



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Имплементација подршке за репродукцију видео садржаја на Андроид платформама за проширену стварност

ДИПЛОМСКИ РАД
- Основне академске студије -

Ментор: проф. др. Илија Башичевић

Студент: Душан Стојковић РА 140\2019

Нови Сад, 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 :	Душан Стојковић
Ментор, МН :	проф. др. Илија Башичевић
Наслов рада, НР :	Имплементација подршке за репродукцију видео садржаја на Андроид платформама за проширену стварност
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)	7/74/6/32/26/0/0
Научна област, НО :	Електротехника и рачунарство
Научна дисциплина, НД :	Рачунарска техника
Предметна одредница/Кључне речи, ПО :	Виртуелна стварност, Погон игара, Рачунарска графика, Програмска подршка за репродукцију видео садржаја
УДК	
Чува се, ЧУ :	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН :	
Извод, ИЗ :	Овај рад је један начин да се направи програмска подршка за приказ видео садржаја у проширеној стварности. Циљ је интегрисати видео плејер у погон игара <i>Unity</i> као проширење за приказ видео садржаја у 3D простору. Имплементација подржава да се више видео канала могу исцртавати истовремено у 3D простору. Преносивост је демонстрирана на Андроид платформама (Poco X3 NFC, Oculus Quest 2, NReal Air). Провера исправности се заснива на снимању и анализи оквира који се исцртавају на екрану у RenderDoc алату. Употребом Андроид алата за профилисање анализиране су перформансе решења.
Датум прихватања теме, ДП :	
Датум одбране, ДО :	
Чланови комисије, КО :	Председник: проф. др. Мирослав Поповић
	Члан: проф. др. Иван Каштелан
	Члан, ментор: Проф. др. Илија Башичевић
	Потпис ментора



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 :	Dušan Stojković
Mentor, MN :	Ilija Bašičević, PhD
Title, TI :	Implementation of video playback software on Android extended reality platforms
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)	7/74/6/32/26/0/0
Scientific field, SF :	Electrical Engineering
Scientific discipline, SD :	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, S/KW :	Virtual reality, game engine, computer graphics, software for video reproduction
UC	
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, N :	
Abstract, AB :	This paper describes a software solution for video reproduction in extended reality. The goal is to integrate a video player into the Unity game engine as a plugin to display video content in 3D space. The implementation supports displaying multiple video streams at the same time in 3D space. The portability of the solution is demonstrated on multiple Android platforms (Poco X3 NFC, Oculus Quest 2, NReal Air). Verification of the solution is based on capturing and analysis of rendered frames in the RenderDoc tool. Using Android profiling tools the performance of the solution was analyzed.
Accepted by the Scientific Board on, ASB :	
Defended on, DE :	
Defended Board, DB :	President: Miroslav Popović, PhD
	Member: Ivan Kaštelan, PhD
	Member, Mentor: Ilija Bašičević, PhD
	Mentor's sign

Zahvalnost

Zahvalan sam porodici i prijateljima na podršci tokom čitavog obrazovanja.

Zahvaljujem se mentoru, kolegama i saradnicima na stručnoj podršci tokom razvojnog projekta.

SADRŽAJ

1. Uvod.....	8
2. Teorijske osnove	10
2.1 Android XR uređaji	10
2.1.1 VR uređaji	11
2.1.2 AR uređaji	11
2.1.3 MR uređaji.....	12
2.2 <i>OpenXR</i> standard za razvoj XR aplikacija.....	13
2.3 XR aplikacije na Android platformama.....	14
2.3.1 Pravljenje XR Android aplikacija u Unity pogonu igara	15
2.3.2 Modelovanje <i>Unity</i> scene u obliku XR Android aplikacije	17
2.3.3 <i>Android</i> ciklus aplikacije.....	18
2.3.4 Životni ciklus <i>Unity</i> skripti.	20
2.3.5 Dinamičko učitavanje Android biblioteka putem <i>Unity</i> skripti	23
2.3.6 Standardna grafička procesna linija <i>Unity</i> pogona.....	24
2.3.6.1 Komandni baferi SRP	25
2.3.6.2 Režimi iscrtavanja na XR uređajima	26
2.4 OpenGL ES biblioteka na Android platformama	27
2.4.1 OpenGL ES za prikaz videa u obliku ekrana pozicioniranog u 3D prostoru ..	28
2.4.2 Teksture za prikaz video strima u Androidu	30
3. Koncept rešenja.....	32
3.1 XR profili za interakciju	34
3.1.1 Podsystem za interakciju sa plejerom	35
3.2 Tok izvršavanja aplikacije	36
3.3 Upravljanje teksturama za prikaz videa u C++ kodu.....	39
3.4 Uključivanje grafičkih biblioteka u C++ biblioteku	41
3.5 Problem linearnog prostora boja	41
4. Programsko rešenje	43
4.1 Unity skripte	43
4.1.1 Podešavanje Unity radnog okruženja za XR interakciju.....	43
4.1.2 Realizacija upravljanja plejerom unutar Unity scene upotrebom	
PlayerInteractor klase	48
4.1.3 Realizacija proširivanja grafičkog toka SRP upotrebom NativeTexturePlugin	
klase	49
4.1.4 Realizacija SceneReload klase	50
4.2 Realizacija Java omotača za upravljanje plejerom	51

4.2.1	Realizacija PlayerWrapper klase.....	51
4.2.2	Realizacija PlayerListener klase.....	53
4.2.3	Realizacija UnityMessenger klase	55
4.3	Realizacija OpenGL ES biblioteke za iscrtavanje videa	55
4.3.1	Realizacija RenderingPlugin modula	56
4.3.1.1	Realizacija funkcija koje se povezuju u Unity pogon.....	57
4.3.1.2	Realizacija JNI funkcija RenderingPlugin modula.....	58
4.3.1.3	Funkcije koje se dinamički uvezuju u NativeTexturePlugin C# klasu	58
4.3.1.4	Realizacija internih funkcija	59
4.3.2	Realizacija GLHelpers modula	59
4.3.3	Realizacija ShaderHelper klase	61
5.	Evaluacija.....	63
5.1	Opis okruženja za ispitivanje	63
5.2	Poređenje kvaliteta slike sa i bez gama aproksimacije	65
5.3	Ispitivanje komandnih bafera.....	66
5.3.1	Ispitivanje komandnih bafera na <i>Oculus Quest 2</i>	67
5.3.2	Ispitivanje komandnih bafera na Poco X3 NFC telefonu	68
5.3.3	Ispitivanje komandnih bafera na NReal Air naočarima	69
5.4	Ispitivanje performansi Android aplikacije	69
5.5	Ispitivanje toka izvršavanja i upravljanja resursima	70
5.6	Diskusija rezultata.....	71
6.	Zaključak	73
7.	Literatura.....	75

