitejšou časťou analýzy je predstavenie funkcionality nDisplay, tento modul je od verzie 4.22 Unreal Enginu oficiálne dostupný a uľahčuje tvorbu aplikácii pre systémy viacerých obrazoviek, medzi ktoré patri aj CAVE v LIRKIS laboratóriu, vďaka čomu je vhodný pre riešenie nášho problému.

V druhej časti analýzy sme si predstavili niekoľko systémov, ktoré využívajú herné jadra pre účely, ktoré nemusia súvisieť iba s hrami. Medzi najužitočnejšie sa pre nás javila analýza CaveUT a LIRKIS CAVE, ktoré boli čiastočne podobné tým, že mali centrálne spúšťanie zo vzdialeného stroja, čím umožňujú jednoduché spustenie celého systému. Nakoľko je toto jedna z funkcionalít SuperEnginu, musíme to zobrať do úvahy pri návrhu riešenia.

Ďalej sme analyzovali predošlé riešenie, ktoré využíva nDisplay v LIRKIS CAVE. V riešení bol vytvorený konfiguračný súbor, ktorý využijeme, ako začiatočný bod nášho riešenia. Nedostatky konfiguračného súboru boli analyzované a v rámci nášho riešenia budú opravené. Veľkým prínosom z tohoto riešenia pre nás budú vyriešené problémy zo systémom *Optitrack* a aplikáciou Motive, nakoľko konfigurácia mena *Rigid Body* v Motive a VRPN funkcionalita je v oficiálnej dokumentácii nedostatočne opísaná.

6 Návrh a implementácia vizualizačného systému

Súčasné jadro *SuperEngine* ponúka viac, ako iba zobrazovanie scény. Všetky jeho funkcionality by mali byť obsiahnuté aj v novom riešení, aby bolo možné ho nahradiť. Tieto požiadavky sú zosumarizované v tabuľke 6.1.

Na základe týchto požiadaviek je možné navrhnúť vhodnú architektúru riešenia tak, aby zahrňovala každú funkcionalitu.

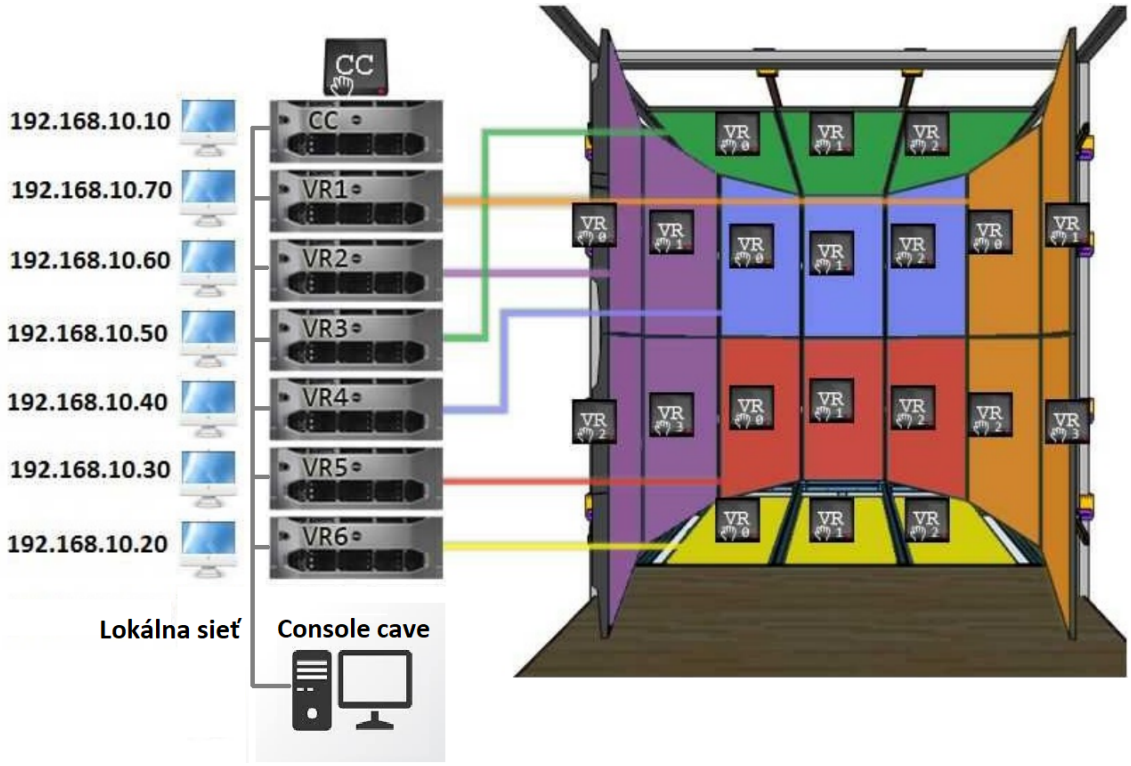
6.1 Architektúra

Z analýzy sme získali poznatky o module nDisplay, Unreal Enginu, ktoré sa javí ako vhodné pre naše riešenie. V tejto časti si predstavíme aplikácie a časti, ktoré budú súčasťou riešenia. Taktiež si popíšeme, na ktorých strojoch budú spustené tieto aplikácie. Usporiadanie počítačov v klastri, ich mená a prípadne ku ktorým monitorom sú pripojené môžeme vidieť na Obr. 6.1. Počítač *Console_Cave*, ktorý je taktiež pripojený k sieti, nie je súčasťou klastra, slúži hlavne ako konzola pre administráciu systému, spúšťanie a vypínanie aplikácií.

Naše riešenie bude pozostávať z *nDisplayLauncher*, *nDisplayListener*, sledovania polohy a konfiguračného súboru. Aplikácia *nDisplayLauncher* bude mať za úlohu zapínať a vypínať aplikácie Unreal Enginu v CAVE systéme, možnosť výberu typu zobrazenia, stereoskopické alebo mono, a výber konfiguračného súboru. *nDisplayLauncher* bude komunikovať cez TCP protokol s *nDisplayListener*-mi, ktoré sú spustené na počítačoch, ktoré boli v konfiguračnom súbore zapísané ako uzly klastra. *nDisplayListener*, následne spustí podľa príkazu od *nDisplayLauncher*-a požadované inštancie Unreal Engine aplikácie. Pri správnej konfigurácií bude nastavený vstup z *Optitrack* systému cez VRPN. Počítač *CC*, na ktorom je spustená aplikácia

ID	Funkcionalita
1	*Vizualizácia*
1.1	Stereoskopia
1.2	Synchronizovaný obraz medzi obrazovkami
1.3	Zobrazenie mimo os, podľa pozície používateľa
2	*Administrácia*
2.1	Centrálne spustenie/vypnutie systému
2.2	Zmena scény
3	*Sledovanie polohy*
3.1	Sledovanie značiek
3.2	Rekonštrukcia dát z kamier
3.3	Posielanie pozície

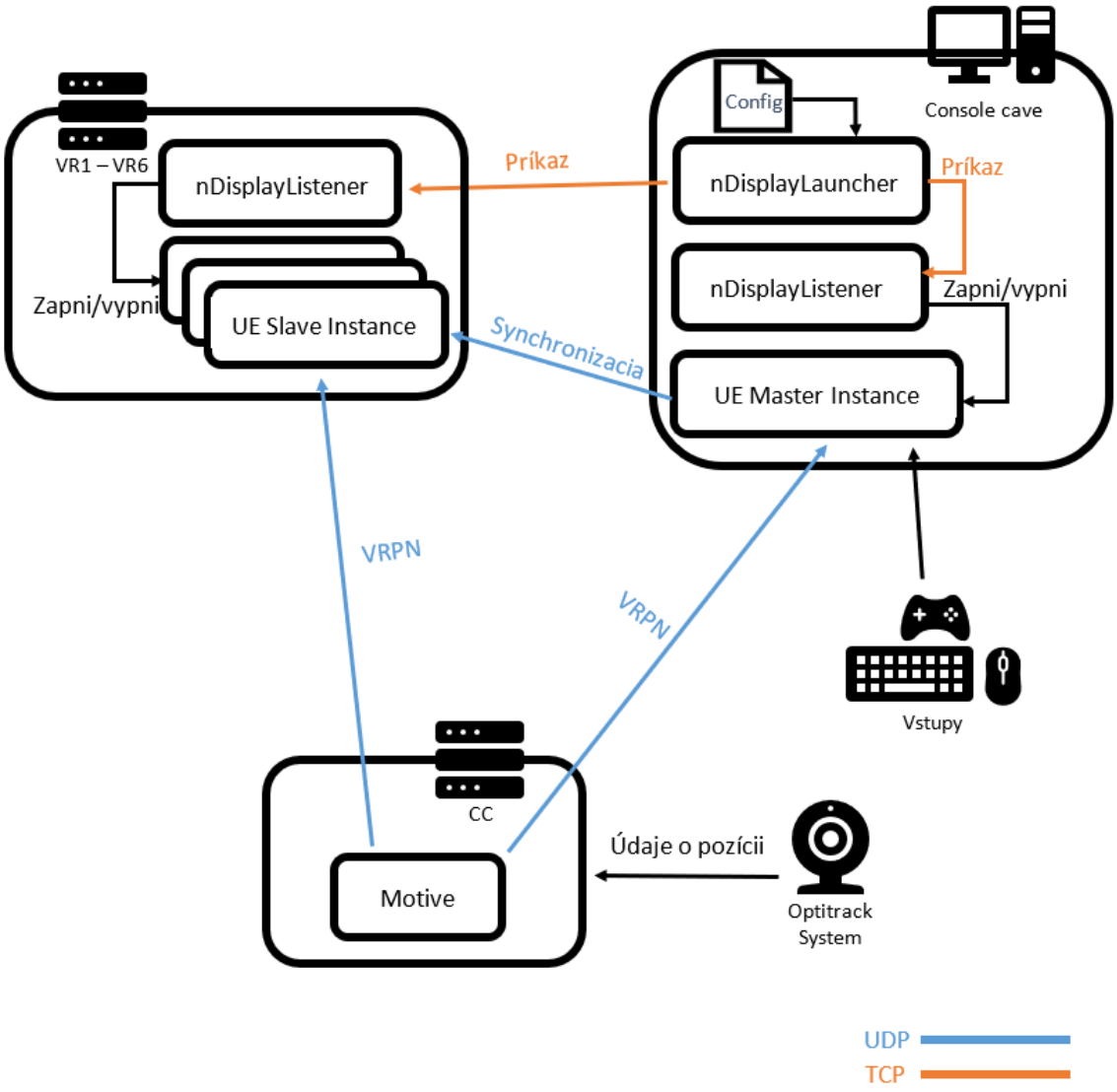
Tabuľka 6.1: Požadované funkcionality



Obr. 6.1: Usporiadanie počítačov v klastri.

Motive, posiela údaje o snímanej polohe pomocou UDP protokolu. Vstupy z klávesnice, myši, herných ovládačov a iných periférnych zariadení sú spracované v

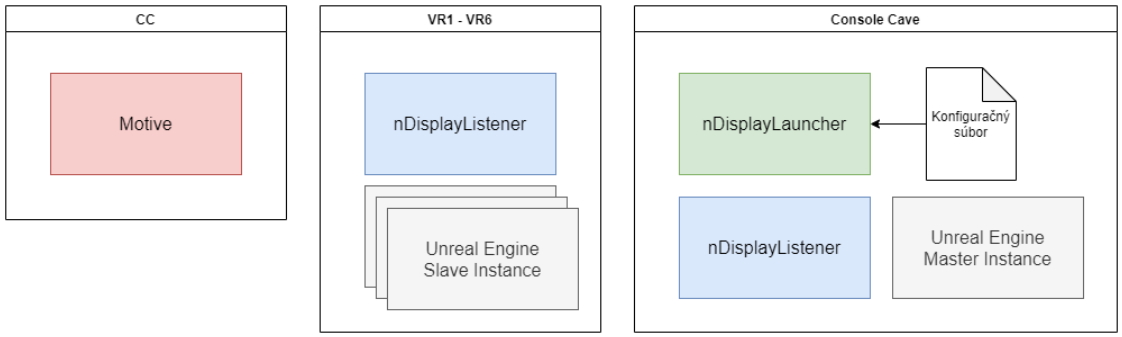
master inštancii aplikácie a do ostatných uzlov je distribuovaný stav scény. Periférne zariadenia preto musia byť pripojené k počítaču, ktorý bude master. Master inštancia je taktiež zodpovedná za synchronizáciu slave inštancii tak, aby bol stav sveta rovnaký vo všetkých uzloch klastra. Prehľad opísanej architektúry a dátových tokov môžeme vidieť na obr. 6.2.



