УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Јована Кнежевић**
Број индекса: **РА 162-2019**

Тема рада: **Једно решење репродукције 360°-видео снимака у окружењу виртуелне стварности**

Ментор рада: **проф. др Илија Башичевић**

Нови Сад, јул, 2023

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Завршни (Bachelor) рад
Аутор, **АУ**:		**Јована Кнежевић**
Ментор, **МН**:		**проф. др Илија Башичевић**
Наслов рада, **НР**:		**Једно решење репродукције 360°-видео снимака у окружењу виртуелне стварности**
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		**2023**
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		**7/42/0/4/20/0/0**
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кључне речи, **ПО**:		**Виртуелна стварност, 360°-видео, шејдер, платформа за развој видео игара**
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		**Фокус овог дипломског рада је репродукција видео снимака унутар 360°-видеа, познатих као сферични видеи, у виртуелној стварности. Циљ рада је имплементација репродукције видео снимака у ВР окружењу унутар 360° користећи Јунити, популарну платформу за развој интерактивних тродимензионалних апликација. Главни изазов је правилан приказ различитих типова 360°-видеа унутар сфере, укључујући моноскопске и стереоскопске видее.**
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		**14.7.2023.**
Чланови комисије, **КО**:	Председник:	**Проф. др Иван Каштелан**
	Члан:	**Проф. др Мирослав Поповић** — Потпис ментора
	Члан, ментор:	**Проф. др Илија Башичевић**

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		**Jovana Knežević**	
Mentor, **MN**:		**Ilija Bašičević, PhD**	
Title, **TI**:		**A Solution for 360-Degree Video Playback in Virtual Reality Environment.**	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		**2023**	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		**7/42/0/4/20/0/0**	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		**Virtual reality, 360°-video, shader, platform for developing interactive 3D applications**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**This paper focuses on the playback of video content within 360-degree videos, also known as spherical videos, in a VR environment. The objective of the project is to implement video playback in a VR environment within the context of 360 degrees using Unity, a popular platform for developing interactive 3D applications. The main challenge lies in properly displaying different types of 360-degree videos in a VR environment, including monoscopic and stereoscopic videos.**	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:		**7/14/2023**	
Defended Board, **DB**:	President:	**Ivan Kaštelan, PhD**	
	Member:	**Miroslav Popović, PhD**	Menthor's sign
	Member, Mentor:	**Ilija Bašičević, PhD**	

Zahvalnost

Zahvaljujem se članovima porodice i prijateljima na pruženoj podršci tokom celokupnog studiranja.

Zahvaljujem se mentoru prof. dr Iliji Bašičeviću, mentorima Saši Jagodinu i Mihajlu Milotiću, kao i saradnicima iz tima na stručnoj pomoći i savetima tokom izrade ovog rada.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

VR - *Virtual Reality,* virtuelna stvarnost

AR - *Augmented Reality*, proširena stvarnost

XR - *Extended Reality***,** produžena stvarnost

AOSP - *Android Open-Source Project,* inicijativa Gugla za učestvovanje u razvijanju Android steka

API - *Application Programming Interfaces,* softverska sprega za komunikaciju između programskih modula

APK - *Android Package*, format u koji se pakuje Android aplikacija

SDK - *Software Development Kit,* razvojni komplet softvera

GPU - *Graphics Processing Unit,* grafička procesorska jedinica

GLSL - *OpenGL Shading Language*, jezik za senčenje visokog nivoa sa sintaksom zasnovanom na programskom jeziku C

3DoF - *Three Degrees of Freedom*, tri stepena slobode

6DoF - *Six Degrees of Freedom,* šest stepeni slobode

TB - *Top/Bottom*, gore/dole tip stereoskopskog videa

SBS - *Side-By-Side*, levo/desno tip stereoskopskog videa

UI - *User Interface,* korisnički prostor

1. Uvod

U eri digitalne transformacije i brzog tehnološkog napretka, virtuelna stvarnost (engl. Virtual Reality - VR) se sve više koristi u različitim sektorima, od zabave i obrazovanja, do medicine i inženjeringa. Iz tog razloga, razvoj tehnika i alata koji poboljšavaju korisničko iskustvo u VR okruženju postaju sve važniji. Ovaj diplomski rad se fokusira na jedan specifičan aspekt VR-a, a to je reprodukcija video snimaka unutar 360°-videa, još poznati i kao sferični videi.

Zadatak projekta je da se implementira reprodukcija video snimaka u VR okruženju unutar 360° koristeći Juniti (eng. Unity), popularnu platformu za razvoj interaktivnih trodimenzionalnih (3D) aplikacija. Konkretno, rad je usredsređen na stvaranje nove scene za prikaz 360°-video snimaka, jer su u strukturi projekta već postojale tri scene.

Osnovni problem koji je trebalo rešiti je kako različite tipove 360°-video snimaka pravilno prikazati u VR okruženju. Ovaj izazov se ogledao u potrebi da se razumeju i realizuju tehnički aspekti dva glavna tipa 360°-videa: monoskopskog (eng. Monoscopic) i stereoskopskog (eng. Stereoscopic). Monoskopski video sastoji se od jedne slike projektovane na sferu oko korisnika, dok stereoskopski video zahteva dve projekcije - jednu za svako oko. Drugi problem je bio pravilno mapiranje teksture (eng. Texture) na sferu, što je posebno složeno za stereoskopske video snimke.

Jedan od izazova prilikom izrade ovog projekta bio je prikaz videa unutar sfere. Kako bi video bio vidljiv iz unutrašnje perspektive sfere, bilo je neophodno invertovati njene normale.

Cilj ovog rada je reprodukcija 360°-video snimaka u VR okruženju unutar sfere preko video plejera i njihovo pravilno mapiranje na istu, kao i interakcija sa korisnikom preko kontrolera, koji omogućavaju odabir i prikazivanje različitih video zapisa.

Drugo poglavlje obuhvata teorijske osnove na kojima je bazirana realizacija rešenja. Navedene osnove su neophodne za razumevanje samog rešenja.

U trećem poglavlju predstavljen je koncept rešenja u kom se detaljno opisuje ideja iza samog rešenja, uključujući prilagođavanje scene za prikaz videa, odabir tipova 360°-videa, postavljanje kamere i interakciju sa video sadržajem putem kontrolera, kao i proces reprodukcije i prikaza videa na sferi.

Četvrto poglavlje prikazuje programsko rešenje. Svi moduli pomenuti u konceptu rešenja su realizovani programski.

U petom poglavlju će biti izvršeno ispitivanje implementiranih funkcionalnosti u okviru *Unity* razvojnog okruženja. Cilj ovog ispitivanja je provera da li implementirane funkcije funkcionišu kako se očekuje i da li se postiže željeni rezultat prilikom interakcije sa korisnikom.

U šestom poglavlju će biti iznet zaključak rada. U ovom delu će biti sumirani rezultati implementacije programa za reprodukciju 360°-video snimaka u VR okruženju, kao i opisane mogućnosti za dalji razvoj.

2. Teorijske osnove

U ovom poglavlju će biti predstavljene teorijske osnove na kojima se rad zasniva: virtuelna stvarnost, Android operativni sistem, 360°-video, proširena stvarnost, video plejer i šejderi. Ove tehnologije su ključne za razumevanje rešenja koje je predstavljeno u ovom radu.

2.1 Virtuelna stvarnost

Virtuelna stvarnost je tehnologija koja koristi napredne računarske sisteme kako bi stvorila interaktivno okruženje koje simulira realnost ili imaginarni svet. Ova tehnologija je revolucionisala način na koji ljudi doživljavaju i interaguju sa digitalnim sadržajem.

VR je tehnologija koja koristi posebne uređaje poput VR naočara ili kaciga (eng. VR Headset) kako bi korisnicima pružila osećaj da su prisutni i interaguju sa digitalnim svetom koji je stvoren oko njih. Ovi uređaji obično koriste prikaz u visokoj rezoluciji i široki ugao gledanja kako bi omogućili korisnicima da se potpuno urone u virtuelno okruženje. Kroz upotrebu senzora za praćenje pokreta i kontrolera za interakciju, korisnici mogu da istražuju virtuelni prostor, manipulišu objektima i komuniciraju sa virtuelnim entitetima.

VR tehnologija kombinuje fizičku arhitekturu (eng. Hardware) i programsku podršku (eng. Software) kako bi stvorila iskustva koja "prevare" oko i mozak. Fizička arhitektura podržava senzorne stimulacije poput zvuka, dodira, mirisa ili intenziteta toplote, dok programska podrška kreira virtualno okruženje.

Kada govorimo o funkcionisanju oka i mozga u kontekstu 3D VR iskustva, imamo na umu da su naše oči udaljene otprilike tri inča jedno od drugog, stvaraju dva blago različita vizuelna prikaza, dok mozak te prikaze objedinjuje. VR aplikacije simuliraju ovaj fenomen kroz par identičnih slika prikazanih iz različitih perspektiva. Umesto da na ekranu imamo jednu sliku koja pokriva čitav prostor, prikazujemo dve slike koje su identične, ali su

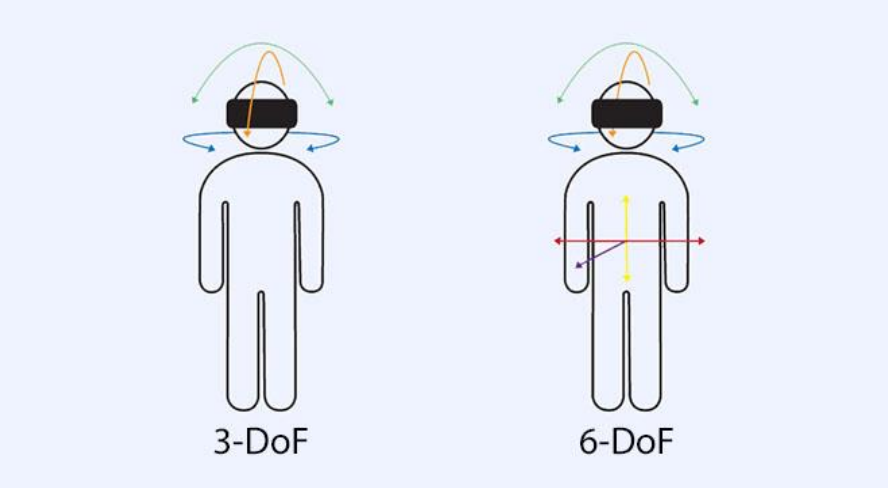
napravljene tako da se razlikuju za svako oko. Na taj način, VR tehnologija "prevari" mozak posmatrača, stvarajući iluziju dubine i prihvatanje višedimenzionalne slike.