SPISAK SLIKA

Slika 1 <i>Oculus Quest 2</i> , Android uređaj za VR.....	11
Slika 2 <i>NReal Air AR</i> naočare.....	12
Slika 3 <i>Lenovo Thinkreality MR</i> naočare.	13
Slika 4 Tok izvršavanja XR sistema.	14
Slika 5 Unity radno okruženje.	16
Slika 6. Android životni ciklus aktivnosti.	19
Slika 7 <i>MonoBehaviour</i> tok izvršavanja.....	23
Slika 8 Seckanje preglednog prostora	25
Slika 9 Ekran iscrtan u 3D prostoru primenom šejdera interpolacije.....	29
Slika 10 Android aktivnost kontinualne obrade video strima.	31
Slika 11 Arhitektura modula koji čine rešenje	32
Slika 12 Dijagram toka iscrtavanja videa unutar Unity scene.....	38
Slika 13 Konfiguracija rešenja za iscrtavanje videa u komandni bafer.....	39
Slika 14 Oslobođanje resursa iscrtavanja video ekrana.	39
Slika 15 Osnovni set XR interakcija.	44
Slika 16 Menadžer XR interakcija.....	44
Slika 17 Podešavanje <i>XROrigin</i> objekta u inspektoru.....	44
Slika 18 <i>XROrigin</i> objekat i njegova deca unutar scene.	45
Slika 19 Konfiguracija kamera objekta.	45
Slika 20 Menadžer kontrolera baziran na akcijama.....	46
Slika 21 Osnovna komponenta kontrolera.	46
Slika 22 Komponente za interakciju zrakom.....	47
Slika 23 Komponente interaktivnog ekrana u sceni.	47
Slika 24 Prikaz Unity scene za ispitivanje rešenja.	64
Slika 25 Prozor <i>RenderDoc</i> alata kada je zakačen na <i>Oculus Quest 2</i>	64
Slika 26 <i>Perfetto</i> alat za analizu Android/Linux aplikacija.....	65
Slika 27 Video iscrtan šejderom bez gama aproksimacije.	66
Slika 28 Video iscrtan šejderom sa gama aproksimacijom.	66
Slika 29 Okvir slike uhvaćen <i>RenderDoc</i> alatom na <i>Oculus Quest 2</i> u višestrukom režimu.	67
Slika 30 Okvir slike uhvaćen <i>RenderDoc</i> alatom na <i>Oculus Quest 2</i> u jednoprolaznom režimu.	68
Slika 31 Okvir slike uhvaćen <i>RenderDoc</i> alatom na <i>Poco X3 NFC</i> u višestrukom režimu.	68

Slika 32 Okvir slike uhvaćen RenderDoc alatom na Poco X3 NFC u jednoprolaznom režimu.	69
Slika 33 Grafik CPU performansi Android aplikacije sa četiri video ekrana na Poco X3 NFC.	69
Slika 34 Grafik memorijskih performansi Android aplikacije sa četiri video ekrana na Poco X3 NFC.	70

SPISAK TABELA

Tabela 1. Poređenje pogona igara.....	15
Tabela 2 Stanja plejera.	48
Tabela 3 Polja klase <i>PlayerInteractor</i>	48
Tabela 4 Metode <i>PlayerInteractor</i> klase.	49
Tabela 5 Polja <i>NativeTexturePlugin</i> klase.....	50
Tabela 6 Metode <i>NativeTexturePlugin</i> klase.....	50
Tabela 7 Metode <i>SceneReload</i> klase.....	51
Tabela 8 Polja <i>PlayerWrapper</i> klase.	52
Tabela 9 Metode <i>PlayerWrapper</i> klase.	53
Tabela 10 Polja <i>PlayerListener</i> klase.....	54
Tabela 11 Metode <i>PlayerListener</i> klase.....	55
Tabela 12 Metode <i>UnityMessenger</i> klase.	55
Tabela 13 Datoteke unutar C++ grafičke biblioteke.	56
Tabela 14 Polja <i>JNIRendering</i> strukture.	56
Tabela 15 Globalne promenljive unutar <i>RenderingPlugin</i> datoteke.....	57
Tabela 16 Funkcije koje se povezuju sa Unity pogonom.	58
Tabela 17 JNI metode unutar <i>RenderingPlugin</i> datoteke.	58
Tabela 18 Funkcije koje se dinamički uvezuju sa <i>NativeTexturePlugin</i> klasom.	59
Tabela 19 Interne funkcije <i>RenderingPlugin</i> datoteke.	59
Tabela 20 Promenljive <i>GLHelpers</i> modula.	60
Tabela 21 Funkcije <i>GLHelpers</i> modula.....	61
Tabela 22 Polja <i>ShaderHelper</i> klase.	61
Tabela 23 Javne metode <i>ShaderHelper</i> klase.	62
Tabela 24 Privatne metode <i>ShaderHelper</i> klase.....	62

SKRAĆENICE

VR	- <i>Virtual Reality, Virtelna stvarnost</i>
XR	- <i>Extended Reality, Proširena stvarnost</i>
ADB	- <i>Android Debug bridge, Android most za otklanjanje greški</i>
AR	- <i>Augmented Reality, Izmenjena stvarnost</i>
MR	- <i>Mixed Reality, Izmešana stvarnost</i>
GPU	- <i>Graphics Processing Unit, grafička procesorska jedinica</i>
SOC	- <i>System on chip, Sistem na čipu</i>
JVM	- <i>Java virtual machine, Java virtuelna mašina</i>
API	- <i>Application program interface, Programski interfejs za aplikacije</i>
JNI	- <i>Java native interface, Java nativni interfejs</i>
SRP	- <i>Standard render pipeline, Standardna grafička procesna linija</i>
HAL	- <i>Hardware abstraction layer, sloj hardverske apstrakcije</i>
IPC	- <i>Inter process communication, Među proceska komunikacija</i>
UI	- <i>User Interface, Korisnički interfejs</i>
FPS	- <i>Frames per second, Brzina smenjivanja slika</i>
URI	- <i>Uniform Resource Identifier, Jedinstveno ime resursa</i>
NDK	- <i>Native Development Kit, Alati za nativni razvoj</i>
URP	- <i>Universal Rendering Pipeline, Univerzalna grafička procesna linija</i>

1. Uvod

Reprodukcija videa u proširenoj stvarnosti se fundamentalno razlikuje od prikaza na ekranu po tome što se video mora iscertavati u 3D prostoru. Prikaz videa u 3D prostoru na ekranu zahteva da se prozor kroz koji korisnik gleda 3D svet pomera. Pomeranje prikaza u 3D prostoru se na računarima opšte namene postizalo tastaturom i mišem. Naspram toga, uređaji za virtuelnu stvarnost, kao *Oculus Quest 2*, su kretanje prozora kroz koji korisnik gleda u virtuelni prostor izvršili senzorima koji prate kretanje glave i džojsticima koji omogućuju pomeranje celog tela u 3D prostoru. Namenske platforme za *XR - proširenu stvarnost* su uznapredovale dosta od njihovih prvobitnih mogućnosti.

Platforme za virtuelnu stvarnost koje su korišćene u ovom radu su bazirane na Android platformi. Specifično, fokus je na računarskoj grafici i reprodukciji videa na Android platformama zbog toga što video plejer koji će služiti za dekodovanje videa u ovom radu dat kao Android arhiva. Jedan od aspekta od velikog značaja je prenosivost rešenja koja se bazira na *OpenXR* standardu za razvoj aplikacija za proširenu stvarnost.