Obr. 6.2: Prehľad architektúry a dátových tokov.

Na diagrame 6.3 môžeme vidieť, ktoré aplikácie budú spustené na jednotlivých počítačoch.

Potom ako budú nakonfigurované moduly, bude možné vytvoriť ukážkovú ap-



Obr. 6.3: Aplikácie spúšťané na počítačoch v klastri.

likáciu pre náš CAVE systém. Aplikácia bude spočívať zo scény, po ktorej sa budeme môcť voľne pohybovať za použitia herného ovládača. Scéna musí pozostávať z prostredia, ktoré má na každej stene rozoznateľne spojité čiary. Ako jednou z implementácii takejto scény by mohla byť napríklad izba budovy, nakoľko v nej sú steny naokolo z každej strany a jasne vieme vidieť a validovať, či prechody medzi obrazovkami sú plynulé.

6.1.1 nDisplayLauncher

NDisplayLauncher je časť nDisplay modulu, ktorá bude slúžiť ako modul pre administráciu. Tento modul ma v sebe zahrnutú funkcionalitu spúšťania a vypínania aplikácií, a zároveň aj zmenu scén, a prípadne konfigurácií systému. Modul budeme spúšťať z počítača *Console_Cave*, ktorý je umiestnený v sieti. Dôvod pre výber tohoto počítača je, že v riešení ktoré využíva SuperEngine sa používa taktiež pre spúšťanie systému. Prechod zo SuperEnginu na Unreal Engine je tak jednoduchý a môže fungovať počas behu systému.

6.1.2 nDisplayListener

Ďalšou časťou bude *nDisplayListener*. Tento listener musí byť spustený na každom stroji, na ktorom bude bežať inštancia aplikácie, inak nebude možné spustenie aplikácie. Počítače, ktoré sa starajú o zobrazovanie sú s menom *VR1-VR6*.

Jednou možnosťou je pred spustením systému manuálne spustiť na každom stroji listener. Táto možnosť však nie je uspokojivá nakoľko jednotlivé počítače, ktoré slúžia pre vykresľovanie nemajú vstupné zariadenia. Druhy dôvod, prečo je toto riešenie nepostačujúce je, že nenaplňuje požiadavku funkcionality na cen-

trálne spustenie systému, nakoľko predtým ako by sme mohli spustiť systém by sme museli spustiť listener-y.

Druhá možnosť ako zabezpečiť beh listenera na každom stroji je, že ho spustíme pri štarte operačného systému, podobne ako to bolo uskutočnené v CaveService module LIRKIS CAVE, ktoré sme si predstavili v analýze. Nakoľko je toto riešenie dostačujúce, a zároveň spĺňa požiadavku na funkcionalitu, je vhodné pre náš systém.

6.1.3 Sledovanie polohy

Sledovanie polohy používateľa je treťou časťou, ktorú musíme zahrnúť do riešenia. Toto sledovanie sa uskutočňuje pomocou systému *OptiTrack*, ktorý následne údaje o pozícii značiek posiela programu *Motive*. Ten tieto údaje rekonštruuje, čím získa pozíciu v priestore. Program *Motive* je spustený na počítači s menom *CC*, ktorý je pripojený k sieti.

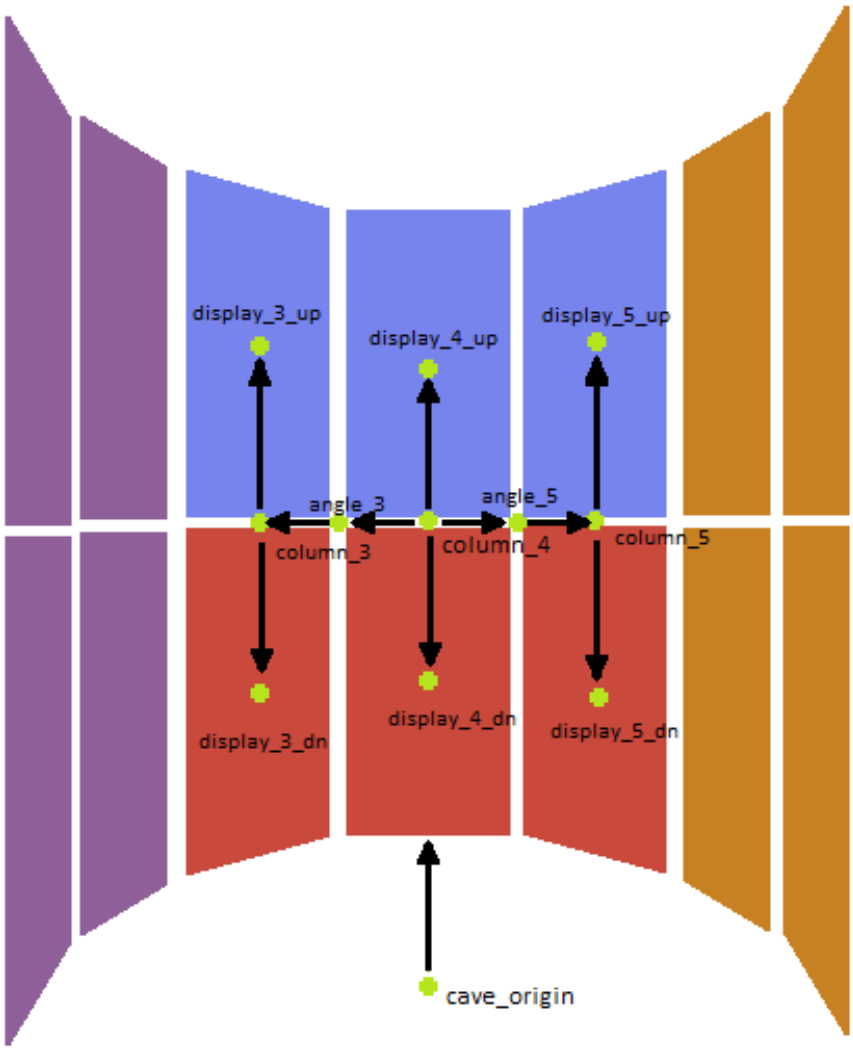
NDisplay má možnosť prijímať vstup zo sledovacích zariadení cez rozhranie VRPN. Aplikácia *Motive* má pre toto rozhranie podporu a je možné z nej vysielať údaje. Preto využijeme túto možnosť a na štandardnom porte 3883 budeme vysielať údaje o pozícii.

6.1.4 Konfigurácia

Konfiguračný súbor systému pre nDisplay je posledná a najdôležitejšia časť, ktorú je nutné implementovať. Tento súbor popisuje relatívne pozície obrazoviek systému, ktoré počítače sú zodpovedné za obrazovky, aké veľké by malo byť okno pre obraz, ktoré počítače sú v sieti, odkiaľ sa máju prijímať vstupy, ako napríklad sledovanie pozície alebo vstupy z klávesnice a iné všeobecné nastavenia.

Pre vytvorenie konfiguračného súboru bude nutné uskutočniť meranie celého systému, aby sme vedeli zistiť pozíciu každej obrazovky. Zadefinovanie hierarchie monitorov a ich pozícií je možné uskutočniť rôznymi spôsobmi. My využijeme spôsob, ktorý je navrhnutý na oficiálnej stranke nDisplay-u [11].

Spôsob spočíva v tom, že na začiatku máme istý začiatočný bod systému, zadefinovaný ako *Scene node* komponent, a od tohoto bodu potom tvoríme nové uzly, ktoré dedia od bodu, ktorý už bol zadefinovaný. Takýmto spôsobom sa tvorí hierarchia, ktorá popisuje systém. Tieto uzly sa následne môžu použiť pre zadefi-



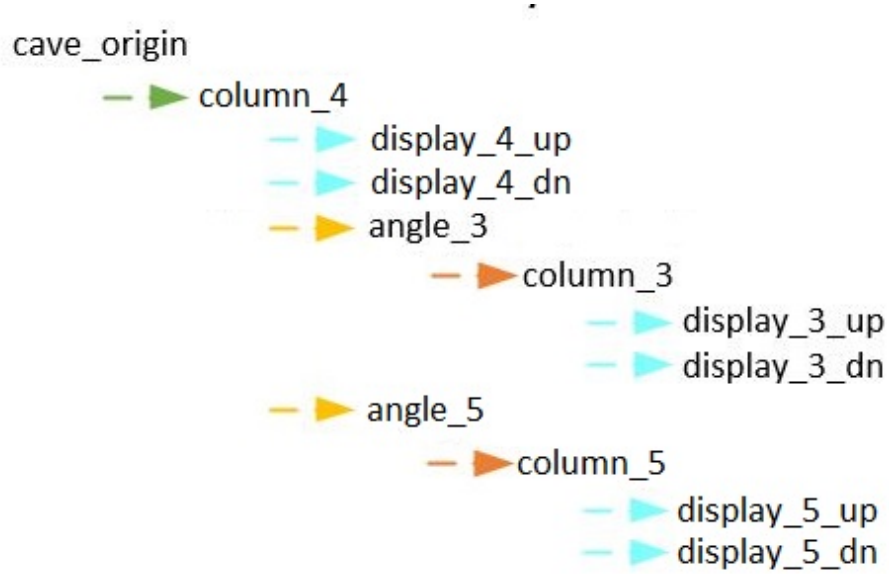
Obr. 6.4: Čiastočna hierarchia pre 6 monitorov.

novanie pozície obrazovky, ktorá sa definuje v komponente *Screen* a vyžaduje si veľkosť obrazovej časti monitora.

Poslednou časťou, ktorú potrebujeme zadefinovať sú komponenty *Cluster node*, ktoré definujú inštanciu aplikácie systému. Je pri nej potrebné zadefinovať IP adresu počítača, na ktorom sa má inštancia spustiť, veľkosť obrazovky, a relatívnu pozíciu v systéme.

Definíciou týchto komponentov si vytvoríme hierarchiu, ktorá popíše náš systém. Logickú hierarchiu si vieme vytvoriť aj bez merania a neskôr do nej dosadiť požadované namerané hodnoty. Ukážku ako bude vyzerať naša hierarchia pre 6 monitorov predného radu uvidíme na Obr. 6.4.

Opis hierarchie je nasledovný: po tom ako si zadefinujeme *cave_origin* sa posunieme na pozícií medzi štvrtým monitorom hore a štvrtým monitorom dole. V tejto pozícii si zadefinujeme stĺpec *column_4* od neho sa posunieme hore o polovicu veľkosti monitora a zadefinujeme si pozíciu *display_4_up*. Analogicky si zadefinujeme pozíciu *display_4_dn*. Od bodu stĺpca sa posunieme vľavo a zadefinujeme si bod uhla *angle_3*. V ňom sa otočíme vľavo o 36 stupňov a zadefinujeme si stĺpec *column_3*. Odtiaľ si analogicky môžeme definovať zvyšné stĺpce a monitory. Pre lepšiu predstavu, je hierarchia zobrazená aj na Obr. 6.5, na ktorom vidno, ktorý uzol je rodič, ku ktorému je nova pozícia relatívna.



Obr. 6.5: Hierarchia uzlov virtuálneho sveta.

6.2 Implementácia vizualizačného systému

Implementácia navrhnutého riešenia bude pozostáva z troch časti:

- Automatizované spúšťanie *nDisplayListener*-ov na všetkých počítačoch klastra pri spustení.
- Nastavenie *Motive*, aby odosielal dáta cez VRPN.
- Vytvorenie konfiguráčneho súboru pre nDisplay.