2.1.1 Stepeni slobode

Ključni koncepti virtuelne stvarnosti su “tri stepena slobode” (eng. Three Degrees of Freedom - 3DoF) i “šest stepeni slobode” (eng. Six Degrees of Freedom - 6DoF).

3DoF se odnosi na mogućnost praćenja korisnikovih rotacionih pokreta glave. Ovo znači da korisnik može da okreće glavu levo-desno, gore-dole i naginje je, ali nema mogućnost kretanja napred-nazad ili levo-desno u virtuelnom prostoru. Ova vrsta VR iskustva je najčešća u pristupačnim VR uređajima, kao što su neke VR naočare za mobilne telefone.

S druge strane, **6DoF** dodaje tri dodatna stepena slobode, omogućavajući korisniku da se slobodno kreće u virtuelnom prostoru. Ovo znači da korisnik može da hoda napred-nazad, levo-desno i penje se ili spušta u virtuelnom okruženju. Ovakav oblik VR iskustva zahteva naprednije i skuplje uređaje, kao što su VR naočare povezane sa računarom ili konzolom za video igre. 6DoF pruža mnogo realističnije iskustvo, jer korisnici mogu da istražuju virtuelni prostor na način koji je sličan stvarnom kretanju.



Slika 2.1 3DoF I 6DoF

2.2 Android

Android je operativni sistem otvorenog koda (eng. Android Open-Source Project - AOSP), koji je razvio Gugl (Google), a koristi se većinom na mobilnim uređajima poput pametnih telefona i tableta. Međutim, Android je prilagodljiv i može se koristiti na različitim uređajima, uključujući televizore (Android TV), automobile (Android Auto) i pametne satove (Wear OS).

Glavna prednost Androida je njegova prilagodljivost i otvorena priroda. Android koristi *Linux* kernel i omogućava proizvođačima uređaja da prilagode programsku podršku za svoje specifične potrebe. To uključuje mogućnost optimizacije za različite konfiguracije fizičke arhitekture, kao i mogućnost prilagođavanja korisničke sprege (eng. Interface).

Android je takođe otvoren za razvoj aplikacija. *Google* nudi Android SDK (eng. Software Development Kit - SDK) koji programerima pruža alate i API-je (eng. Application Programming Interfaces - API) za razvoj aplikacija. Ovo uključuje mogućnost pristupa komponentama uređaja na fizičkoj arhitekturi, kao što su senzori, kamera i grafički čipovi, što je posebno važno za razvoj VR aplikacija [1].

2.2.1 Meta Quest 2

Meta Quest 2, prethodno poznat kao Oculus Quest 2, su samostalne VR naočare koje ne zahtevaju dodatnu opremu, poput računara ili konzole, da bi funkcionisale. To je druga generacija Quest-a, koja donosi brojna poboljšanja u odnosu na originalni model, uključujući bolju rezoluciju, više RAM-a i brži procesor, ali i manju težinu.

Slika 2.2 Meta Quest 2

Jedna od ključnih karakteristika Quest 2 naočara je da koristi prilagođenu verziju Android operativnog sistema. Za prvo podešavanje, mora se koristiti pametni telefon na kojem se izvršava Meta Quest aplikacija. Iako korisnička sprega i iskustvo koje Quest 2 pruža korisniku značajno varira u odnosu na tradicionalnu Android spregu, temelj sistema je i dalje Android [2].

Ova činjenica ima nekoliko prednosti. Prvo, Android je već optimizovan za mobilne uređaje, što znači da je dobro prilagođen za efikasno korišćenje baterije i resursa, što je ključno za samostalne VR naočare. Drugo, postojeći Android API-ji i alati za razvoj mogu se koristiti za izradu aplikacija za Quest 2, što olakšava razvoj za ovu platformu.

Ove informacije ukazuju na to da je Android ključni deo ekosistema Meta Quest 2. Koristi se za pokretanje operativnog sistema naočara, a razvojni ekosistem Androida, uključujući AOSP i SDK, koristi se za izradu aplikacija za Quest 2. Android takođe pruža osnovu za interakciju sa drugim uređajima, kao što je potrebno za prvo podešavanje kroz Meta Quest aplikaciju na pametnom telefonu.

2.3 360°-video

U poslednjoj deceniji, napredak u tehnologiji je doveo do razvoja novih medijskih formata koji značajno menjaju način na koji konzumiramo sadržaj. Jedan od takvih inovativnih formata je 360°-video, poznat i kao sferni video (engl. Spherical video), koji omogućava korisnicima da potpuno urone u virtuelni svet.

360°-video je digitalni video format koji omogućava korisnicima da istražuju okolinu u svim pravcima. To se postiže korišćenjem posebnih kamera koje snimaju pogled iz svih pravaca u istom trenutku.

Da bi se dobio jedan ovakav sadržaj, obično se snima uz pomoć više kamera, a može i pomoću posebne kamere koja snima celokupnu scenu sa jedinstvene tačke. Takva kamera ima više objektiva koji su ugrađeni u sam uređaj i u istom trenutku snima uglove koji se preklapaju.

Slika 2.3 Specijalna 360° kamera

Metoda koja je poznata kao “video spajanje” (engl. video stiching), sve zasebne snimke spaja u jedan sferičan video, dok se kontrasti i boje prilagođavaju kako bi bili u skladu sa ostalima. Ovaj proces se može izvesti korišćenjem same kamere ili pomoću programske podrške koja analizira vizuelni i zvučni sadržaj svake kamere i usklađuje ih.

Slika 2.4 Primer video spajanja

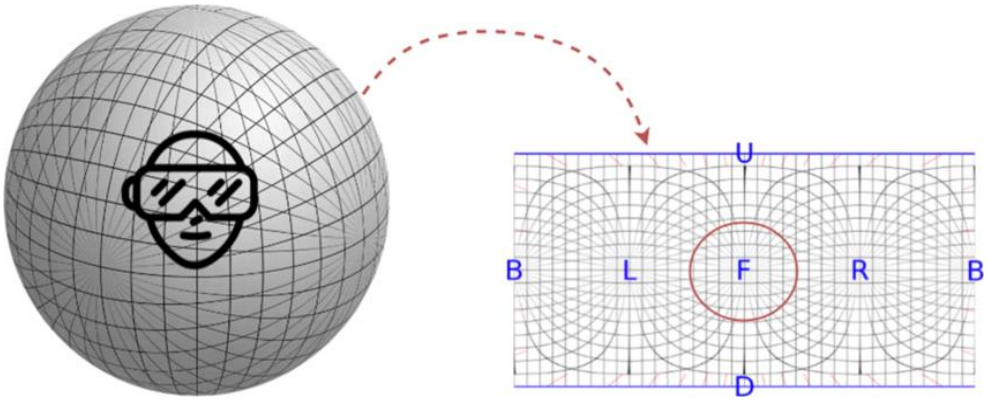
2.3.1 Monoskopski 360°-video

Osnovna podela 360°-videa su monoskopski i stereoskopski videi.

Monoskopski video je ravni dvodimenzionalni (2D) prikaz koji se može gledati na bilo kom ekranu. Korisnici mogu istraživati svaki deo scene, ali bez pravog osećaja dubine. Gledanje monoskopskog videa je kao da gledate oko sebe sa samo jednim otvorenim okom.

Ovakav tip videa obično ima odnos stranica 2:1, ekvirektangularne (eng. Equirectangular) projekcije. Ekvirektangularna projekcija se moze zamisliti kao mapa sveta sa globusa, koja je razvijena na ravan. Česte rezolucije ovakvih videa su 3840x1920, 4096x2048, 5760x2880 i 7680x3840.

Kada kažemo “mono”, pomislimo da je video jednokanalni, ali se on zapravo prikazuje na oba oka u virtuelnoj stvarnosti.



Slika 2.5 Nastanak ekvirektangularne projekcije

2.3.2 Stereoskopski 360°-video

Stereoskopski 360°-video, za razliku od monoskopskog videa pruža dubinu i trodimenzionalno (3D) iskustvo. Ovakav tip videa ima dva kanala, za oba oka, sa blago različitom perspektivom koja pruža percepciju dubine, kao što je to slučaj sa 3D filmovima u bioskopu.

Najčešće je prikazan u gornjem/donjem (engl. Top/Bottom, TB) formatu, gde gornja slika predstavlja prikaz levog oka, dok donja slika predstavlja prikaz desnog oka. Neki od formata su i levo/desno (engl. Side-By-Side, SBS) i ekvirektangularni format.

Slika 2.6 Primer “levo/desno” i “gore/dole” tipova

Posebne kamere se koriste za snimanje obe perspektive istovremeno. Pošto su oba kanala smeštena u isti video kontejner, to znači da se rezolucija suštinski deli na pola za krajnjeg gledaoca. Da bi se to nadoknadilo, stereoskopski 360°-video zapisi trebaju biti isporučeni sa dvostrukom rezolucijom za razliku od monoskopskih, što može biti izazovno za većinu platformi za strimovanje i za samu fizičku arhitekturu.

Česte stereoskopske rezolucije mogu biti 3840x3840, 5120x5120 i čak 7680x7680. Za fizicku arhitekturu niže klase, obično se isporučuje u rezoluciji 3840x2160, a oba "rastegnuta" stereoskopska kanala su smeštena u taj kontejner, ali na ovoj rezoluciji se gubi velika količina detalja. Iz tog razloga, sadržaj koji ima rezoluciju 4K, često može izgledati zamućeno.

Prilikom gledanja 360°-video sadržaja, gledalac vidi samo mali deo 360°-videa u datom trenutku unutar svog vidnog polja. To znači da 3840x1920 360°-video zapravo prikazuje samo oko 1280x720 piksela u portalu gledanja u datom trenutku. Zbog toga VR video sadržaj ponekad izgleda kao televizija iz 1990-ih. Iz tog razloga, svaki piksel je važan!