Unity pogon igara predstavlja osnovu rada zbog njegove snažne podrške od XR zajednice. Pravljenje XR aplikacija je dobro dokumentovano i postoje funkcionalnosti za interakciju sa virtuelnim 3D objektima, rukama ili kontrolerima. Upotrebom Unity pogona igara se postiže visok nivo apstrakcije za modeliranje virtuelnog prostora i implementaciju interakcije korisnika sa okruženjem u virtuelnoj stvarnosti. Takođe poseduje izuzetno visok stepen integracije sa grafičkim funkcionalnostima Androida.

Računarska grafika u aplikaciji je razvijena po OpenGL ES 3.1 standardu jer sadrži proširenja za reprodukciju video sadržaja na Android platformama.

Zadatak ovog rada je da se proširi Unity pogon igara sa programskom podrškom za reprodukciju video sadržaja. Cilj je da se postojeći video plejer koji je Android biblioteka

integriše u Unity, njegove funkcionalnosti budu upotrebljive u Unity skriptama i video reprodukuje unutar Unity scene.

Razvijen je API u Android Studio radnom okruženju koji služi kao među sloj između Unity skripti i video plejera. Za otklanjanje greški u iscertavanju video okvira je korišćen *RenderDoc* alat, a za otklanjanje greški u logici ADB. Za pisanje Unity skripti korišćen je Visual Studio 2022.

Rad se sastoji iz sedam poglavlja.

Drugo poglavlje, za početak, se bavi terminom XR i objašnjava slojeve njegovog značenja. Zatim se uvodi *OpenXR* standard za razvoj XR aplikacija. Pokazuje se koja je uloga *OpenXR* standarda u XR aplikacijama i govori se o osnovama *OpenXR* toka izvršavanja. Pogon igara pruža podršku na visokom nivou za razvoj, održavanje i proširivanje projekta. Pogoni igara poseduju integrisano radno okruženje za razvoj aplikacija. U narednom delu se vrši poređenje nekoliko varijanti za razvoj aplikacija, sa i bez pogona. Izdvaja se Unity pogon igara kao alat za pravljenje Android aplikacija za XR. Spominje se uloga pogona igara u modelovanju scene u kojoj se korisnik nalazi. Uvodi se koncept među sloja koji služi kao sprega između Unity skripti i Android Java objekta, kao i dinamičkih C/C++ biblioteka po *OpenGL ES* standardu za računarsku grafiku. Objašnjava se metod za proširenje protočne strukture grafike koju Unity koristi sa korakom za video prikaz. Izdvajaju se osnove računarske grafike po *OpenGL ES* standardu koje su dovoljne za prikaz video sadržaja u 3D prostoru. Za kraj se uvode specifičnosti za reprodukciju video sadržaja na Android platformama za XR iz aspekta programske podrške.

U trećem poglavlju prikazana je arhitektura programske podrške rešenja. Tri značajne celine ove programske podrške su: video plejer, sloj između plejera i Unity pogona igara, Unity scena koja sadrži skripte neophodne reprodukciju videa kao i skripte za interakciju korisnika sa aplikacijom.

U četvrtom poglavlju se opisuju moduli programske podrške koji su podeljeni u celine po programskom jeziku: proširenje protočne strukture grafike *Unity* pogona (C++), sloj između Unity skripti i video plejera (Java), Unity skripte za reprodukciju videa i interakciju sa korisnikom aplikacije (C#) i program za GPU namenjen za iscertavanje (*engl. rendering*) videa (GLSL).

U petom poglavlju se ispitivanja dele u dve kategorije: ispituje se ponašanje proširenja protočne strukture *Unity* pogona i redosled događaja neophodnih da se video reprodukuje tako što korisnik interaguje sa Unity scenom. Na kraju se daje se prikaz rezultata u slučajevima kada se reprodukuju jedan ili više videa istovremeno.

Šesto poglavlje je zaključak ispitivanja sprovedenog i kratak osvrt na dalja unapređenja realizovane arhitekture za video reprodukciju u *Unity* pogonu.

2. Teorijske osnove

Android je operativni sistem razvijen od strane kompanije *Google* i koristi se na širokom spektru pametnih telefona, tableta i drugih uređaja. U poslednje vreme, Android je takođe postao popularan u svetu proširene stvarnosti *XR*.

XR uređaji, koji uključuju virtualnu stvarnost *VR*, izmenjenu stvarnost *AR* i izmešanu stvarnost *MR* pružaju korisnicima jedinstveno iskustvo koje kombinuje digitalne elemente sa stvarnim svetom. Android platforma je postala izuzetno popularna među razvojnim timovima i korisnicima koji žele da istraže i iskoriste mogućnosti *XR* tehnologije.

Uz sve veći broj *XR* uređaja koji koriste Android, postoji i veliki broj razvojnih alata i resursa koji pomažu programerima da stvaraju inovativne *XR* aplikacije. *XR* uređaji na Android platformi često koriste senzore poput akcelerometra, žiroskopa i kamera kako bi pratili pokrete korisnika i omogućili interakciju sa digitalnim sadržajem u realnom svetu. Na primer, možete koristiti *AR* uređaje za projekciju virtuelnih objekata u stvarnom okruženju, dok *VR* uređaji korisnicima pružaju potpuno uronjeno iskustvo u virtuelni svet.

2.1 Android *XR* uređaji

Glavna osobina koji Android *XR* uređaji imaju zajedničku je dizajniranje *XR* uređaja kao perifernog dodatka za Android telefon ili personalni računar. Ideja je da se smanjuje cena uređaja i omogućuje znatno brži odziv aplikacije jer se sva obrada događa na Android telefonu, personalnom računaru, negde na oblaku ili na konzoli za igrice. U ovom režimu *XR* uređaj u sebi koristi senzore i sistem za prikaz sadržaja tako da je snažan SOC za kompleksne grafičke obrade informacija nepotreban.

Android platforme za *XR* se dele po svojim mogućnostima. U nastavku ovog poglavlja navode se neki primeri *XR* uređaja, kako bi se pokazale razlike između *VR*, *AR* i *MR*. Koncept

XR-a je uveden kako bi se grupisali uređaji po svojim mogućnostima i uveo standard za razvoj aplikacija na agnostički način prema platformi.

2.1.1 VR uređaji

Putem VR uređaja omogućuje se korisnicima da se u potpunosti urone u digitalna iskustva. Korisnici mogu istraživati virtualne prostore, učestvovati u simulacijama i interaktivnim igrama, ili jednostavno uživati u vizuelnom sadržaju koji izgleda kao da je stvaran.

VR tehnologija koristi napredne senzore i ekrane visoke rezolucije kako bi stvorila osećaj prisutnosti duboko u virtualnom svetu. Uređaji poput VR naočara *Oculus Quest 2* omogućavaju korisnicima da vide i čuju virtualni sadržaj, dok kontroleri omogućavaju interakciju sa okruženjem.



Slika 1 *Oculus Quest 2*, Android uređaj za VR.

Sa VR uređajima, korisnici mogu doživeti avanture, putovati na udaljene lokacije, prisustvovati sportskim događajima ili umetničkim performansima, ili čak naučiti nove veštine kroz realistične simulacije. Ova tehnologija ima široku primenu u zabavi, obrazovanju, medicini, arhitekturi i mnogim drugim oblastima. Uz stalni razvoj VR tehnologije, očekuje se da će VR iskustva postati još impresivnija i pristupačnija.