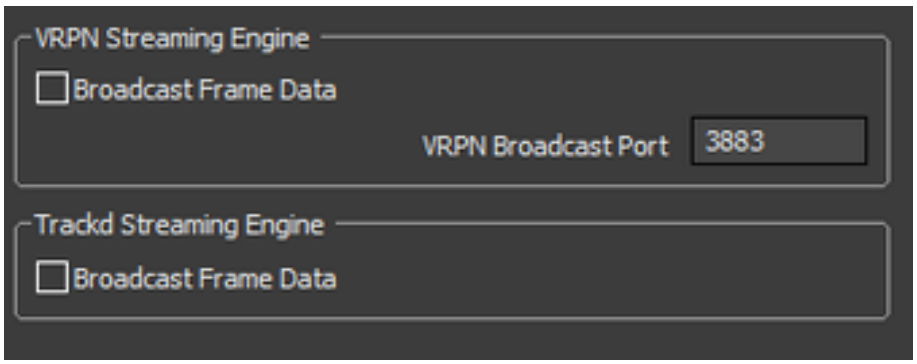
6.2.1 Automatizované spúšťanie nDisplayListener

Aplikácia *nDisplayListener* sa bude spúšťať pri štarte počítačov klastra. Pre prístup k aplikácií bol *nDisplayListener* umiestnený do priečinku, ktorý je v domácej skupine a má k nemu prístup každý počitač klastra, aj *Console_Cave*.

Nastavenie spúšťania aplikácie pri štarte systému je v operačnom systéme Windows 10 pomerne jednoduché. Najprv je nutné vytvoriť si odkaz na aplikáciu, ktorú chceme spúšťať. Následne si otvorime priečinok *Startup*, ktorý sa nachádza na ceste *C:\\Users\meno používateľa\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup* . Otvoriť ho môžeme aj pomocou príkazu *shell:startup*. Nakoniec odkaz na aplikáciu vložíme do tohoto priečinku. Tento postup sme zopakovali pre každý počítač a následne sme reštartom overili funkčnosť spúšťania aplikácie.

6.2.2 Nastavenie Motive

Odosielanie dát cez VRPN sa v aplikácii *Motive* nastavuje v záložke *Streaming pane*. Tam je v časti *VRPN Streaming Engine* potrebné zaškrtnuť položku *Broadcast Frame Data* a nastaviť číslo portu (obr. 6.6). V našom prípade sme ponechali štandardný port 3883.



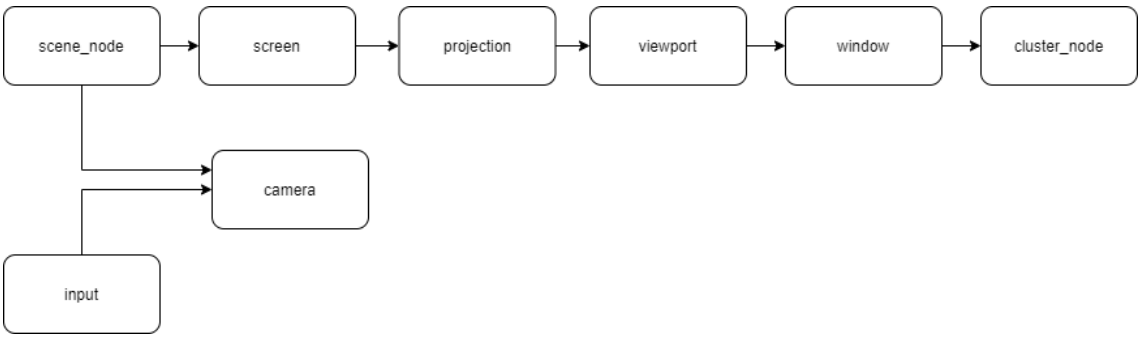
Obr. 6.6: Záložka Streaming pane.

Ďalej sme si v záložke *Assets* vybrali zadefinované rigid body s názvom *Rigid Body 1* a zmenili jeho názov na *Optitrack*. Zmena názvu je potrebná kvôli tomu, že do konfiguračného súboru sa nedá zapísať meno VRPN vstupu, ktoré obsahuje biele znaky. Funkčnosť VRPN sme si overili pomocou konzolového VRPN klienta, ktorý bol spustený z počítača v sieti a nastavený, aby prijímal dáta zo zdroja *Optit-*

rack@192.168.10.10, ktorý zodpovedá *Motive*-u. Po úspešnom pripojení sa prijaté dáta vypisovali do konzoly.

6.2.3 Konfiguračný súbor

Ako počiatočný bod tvorby konfiguračného súboru slúži súbor vytvorený v rámci predošlej práce pre LIRKIS CAVE [3]. Pre účely tejto práce budeme využívať verziu 23 konfiguračného súboru, ktorá zodpovedá Unreal Engine verzii 4.24. Nakoľko je táto verzia pomerne dosť odlišná od verzie 21, ktorá bola využitá v práci P.Romanovej, z konfiguračného súboru využijeme definície *cluster_node*, a ostatné atribúty si zadefinujeme na novo. Niektoré atribúty obsahujú referenciu na iné atribúty, preto ich budeme definovať postupne podľa toho, ako sú na sebe závisle. Závislosť jednotlivých atribútov od iných môžeme vidieť na Obr. 6.7.



Obr. 6.7: Závislosti atribútov.

Stereoskopia a mimo-osové zobrazenie

Pre funkčnosť mimo-osového zobrazenia je potrebné zadefinovať polohu používateľa na vstup. Tá sa definuje pomocou atribútu *input* nasledovne:

```
[input] id=Optitrack type=tracker addr=Optitrack@192.168.10.10
loc="X=0,Y=0,Z=0" rot="P=0,Y=0,R=0" front=-Z right=X up=Y
```

Adresa v parametri *addr* zodpovedá VRPN prúdu, ktorý sme nastavili v *Motive*. Parametre front, right a up určujú ako sa majú mapovať osi z VRPN do pozície v Unreale, pre správne fungovanie mimo-osového zobrazenia je dôležité, aby boli nastavené správne.

Vytvoríme si uzol scény *socket_cam*, ktorého pozícia je určovaná vstupom zo sledovacieho zariadenia. Následne nastavíme, aby atribút *camera* referoval tento

Šírka monitora	0.708 m
Šírka obrazovky	0.682 m
Výška monitora	1.230 m
Výška obrazovky	1.207 m
Dolný okraj	0.012 m
Horný okraj	0.011 m
Ľavý okraj	0.016 m
Pravý okraj	0.010 m

Tabuľka 6.2: Výsledky merania obrazovky

uzol, čím zabezpečíme zmenu pohľadu kamery, v závislosti od pozície používateľa.

```
[scene_node] id=socket_cam loc="X=0,Y=0,Z=0" rot="P=0,Y=0,R=0"
tracker_id=Optitrack tracker_ch=0
[camera] id="camera_dynamic" loc="X=0,Y=0,Z=0" parent=socket_cam
eye_swap="true" eye_dist="0.064"
```

Meranie obrazoviek

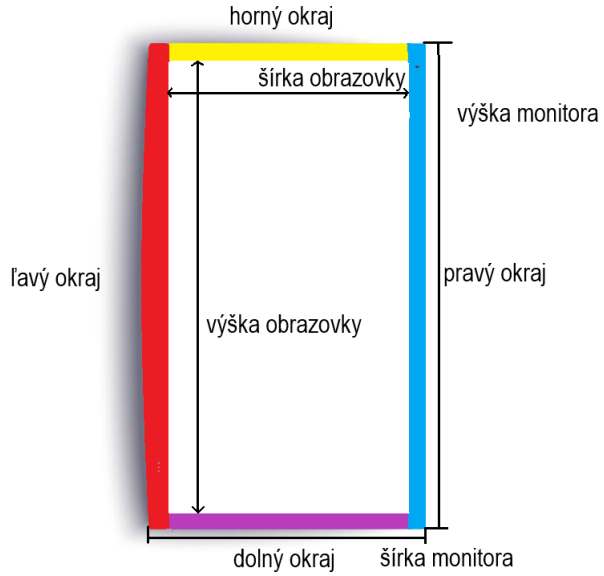
Predtým, ako bude možné vytvoriť hierarchiu obrazoviek pre náš systém je potrebne uskutočniť merania systému.

Podľa stránky výrobcu obrazoviek, ktoré tvoria CAVE systém je výška monitora 1,23 metra a šírka 0,708 metra. Avšak tieto rozmery zodpovedajú celej obrazovke aj s okrajmi, nie len zobrazovacej časti. Preto je nutné odmerať jednotlivé okraje a zistiť tak skutočný stred zobrazovacej časti. Výsledky merania obrazovky je možné vidieť v tabuľke 6.2. Na obr. 6.8 sú rozpísané jednotlivé časti obrazovky.

Okrem rozmerov obrazovky bude potrebné vziať do úvahy aj medzery medzi jednotlivými stĺpcami obrazoviek. Tieto rozmery môžeme vidieť v tabuľke 6.3 a číselné označenie stĺpcov je viditeľné na Obr. 6.9.

Hierarchia uzlov scény

Hierarchiu si budeme určovať podľa postupu definovaného v návrhovej časti. Postupným definovaním uzlov sa dostane k určení stredov všetkých monitorov. Názvy týchto uzlov a ktoré monitory predstavujú môžeme vidieť na Obr. 6.10.



Obr. 6.8: Časti meraného monitora.

Stredovým bodom systému je *cave_origin*.

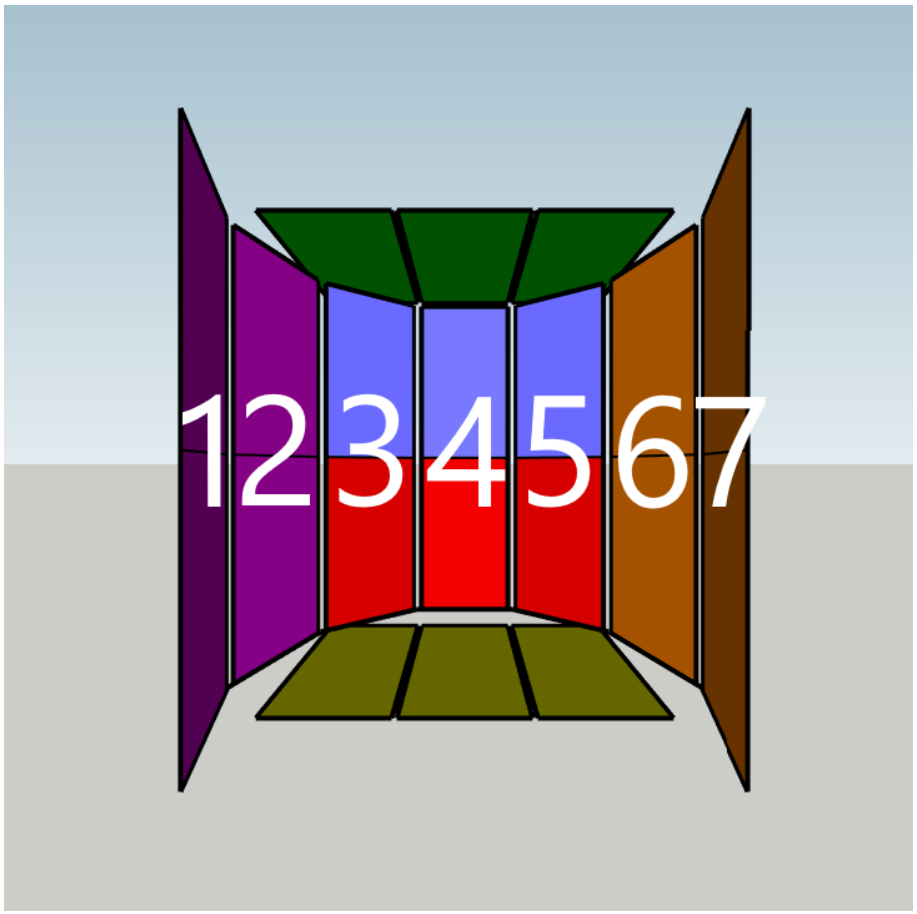
```
[scene_node] id=cave_origin loc="X=0,Y=0,Z=0" rot="P=0,Y=0,R=0"
```

Na neho sa referuje uzol *column_4*, ktorý označuje stredný stĺpec systému. Od stredu systému je posunutý vpred o 1,05 metra a nahor o výšku monitora, teda o 1,23 metra. Je položený v strede medzi hornou a dolnou obrazovkou stĺpca.

```
[scene_node] id=column_4 loc="X=1.05,Y=0,Z=1.23" rot="P=0,Y=0,R=0"
parent=cave_origin
```