2.4 OpenXR

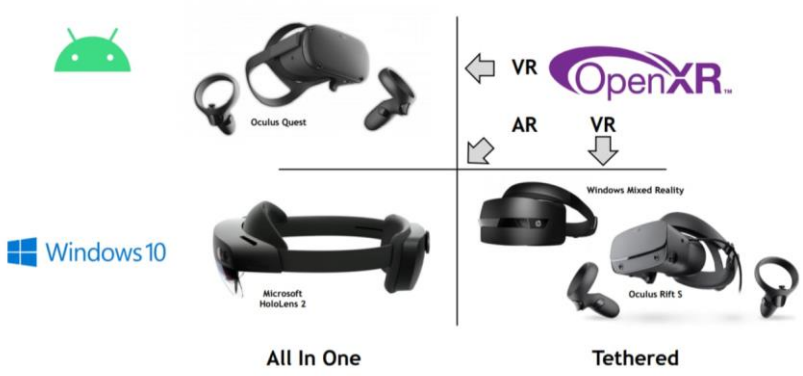
Produžena stvarnost (eng. Extended Reality - XR) je otvoreni standard koji je osmišljen da pojednostavi razvoj proširene stvarnosti (eng, Augmented Reality - AR) i virtualne stvarnosti programske podrške. Ime "OpenXR" dolazi od "Open Extended Reality", gde je "Extended Reality" zajednički naziv za sve kombinacije AR, VR i mešane stvarnosti (eng. Mixed Reality - MR).

Standard je razvijen od strane *Khronos* Grupe, udruženje koje je poznato po razvoju otvorenih standarda kao što su *OpenGL* i *Vulkan*.

OpenXR ima za cilj da omogući razvoj jedinstvene aplikacije koja može da radi na različitim uređajima za AR i VR, bez obzira na njihovu specifičnu konfiguraciju fizičke arhitekture ili platformu. Ovaj standard pruža jednostavan pristup fizičkoj arhitekturi i funkcionalnostima uređaja, dok istovremeno smanjuje kompleksnost razvojnog procesa.

Prvi deo OpenXR-a je API, koji pruža jedinstveni skup standardnih funkcija za pristup funkcionalnostima AR i VR uređaja. Ovaj API omogućava kreiranje, upravljanje i interakciju sa virtuelnim okruženjima.

Naredni deo je standardna biblioteka (eng. Runtime), sloj programske podrške koji funkcioniše kao posrednik između aplikacija i fizičke arhitekture. OpenXR standardna biblioteka se instalira na korisnikovom računaru ili uređaju i omogućava komunikaciju između aplikacija koje koriste OpenXR API i različitih AR/VR uređaja.



Slika 2.7 OpenXR i uredjaji

OpenXR podržava programske dodatke (eng. Plugin), što znači da se mogu koristiti dodatni moduli, za podršku specifičnim funkcionalnostima ili uređajima. Ovi programski dodaci proširuju mogućnosti OpenXR API-ja i omogućavaju prilagođavanje aplikacija specifičnim potrebama.

Još jedan važan koncept u okviru OpenXR-a su **sesije** i **uređaji**. Sesija predstavlja zahtev aplikacije da prikaže XR sadržaj korisniku. Sesija omogućava komunikaciju i koordinaciju između aplikacije i standardne biblioteke (programske podrške koja implementira OpenXR API) putem funkcija za kontrolu sesije i događaja stanja sesije, dok uređaj predstavlja fizičku arhitekturu koja se koristi za prikaz i interakciju sa virtuelnim svetom. OpenXR podržava različite vrste uređaja, kao što su VR naočare, AR naočare, kontroleri i druge ulazne periferije [3].

2.5 Video plejer

Video plejer (eng. Player) je ključna komponenta u digitalnom medijskom ekosistemu, omogućavajući reprodukciju video i audio sadržaja u digitalnom formatu. Ključ za ovu funkcionalnost je programska ili komponenta fizičke arhitekture, poznata kao kodek. Kodek je algoritam koji kodira (kompresuje) i/ili dekodira (dekompresuje) digitalne medijske datoteke, omogućavajući plejeru da reprodukuje i stvara ove datoteke. Interesantno je napomenuti da neki kodeci mogu imati samo jednu od ovih funkcija - kodiranje ili dekodiranje.

Funkcionalnost video kodeka može se grubo podeliti u tri kategorije. Prva je uzorkovanje izlaznog signala kamere, koji je obično analogni. Drugi korak je konverzija analognog signala u digitalni, gde se svaki uzorak obično konvertuje u 8 bitova - ovaj proces se naziva kvantizacija. Poslednji korak je kompresija binarnog toka podataka. Kompresija može biti sa gubicima (eng. lossy) ili bez gubitaka (eng. lossless). Cilj je smanjiti veličinu podataka bez značajnog uticaja na kvalitet reprodukcije. Kada video plejer dobije kompresovani video tok, koristi odgovarajući kodek da dekompresuje video i audio podatke, a zatim renderuje video na ekranu i reprodukuje audio na zvučnicima ili slušalicama. Tokom ovog procesa, video plejer mora održavati sinhronizaciju između video i audio traka, osiguravajući da se slika i zvuk uvek poklapaju. Video plejeri nude korisničke sprege koje omogućavaju korisnicima da interaguju sa video reprodukcijom, pružajući kontrolu za pauziranje, premotavanje, preskakanje, promenu brzine reprodukcije i druge opcije.

2.6 Šejder

Šejder (eng. Shader) je tip programa koji se koristi u 3D računarskoj grafici za određivanje konačnog izgleda objekta na ekranu. Ovi programi su izuzetno fleksibilni i mogu se programirati da proizvedu vrlo složene efekte. Najčešće se koriste u računarskim igrama i

simulacijama kako bi se stvorili efekti kao što su refleksija svetlosti, senke, atmosferski efekti i mnogi drugi. Šejderi omogućavaju programerima da kontrolišu kako se svetlost ponaša kada padne na objekat, kako se boje mešaju i još mnogo toga.

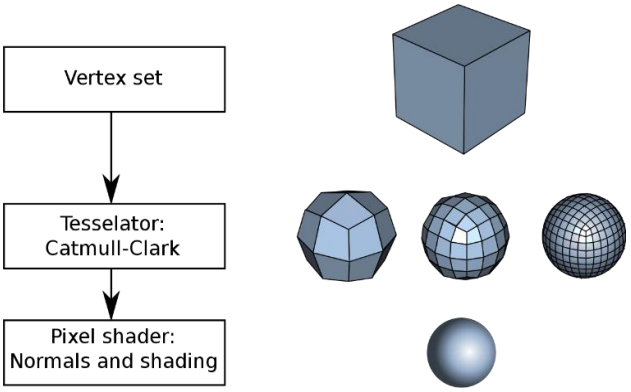
Postoje različite vrste šejdera, uključujući *vertex* šejder, fragment (ili piksel) šejder, i geometrijski (eng. Geometry) šejder, svaki sa svojom specifičnom ulogom:

- ***Vertex*** šejder - Ovaj tip šejdera se koristi za obradu svakog vrha modela. Može da kontroliše atribute kao što su pozicija, boja i tekstura koordinata.
- **Fragment** (ili piksel) šejder - Ovaj tip shadera se koristi za obradu svakog fragmenta, što je osnovna jedinica rasterizacije, koja često odgovara jednom pikselu na ekranu. Kontroliše konačnu boju i druge atribute svakog piksela na ekranu.
- **Geometrijski** šejder - Ovaj tip šejdera se koristi za manipulaciju i kreiranje novih primitiva (kao što su tačke, linije i poligoni) u 3D modelu.

Šejderi se obično pišu u posebnim jezicima za šejdere, kao što su GLSL (OpenGL Shading Language) ili HLSL (High-Level Shading Language za DirectX). Ovi jezici dozvoljavaju pisanje koda koji se izvršava direktno na grafičkom procesoru (eng. Graphics Processing Unit - GPU), omogućavajući velike performanse potrebne za realističnu 3D grafiku [4].

2.6.1 OpenGL Shading Language

GLSL je sličan C jeziku, i omogućava pisanje prilagođenih *vertex*, fragment (piksel), geometrijskih šejdera i šejdera za usitnjavanje (eng. tessellation). Kako bi omogućio maksimalnu fleksibilnost, GLSL dozvoljava programerima da definišu svoje varijable, funkcije i operacije, što znači da možete realizovati veoma kompleksne i prilagođene efekte.



Slika 2.8 Primer šejdera

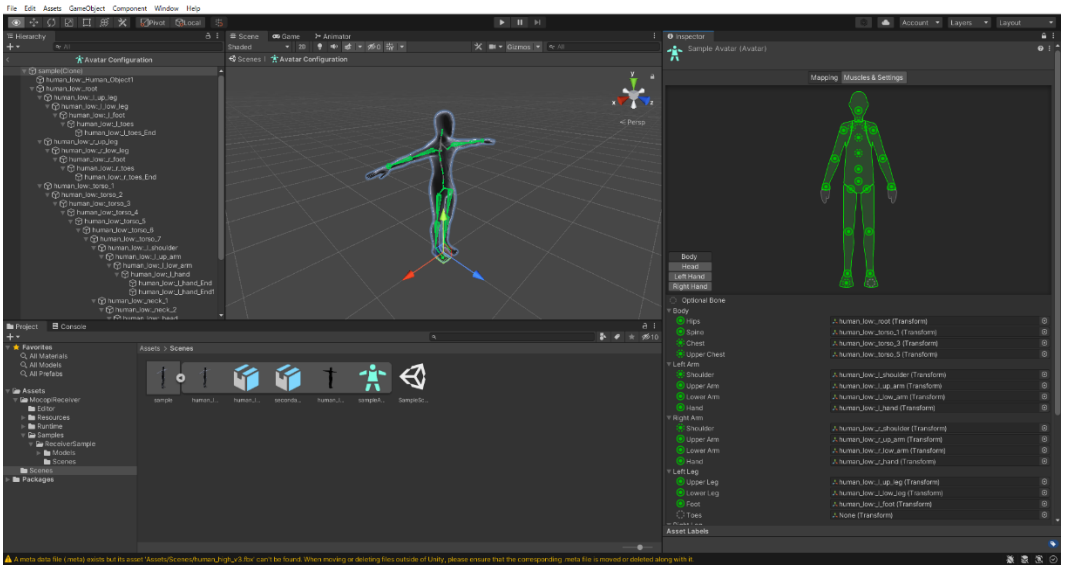
Različiti tipovi šejdera u GLSL-u se koriste za različite aspekte renderinga. *Vertex*, fragment i geometrijski šejderi su objašnjeni, a što se tiče šejdera za usitnjavanje, koristi se za dinamično dodavanje detalja 3D modelima na osnovu gledaoca i pozicije kamere, dok *compute* šejder omogućava generalnu obradu podataka na grafičkom procesoru izvan konteksta renderinga.

Svaki od ovih šejdera se izvršava na grafičkom procesoru, omogućavajući brzo izvršavanje paralelnih operacija potrebnih za obradu grafike u realnom vremenu. Pored toga, GLSL podržava širok spektar matematičkih funkcija, što ga čini idealnim za operacije koje su česte u računarskoj grafici, kao što su transformacije koordinata i operacije sa bojama [5].