2.1.2 AR uređaji

AR tehnologija koristi kameru i senzore uređaja kako bi prepoznala stvarne objekte i prostor i na njih projektovala virtualne elemente. To omogućava korisnicima da interaguju sa virtualnim sadržajem na stvaran način.

Sa AR uređajima, korisnici mogu poboljšati svoje svakodnevne aktivnosti i oblasti kao što su obrazovanje, dizajn, medicina i mnoge druge. Mogu koristiti AR za navigaciju,

obogaćivanje informacija, igranje igara, virtuelno isprobavanje odeće ili nameštaja, ili čak za obuku i simulacije.

AR tehnologija je postala sve dostupnija i podržana na mnogim pametnim telefonima i tabletima putem specijalizovanih aplikacija. Pored toga, postoje i posebni AR uređaji poput naočara ili headsetova koji pružaju bogatije i dublje AR iskustvo.

Sve veći broj kompanija, razvojnih timova i kreativnih umova radi na stvaranju inovativnih AR iskustava i aplikacija. Očekuje se da će AR tehnologija nastaviti da napreduje i pruži još šire mogućnosti za korisnike u budućnosti.



Slika 2 *NReal Air AR* naočare.

2.1.3 MR uređaji

MR uređaji kombinuju elemente VR i AR kako bi korisnicima pružili jedinstveno iskustvo koje integriše virtualne objekte u stvarni svet. MR tehnologija koristi napredne senzore i kamere kako bi pratila okolinu i prepoznala stvarne objekte i površine. Zatim, virtualni elementi se prikazuju korisnicima na način koji je povezan i interaktivan sa stvarnim okruženjem.

Sa MR uređajima, korisnici mogu doživeti kombinaciju virtuelnih i stvarnih elemenata koji se međusobno prožimaju. Mogu interagovati sa virtualnim objektima i informacijama, istovremeno održavajući kontakt sa stvarnim svetom.

MR tehnologija ima široku primenu u raznim oblastima. Može se koristiti za obrazovanje, dizajn, simulacije, komunikaciju i mnoge druge svrhe. Korisnici mogu dobiti obogaćene

informacije, obavljati zadatke i doživeti duboka iskustva koja kombinuju najbolje od VR i AR tehnologija. Iako je MR tehnologija još u razvoju, očekuje se da će imati sve veći uticaj u budućnosti zato što MR suštinski poboljšana, interaktivnija verzija AR.



Slika 3 Lenovo ThinkReality MR naočare.

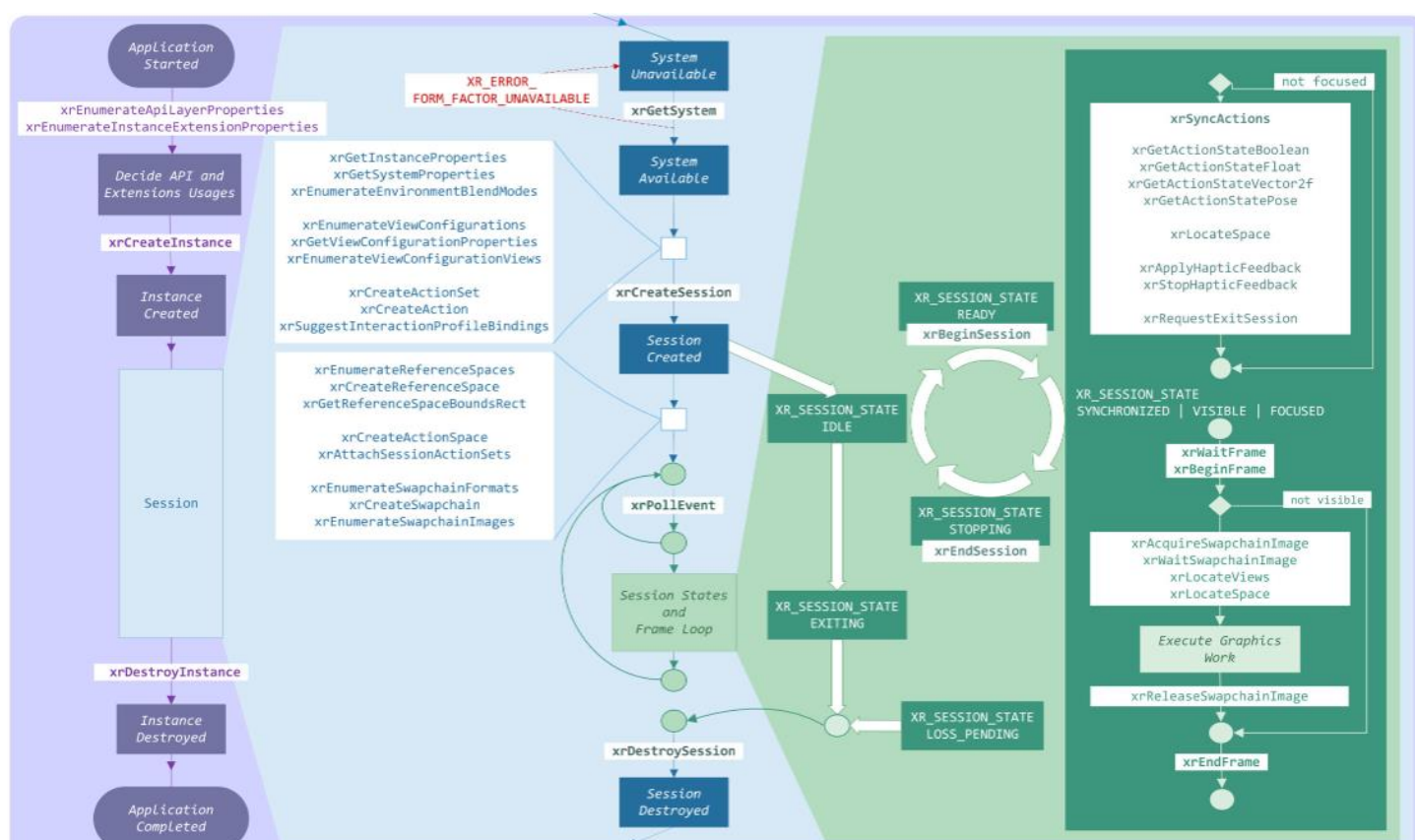
2.2 **OpenXR standard za razvoj XR aplikacija**

OpenXR je standardni mehanizam za interakciju sa VR/AR/XR sistemima na platformski agnostičan način. Predstavlja spregu između stvarnog i virtuelnog okruženja generisanog od strane računara kroz interakciju čoveka i mašine. OpenXR je interfejs između aplikacije i ulaznog ili izlaznog XR toka izvršavanja sistema (engl. *XR runtime system*). Tok izvršavanja XR sistema (u nastavku tok izvršavanja sistema) može pružati funkcionalnosti kao što su kompozicija okvira (engl. *Frame composition*), upravljanje perifernim jedinicama (engl. *Peripheral management*) i merenjima senzora za praćenje kretanja. Opciono, tok izvršavanja sistema može podržavati proširenja koja su specifična za uređaje, ovakva proširenja pružaju pristup raznom hardveru kroz jedan zajednički interfejs.