ID ľavého stĺpca	ID pravého stĺpca	Medzera
1	2	0.010 m
2	3	0.010 m
3	4	0.010 m
4	5	0.007 m
5	6	0.006 m
6	7	0.010 m

Tabuľka 6.3: Medzery medzi obrazovkami

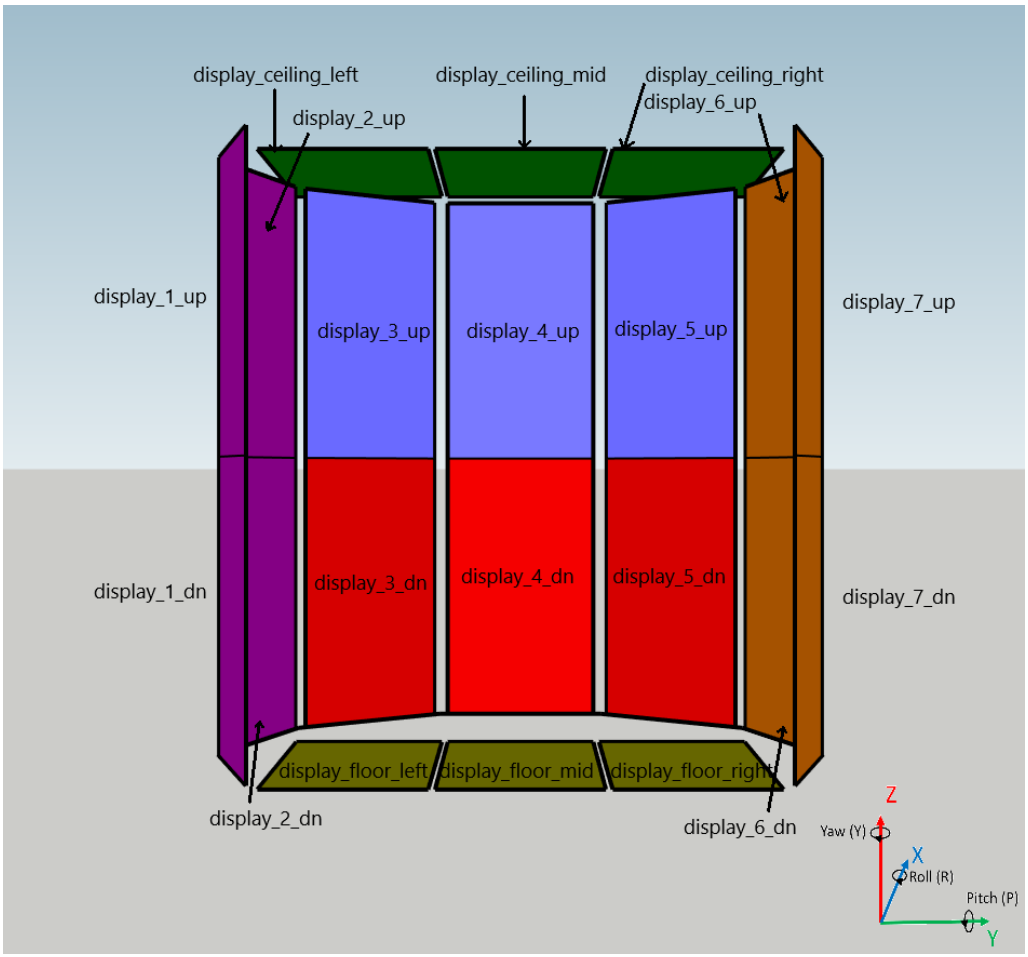


Obr. 6.9: Číselné označenie stĺpcov.

Uzly *display_4_up* a *display_4_dn* označujú stred hornej a dolnej obrazovky, referujú sa na uzol *column_4*, preto je nutné ich voči nemu správne posunúť. Stred hornej obrazovky vypočítame súčtom dolného okraja a polovice výšky obrazovky, t.j `0,012 + 1,207/2 = 0,6155`. Dolná obrazovka má podobný vzorec, akurát pripočítame horný okraj namiesto dolného okraja. Keďže je stred dolnej obrazovky pod bodom *column_4*, musíme dať zápornu hodnotu. Výsledný výpočet potom bude `-(0,011 + 1,207/2) = -0,6145`

```
[scene_node] id=display_4_up loc="X=0,Y=0,Z=0.6155" rot="P=0,Y=0,R=0"
parent=column_4
[scene_node] id=display_4_dn loc="X=0,Y=0,Z=-0.6145" rot="P=0,Y=0,R=0"
parent=column_4
```

Ďalší uzol, ktorý je potrebné zadefinovať je uzol *angle_3*, tento uzol definuje bod, v ktorom sa dotýkajú okraje dvoch obrazoviek. Jeho posun vypočítame súč-



Obr. 6.10: Názvy uzlov označujúcich monitory.

tom polovice šírky obrazovky a ľavého okraja, keďže obrazovka 3 je naľavo od obrazovky 4, voči ktorej je posunutá. K súčtu je potrebné ešte pripočítať medzeru medzi obrazovkami, aby sedeli skutočné rozmery. Ďalej je nutné otočiť obrazovku o uhol 36 stupňov v zápornom smere.

```
[scene_node] id=angle_3 loc="X=0,Y=-0.367,Z=0" rot="P=0,Y=-36,R=0"
parent=column_4
```

Teraz môžeme zadefinovať uzol *column_3*, ktorý zodpovedá stredu tretieho stĺpca obrazoviek. Vypočítame ho súčtom polovice obrazovky a pravého okraja obrazovky.

```
[scene_node] id=column_3 loc="X=0,Y=-0.351,Z=0" rot="P=0,Y=0,R=0"
parent=angle_3
```

Uzly *display_3_up* a *display_3_dn* si definujeme analogicky podľa postupu pri definícií displeja 4, budu relatívne voči uzlu *column_3*. Uzol *display_3_up* posunieme o 0,6155m hore a uzol *display_3_dn* posunieme o 0,6145 dole.

```
[scene_node] id=display_3_up loc="X=0,Y=0,Z=0.6155" rot="P=0,Y=0,R=0"
parent=column_3
[scene_node] id=display_3_dn loc="X=0,Y=0,Z=-0.6154" rot="P=0,Y=0,R=0"
parent=column_3
```

Zadefinovanie prvého a druhého stĺpca obrazoviek bude veľmi podobné definícii tretieho stĺpca. Nakoľko je medzera medzi stĺpcami 1,2 a 2, 3 rovnaká, ako medzera medzi stĺpcami 3 a 4 (tab. 6.3), uzly *angle_2*, *column_2*, *display_2_up* a *display_2_dn* budú zadefinované rovnako ako uzly *angle_3*, *column_3*, *display_3_up* a *display_3_dn*. Jedinou výnimkou bude parameter *parent*, na ktorý sa odkazujú. Identifikátor jednotlivých atribútov sa znížil o jedna, takže keď znížime o jedna identifikátory rodičov, budeme odkazovať na správne uzly. Zadefinovanie uzlov *angle_1*, *column_1*, *display_1_up* a *display_1_dn* bude potom prebiehať rovnako, ako definovanie druhého stĺpca.

```
[scene_node] id=angle_1 ... parent=column_2
[scene_node] id=column_1 ... parent=angle_1
[scene_node] id=display_1_up ... parent=column_1
[scene_node] id=display_1_dn ... parent=column_1

[scene_node] id=angle_2 ... parent=column_3
[scene_node] id=column_2 ... parent=angle_2
[scene_node] id=display_2_up ... parent=column_2
[scene_node] id=display_2_dn ... parent=column_2
```

Tento zjednodušený postup bol možný iba vďaka tomu, že medzery medzi jednotlivými stĺpcami boli rovnaké, ak by medzery boli rôzne, je nutné počítať posun medzi obrazovkami zvlášť pre každý stĺpec.

Stĺpce naľavo od stredného, štvrtého stĺpca máme zadefinované a môžeme začať zapisovať stĺpce napravo. Pre uzly uhlov stĺpcov, ktoré sú napravo od stredu bude platiť vzorec pre výpočet posunutia:
`šírka obrazovky/2 + pravý okraj + medzera medzi stĺpcami`.

Dosadením do vzorca nám vzniknú výsledné hodnoty pre jednotlive uhly stĺpcov, ktoré môžeme dosadiť do uzlov. V uzloch je zadefinovaná aj rotácia o 36 stupňov, ktorá zodpovedá natočeniu monitorov.

```
[scene_node] id=angle_5 loc="X=0,Y=0.358,Z=0" rot="P=0,Y=36,R=0"
parent=column_4
[scene_node] id=angle_6 loc="X=0,Y=0.357,Z=0" rot="P=0,Y=36,R=0"
parent=column_5
[scene_node] id=angle_7 loc="X=0,Y=0.361,Z=0" rot="P=0,Y=36,R=0"
parent=column_6
```

Posunutie pre uzly stĺpcov *column_5*, *column_6* a *column_7* si vypočítame pomocou súčtu ľavého okraja a polovice šírky obrazovky: `0.016 + 0.682/2 = 0.357`. Ako rodiča týchto stĺpcov si nastavíme jemu zodpovedajúci uhol. Pre *column_5* (*6,7*) to bude *angle_5* (*6,7*). Uzly displejov budú definované rovnako pre všetky displeje, nakoľko je nutné ich iba posunúť nahor, alebo nadol od stredu stĺpca. Číslo stĺpca rodiča musí zodpovedať číslu displeja, aby bola hierarchia správne definovaná.

```
[scene_node] id=display_5(6,7)_up loc="X=0,Y=0,Z=0.6155"
rot="P=0,Y=0,R=0" parent=column_5(6,7)
[scene_node] id=display_5(6,7)_dn loc="X=0,Y=0,Z=-0.6145"
rot="P=0,Y=0,R=0" parent=column_5(6,7)
```

Potom ako máme zadefinované všetky vertikálne položené obrazovky, môžeme prejsť na definíciu podlahového a stropného radu obrazoviek.

Rad monitorov, ktorý predstavuje podlahu, je uložený pod priehľadným sklom 0,11m pod stredom jaskyne. Pre jeho definíciu si najprv vytvoríme uzol *angle_floor*, ktorý posunieme dopredu o 1,05m, čím bude zarovnaný so stredným, dolným monitorom. Následne bude posunutý nadol o 0,11 metra a otočený o -90 stupňov, aby zodpovedal horizontálnej polohe monitorov.

```
[scene_node] id=angle_floor loc="X=1.05,Y=0,Z=-0.11" rot="P=-90,Y=0,R=0"
parent=cave_origin
```

Vytvoríme si uzol *column_floor*, ktorý definuje rad podlahy. Jeho pozíciu si nastavíme na stred strednej obrazovky podlahy. Keďže sa odkazuje na *angle_floor*, je

potrebné posunúť sa o polovicu obrazovky a horný okraj nadol, čo zodpovedá 0,6145m.

```
[scene_node] id=column_floor loc="X=0,Y=0,Z=-0.6145" rot="P=0,Y=0,R=0"
parent=angle_floor
```

Uzol *display_floor_left* označuje stred ľavej obrazovky podlahy. Jeho pozícia je relatívna, voči *column_floor*. Posunutie je o súčet šírky monitora a medzery medzi monitormi podlahy. Výsledne posunutie je `-0.719`, pretože smer vľavo je záporný smer osy Y. Medzera medzi stredným a ľavým monitorom je rovnaká ako medzera medzi pravým a stredným monitorom, preto bude posunutie rovnaké, rozdiel bude iba smer posunutia. Pravý monitor je definovaný uzlom *display_floor_left*. Definícia stredného monitora podlahy nie je potrebná, keďže pozícia oproti *column_floor* sa nemení, avšak zaviedli sme ju pre jednotnosť označenia monitorov.

```
[scene_node] id=display_floor_left loc="X=0,Y=-0.719,Z=0"
rot="P=0,Y=0,R=0" parent=column_floor
[scene_node] id=display_floor_mid loc="X=0,Y=0,Z=0"
rot="P=0,Y=0,R=0" parent=column_floor
[scene_node] id=display_floor_right loc="X=0,Y=0.719,Z=0"
rot="P=0,Y=0,R=0" parent=column_floor
```