2.7 Unity

Juniti (eng. Unity) je programsko okruženje koje omogućava razvoj aplikacija i igara za različite platforme, kao što su Windows, MacOS, Linux, Android, iOS, i mnoge druge. Njegovo jezgro napisano je u C/C++ programskom jeziku, dok je Unity program za uređivanje grafičke korisničke sprege (eng. UI Editor) baziran na jeziku C#, koji služi za kreiranje 2D i 3D igara. Korisnicima se pruža mogućnost pristupa jezgru Unity-a kroz API koji se može koristiti u .NET *Framework*-u, kao i podrška za jezike poput C#, Boo i JavaScript-a [6].

Osim toga, Unity ima podršku za različite audio i video formate, uključujući 360°-video, što ga čini idealnim za kreiranje interaktivnih multimedijalnih iskustava.



Slika 2.9 Unity okruženje

U kontekstu ovog rada, Unity predstavlja platformu koja omogućava integraciju svih prethodno navedenih pojmova: VR, Android, OpenXR, video plejer, 360°-video, i šejdere. Kroz Unity, mogu se kreirati VR aplikacije za Android uređaje, koristeći OpenXR za kompatibilnost sa različitim VR naočarama, u ovom slučaju Meta Quest 2. Takođe, može se koristiti video plejer za prikazivanje 360°-video sadržaja unutar VR okruženja, a sve to koristeći različite šejdere za postizanje željenog vizuelnog efekta.

3. Koncept rešenja

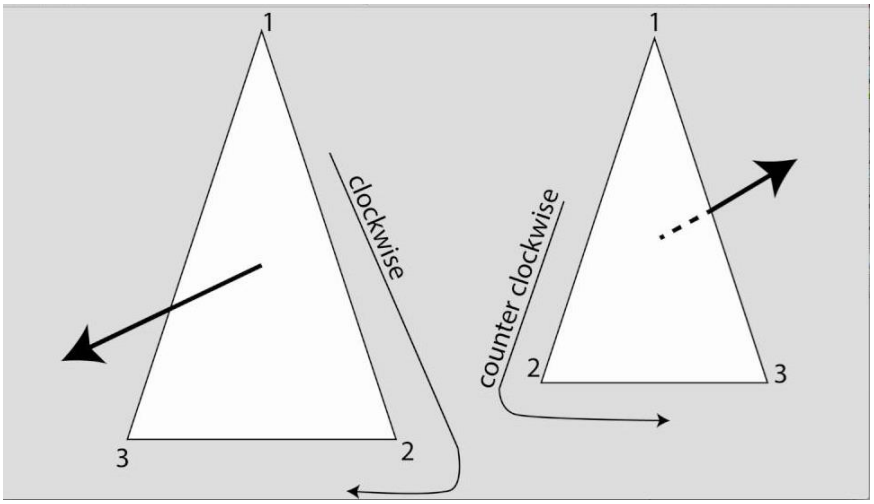
U ovom poglavlju biće reči o ideji iza samog rešenja. Modul je organizovan hijerarhijski i čine ga nekoliko modula koji međusobno komuniciraju. Kroz prilagođavanje scene za prikaz videa, odabir tipova 360°-videa, postavljanje kamere i interakciju sa video sadržajem putem kontrolera, kao i proces reprodukcije i prikaza videa na sferi, sistem je osmišljen da pruži korisnicima uzbudljivo i interaktivno iskustvo gledanja 360°-video sadržaja.

3.1 Prilagođavanje scene za prikaz videa

Za uspešno prikazivanje video sadržaja, prvi korak je stvaranje sfere u Unity sceni na kojoj će se videi prikazivati. Sfera može biti kreirana u odgovarajućoj programsoj podršci namenjenoj za modelovanje, ili se može iskoristiti već postojeća sfera koju nudi Unity. Međutim, ta sfera prikazuje svoju spoljašnost, dok mi želimo da prikažemo unutrašnjost.

Proces koji ovo rešava se naziva "invertovanje normala“. Na slici je prikazan jedan trougao sa sfere koji čini mrežu. Svaka od tačaka trougla je sačuvana kao 3D vrh sa (x,y,z) vrednostima što predstavlja njihovu poziciju u 3D prostoru. Normala je vektor koji naleže na ravan, odnosno poligon pod ugom od 90 stepeni. Ona govori Unity-u na kojoj strani treba prikazati teksturu. Stoga, ono što treba uraditi je okrenuti vrhove tako da budu rasporedjeni u suprotnom smeru (suprotno od kazaljke na satu). Tada će normala biti na drugoj strani, što znači da će tekstura biti prikazana sa unutrašnje strane.

Postoji više načina kako to uraditi u Unity-u, za ovaj rad ćemo koristiti šejder. Međutim, postoji još jedna stvar o kojoj treba voditi računa. Kada bi pustili video, na ovako izmenjenoj sferi, izgledaće unazad (kao kada bi pisali po staklu pa pogledali sa druge strane). Kako bi video izgledao sasvim pravilno, treba zameniti x (horizontalna koordinata teksture) sa 1-x. Ovim promenama dobijamo sferu koja prikazuje sadržaj sa unutrašnje strane.

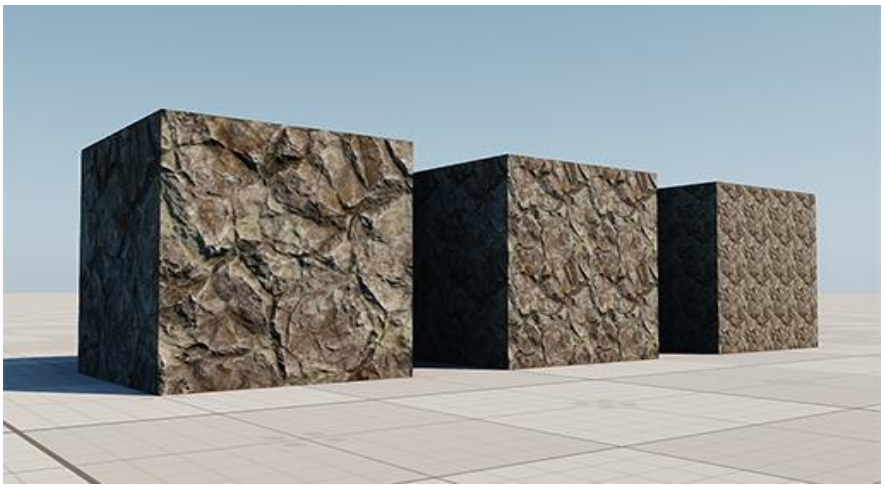


Slika 3.1 Invertovanje normala

3.2 Prikaz različitih tipova 360°-videa

Sledeći korak je prikaz različitih tipova 360°-videa. U teorijskim osnovama, navedena su dva tipa ovakvih videa, monoskopski – sa jednim kanalom, i stereoskopski – sa dva kanala. Kako bi pravilno prikazali različite sadržaje, potrebno je pravilno odrediti na koji način predstaviti svaki podtip zasebno.

Za monoskopske videe, jedna slika se prikazuje na oba oka, dok za stereoskopske videe, potrebno je imati dve sfere umesto samo jedne. Razlog za to je jer će slika za levo oko biti prikazana na jednoj sferi, dok će slika za desno oko biti na drugoj sferi. Takođe, treba napraviti dva materijala, po jedan za svaku sferu. U Unity-u, materijali se koriste za definisanje izgleda objekata. Svaki materijal može imati svoj šejder koji kontroliše kako će se materijal ponašati u pogledu osvetljenja, senke, tekstura i drugih vizuelnih efekata. Nakon toga, potrebno je prilagoditi polja Tiling (ponavljanje) i Offset (pomeraj) za svaki tip videa kako bi se pravilno prikazali na obe sfere.



Slika 3.2 Primer pomeraja i ponavljanja tektsture

U šejderu, ova polja su parametri koji se koriste za kontrolu ponavljanja i pomeranja tekstura na površini objekta. Ponavljanje se odnosi na to koliko puta se tekstura ponavlja po horizontalnoj i vertikalnoj osi na površini objekta. Parametar pomeraj se odnosi na pomeranje teksture po horizontalnoj i vertikalnoj osi na površini objekta.

3.3 Kamera i interakcija sa video sadržajem

Dalje, treba napomenuti kako je kamera postavljena u sceni. *GenericVRPlayer* predstavlja korisnika koji koristi aplikaciju, što znači da on na sebi ima kameru i objekte koji predstavljaju ruke. Način na koji je ovaj deo povezan je uz pomoć XROrigin-a.

XROrigin je referenca na objekat (eng. GameObject) i predstavlja referentnu tačku u virtualnom prostoru, odnosno koordinatni sistem u kojem se odvija praćenje položaja i orijentacije korisnika ili uređaja. Svi ostali objekti i elementi u virtuelnom prostoru se pozicioniraju u odnosu na tu tačku. Tako je GenericVRPlayer postavljen unutar sfere. A na osnovu podataka koje dobija od VR uređaja, Unity može precizno pozicionirati i rotirati korisnikovu perspektivu unutar virtuelnog sveta.

Meta Quest 2 uređaj je opremljen senzorima kao što su akcelerometar, žiroskop i senzor za praćenje položaja. Ovi senzori generišu pokrete i rotacije uređaja i zatim ih obrađuju.

Unity kao razvojno okruženje koristi te podatke koje dobija od VR uređaja pomoću OpenXR-a, kako bi precizno pozicionirao i rotirao perspektivu korisnika unutar virtuelnog sveta. Na temelju tih podataka, Unity može ažurirati poziciju i rotaciju XR-origin objekta i drugih objekata u sceni kako bi odražavali stvarni položaj i rotaciju korisnika u VR prostoru, što omogućuje potpun doživljaj kada korisnik okreće glavu ili se kreće u prostoru, jer se virtuelni svet prilagođava tim promenama.

U sceni je fokus na gledanju videa, stoga je mogućnost slobodnog kretanja isključena.

Za interakciju korisnika sa objektima koristi se Input Action Manager. Unity je uveo paket Input System koji omogućuje korišćenje ovog menadžera kako bi se pojednostavilo definisanje i upravljanje ulaznim aktivnosti unutar Unity-a. To olakšava povezivanje sa funkcionalnostima u aplikaciji, bez obzira na platformu na kojoj se izvršava. Ovaj menadžer omogućava registraciju korisnika putem kontrolera. U ovom slučaju, dugmići na kontroleru koji će biti korišćeni za promenu video sadržaja su A, B, X i Y. Kada korisnik pritisne određeno dugme, Input Action Manager registruje tu promenu i pokreće odgovarajuću funkcionalnost u kodu.