Za razvoj aplikacija, OpenXR je skup funkcija koje putem toka izvršavanja sistema omogućuju korišćenje operacija kao što su pristup kontrolerima/stanju periferija, dobavljanju trenutnih ili očekivanih vrednosti za praćenje kretanja i predaji iscrtanih okvira. Iz aspekta

pravljenja programske podrške za OpenXR na platformama koje je nemaju, potrebno je napisati biblioteku na ciljanom uređaju koja spaja svaku OpenXR funkciju sa grafičkim hardverom onako kako je prikladno ciljanom uređaju.

Tipičan OpenXR program započinje sa radom tako što pozove funkciju za pravljenje instance koja uspostavlja vezu sa tokom izvršavanja sistema. Nakon toga poziva se sistem koji bira za upotrebu odgovarajući displej i podskup za ulaz, praćenje i grafičke uređaje. Potom se pravi poziv koji kreira bafere u koje će aplikacija iscrtavati jedan ili više prikaza upotrebom odgovarajuće grafičke biblioteke za platformu. Na kraju pozivi se prave da bi se napravila sesija i započela petlja za iscrtavanje XR aplikacije.



Slika 4 Tok izvršavanja XR sistema.

Specifikacija OpenXR nije nužno model za implementaciju toka izvršavanja. Implementacija¹ toka izvršavanja mora da proizvodi rezultate koji su slični OpenXR, ali može određene procedure raditi efikasnije nego po OpenXR specifikaciji.

2.3 XR aplikacije na Android platformama

U suštini ovo poglavlje će se baviti analizom raznih načina za pravljenje aplikacija za XR platforme. Za vreme faze istraživanja urađena je komparativna analiza moguće podrške za video reprodukciju na više ekrana istovremeno, tako da se sa svakim može interagovati u XR

¹ <https://www.khronos.org/files/openxr-10-reference-guide.pdf>

aplikaciji. Neki od razloga zbog koji su odbačeni drugi pogoni za postizanje reprodukcije videa u XR:

Pogon	Programski jezici	Otvorenost koda	Prenosivost	Grafika	Zajednica
Unreal	C++, Blueprints	Otvoren	Podržava većinu XR uređaja.	OpenGL ES, Vulkan, Metal, DirectX	Velika zajednica sa dobrom dokumentacijom.
Unity	C#	Zatvoren	Podržava većinu XR uređaja.	OpenGL ES, Vulkan	Velika zajednica sa dobrom dokumentacijom.
Godot	C++, GDScript, C#	Otvoren	Podržava većinu XR uređaja.	OpenGL ES	Slabija zajednica za XR.
Bez pogona	C++	Proizvoljno	Zavisi od implementacije.	OpenGL ES	Grupe koje prave svoje pogone.

Tabela 1. Poređenje pogona igara.

Prvo je odbačena varijanta bez pogona zato što bi se projekat sveo na implementaciju čitave podrške za XR grafiku i interakciju sa okruženjem, što nije u prvom planu zahteva, već reprodukcija videa. *Godot* pogon igara je varijanta otvorenog koda, ali njegova najjača zajednica je kod 2D igara, tako da je količina mogućnosti na *Godot* pogonu ograničena.

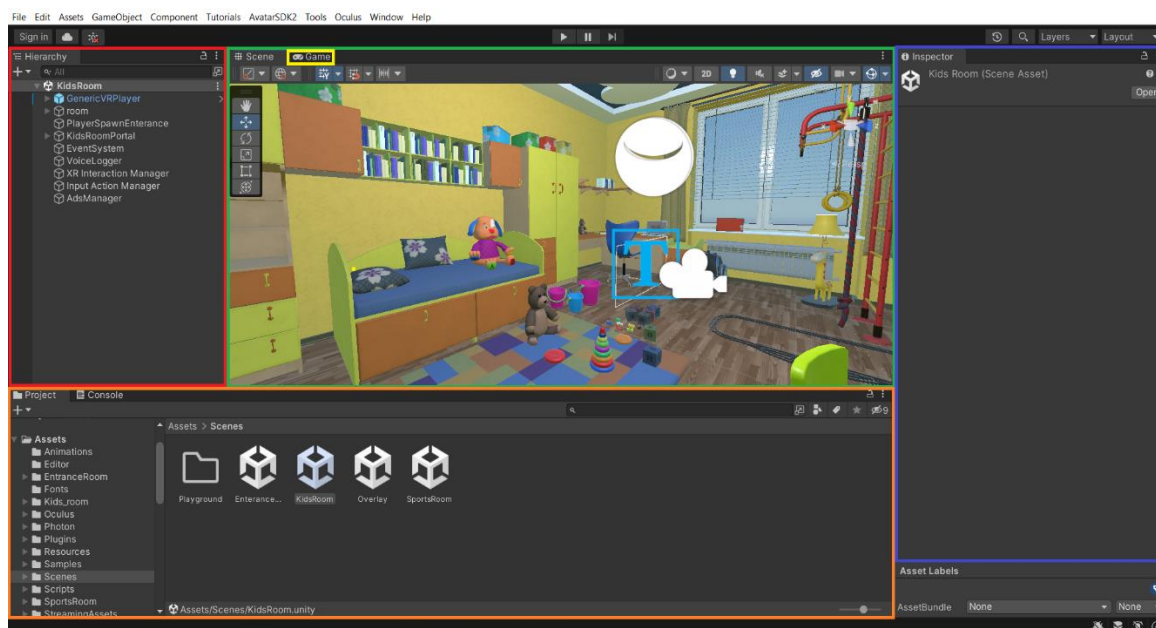
Unreal pogon igara je otpao zbog toga što podrška za reprodukcija videa na Android-u, na način koji je obrazložen detaljno u poglavlju 2.4.2, ograničena. Glavni problem je kreiranje bafera potrošača video okvira na površini za prikaz. Između ostalog, sprega *Unreal* pogona sa Android arhivom video plejera nije trivijalna.

Rezultati istraživanja su izdvojili varijantu *Unity* aplikacije po OpenGL ES 3.1 standardu kao najbolju za ovaj rad. U poglavlju 2.4, dat je kratak osvrt podrške za grafiku video prikaza po OpenGL ES 3.1. standardu na XR platformama.

2.3.1 Pravljenje XR Android aplikacija u Unity pogonu igara

Unity radno okruženje je moćan alat koji pruža širok spektar mogućnosti koje olakšavaju kreiranje, održavanje i upravljanje projektima².

² <https://docs.unity3d.com/Manual/UsingTheEditor.html>



Slika 5 Unity radno okruženje.