Ako poslednú časť pre hierarchiu obrazoviek si potrebujeme zadefinovať stropný rad monitorov. Tento postup bude veľmi podobný definícii podlahového radu, nakoľko je tiež uložený horizontálne a má rovnaký počet monitorov. Preto si definujeme uzol *angle_ceiling*, ktorý bude vo vzdialenosti 2,47m od podlahy a bude otočený o 90 stupňov.

```
[scene_node] id=angle_ceiling loc="X=1.05,Y=0,Z=2.435" rot="P=90,Y=0,R=0"
parent=cave_origin
```

Následne sa vytvorí uzol *column_ceiling*, označujúci rad troch monitorov. Jeho pozícia bude nastavená na stred stredného monitora, preto je potrebné ho posunúť o polovicu obrazovky a horný okraj, teda 0,6145m. Od uzla *column_ceiling* môžeme relatívne zadefinovať tri stropné monitory. Ľavý monitor bude definovaný *display_ceiling_left* a bude posunutý vľavo o súčet šírky monitora a medzery medzi monitormi. Toto posunutie je v tomto prípade rovnaké, ako pri ľavom monitore

podlahy, keďže medzera medzi monitormi je rovnaká a veľkosť monitora je konštantná. Pravý monitor je definovaný *display_ceiling_right* a posunutie je vypočítane rovnako, ako pri predošlom monitore. Stredný monitor je definovaný uzlom *display_ceiling_mid* a slúži len ako označenie monitora, pretože oproti jeho rodičovi nemá zmenu pozície.

```
[scene_node] id=column_ceiling loc="X=0,Y=-0.01,Z=0.6154"
rot="P=0,Y=0,R=0" parent=angle_ceiling
[scene_node] id=display_ceiling_left loc="X=0,Y=-0.719,Z=0"
rot="P=0,Y=0,R=0" parent=column_ceiling
[scene_node] id=display_ceiling_mid loc="X=0,Y=0,Z=0"
rot="P=0,Y=0,R=0" parent=column_ceiling
[scene_node] id=display_ceiling_right loc="X=0,Y=0.719,Z=0"
rot="P=0,Y=0,R=0" parent=column_ceiling
```

Atribúty obrazovky, projekcie, pohľadu a okna

Po definícii hierarchie uzlov scény môžeme prejsť na definíciu ostatných atribútov. Atribút *screen* môžeme pridať do konfiguračného súboru, pretože vieme výšku a šírku obrazovky. Údaje o veľkosti zapíšeme v metroch do parametru *size*. Týmto spôsobom si postupne zadefinujeme všetkých 20 obrazoviek, rozdiel medzi jednotlivými obrazovkami bude spočívať iba v parametroch *parent* a *id*. Zavedieme si konvenciu pomenovania obrazoviek tak, aby identifikátor rodiča zodpovedal identifikátoru obrazovky. Obrazovky majú prefix `screen` podobne, ako uzly scény, ktoré označujú stred obrazoviek majú prefix `display`.

```
[screen] id=screen_1_up size="X=0.682,Y=1.207" parent=display_1_up
```

Ďalší atribút, ktorý si zadefinujeme je *projection*, ktorý popisuje ako sa má vypočítavať projekcia. Tento atribút je potrebný najmä pri použití projektorov, ktorých obraz sa prekrýva, v našom prípade použijeme typ projekcie *simple*. Každý atribút *projection* sa musí odkazovať na atribút *screen*. Znova si zavedieme pomenovanie atribútov tak, aby zodpovedalo pomenovaniu obrazovky, na ktorú sa odkazuje. Pridáme si prefix `proj_`.

```
[projection] id="proj_screen_1_up" type="simple" screen="screen_1_up"
```

Atribút pohľadu *viewport* popisuje časť okna inštancie Unreal Enginu. V našom prípade bude každý počítač mať toľko inštancií, koľko má obrazoviek, preto si pre každu obrazovku vytvoríme jeden *viewport*. Parametre *x* a *y* označujú ofset v okne inštancie, v našom prípade budu mať vždy hodnotu *0*. Parametre *width* a *height* označujú šírku a výšku, preto do nich dosadíme rozlíšenie obrazovky. Identifikátor pohľadu bude tvorený podľa identifikátora projekcie, kde prefix *proj_screen*, nahradíme prefixom *vp_node*.

```
[viewport] id=vp_node_1_dn x=0 y=0 width=1080 height=1920
projection="proj_screen_1_up"
```

Atribút okna *window* bude popisovať okno, v ktorom je spustená inštancia Unreal Enginu. Do tohoto atribútu je potrebné vložiť parameter *viewports*, ktorý označuje, aké pohľady majú byť prítomné v okne inštancie. V našom prípade bude pomer medzi oknami a pohľadmi 1 : 1. Identifikátor okna bude podľa identifikátora pohľadu, ktorý je parametrom. Prefix *vp* nahradíme prefixom *wnd*. Parameter *fullscreen*, označujúci režim plnej obrazovky nastavíme na *true*, pretože nechceme aby lišty okien inštancie rušili dojem spojitosti. Parametre *ResX* a *ResY* označujú rozlíšenie okna, ktoré bude rovnaké ako rozlíšenie monitora.

Nakoniec zadefinujeme parametre *WinX* a *WinY*, ktoré označujú pozíciu okna na ploche počítača. Počítače, ktoré vykresľujú na tri monitory v rade majú ako hlavný monitor počítač, ktorý je vľavo. Preto je pre nastavenie okna na pozíciu druhého a tretieho monitora nutné nastaviť *WinX=1080* a *WinX=2160*. V nasledujúcom úryvku súboru je ukážka monitorov, ovládaných počítačom *VR5*

```
[window] id=wnd_node_3_dn viewports=vp_node_3_dn fullscreen=true
ResX="1080" ResY="1920"
[window] id=wnd_node_4_dn viewports=vp_node_4_dn fullscreen=true
ResX="1080" ResY="1920" WinX=1080
[window] id=wnd_node_5_dn viewports=vp_node_5_dn fullscreen=true
ResX="1080" ResY="1920" WinX=2160
```

Pre počítače, ktoré ovládajú štyri monitory je dolný ľavý monitor hlavným monitorom. Pre hlavný monitor nie je nutné zadávať parametre *WinX* a *WinY*, pretože majú predvolenú hodnotu *0*. Dolný pravý počítač musí mať parameter

WinX=1080. Horný ľavý monitor má nastavený parameter *WinY=-1920*. Horný pravý monitor ma nastavené parametre *WinX=1080* a *WinY=-1920*.

Ako posledné atribúty je potrebné definovať *cluster_node*. Predstavujú uzly klastra, jednú inštanciu Unreal Enginu. Dôležitý parameter je *addr*, ktorý označuje IP adresu počítača, na ktorom je spustená inštancia. Jeden uzol musí predstavovať master inštanciu, ktorá sa definuje pomocou parametra *master=true*. V našom prípade nastavíme, aby master inštancia bola spustená na počítači *Console_cave*, aby bolo možné k nemu pripojiť vstupné zariadenia, ako napríklad herný ovládač. Po definovaní uzlov klastra je možné spustiť aplikácie vytvorené pre nDisplay pomocou *nDisplayLauncher* aplikácie a otestovať prechody medzi obrazovkami.

7 Návrh a implementácia scény

Pre správne otestovanie je potrebné vytvoriť ukážkovú scénu, ktorá bude čo najlepšie ukazovať správne a nesprávne prechody a zarovnania v jaskyni. Taktiež chceme importovať vopred vytvorené scény pre Unreal Engine, ktoré ukazujú grafické možnosti enginu.

7.1 Návrh scény

Scéna, ktorú vytvoríme bude založená na ukážkovom projekte pre nDisplay. Scéna v tomto projekte obsahuje niekoľko geometrických objektov. Textúra prostredia obsahuje kolmé čiary, ktoré nám pomôžu pri testovaní, či prechody medzi obrazovkami sú zarovnané, a či mimo-osové zobrazenie funguje správne. Modifikovaná scéna bude obsahovať objekty z každej strany, aby bolo možné na každom monitore otestovať stereoskopické zobrazenie.

Scéna bude obsahovať funkcionalitu manipulácie objektu. Týmto objektom sa bude možné pohybovať po scéne a bude možné meniť tvar objektu, aby sme čo najlepšie otestovali zobrazovanie v jaskyni. Ovládanie objektu bude prispôsobené pre klávesnicu a herný ovládač.

Charakter používateľa bude statický a nebude ním možné pohybovať, ale bude sa môcť otáčať po horizontálnej osi pomocou myši, alebo herného ovládača. Otáčanie po horizontálnej osi je pridané, aby sme mohli vidieť časti scény, ktoré sú za používateľom, nakoľko monitory nie sú na zadnej strane jaskyne. Otáčanie po vertikálnej osi nie je potrebné, pretože pohľady nahor a nadol sú viditeľné na stropných a podlahových monitoroch.

Počas behu aplikácie bude možné zmeniť scénu. Druhou ukážkovou scénou bude scéna *HQ Residential House*, ktorá je voľne dostupná na stiahnutie a použitie v Unreal Engine. Predstavuje demonštráciu domu v americkom štýle s vysokými

detailami textúr a modelov. Do tejto scény pridáme charakter, ktorý bude možné ovládať pomocou klávesnice a myši alebo herného ovládača. Po scéne sa tak budeme môcť pohybovať.

7.2 Implementácia scény

Implementáciu začneme vytvorením projektu. Pri vytváraní projektu vyberieme možnosť *Film, Television and Live Events* a vyberieme šablónu *nDisplay*. Tento projekt automaticky obsahuje zapnutý *nDisplay* plugin.

Po spustení projektu je zobrazená ukážková scéna. Túto scénu sme si upravili tak, aby bola väčšia. Tiež sme pridali viacero objektov, aby sme videli prechody medzi obrazovkami a hĺbku pri stereoskopickom zobrazení. Vkladanie nových objektov do scény sa uskutočňuje presunutím objektu z okna *Content Browser* do okna *Viewport*. Pokiaľ chceme zmeniť pozíciu objektu môžeme ho posunúť priamo v scéne. Druhá možnosť je vybrať objekt, ktorý chceme presunúť a nastaviť vektor *Location* na požadovanú hodnotu.

7.2.1 Pridanie *nDisplay* pluginu do existujúceho projektu

V prípade, že chceme nastaviť existujúci projekt, aby bol spustiteľný ako *nDisplay* projekt, potrebujeme povoliť *nDisplay* plugin. Postup pre nastavenie projektu je nasledovný:

1. V záložke *Edit -> Plugins* vyhľadať *nDisplay* a zaškrtnúť políčko *Enabled*.
2. Prejsť do záložky *Settings -> Project Settings -> Plugins -> nDisplay* a zaškrtnúť políčko *Enabled*.
3. V záložke *Settings -> Project Settings -> Plugins -> Description -> Settings* zaškrtnúť políčko *Use Borderless Window*.
4. Reštartovať Unreal Editor a otvoriť projekt.

Aplikácia, vytvorená s povoleným *nDisplay* pluginom je potom spustiteľná pomocou *nDisplayLauncher*-a.

7.2.2 Pohyb objektom

Ako prvé sme nastavili vstupy od používateľa nastavíme v položke *Settings -> Project Settings -> Engine -> Input*. Pre pohyb a otáčanie sme si vytvorili *Axis mappings*, ktoré označujú vstupy, ktoré majú určitú hodnotu, napríklad posunutie myšou po osi X. Pre akcie, ako napríklad zmena scény, sme vytvorili *Action mappings*, ktoré označujú vstupy, ktoré vysielajú udalosť o tom, že boli stlačené a uvolnené.

Potom ako sme si zadefinovali vstupy, môžeme vytvoriť blueprint skript, ktorý bude odchytávať udalosti týchto vstupov a reagovať na nich. Vytvorili sme si triedu *MoveItems_Character_Blueprint*, ktorá dedí od triedy *Character*. *Character* je trieda, ktorá predstavuje objekt, ktorý sa môže pohybovať. Inštanciu tohoto objektu si vložíme do scény a nastavíme aby pri spustení scény bol hráč automaticky vložený ako vlastník tejto inštancie pomocou parametra *AutoPossesPlayer*. V návrhu sme uviedli, že chceme počas behu vedieť meniť model objektu, ktorý ovládame. Pre tento účel je možné využiť triedu *SwitchActor*, ktorá predstavuje objekt, ktorý má v sebe niekoľko objektov, z ktorých je súčasne viditeľný vždy iba jeden. To, ktorý objekt je viditeľný predstavuje parameter *SelectedOption*.