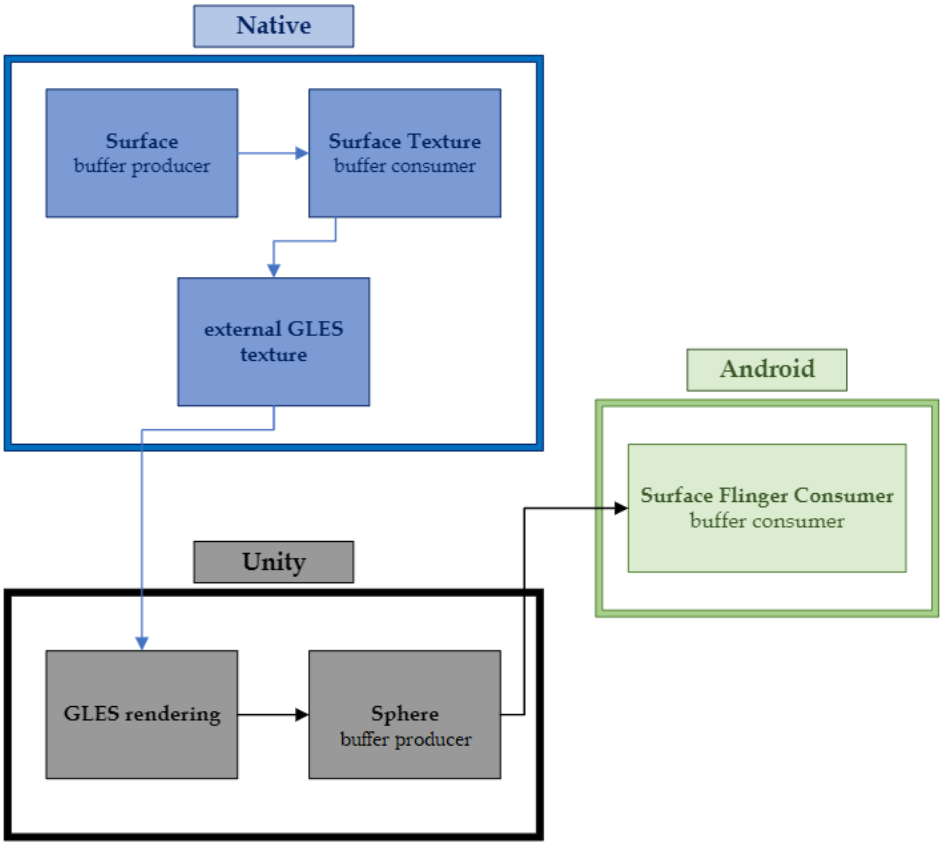
Uključivanje kontrolera omogućava korisnicima da menjaju videozapise. Sledeće što treba objasniti je kako se video iscrtava unutar sfere.

3.4 Proces reprodukcije i prikaza 360°-videa na sferi

Video plejer je ključna komponenta koja generiše video okvire (eng. frame). Svaki okvir predstavlja bafer podataka koji se koristi za prikazivanje slike. U kontekstu Unity-ja, koristi se SurfaceTexture i spoljna GLES tekstura (eng. External GLES texture), za prikazivanje video okvira na objektu u 3D sceni. SurfaceTexture kombinuje Surface i spoljnu GLES teksturu i koristi BufferQueue za upravljanje baferima koje generiše video plejer.

Kada se generiše novi okvir, on se smešta na površinu (eng. Surface), koja pruža način za iscrtavanje sadržaja. SurfaceTexture obaveštava aplikaciju kada se novi okvir stavi u red (eng. BufferQueue), zatim aplikacija preuzima novi bafer i čini ga dostupnim spoljnoj GLES teksturi.

GLES Rendering se koristi za iscrtavanje spoljne teksture, koja sadrži video okvir, na objektu u 3D sceni. Da bi se postiglo bojenje piksela na ekranu, koristi se šejder, program koji se izvršava na GPU-u i definiše kako će pikseli na ekranu biti obojeni na osnovu ulaznih podataka, u ovom slučaju, spoljne teksture [7].



Slika 3.3 Prikaz procesa reprodukcije

Na kraju, Unity scena pruža prostor za prikazivanje videa, koji se mapiraju na sferu u Unity sceni, što znači da će video okviri biti prikazani kao tekstura na površini sfere.

U ovom opisu nisu obuhvaćeni detalji o video koderu, jer nije relevantan za ovu specifičnu realizaciju. U drugim situacijama, video koder bi bio deo sistema koji kodira video okvire u odgovarajući format za snimanje ili prenos.

Ovaj koncept rešenja pruža osnovni pregled kako biste mogli da prikažete 360°-videe unutar sfere u VR sceni korišćenjem Unity platforme.

4. Programsko rešenje

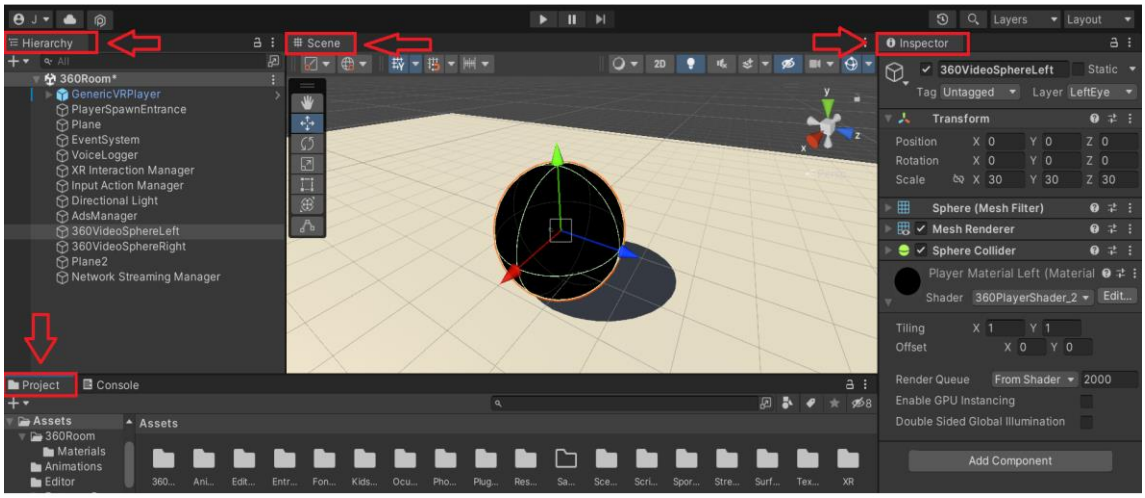
U ovom delu fokus će biti na programskom rešenju, detaljno objašnjenje izrade funkcionalnosti i tehničkih aspekata koji omogućavaju rad sistema za reprodukciju 360°-video snimaka u VR okruženju. Glavni alati i tehnologije korišćeni za razvoj su Microsoft Visual Studio, Unity razvojno okruženje, jezik C# za pisanje skripti i GLSL za modifikaciju šejdera.

4.1 Unity scena i struktura objekata

Bitne stvari koje se nalaze u sceni, potrebne da bi se pustio video su: dve sfere na kojima se prikazuju videi, *GenericVRPlayer* pomoću kog se vrši interakcija i gledaju videi, Input Action Manager, XR Interaction Manager, Event System.

Povezivanje skripti sa objektima u Unity-u omogućava izvršavanje određenih funkcionalnosti i logike na tim objektima. U nastavku je kratko objašnjenje kako su skripte povezane sa objektima u Unity-u:

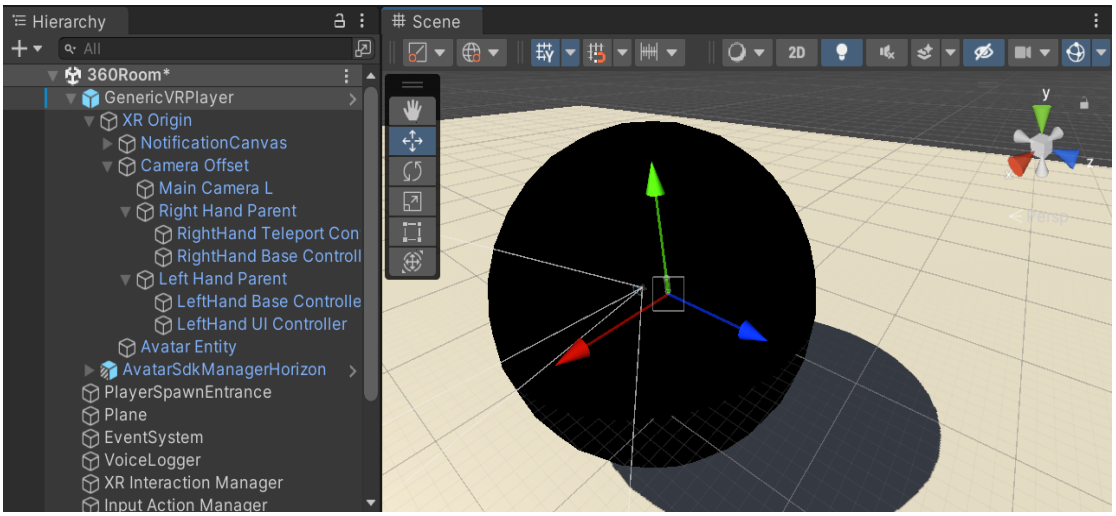
Za povezivanje skripte sa objektom, prvo je potrebno kreirati skriptu u Unity razvojnom okruženju ili u spoljnom uređivaču kao što je Microsoft Visual Studio. Nakon što je skripta kreirana, može se prikačiti na određeni objekat u sceni, klikom na njega, u *Inspector* polju se mogu videti sve komponente koje su vezane za objekat. Prevlačenje skripte se obično radi iz *Project* prozora sve do odgovarajućeg objekta u *Hierarchy* prozoru. Kada je skripta prevučena na objekat, može se pristupiti i manipulisati komponentama i svojstvima tog objekta putem koda u skripti, a takođe omogućava komunikaciju između objekata u sceni. Skripte mogu razmenjivati podatke, pozivati funkcije jedna drugoj.