- **Crvenom** bojom je obeležen prikaz **Hijerarhije**. Hijerarhija prikazuje strukturu objekata u trenutnoj sceni. Ovde se nalaze svi kreirani objekti, kao i njihovi roditelji, deca i hijerarhijski odnosi među njima. Hijerarhija omogućava organizovanje objekta i manipulisanje njihovim pozicijama i transformacijama.
- **Plavom** bojom je obeležen **Inspektor**. Inspektor je prozor u Unity radnom okruženju koji prikazuje detalje i podešavanja objekata ili komponenti izabranih u hijerarhiji ili sceni. Omogućava da se prilagode objekti, podešavaju parametri, pridružuju skripte i pristupanje različitim svojstvima objekata.
- **Zelenom** bojom obeležena je **Scena**. Scena je prostor u Unity radnom okruženju gde se kreira i uređuje vizuelna predstava igre ili aplikacije. Ovde se mogu postavljati objekti, podešavati njihove pozicije, rotacije i skale, kao i pridruživati materijali i skripte. Scene omogućavaju kreiranje složenih svetova i postavljaju interakcije između objekata.
- **Žutom** bojom obeležen je prozor **Igre**. Prozor „Game“ u Unity editoru prikazuje trenutno stanje igre ili aplikacije kao što bi izgledala u stvarnom vremenu. Ovde se može testirati igra i proveriti kako će izgledati i funkcionisati za krajnje korisnike. Omogućava proveravanje ispravnost logike i vizuelnih efekta u realnom vremenu.
- **Narandžastom** bojom obeležen prozor **Projekta**. Prikazuje strukturu i sadržaj projekta. Ovde se nalaze svi fajlovi, resursi, skripte, materijali i sve druge

komponente projekta. Omogućava organizovanje fajlova i njihovu manipulaciju, kao i pristup resursima koji se koriste u igri ili aplikaciji.

Menadžer paketa je funkcionalnost Unity radnog okruženja koja omogućava preuzimanje, instaliranje i upravljanje paketima ili dodacima za proširenje funkcionalnosti³. Ovde se mogu pronaći i uključiti razni moduli, biblioteke, alati i resursi iz Unity prodavnice (engl. Asset Store) ili drugih izvora. Programaska podrška za reprodukciju videa opisana u ovom radu je naposljetku jedan paket, proširenje funkcionalnosti Unity pogona.

2.3.2 Modelovanje *Unity* scene u obliku XR Android aplikacije

Unity pogon igara pruža veliki broj opcija za podešavanje projekta. Jedan projekat se može podesiti da pravi aplikaciju za sve širokorašprostranjene operativne sisteme (*Android*, *IOS*, *Windows*, *Mac*, *Linux*), aplikacije za web pregledače, kao i serverske aplikacije. Postoje razne Android platforme koje podržavaju *OpenXR* standard, počevši od telefona do namenskih uređaja za virtuelnu ili proširenu stvarnost. Takođe pruža izuzetno korisnu mogućnosti za otklanjanje greški na *Oculus* uređajima (engl. *Oculus Link*). Ovi alati omogućuju pokretanje aplikacija u samom radnom okruženju koji je spregnut sa *Oculus* uređajima.

Scena predstavlja vidljiv deo aplikacije koji može da se menja u okviru *Unity* radnog okruženja. U scenskom prikazu se postavljaju svi objekti u 3D prostoru. Svaki objekat u sceni ima komponente vezane za sebe koje se zovu skripte. Pisane su u *C#* programskom jeziku. Programiranje u *Unity* pogonu se zasniva na pisanju i primeni skripti nad objektima unutar scene. Pre nego što se scena učita, aplikaciju pokreće *Android* operativni sistem koji u pozadini započinje i *XR* aplikaciju kako bi se učitavale vrednosti sa senzora i primenjivale na prozor kroz koji se scena vidi.

Svojstvo zbog kog se *Unity* izdvaja kao jedan od najboljih alata za stvaranje *XR* aplikacija je širok dijapazon interakcija koje korisnik može imati sa scenom oko sebe. Klasična scena kreirana za računare je vrlo slična *XR* sceni. Potrebno je samo dodati sistem za *XR* interakciju kako bi se omogućio korišćene namenskih funkcionalnosti *XR* platformi⁴. Komplet alata za *XR* interakcije koji se u ovom radu koristi je Unity paket koji u sebi sadrži komponente koje omogućuju korišćenje Unity sistema za korisnički unos na *XR* platformama. Jezgro ovog sistema predstavljaju komponente za interaktivnost, one koje započinju interakciju (engl. *Interactor*) i one koje odgovaraju na interakciju (engl. *Interactable*), kao i menadžer koji spaja ove dve vrste komponenti zajedno. Takođe poseduje komponente koje se koriste za kretanje i vizualizaciju interakcije.

³ <https://docs.unity3d.com/2021.3/Documentation/Manual/class-PackageManager.html>

⁴ <https://docs.unity3d.com/2021.3/Documentation/Manual/xr-scene-setup.html>

Komplet alata za XR interakcije podržava sledeće komponente:

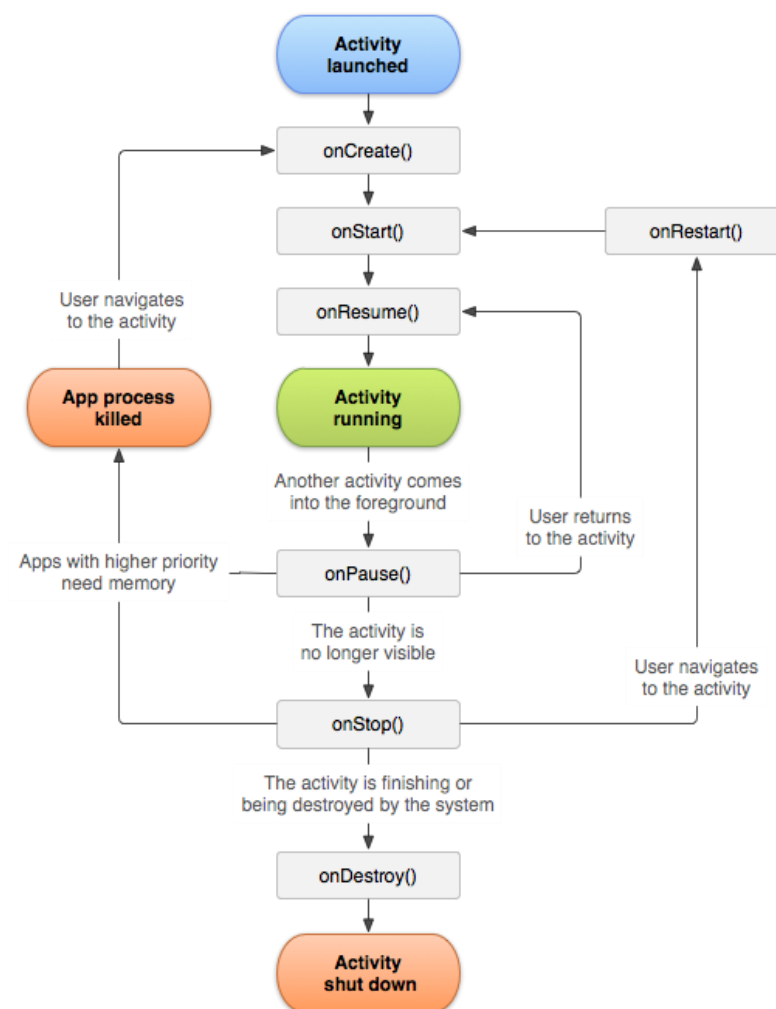
- Pristup kontrolerima na raznim XR platformama: *Meta Quest (Oculus)*, *OpenXR*, *Windows Mixed Reality* itd.
- Osnovne mogućnosti biranja, selekcije i hvatanja objekata.
- Haptički odziv putem XR kontrolera.
- Vizuelni odziv (iscrtavanje zraka/pokazivača) koji prikazuje moguće interakcije sa objektima unutar scene.
- Osnovne UI interakcije sa XR kontrolerima.
- Alate za interakciju sa koordinatnim početkom XR scene (engl. *XR Origin*) – prozorom kroz koji korisnik vidi XR svet.