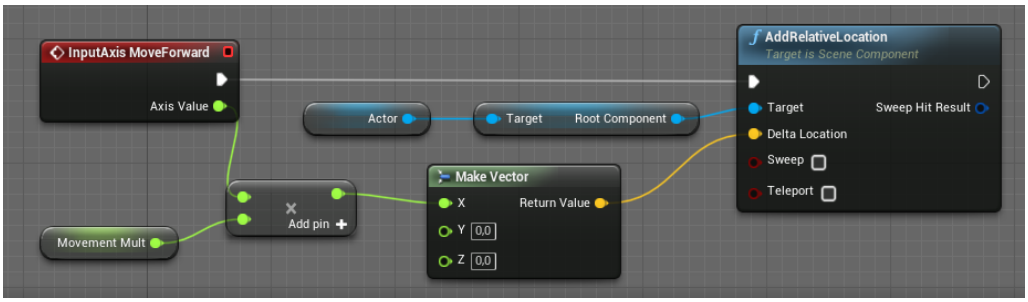
Trieda *MoveItems_Character_Blueprint* obsahuje skript, ktorý reaguje na udalosti a pohybuje objektom. Obsahuje premennú *Actor*, ktorá je referenciou na inštanciu triedy *SwitchActor*, ktorou chceme pohybovať. Do tejto premennej sa pri začatí scény nastaví referencia na inštanciu, aby sa nemusela prehľadávať celá scéna zakaždým, keď budeme chcieť posunúť týmto objektom.

Rýchlosť pohybu objektom je uložená v premennej *movementMult* a môžeme ju upravovať počas behu. Pre lepšiu viditeľnosť bol pridaný aj skript, ktorý zobrazuje súčasnú hodnotu tejto premennej na obrazovke. Štandardne je toto zobrazenie vypnuté, aby nerušilo dojem scény a slúži najmä pre ladenie.

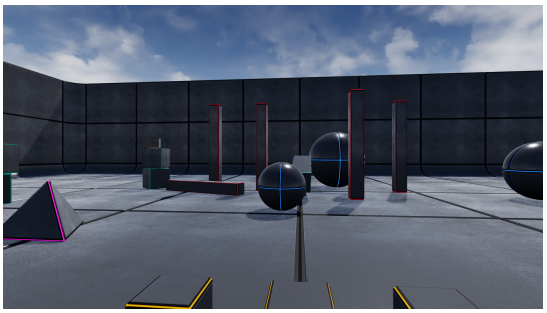
Časť skriptu, ktorý zabezpečuje pohyb objektu je možné vidieť na Obr. 7.1. Hodnotu vstupu vynásobuje o rýchlosť pohybu, následne vytvára vektor, ktorý sa pohybuje v smere dopredu, teda po osi X. Pozícia objektu sa zmení o veľkosť vektora, čím ho posunieme.

7.2.3 Scéna *HQ Residential House*

Existujúcu scénu sme pridali do projektu vložením súborov do priečinku *Content*. Do scény bolo potrebné pridať inštanciu novovytvorenej triedy *Demo_Character_Blueprint*,



Obr. 7.1: Blueprint pre pohyb objektom po osi X.



(a) Scéna s pohyblivým objektom.

(b) Scéna *HQ Residential House.*

Obr. 7.2: Scény testovacieho projektu

ktorá dedí od triedy *Character* a obsahuje skript pre pohyb používateľa. Zmena medzi touto scénou a predošlou scénou je možná počas behu stlačením kláves *Shift+L*, alebo tlačidla *X* na hernom ovládači. Zmena scény je zabezpečená pomocou funkcie Unreal Enginu *OpenLevel*, ktorá na základe parametra nájde scénu a otvorí ju.

Po vytvorení týchto testovacích scén je možné vytvoriť spustiteľnú aplikáciu. Táto aplikácia je spustiteľná pomocou *nDisplayLauncher*-a, ale je možné ju spustiť aj samostatne. Vytvorenie aplikácie je v Unreal Editore v záložke *File -> Package -> Windows -> Windows (64-bit)*.

Výslednú testovacie scénu s pohyblivým objektom môžeme vidieť na Obr. 7.2a a scénu *HQ Residential House* môžeme vidieť na Obr. 7.2b.

Ovládanie a inštalácia scény sú podrobne popísané v Prílohe B: Používateľská príručka.

8 Vyhodnotenie

Vyhodnocovanie riešenia prebiehalo postupným testovaním v rámci CAVE systému. Prvá verzia riešenia bola vytvorená na základe predošlých konfigurácií vytvorených pre LIRKIS CAVE. Do tejto konfigurácie bol pridaný stropný a podlahový rad monitorov. Toto riešenie si vyskúšal aj Ing. Marián Ďuratný z firmy Slovakia Supercomputers, ktorý stál pri vzniku LIRKIS CAVE a softvéru SuperEngine, ktorý slúži ako vizualizačné jadro pre LIRKIS CAVE. Voči tomuto riešeniu mal isté výhrady. Prechody medzi obrazovkami sa zdali pomerne dobré, avšak stereoskopické zobrazenie bolo úplne nefunkčné. V tejto verzii riešenia bolo funkčné získavanie dát z *Optitracku*, ako aj mimo-osové zobrazenie na základe pozície. Spúšťanie systému bolo v tejto verzii stále pomerne náročne, nakoľko listenery bolo potrebne spúšťať na každom počítači zvlášť, čo neskôr viedlo k požiadavke centrálneho spúšťania.

V druhej verzii bolo prvoradé implementovať spúšťanie listenerov pri štarte, aby sa pri ďalšom testovaní zbytočne nestrácal čas. Implementácia prebehla úspešne a overilo sa, že spúšťanie prebieha správne, a že listenery nemajú vedľajšie účinky na jadro SuperEngine, takže môžu byť spustené súčasne. Ďalej sa vykonali prvé merania systému a obrazoviek. Ukázalo sa, že rozmer obrazovky, ktorý sme pôvodne zadávali do konfiguračného súboru bol v skutočnosti rozmer celého monitora. Táto chyba bola dôvodom niektorých nepresností pri prechodoch a nesprávnom stereoskopickom zobrazení.

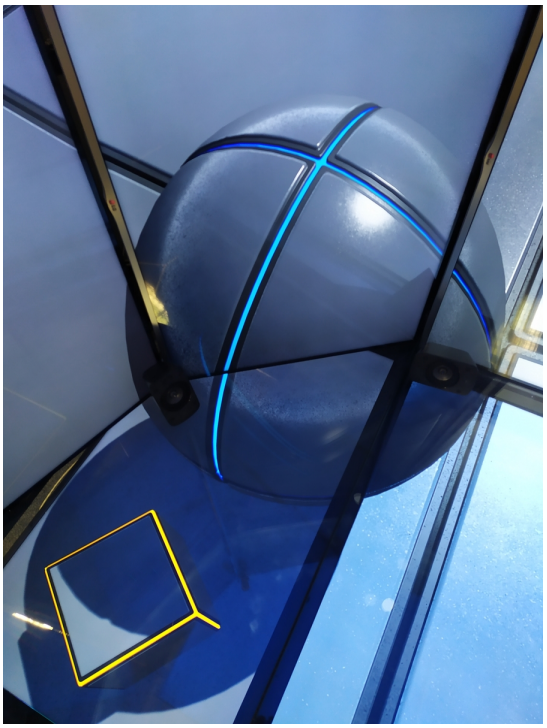
V tretej verzii bol konfiguračný súbor upravený tak, aby zahŕňal vykonané merania obrazoviek. Pri konfigurácii sme brali do úvahy aj medzery medzi jednotlivými obrazovkami, aby sme dosiahli čo najväčšiu vernosť prostredia. Avšak stereoskopické zobrazenie stále nebolo funkčné. Nefunkčnosť stereoskopického zobrazenia nás dohnala k myšlienke, že údaje, ktoré dostáva Unreal Engine od *Motivu* cez VRPN nemusia byť správne. Preto bol do testovacieho projektu pri-

daný odchyt týchto údajov cez blueprint skript. Údaje boli následne vypisované na obrazovku a boli porovnané s údajmi odchytenými priamo z *Motivu*, aby sa dokázalo, že sú rovnaké. Tieto údaje boli rovnaké, avšak popri tomto testovaní sme zistili, že vzdialenosť stredného monitora od stredu jaskyne bola nesprávne určená. Dôvodom, prečo meranie vzdialenosti bolo nesprávne, bol zle určený stred jaskyne. Po pridaní výpisu údajov o pozícii sme stred jaskyne určili presne a zistili sme, že pôvodná hodnota 1,25m pre vzdialenosť stredného monitora musí byť upravená na hodnotu 1,05m.

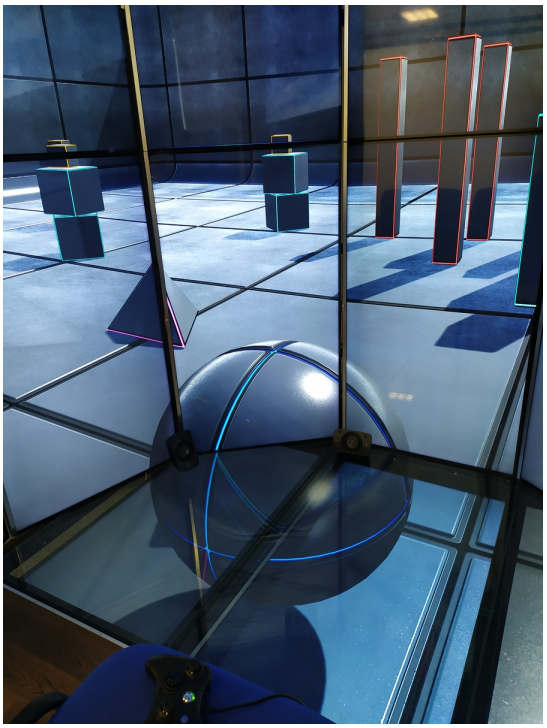
Následne, keď stereoskopické zobrazenie bolo funkčné, bolo možné porovnať ho so stereoskopickým zobrazením vizualizačného jadra SuperEngine. V oboch prípadoch bolo jasne vidieť hĺbku medzi objektami, ktoré sú v rôznej vzdialenosti. Ukážková scéna v SuperEngine, ktorú sme porovnávali obsahovala drón, ktorým je možné pohybovať sa po scéne a umiestniť ho napríklad tak, aby bol viditeľný na podlahovom rade monitorov. Táto funkcionalita scény sa nám zdala ako vhodná pre naše riešenie, preto bola v ďalšej verzii testovacieho projektu pridaná.

Finálna testovacia scéna obsahovala objekt, ktorým môžeme pohybovať pomocou herného ovládača alebo klávesnice. Herný ovládač sa javí ako dobrá možnosť, pre ovládanie, nakoľko používateľ v jaskyni často stojí a potrebuje sa po nej pohybovať, čo mu kompaktnosť ovládača umožňuje. Počas testovania finálnej scény bolo za pomoci pohyblivého objektu postupne odskúšané stereoskopické zobrazenie a prechody medzi obrazovkami. Nedostatočné prechody na stropnom a podlahovom rade monitorov sa opravili upravením pozície týchto monitorov. Výsledne prechody môžeme z rôznych pohľadov vidieť na Obr. 8.1.