Slika 4.1 Prikaz *Unity* prozora

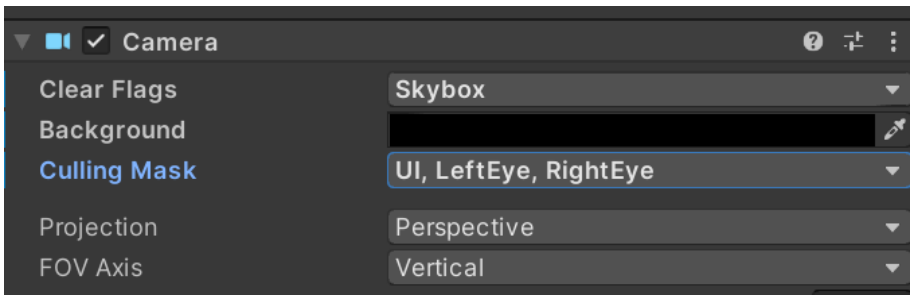
❖ **GenericVRPlayer**

Kamera kao svoje polje sadrži raslojavanje (eng. *Culling Mask)*, a njegovo značenje je koji sloj (eng. Layer) želimo da kamera prikazuje (eng. Render). U *Unity*-u, slojevi su kategorije ili grupe objekata koji se koriste za organizaciju i upravljanje elementima u sceni. Svaki objekat u *Unity* sceni može biti dodeljen određenom sloju, što omogućava lakšu kontrolu objekata koji međusobno interaguju i kako se ponašaju u različitim situacijama. U ovom slucaju, slojeve koristimo za kontrolu redosleda prikazivanja videa u sceni.



Slika 4.2 Struktura *GenericVRPlayer*-a i njegova pozicija u sceni

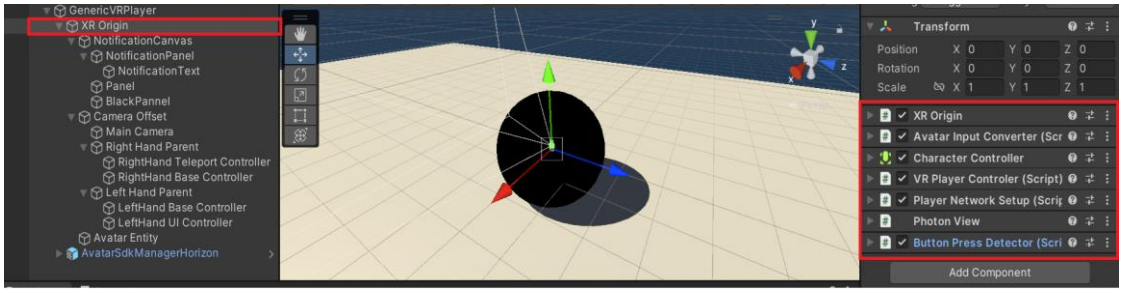
To je bitno, jer za stereoskopske videe, kao sto je već rečeno, želimo da se na svakom oku prikaže odgovarajuća slika. Iz tog razloga pravimo dva sloja. Zatim, jednoj sferi treba staviti jedan sloj, a drugoj drugi, i u malopre pomenutom polju na kameri *Culling Mask*, treba dodati oba sloja. Na taj način smo obezbedili dobar prikaz i jednom i drugom oku.



Slika 4.3 Polje *Culling Mask* sa odabranim slojevima

❖ **XROrigin**

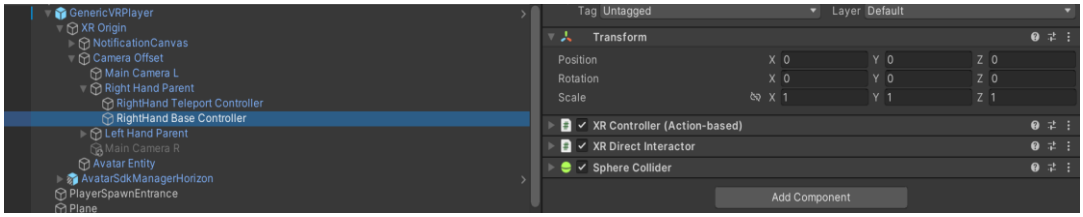
Komponente sa slike (Slika 4.4) zajedno omogućavaju praćenje kretanja korisnika (*XROrigin*), kontrolu avatara (*AvatarInputConverter*), implementaciju mrežne funkcionalnosti (*PhotonView* i *PlayerNetworkSetup*) i detekciju interakcije sa virtuelnim objektima u okviru virtuelne stvarnosti (*ButtonPressDetector*).



Slika 4.4 *XROrigin* sa svojim komponentama

❖ **Kontroleri**

Kada je reč o kontrolerima, kako bi ih mogli koristiti, skripte koje objekti moraju imati na sebi date su na slici:



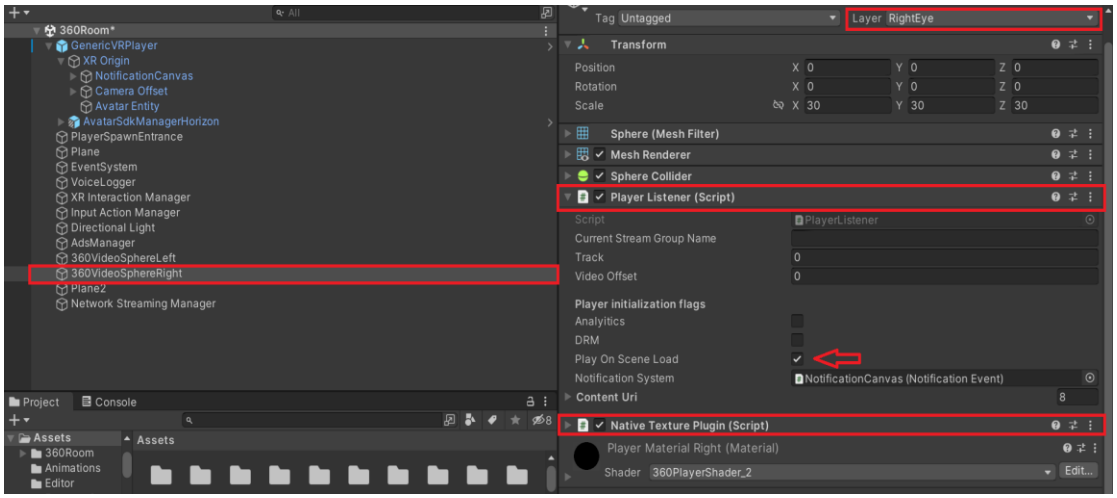
Slika 4.5 Komponente desnog kontrolera

Skripte *XRController (Action-Based)* i *XRDirectInteractor* se koriste u kontekstu interakcije korisnika sa virtuelnim objektima u sceni koristeći kontrolere u okviru *XR* sistema.

Koristeći praćenje i korisnički unos sa kontrolera, korisnik može da interaguje sa virtuelnim objektima na način koji simulira fizičko prisustvo i omogućava im da izvršavaju različite akcije, kao što su hvatanje, pomeranje ili aktiviranje funkcionalnosti objekata. Ove dve skripte, takođe se primenjuju na levi kontroler.

❖ Sfere

U sceni su prisutne dve sfere, od kojih svaka koristi isti šejder (*360PlayerShader_2*), ali ima svoj materijal (*PlayerMaterialRight* i *PlayerMaterialLeft*). Primenjene skripte na sferi su *PlayerListener* i *NativeTexturePlugin*. Dovoljno je na jednu sferu staviti ove dve skripte, a u nastavku će one biti objašnjene. Za desnu sferu je postavljen sloj *RightEye*, dok je za levu sferu izabran drugi sloj *LeftEye*, što je objašnjeno ranije.



Slika 4.6 Komponente desne sfere

U ovom delu analiziraćemo dve ključne klase, *PlayerListener* i *NativeTexturePlugin*, koje omogućavaju integraciju video plejera u *Unity* okruženje.

***PlayerListener* klasa**:

PlayerListener je klasa koja pruža funkcionalnost za kontrolu video plejera u *Unity* okruženju. Glavna funkcija ove klase je inicjializacija plejera, i mogućnost startovanja i zaustavljanja reprodukcije sadržaja na plejeru. Metode poput *StartPlayer()* i *StopPlayer()* omogućavaju kontrolu reprodukcije sadržaja na plejeru. Takođe, ova klasa može primati Android poruke koje se obrađuju radi ažuriranja korisničke sprege u *Unity* okruženju.

-Kreira se nova tekstura *Texture2D* sa postavljenim dimenzijama i formatom.

-Filtriranje teksture je postavljeno na *Point* kako bi se pikseli jasno videli.

-Metoda *Apply()* se poziva na teksturi kako bi se ona stvarno prenela na GPU.

-Zatim se nova tekstura postavlja na materijal objekta koji sadrži *PlayerListener* skriptu.

-Na kraju se poziva *SetTextureFromUnity()* funkcija iz *NativeTexturePlugin* skripte kako bi se predao pokazivač na teksturu i njene dimenzije.

Zatim se vrši inicijalizacija plejera tako što se kreira nova instanca klase *PlayerWrapper* iz *AndroidJavaClass*. Na kraju se ispisuje poruka u konzoli *Unity*-a koja označava da je inicijalizacija plejera završena.

I ukoliko je *playOnSceneLoad* parametar otkačen u *Inspector*-u, u tom slučaju se poziva *StartPlayer()* funkcija.

***NativeTexturePlugin* klasa:**

NativeTexturePlugin je klasa koja omogućava prikazivanje video sadržaja unutar *Unity* scene kroz spoljašnji *OpenGL* kontekst. Ova klasa se koristi za prikazivanje video sadržaja koji dolazi iz plejera na 3D objektima u sceni.

Metoda *SetTextureFromUnity()* omogućava postavljanje teksture za prikaz na materijalima objekata u sceni. Ova metoda se poziva iz *Unity*-a i prima pokazivač na teksturu, širinu i visinu teksture kao argumente. Kada se ova metoda pozove, pokazivač na teksturu se prenosi u *Android* ili *C++* deo koda, gde se koristi za postavljanje teksture na materijal objekta.

Metoda *SetMeshBuffersFromUnity()* omogućava postavljanje podataka o mreži (*vertex, normal, UV*) za prikaz na objektima.