Android, *OpenXR* i *Unity* efektivno imaju različite životne cikluse svojih respektivnih zona unutar jedne aplikacije. XR aplikacije sadrže više slojeva koji se odvijaju i interaguju međusobno paralelno.

U nastavku će se govoriti o konceptu životnog ciklusa. Koncept životnog ciklusa se odnosi na niz stanja i događaja kroz koje prolazi aplikacija od svog pokretanja do završetka. Razumevanje životnog ciklusa aplikacije je ključno za razvoj pouzdanih aplikacija brzog odziva, koje efikasno obrađuju korisničke interakcije i sistemske događaje.

2.3.3 *Android* ciklus aplikacije

Životni ciklus *Android* aplikacija je vođen skupom metoda povratnog poziva koje se pozivaju u različitim fazama dok aplikacija prelazi između različitih stanja. Ova stanja uključuju:



Slika 6. Android životni ciklus aktivnosti.

1. **Kreiran:** Ovo je početno stanje aplikacije kada se prvi put pokrene, ali još uvek nije prikazana korisniku. U ovom stanju se dodeljuju resursi aplikacije i obavljaju osnovni zadaci inicijalizacije.
2. **Pokrenut:** Aplikacija ulazi u ovo stanje kada postaje vidljiva korisniku, ali korisnik još uvek ne može da interaguje sa njom. To se dešava kada se korisnički interfejs aplikacije priprema za prikaz, a zadaci inicijalizacije se završavaju.
3. **Pokrenut sa fokusom:** Kada je aplikacija u stanju pokrenutog sa fokusom, ona je u prvom planu i aktivno se interaguje sa korisnikom. Ovo je stanje u kojem aplikacija prima korisničke unose i obavlja ključne operacije.
4. **Pauziran:** Pauzirano stanje nastupa kada aplikacija gubi fokus, ali ostaje delimično vidljiva. Na primer, kada dijalog ili obaveštenje druge aplikacije prekrije trenutnu aplikaciju. U ovom stanju, aplikacija je još uvek vidljiva korisniku, ali ne može odgovoriti na korisničke unose.
5. **Zaustavljen:** Kada je aplikacija u stanju zaustavljenog, ona više nije vidljiva korisniku. To može da se dogodi kada je aplikacija minimizirana, zatvorena ili kada

druga aplikacija preuzme prioritet. U ovom stanju, aplikacija zadržava svoje stanje instance, ali je možda oslobodila neke resurse.

6. **Uništen:** Konačno stanje životnog ciklusa aplikacije je kada je potpuno završena ili uklonjena iz memorije. To se događa kada sistem mora da oslobodi resurse i odluči da uništi aplikaciju. Stanje aplikacije i aktivne promenljive su izgubljene u ovom stanju.

Kroz čitav životni ciklus Android aktivnosti, različite metode povratnog poziva se mogu redefinisati (engl. *Override*) kako bi se izvršile određene akcije ili obradili događaji u različitim fazama. Ove metode uključuju *onCreate()*, *onStart()*, *onResume()*, *onPause()*, *onStop()* i *onDestroy()*, između ostalih. Pravilnim upravljanjem ovim metodama, programeri mogu kontrolisati ponašanje svojih Android aplikacija i obezbediti besprekorno korisničko iskustvo.

Životni ciklus Android aplikacija omogućava obrade događaja, čuvanje i obnovu podatka, efikasno upravljanje resursima i graciozno rukovanje prelascima između stanja. To je ključno za izgradnju efikasnih, dobro dizajniranih i korisnički prijatnih Android aplikacija.

2.3.4 Životni ciklus *Unity* skripti.

Osnovna klasa iz koje svaka Unity skripta koja se pokreće unutar aplikacije je izvedena se zove *MonoBehaviour*. *MonoBehaviour* skripte poseduju svoj životni ciklus nezavisno od Android životnog ciklusa. Životni ciklus *Unity* skripti može biti značajno kraći od Android aktivnosti aplikacije, time što Unity skripta može biti kreirana i uništena dok je aplikacija bez prestanka u fokusu. Uvode se sledeća stanja kroz koja Unity skripte prolaze kao najznačajnija za stvaranje dinamičkog interaktivnog iskustva u Unity pogonu:

Inicijalizacija (engl. *Initialization*) prilikom pokretanja, *MonoBehavior* skripta prolazi kroz inicijalizaciju. Ova faza se odvija kada se objekat na kojem je skripta prikazana kreira. U ovoj fazi mogu se postaviti početne vrednosti promenljivih, pokrenuti određeni kod ili izvršiti bilo kakve druge radnje koje su potrebne za pripremu skripte za dalje izvršavanje.

Učitavanje (engl. *Awake*) se odvija odmah nakon inicijalizacije. U ovoj fazi skripta je spremna za rad i može pristupiti drugim objektima i komponentama u sceni. Učitavanje se obično koristi za inicijalizaciju referenci na druge objekte, postavljanje parametara ili pokretanje dodatnih radnji koje su potrebne pre nego što se scena učitava ili nakon što se objekat kreira dinamički.