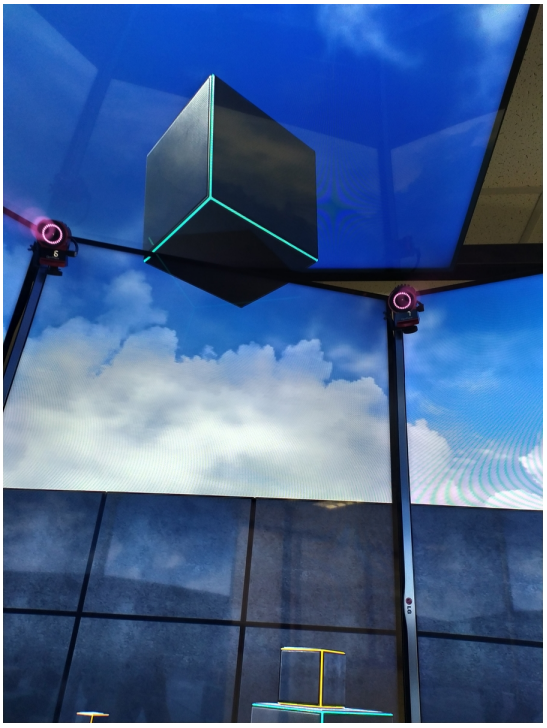
Pre ukážku vizualizačných schopnosti Unreal Enginu, bolo pre spustenie pomocou *nDisplay* upravených niekoľko scén. Tieto scény sú voľne dostupné na stiahnutie a modely využité v týchto scénach sú dostupné pre použitie v Unreal Engine projektoch. Scéna *Infinity Blade: Fire Lands* bola odskúšaná a vizuálne efekty a prostredie tejto scény bolo vierohodné. Prechody medzi obrazovkami boli plynulé. Stereoskopické zobrazenie fungovalo správne a na objektoch scény bola viditeľná hĺbka. Mimo-osové zobrazenie taktiež reagovalo správne na pohyb používateľa po prostredí jaskyne. Ukážku tejto scény na predných radoch monitorov môžeme vidieť na Obr. 8.2. Okrem tejto scény bolo pre *nDisplay* upravených pár ďalších scén, ktoré boli postupne odskúšané. Pri každom upravovaní scén sa ukázalo, že ich importovanie pre použitie je veľmi intuitívne a prebieha podľa postupu



(a) Pohľad nadol



(b) Bočný pohľad

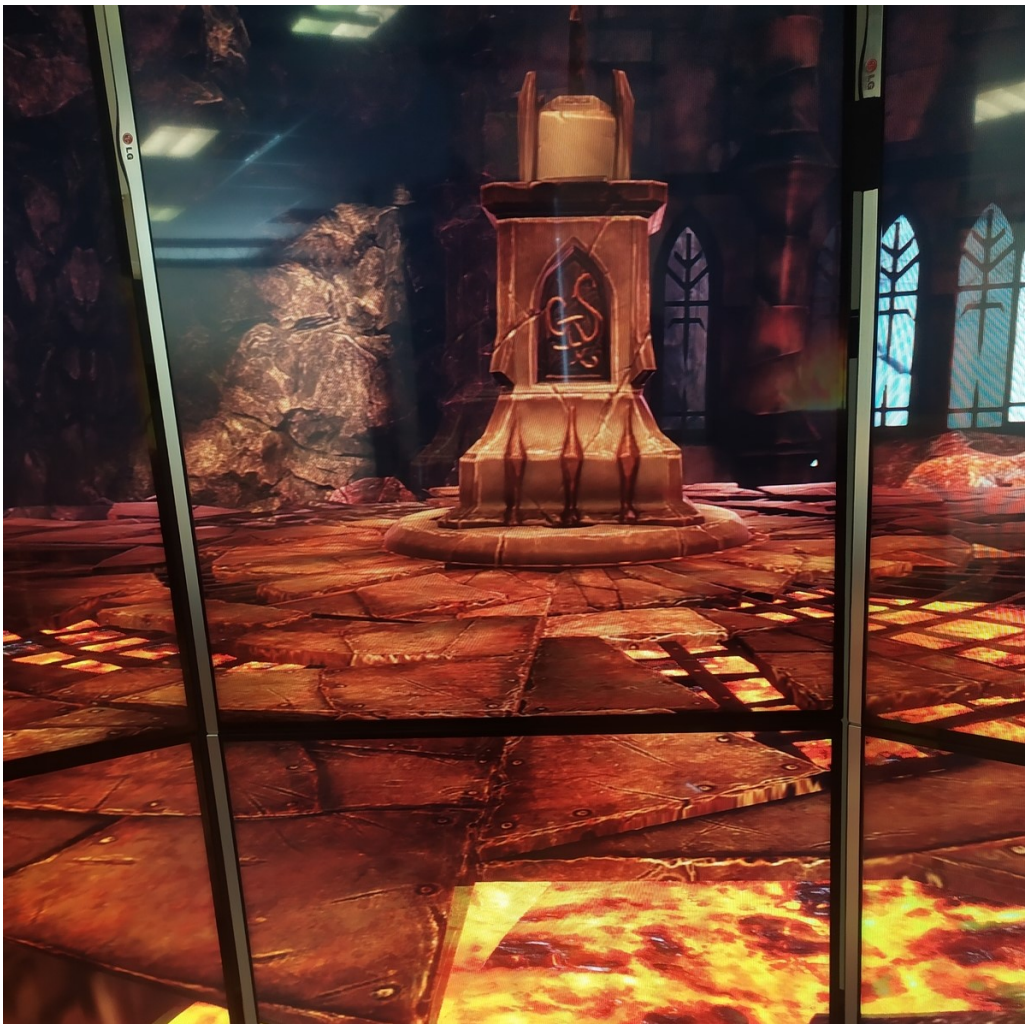


(c) Pohľad nahor

(d) Priamy pohľad

Obr. 8.1: Prechody medzi monitormi jaskyne.

pridania *nDisplay* pluginu do projektu.



Obr. 8.2: Ukážka scény *Infinity Blade: Fire Lands*.

Sledovanie pohybu je ovplyvnené mŕtvym uhlom, z ktorého sledovacie kamery nemôžu dostatočne vidieť značky. To spôsobuje problém v prípade, že používateľ je blízko monitorov, kde obraz nezodpovedá pozícií používateľa a často seká. Túto chybu nie je možné vyriešiť pomocou softvéru, avšak pomôcť by mohlo pridanie viacerých sledovacích kamier, alebo iné umiestnenie sledovacích značiek.

Finálne riešenie spĺňa všetky požiadavky, ktoré boli určené pri návrhu systému a je schopné nahradenia súčasného jadra SuperEngine.

9 Záver

Použitie Unreal Enginu ako vizualizačné jadro v LIRKIS CAVE bolo implementované za pomoci *nDisplay* pluginu. Pre tento plugin bol vytvorený konfiguračný súbor, ktorý popisuje hierarchiu obrazoviek, IP adresy počítačov klastra, vstupy a iné. Konfigurácia bola vytvorená na základe merania a postupného testovania funkcionality jaskyne. Postupnou implementáciou boli dosiahnuté výsledky zobrazenia, ktoré sú porovnateľné so súčasným zobrazovacím jadrom SuperEngine. Taktiež bolo zabezpečené spúšťanie a vypínanie celého systému cez jeden počítač. Sledovanie pozície používateľa a mimo-osové zobrazenie bolo zabezpečené pomocou systému *Optitrack*. Pre prenos údajov sa zvolil štandard VRPN, ktorý je podporovaný *Optitrack*-om aj *nDisplay* pluginom.

Nakoniec bola vytvorená testovacia scéna, ktorú je možné ovládať pomocou herného ovládača. V nej je možné hýbať objektom a meniť jeho vzhľad. Pre testovanie boli taktiež importované voľne dostupné ukážkové scény, ktoré ukazujú grafické možnosti jadra.

Výhodou riešenia je využitie technológie Unreal Engine 4. Herné jadro je pre použitie v CAVE systémoch vhodné, pretože obsahuje mnohé funkcionality, ktoré sú použiteľné pri tvorbe aplikácií. V riešení sa taktiež ukázala jednoduchosť konvertovania Unreal projektu na projekt, ktorý využíva *nDisplay*. To nám umožňuje rýchly štart pri tvorbe ďalších aplikácii pre CAVE systém.

Unreal Engine je v neustálom vývoji a k tomu patrí aj vývoj pluginu *nDisplay*, ktorý môže v budúcnosti priniesť vylepšenú funkcionalitu. Riešenie je vhodné pre použitie do budúcna a pre vytváranie aplikácie pre tento systém. Aplikácie, ktoré je možné vytvoriť siahajú od hier, cez návrh architektúry a dizajn, až po filmovú produkciu a sú limitované iba kreativitou tvorcu.

Literatúra

[1] Miloš Marcinčin. „Využitie herného jadra Unreal pre virtuálno-realitnú jaskyňu“. Dipl. pr. Košice: Technická Univerzita v Košiciach, apr. 2016, s. 48.

[2] Matúš Gabriška. „Prispôsobenie herného jadra Unreal pre virtuálno-realitnú jaskyňu“. Dipl. pr. Košice: Technická Univerzita v Košiciach, apr. 2018.

[3] Petra Romanová. „Konfigurovateľné mimo-osové zobrazenie pre virtuálno-realitnú jaskyňu“. Dipl. pr. Košice: Technická Univerzita v Košiciach, máj 2019.

[4] *Rendering to Multiple Displays with nDisplay*. [Online]. URL: https://docs.unrealengine.com/en-US/Engine/Rendering/nDisplay/index.html.

[5] Marián Hudák, Štefan Korečko a Branislav Sobota. „On architecture and performance of LIRKIS CAVE system“. In: *2017 8th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. IEEE. 2017, s. 000295–000300.

[6] *Unreal Engine*. [Online]. URL: https://www.unrealengine.com/en-US/.

[7] *Installing Unreal Engine*. [Online]. URL: https://docs.unrealengine.com/en-US/GettingStarted/Installation/index.html.

[8] *Unreal Engine 4 on GitHub*. [Online]. URL: https://www.unrealengine.com/en-US/ue4-on-github.

[9] *nDisplay Overview*. [Online]. URL: https://docs.unrealengine.com/en-US/Engine/Rendering/nDisplay/Overview/index.html.

[10] *nDisplayLauncher UI Reference*. [Online]. URL: https://docs.unrealengine.com/en-US/Engine/Rendering/nDisplay/LauncherUI/index.html.

[11] *nDisplay Configuration File Reference*. [Online]. URL: https://docs.unrealengine.com/en-US/Engine/Rendering/nDisplay/Configuration/index.html.

[12] Michael Lewis a Jeffrey Jacobson. „Game engines“. In: *Communications of the ACM* 45.1 (2002), s. 27.

[13] Haroon Shakirat Oluwatosin. „Client-server model“. In: *IOSRJ Comput. Eng* 16.1 (2014), s. 2278–8727.

[14] Brandon K Horton et al. „Game-Engine-Assisted Research platform for Scientific computing (GEARS) in Virtual Reality“. In: *SoftwareX* 9 (2019), s. 112–116.

[15] Dace A Campbell a Maxwell Wells. „A critique of virtual reality in the architectural design process“. In: *University of Washington HITL Technical Report R-94* 3.2 (1994).

[16] Vito Miliano. „Unrealty: application of a 3D game engine to enhance the design, visualization and presentation of commercial real estate“. In: *Proceedings of 1999 International Conference on Virtual Systems and MultiMedia (VSMM'99)*. 1999, s. 508–513.

[17] Antônio Carlos A Mól, Carlos Alexandre F Jorge a Pedro M Couto. „Using a game engine for VR simulations in evacuation planning“. In: *IEEE computer graphics and applications* 28.3 (2008), s. 6–12.

[18] Jeffrey Jacobson et al. „The CaveUT system: immersive entertainment based on a game engine“. In: *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*. ACM. 2005, s. 184–187.

[19] Russell M Taylor et al. „VRPN: a device-independent, network-transparent VR peripheral system“. In: *Proceedings of the ACM symposium on Virtual reality software and technology*. 2001, s. 55–61.

[20] Jeffrey Jacobson. „Using" CaveUT" to build immersive displays with the unreal tournament engine and a PC cluster“. In: *Proceedings of the 2003 symposium on Interactive 3D graphics*. 2003, s. 221–222.

[21] Marc Cavazza et al. „New ways of worldmaking: the Alterne platform for VR art“. In: *Proceedings of the 12th annual ACM international conference on Multimedia*. 2004, s. 80–87.

[22] Keith Miller a Alejandro Crawford. „Childish Gambino's Pharos: Real-Time Dome Projection for Live Concert“. In: *ACM SIGGRAPH 2019 Talks*. SIGGRAPH '19. Los Angeles, California: Association for Computing Machinery, 2019. ISBN: 9781450363174. DOI: 10.1145/3306307.3338114. URL: https://doi.org/10.1145/3306307.3338114.

[23] Robert Kooima. „Generalized perspective projection“. In: *J. Sch. Electron. Eng. Comput. Sci* (2009).

[24] *Qt Framework*. [Online]. URL: https://www.qt.io/.