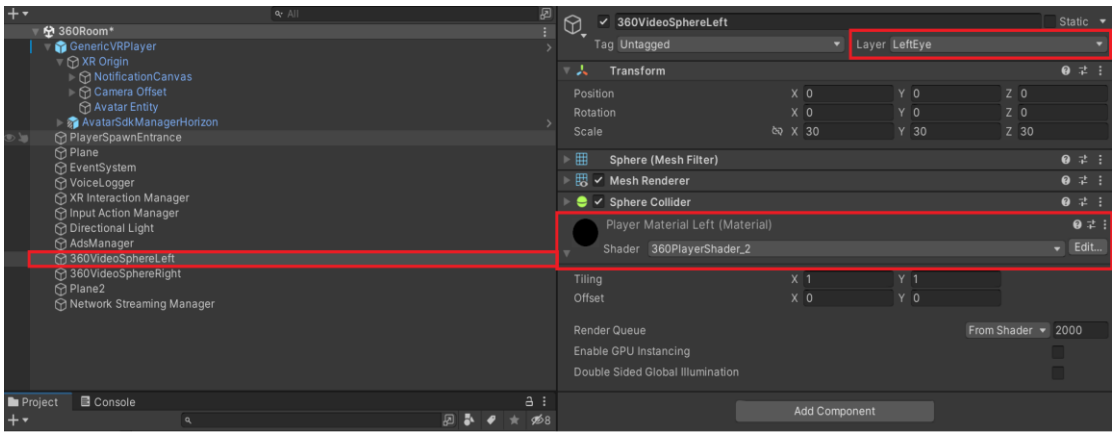
Metoda *SetFrameBufferFromUnity()* se koristi za postavljanje *framebuffer* objekta za renderovanje slike. Kada se ova metoda pozove, pokazivač na *framebuffer* se prenosi u *Android* ili *C++* deo koda, gde se koristi za postavljanje *framebuffera* za renderovanje slike. *Framebuffer* je poseban objekat koji se koristi za skladištenje slika generisanih tokom renderovanja i može se koristiti za različite svrhe, kao što je renderovanje tekstura, postprocesiranje ili prikazivanje slike na ekranu. Postavljanje *framebuffera* omogućava kontrolu nad procesom renderovanja slike u *Unity* sceni.

Kroz metodu *OnRenderEvent()* se vrši ažuriranje tekstura ili f*ramebuffer*-a prilikom renderovanja scene.

Uopšteno, *Android* deo ima ulogu posrednika između *Unity* platforme i nativnog (*C++*) koda kako bi se omogućila integracija i razmena podataka između ove dve platforme.

- **Ponavljanje i pomeraj**

U okviru *PlayerMaterialLeft* materijala, na slici Slika 4.8 vidimo dva polja (ponavljanje i pomeraj) o kojima je bilo reči u konceptu rešenja, a sada treba objasniti kako se svaki tip videa iscrtava na sferu.



Slika 4.7 Komponente leve sfere

		x	y
Left	**Tiling**	1	0.5
	Offset	0	0.5
Right	**Tiling**	1	0.5
	Offset	0	0

Tabela 4.1 Koordinate za TB tip

Ponavljanje za y osu će biti za obe sfere 0.5, jer želimo da se gornja polovina slike ponavlja ne jednom, već pola puta u vertikalnom smeru. Ako se postupak primeni i za levi i za desni materijal, obe sfere će prikazivati samo gornju polovinu teksture. Iz tog razloga, levu stranu teksture, treba pomeriti za polovinu, tako da y pomeraj za levi materijal ima vrednost 0.5. Istom logikom, treba promeniti ponavljanje i pomeraj za ostale tipove videa.

		x	y
Left	**Tiling**	0.5	1
	Offset	0.5	0
Right	**Tiling**	0.5	1
	Offset	0	0

Tabela 4.2 Koordinate za SBS tip

Za monoskopski video, treba nam samo jedan kanal, sto znači da se jedna slika prikazuje na oba oka. Pa iz tog razloga želimo celu sliku da prikažemo, što znači da će i ponavljanje i pomeraj za oba oka biti 1. Sva polja su definisana u skripti *ButtonPressDectecor*.

		x	**y**
Left	**Tiling**	1	1
	Offset	0	0
Right	**Tiling**	1	1
	Offset	0	0

Tabela 4.3 Koordinate za monoskopski tip

Sledeće što treba objasniti je na koji način se menja sadržaj videa. Skripta *ButtonPressDetector* se nalazi u sceni na objektu *XROrigin*, a u nastavku će biti detaljnije analizirana.

```
using UnityEngine;
using UnityEngine.InputSystem;
using IWP;
using System.Collections.Generic;
using World.Interactables;
using Utility;
using System.Collections;
```

Izraz *Using* u skripti, omogućava pristup svim klasama, funkcijama i ostalim definicijama koje su definisane unutar tog imena (eng. namespace) ili modula, bez potrebe da se eksplicitno navode sa svojim punim imenom svaki put kada se koriste.

- `public class ButtonPressDetector : MonoBehaviour {}`

MonoBehaviour je posebna vrsta klase u Unity-u koja omogućava objektima da imaju ponašanje, reaguju na događaje i izvršavaju određenu logiku. Klase koje nasljeđuju *MonoBehaviour* mogu koristiti različite metode koje se automatski pozivaju u određenim trenucima tokom životnog ciklusa objekta.

- `[SerializeField]`

```
private InputActionReference _leftController = null;
```

[SerializeField] je specifičan atribut koji se koristi u *Unity*-u kako bi se polje ili varijabla prikazali u *Inspector*-u, iako su privatni ili zaštićeni (private ili protected) u kodu, što znači da će to polje biti vidljivo u *Inspectoru*, gdje se može pristupiti i uređivati.
Sada ćemo proći kroz funkcije u ovoj skripti:

- `private void Awake();`

Awake funkcija je deo *Monobehaviour* klase, i poziva se prilikom učitavanja instance skripte. Koristi se za postavljanje osnovnih mehanizama detekcije pritiska na dugmadima. Takođe, registruje sistem obaveštenja (eng. Notification System) ako postoji.

Sistem obaveštenja se može implementirati kao deo korisničke sprege (UI) koji prikazuje obaveštenja na ekranu ili kao deo skripte ili komponente koja obrađuje logiku obaveštenja.

- `private IEnumerator Start();`

Start funkcija je korutina (eng. Coroutine). Korutina se koristi da bi omogućila asinhrona i odložena izvršavanja određenih funkcija ili sekvenci koda. Start funkcija dalje poziva:

- `private void FirstNotification();`

FirstNotification funkcija je odgovorna za prikazivanje prvog obaveštenja na sistemu obaveštenja. Postavlja tekst i trajanje obaveštenja. Kada je aplikacija pokrenuta, prvo obaveštenje koje se pojavi je informacija na koji način se menja sadržaj videa, kako bi korisnik odmah znao na koji način da koristi aplikaciju.

- ```
  private void ButtonAPressPerformed(InputAction.CallbackContext obj);
  private void ButtonBPressPerformed(InputAction.CallbackContext obj);
  private void ButtonXPressPerformed(InputAction.CallbackContext obj);
  private void ButtonYPressPerformed(InputAction.CallbackContext obj);
  ```

Sve četiri funkcije se aktiviraju kada se pritisne odgovarajuće dugme na kontroleru. Svaka od ovih funkcija obavlja sledeće korake:

1. Ažurira vrednosti ponavljanja i pomeraja za levu i desnu teksturu: Pre nego što se reprodukuje odgovarajući video, funkcija dobija potrebne vrednosti ponavljanja i pomeraja za odabrano dugme iz Awake funkcije. Te vrednosti se zatim postavljaju na odgovarajuće
  ```

materijale koje koriste dve sfere. Ovo omogućava vizuelno prilagođavanje prikaza na osnovu pritisnutog dugmeta.

2. Reprodukcija odgovarajućeg videa pomoću PlayerListener-a: Nakon ažuriranja vizuelnih parametara, poziva se:

- `private void PlayContent(int buttonIndex);`

PlayContent funkcija koja koristi PlayerListener objekat za pokretanje reprodukcije odgovarajućeg videa koji je povezan sa pritisnutim dugmetom. Na primer, ako je pritisnuto "Dugme A", reprodukovaće se odgovarajući monoskopski video.

3. Postavljanje trenutno pritisnutog indeksa dugmeta: Na kraju, funkcija postavlja "`currentlyPressedButtonIndex`" promenljivu na odgovarajuću vrednost kako bi se zapamtilo koje dugme je trenutno pritisnuto. Ovo omogućava kasnije provere i upravljanje trenutno aktivnim sadržajem u aplikaciji.

- `private void OnDisable();`

Ova funkcija je takođe deo MonoBehaviour klase i poziva se kada je instanca skripte onemogućena. Onemogućava ulazne akcije i zaustavlja plejer ako je trenutno pritisnuto neko dugme.

Sledeća stvar koju treba objasniti je šejder. Šejderi u Unity-u su programi koji omogućavaju detaljno kontrolisanje vizuelnih efekata i materijala objekata u 3D prostoru. Struktura šejdera sastoji se od nekoliko ključnih delova koji definišu kako će objekti biti prikazani i izgledati na ekranu.

Jedan od osnovnih delova šejdera su *Properties*. Ovde se definišu svojstva (eng. properties) koja se mogu koristiti u šejderu. Svojstva su parametri koji omogućavaju prilagođavanje šejdera putem Unity-ja. Na primer, "_Tiling" i "_Offset" su dva svojstva koja su deklarisana kao vektori, i koriste se za podešavanje ponavljanja i pomeraja teksture.

- Properties