Zoznam skratiek

API rozhranie pre programovanie aplikácií.

CAVE Cave Automatic Virtual Environment.

FPS Snímky za sekundu (Frames per second).

HMD Head-mounted display.

JSON JavaScript Object Notation.

LAN lokálna počítačová sieť (Local area network).

TCP Transmission Control Protocol.

UDP User datagram protocol.

VRPN Virtual Reality Peripheral Network.

Slovník

Blueprint Vizuálny skript pre Unreal Engine 4..

Open-source Open-source je počítačový softvér, ktorého zdrojový kód je prístupný pod takou licenciou, ktorá umožňuje študovanie, zmenu a distribuovanie zdrojového kódu pre akýkoľvek účel..

Powerwall Powerwall je rozsiahly displej, z ultra-vysokým rozlíšením, ktorý sa skladá z matice ďalších displejov..

Zoznam príloh

Príloha A DVD médium – záverečná práca v elektronickej podobe

Príloha B Používateľská príručka

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Príloha B: Používateľská príručka

2020 **Bc. Dominik Tóth**

Obsah

Zoznam obrázkov

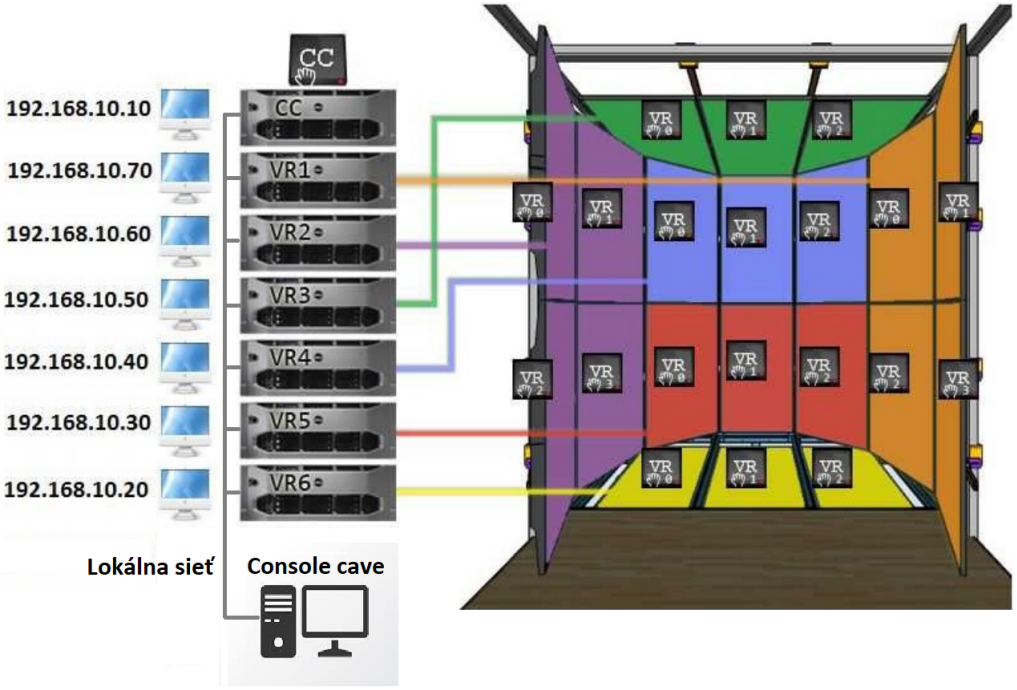
Zoznam tabuliek

1 Používateľská príručka

1 Úvod

V tejto príručke bude opísaný postup inštalácie, spustenia a používania systému. Ďalej bude opísané ovládanie testovacej scény.

V príručke sa budeme odvolávať na mena počítačov, ktoré sú viditeľné na Obr. 1.1



Obr. 1.1: Mená počítačov.

2 Inštalácia systému

Inštalácia systému prebieha nasledujúcimi krokmi:

1. Skopírovanie súboru `bin\nDisplayLauncher.exe` z DVD média do počítača *Console_Cave*.
2. Skopírovanie súboru `bin\nDisplayListener.exe` do zdieľaneho priečinka *unreal*, ktorý sa nachádza na ploche počítača *Console_Cave*.
3. Skopírovanie konfiguračného súboru `bin\LIRKISCAVE_konfigurak.cfg` do zdieľaneho priečinka *unreal*, ktorý sa nachádza na ploche počítača *Console_Cave*.

3 Inštalácia scén

Inštalácia akejkoľvek scény spočíva vo vložení balíčku zo spustiteľným súborom do zdieľaného priečinku *unreal*, ktorý je na ploche počítača *Console_Cave*.

Pre inštaláciu testovacej scény presunieme priečinok `bin\TestScene` z DVD média do priečinku *unreal*.

4 Spustenie systému a scén

Systém sa spúšťa vykonaním nasledujúcich krokov:

1. Zapnutie všetkých počítačov.
2. Vypnutie spustených inštancií SuperEnginu.
3. Na počítači *CC* zapnúť aplikáciu *Motive* a počkať 10 sekúnd, kým začne spracovávať informácie z kamier.
4. V aplikácii *Motive* otvoriť súbor *Unreal_Engine_Working*.
5. Na počítači *Console_Cave* spustiť aplikáciu *nDisplayLauncher.exe*.

Po vykonaní týchto krokov môžeme pomocou *nDisplayLauncher*-a spustiť scénu. Predtým si však spustíme aplikáciu *nDisplayListener.exe* na počítači *Console_Cave*. Kroky pre spustenie scény sú nasledovné (Obr. 1.2):

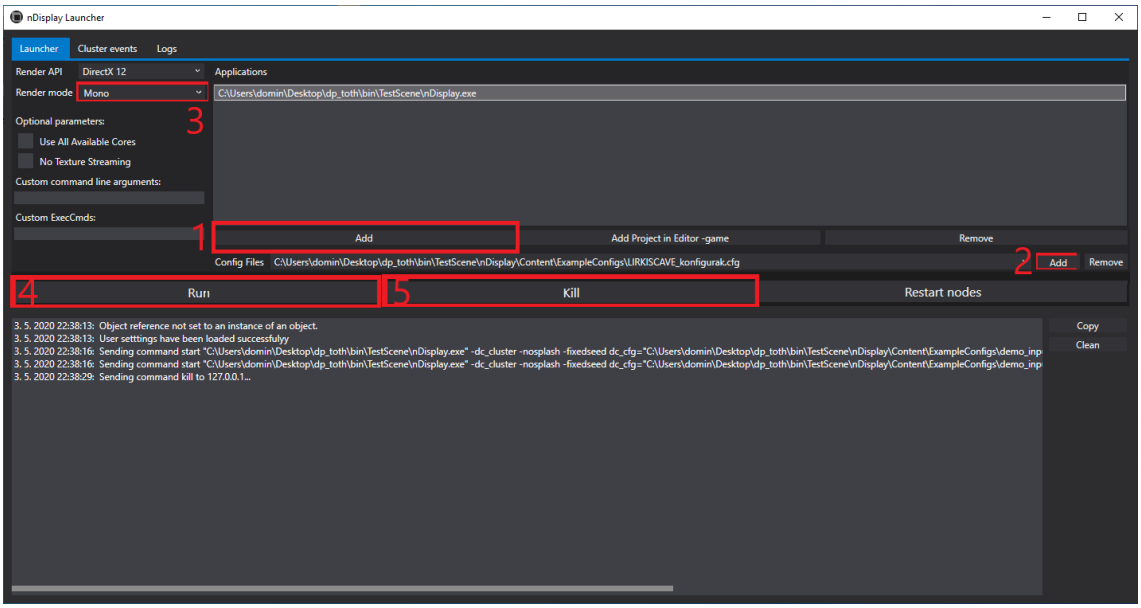
1. Pridanie spustiteľnej scény. Po stlačení tlačidla *Add* vyberieme cestu k spustiteľnému súboru. V prípade testovacej scény, ktorá je nainštalovaná podľa návodu, by mala byť cesta k súboru `unreal\TestScene\nDisplay.exe`. Po pridaní vyberieme scénu kliknutím na jej názov.

2. Kliknutím na tlačidlo *Add* vyberieme konfiguračný súbor *LIRKISCAVE_konfigurak.cfg*, ktorý sme nainštalovali do zdieľaného priečinka *unreal*.

3. Vyberieme *Render mode*. Ak chceme stereoskopické zobrazenie, vyberieme *Side-by-side*, v opačnom prípade vyberieme možnosť *Mono*.

4. Pre spustenie klikneme na tlačidlo *Run*.

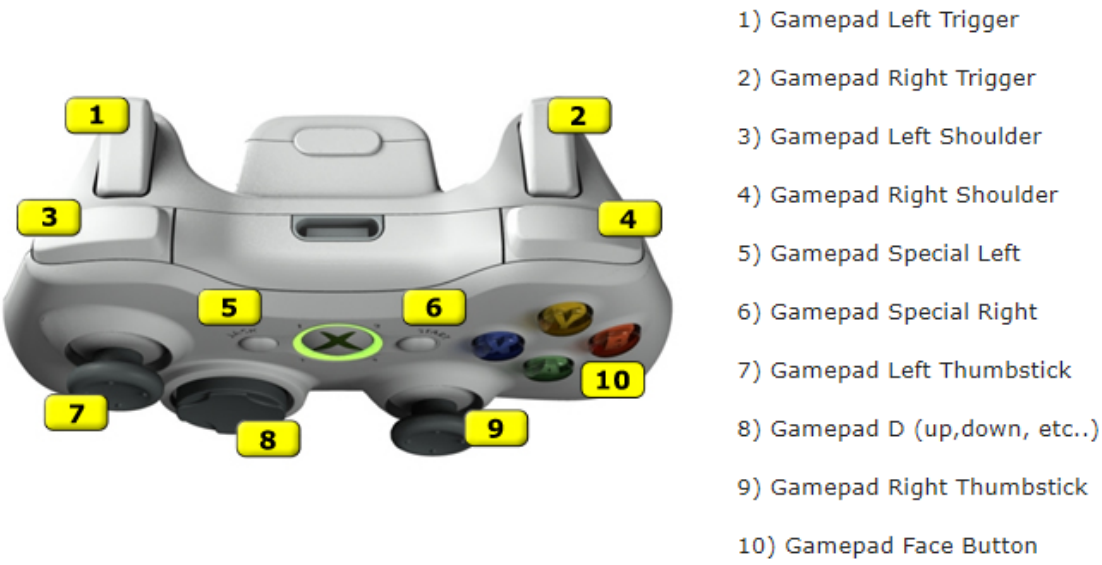
5. Akonáhle chceme aplikáciu vypnúť, klikneme na tlačidlo *Kill*.



Obr. 1.2: Spustenie a vypnutie scény.

5 Ovladanie testovacej scény *TestScene*

Testovaciu scénu je možné ovládať pomocou klávesnice a myši alebo pomocou herného ovládača. Pomenovanie tlačidiel herného ovládača môžeme vidieť na Obr. 1.3. Ovládanie testovacích scén môžeme vidieť v tabuľke 1.1. Akcie označené (a) sú funkčne iba v scéne *HQ Residential House*, (b) iba v scéne s pohybujúcim sa objektom. Zvyšné akcie sú funkčné pre obe scény.



Obr. 1.3: Mena tlačidiel herného ovládača.

Akcia	**Herný ovládač**	**Klávesnica a myš**
Zmena scény	Special Right (Start)	Shift + L
Pohyb vpred	Left Thumbstick Up	W
Pohyb vzad	Left Thumbstick Down	S
Pohyb vpravo	Left Thumbstick Right	D
Pohyb vľavo	Left Thumbstick Left	A
Otáčanie	Right Thumbstick	Myš
Otvorenie/Zatvorenie dverí (a)	Face Button Left (X)	E
Pohyb nahor (b)	Face Button Bottom (A)	E
Pohyb nadol (b)	Face Button Right (B)	Q
Ukázanie/Skrytie rýchlosti (b)	Face Button Top (Y)	P
Zvýšenie rýchlosti pohybu (b)	Right Shoulder	Up
Zníženie rýchlosti pohybu (b)	Left Shoulder	Down
Zmena modelu objektu (b)	Left/Right Trigger	Left/Right

Tabuľka 1.1: Ovládanie scén