```
{
        [ShowAsVector2] _Tiling("Tiling", Vector) = (0, 0, 0, 0)
        [ShowAsVector2] _Offset("Offset", Vector) = (0, 0, 0, 0)
}
```

SubShader je još jedan bitan deo šejdera. To je skup instrukcija za renderovanje objekta koji koristi ovaj šejder. Može postojati više *SubShader*-a unutar jednog šejdera, ali

Unity će automatski odabrati prvi koji podržava trenutnu platformu na kojoj se izvršava aplikacija.

- Cull front

Kada dodamo liniju "Cull front" u šejder, ona odbacuje, odnosno ignoriše prednje strane poligona, što znači da će prikazati zadnje strane svakog poligona koji se renderuje. To daje rezltat izgleda sfere koja je okrenuta naopako.

Tags su dodatni tagovi koji opisuju šejder. Oni pružaju informacije o renderovanju objekta. Na primer, u ovom rešenju, "*RenderType*" je postavljen na "*Opaque*" kako bi se naglasilo da je objekat neproziran.

- Tags { "RenderType" = "Opaque" }

Pass definiše jedan prolaz za renderovanje objekta. U jednom *SubShader*-u može postojati više prolaza, a svaki sadrži instrukcije za izvršavanje *vertex* i fragment šejdera. Svaki prolaz se izvršava nezavisno jedan od drugog i može se koristiti za primenu različitih efekata.

- Pass
  {
  GLSLPROGRAM
  #include "UnityCG.glslinc"
  #version 300 es
  uniform vec2 _Tiling;
  uniform vec2 _Offset;

  }

GLSLPROGRAM - naglašava da će sledeći kod biti napisan u GLSL jeziku.

#include "UnityCG.glslinc" – uključuje datoteku, koji je deo Unity-jeve biblioteke i sadrži korisne funkcije, makroe i definicije za pisanje šejdera u Unity-u.

#version 300 es - Ovo je deo GLSL sintakse gde se naglašava verzija jezika. Koristi se GLSL verzija 3.00, specifična za OpenGL ES 3.0 standard.

Vertex shader je deo šejdera koji manipuliše pozicijom i transformacijama svakog vrha objekta. Ovde se obavljaju izračunavanja vezana za geometriju objekta.

- #ifdef VERTEX
  void main()
  {
  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
  gl_TexCoord[0] = gl_MultiTexCoord0;
  }
  #endif

Svaki *vertex* deo mora imati *main()* funkciju, a ona će biti pozvana za svaki vrh koji se renderuje.

- gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

Ova linija koda transformiše poziciju vrha (*gl_Vertex*) koristeći *Model-View-Projection* matricu (*gl_ModelViewProjectionMatrix*). Ova matrica kombinuje modelnu matricu (koja predstavlja poziciju, rotaciju i skaliranje objekta), matricu pogleda (koja predstavlja pogled kamere) i matricu projekcije (koja predstavlja projekciju 3D scene na 2D ekran). Rezultat ove transformacije se smešta u *gl_Position*, koji predstavlja rezultujuću poziciju vrha koja će biti prosleđena dalje za renderovanje.

- gl_TexCoord[0] = gl_MultiTexCoord0;

Ova linija koda postavlja teksturnu koordinatu (*gl_MultiTexCoord0*) u *gl_TexCoord[0]* vektor. Teksturne koordinate se koriste za uzorkovanje teksture u fragment šejderu kako bi se odredila boja piksela. Ovde se pretpostavlja da se koristi samo jedna teksturna koordinata (*gl_TexCoord[0]*), ali moguće je koristiti više koordinata.

Fragment šejder je deo šejdera koji manipuliše pikselima objekta. On određuje boje i druge karakteristike svakog piksela na ekranu. U fragment šejderu se često koriste teksture i svetlosni modeli za postizanje željenog izgleda objekta.

```
#ifdef FRAGMENT
	#extension GL_OES_EGL_image_external : require
	#extension GL_OES_EGL_image_external_essl3 : require

	uniform samplerExternalOES texSampler0;

	void main()
	{
		vec2 vc2 = gl_TexCoord[0].st;
		vc2.x = 1.0 – vc2.x;
		vc2.y = 1.0 - vc2.y;

		vec4 color = texture2D(texSampler0, vc2.xy * _Tiling + _Offset);
		vec3 sRGB = color.rgb;
		gl_FragColor = vec4(sRGB * (sRGB * (sRGB * 0.305306011 +
	0.682171111) + 0.012522878), color.a);

	}
#endif
```

vec2 vc2 - Ova linija koda uzima koordinate teksture (*gl_TexCoord[0]*) za trenutni piksel i smešta ih u *vc2* vektor.

vc2.x - Invertuje vertikalnu komponentu koordinata teksture kako bi se prilagodio načinu na koji se tekstura prikazuje na ekranu.

vec3 sRGB - Izdvaja RGB komponente boje piksela iz color promenljive i smešta ih u *sRGB* vektor.

gl_FragColor - Primenjuje postupak kolor transformacije na *sRGB* vektor kako bi se dobila konačna boja piksela. Rezultujuća boja se smešta u *gl_FragColor*, koja predstavlja izlaznu boju piksela fragment shadera. *Alpha* vrednost boje se očuva iz *color.a*.

FallBack je rezervni šejder koji se koristi ako trenutni ne radi na određenoj platformi. Kada Unity ne može pronaći podržani šejder za platformu, automatski se koristi rezervni kao zamena. "*Diffuse*" je jedan od ugrađenih Unity šejdera koji pruža osnovno osvetljenje i teksture.

- FallBack "Diffuse", 1

Ova struktura šejdera omogućava stvaranje različitih vizuelnih efekata i prilagođavanje izgleda objekata u Unity-u. Razumevanje ovih delova i njihova pravilna upotreba pomažu u kreiranju impresivnih grafičkih rešenja.

U delu *Properties* u šejderu se nalazi [*ShowAsVector2*]. Skripta *ShowAsVector2Drawer* je smeštena u Editor folderu i koristi se kako bi se prilagodio prikaz vektorskih svojstava u inspektoru materijala. Ova skripta proširuje klasu *MaterialPropertyDrawer*, koja omogućava pravljenje prilagođenih prikaza UI elemenata za svojstva materijala.

Skripta *ShowAsVector2Drawer* se smešta u Editor folder u Unity-u iz razloga što Editor folder služi za smeštanje skripti koje se koriste za proširenje funkcionalnosti Unity Editora. Sve skripte smeštene u ovom folderu se kompajliraju samo prilikom izgradnje samog Editora, a ne prilikom izgradnje finalne igre ili aplikacije.

Ono što je potrebno prikazati u Editoru su samo dve koordinate (x i y) umesto standardnih četiri koordinate (x, y, z, w) za ponavljanje i pomeraj u šejderu. Kroz korišćenje skripte *ShowAsVector2Drawer* i atributa *[ShowAsVector2]* na vektorskim svojstvima u šejderu, postiže se prilagođeni prikaz sa samo dva polja za unos koordinate, čime se pojednostavljuje i smanjuje konfuzija prilikom podešavanja vrednosti.

Slika 4.8 Materijal za sferu

5. Ispitivanje rešenja

U ovom odeljku će biti ispitana implementacija različitih funkcionalnosti u okviru Unity razvojnog okruženja. Cilj ispitivanja je provera da li implementirane funkcije rade prema očekivanjima i da li se postiže željeni rezultat pri interakciji sa korisnikom.

Detalji o ispitivanju i rezultati će biti prikazani u tabeli koja sledi. Tabela sadrži opis svake funkcionalnosti, očekivano ponašanje, i rezultat ispitivanja za svaku funkcionalnost.

Testovi su izvršavani ručno, na Meta Quest 2 uređaju. Primeri izvršenih testova su dati u nastavku. U tabeli 5.1 su prikazani sprovedeni ručni testovi.

Rb.	Funkcija	Opis ispitnog slučaja	Očekivani rezultati	Uspešnost testa
1.	FirstNotification()	Pri ulazu u scenu, treba da bude ispisana notifikacija za korišćenje kontrolera	Ispisana notifikacija ispred korisnika, sa sadržajem dugmića koje treba koristiti za puštanje videa.	Uspešno
2.	Start()	U Unity Editor-u u okviru *PlayerListener* skripte postoji polje `playOnSceneLoad`. Ukoliko je ono otkačeno, nakon uspešno inicijalizovanog plejera, poziva se *StartPlayer()* i pušta se prvi video iz liste.	Prvi video iz liste u *PlayerListener* skripti počinje pri ulasku u scenu.	Uspešno
3.	Šejder: 360PlayerShader_2	Uz pomoć šejdera, koji određuje konačan izgled videa, ukoliko je pravilno napisan, treba prikazati video sa unutrašnje strane.	Svi videi se nalaze sa unutrašnje strane sfere.	Uspešno
4.	ButtonAPressPerformed()	Na osnovu informacija sa prve notifikacije, pri pritisku dugmeta A na kontroleru, biće pušten prvi tip videa.	Pušten prvi video iz liste videa.	Uspešno
5.	ButtonBPressPerformed()	Pri pritisku dugmeta B na kontroleru, biće pušten drugi tip videa.	Pušten drugi video iz liste videa.	Uspešno
6.	ButtonXPressPerformed()	Pri pritisku dugmeta B na kontroleru, biće pušten treći tip videa.	Pušten treći video iz liste videa.	Uspešno
7.	ButtonYPressPerformed()	Pri pritisku dugmeta Y na kontroleru, biće pušten četvrti tip videa.	Pušten četvrti video iz liste videa.	Uspešno
8.	SetVector()	Svaki tip videa podrazumeva različito iscrtavanje na sferi. Funkcija *SetVector()* postavlja za svaki video odgovarajuće x i y koordinate.	Očekivane koordinate za svaki od videa objašnjene su u programskom rešenju.	Uspešno
9.	NotificationAction()	Za svaki pušten video, izlazi odgovarajuća notifikacija koja kaže koji tip videa je pušten.	Tako će pisati za pritisnuto dugme: ***A***-*Monoscopic Video* **B**-*Stereoscopic Equirectengular Video* **X**-*Stereoscopic SBS Video* **Y**-*Stereoscopic Bottom-Up Video*	Uspešno

Tabela 5.1 Ispitivanje funkcionalnosti

6. Zaključak

U zaključku ovog dela možemo sumirati da je u radu opisan programski pristup za reprodukciju 360°-video snimaka u VR okruženju korišćenjem Unity razvojnog okruženja i Microsoft Visual Studio alata. Implementirane su funkcionalnosti koje omogućavaju interakciju sa korisnikom i prikazivanje različitih vrsta 360°-video sadržaja.

Dalji pravci razvoja mogu uključivati unapređenje korisničke sprege, dodavanje dodatnih funkcionalnosti poput kontrolisanja reprodukcije (pauza, stopiranje, premotavanje), mogućnost učitavanja videa sa spoljnih izvora. Takođe, mogu se istražiti mogućnosti optimizacije performansi i podrške za različite VR uređaje.

Implementacija programa za reprodukciju 360°-video snimaka u VR okruženju omogućava korisnicima da uživaju u bogatom i interaktivnom iskustvu virtuelne stvarnosti. Ovaj pristup može biti od koristi za različite svrhe, kao što su obrazovanje, zabava, turizam i druge oblasti gde je imerzivno iskustvo od ključne važnosti.

7. Literatura

[1] Android Developer: https://developer.android.com/ , *učitano 5.6.2023*

[2] Meta , https://www.meta.com/, *učitano 15.6.2023*

[3]OpenXR , https://www.khronos.org/openxr/, *učitano 16.6.2023*

[4]Shaders , https://docs.unity3d.com/Manual/Shaders.html, *učitano 10.6.2023*

[5]Opengl shader language, https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language, *učitano 18.6.2023*

[6] Unity, https://unity.com/, *učitano 5.6.2023*

[7]Android Surface, https://source.android.com/docs/core/graphics/arch-st, *učitano 18.6.2023*