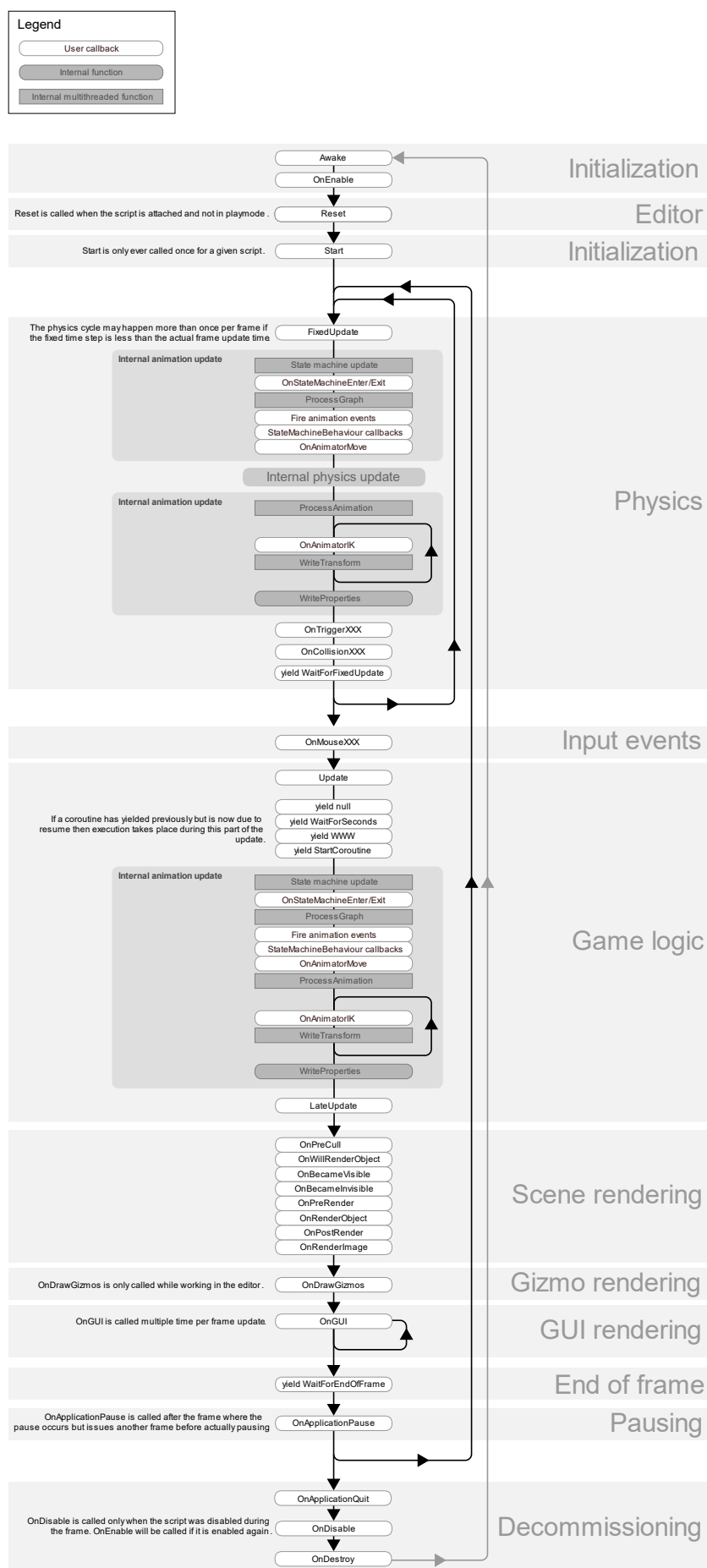
Metoda *Start* se poziva odmah nakon faze učitavanja. Ova faza se koristi za pokretanje logike ili aktiviranje funkcionalnosti skripte koje su potrebne na početku izvršavanja neposredno pre ažuriranja. U ovoj fazi se često pokreću rutine, animacije ili pokretanje drugih skripti.

Ažuriranje (engl. *Update*) se ponavlja svaki za svaki okvir (engl. *Frame*) tokom izvršavanja. U ovoj fazi se obično vrši ažuriranje logike i stanja objekta na osnovu trenutnog vremena, korisničkog unosa ili drugih faktora. *Update* metoda se koristi za reagovanje na promene u stanju igre i izvršavanje odgovarajućih akcija.

Fiksno ažuriranje (engl. *FixedUpdate*): Faza fiksnog ažuriranja se takođe ponavlja za svaki okvir, ali se odvija u fiksnim vremenskim intervalima. Ova faza je posebno korisna za izvršavanje fizike i bilo kakvih drugih operacija koje zahtevaju dosledan vremenski korak. *FixedUpdate* metoda se često koristi za manipulaciju fizike objekta ili kretanje objekta na osnovu unapred definisanih pravila.

Kasno ažuriranje (engl. *LateUpdate*): Faza kasnog ažuriranja se odvija nakon svih faza ažuriranja u skripti. Ova faza se često koristi za izvršavanje radnji koje se odnose na kameru, praćenje ciljeva ili postavljanje krajnjeg stanja objekta nakon ažuriranja svih drugih objekata u sceni.

Uništavanje (engl. *OnDestroy*): Kada se objekat uništi ili skripta ukloni, poziva se ova metoda. Ova faza se koristi za čišćenje resursa, zatvaranje otvorenih veza ili izvršavanje bilo kakvih drugih akcija koje su potrebne pre nego što se skripta potpuno ukloni iz igre.



*Slika 7 MonoBehaviour tok izvršavanja.*⁵

Na *Androidu*, Unity pogon unutar aplikacije se pokreće u *OnCreate* metodi što znači da se *Unity* skripte unutar početne scene kreiraju neposredno pre nego što *Android* aplikacija bude vidljiva korisniku.

2.3.5 Dinamičko učitavanje Android biblioteka putem *Unity* skripti

Ideja pristupa Android bibliotekama za vreme izvršavanja aplikacije je neophodna ako je potrebno upotrebiti neke funkcionalnosti koje su van *Unity* skripti. Ovakve eksterne biblioteke se učitavaju dinamički tokom izvršavanja. Dinamičke biblioteke koje se učitavaju tokom izvršavanja aplikacije se zovu dodaci⁶ (engl. *Plugins*). Dodaci za *Unity* pogon na Android platformama su obično Android arhive (*aar* datoteke).

Android arhive se kompajliraju u *Android Studio* radnom okruženju i vrlo su slične Android aplikacijama (*apk* datoteke). Strukturno su vrlo slične aplikacionim datotekama time što uključuju sve što je neophodno za kompajliranje aplikacije. Namena Android arhiva je drugačija, one služe kao izdvojena celina (engl. *Dependency*) koja se može proizvoljno uključiti za kompajliranje Android aplikacija ili drugih Android arhiva⁷. Ovo je izuzetno korisno kada se prave više različitih aplikacija koje dele jednu istu celinu, Android arhive omogućuju da se ista celina koristi za svaku aplikaciju.

Android Studio podržava direktnu integraciju kompajliranja nativnih biblioteka u proces kreiranja arhive zajedno sa Java kodom upotrebom *CMake* alata. *CMake* je alat otvorenog koda koji se koristi za generisanje nativnih konfiguracija za kompajliranje izvornog koda. Svi uređaji koji su korišćeni za evaluaciju rešenja su ARM64 platforme. Pošto računar na kome se kompajlira aplikacija nije ARM64, već x64, neophodno je koristiti alate za kompajliranje za ciljanu platformu. *CMake* omogućuje generisanje konfiguracija za unakrsno kompajliranje (engl. *Cross compile*).

Ovaj rad koristi dinamičko učitavanje C++ funkcija iz Android arhiva. Pod uslovom da je C/C++ zaglavlje uspešno kompajlirano u dinamičku biblioteku (*.so* fajl) i dinamička biblioteka u direktorijumu *Assets/Plugins/Android/* (ovo važi i za Android arhive), moguće je dinamički učitati bilo koju funkciju iz zaglavlja kao C# metodu unutar Unity skripte putem *DllImport* atributa⁸. Na ovaj način se u ovom radu integrišu funkcije za *OpenGL ES* grafičko iscertavanje video strima.

⁵ https://docs.unity3d.com/uploads/Main/monobehaviour_flowchart.svg

⁶ <https://docs.unity3d.com/2021.3/Documentation/Manual/AndroidAARPlugins.html>

⁷ <https://developer.android.com/studio/projects/android-library>

⁸ <https://docs.unity3d.com/2021.3/Documentation/Manual/NativePlugins.html>

Unity poseduje API za učitavanje i pozivanje Java metoda preko JVM, bile one statičke ili deo nekog Java objekta. Android klase ili objekte je moguće učitati dinamički kreiranjem *AndroidJavaObject* ili *AndroidJavaClass* C# objekta u Unity skriptama⁹. Ovaj mehanizam se koristi za pristup među sloju, zasebnoj Android arhivi. Android arhiva video plejera je zavisnost neophodna za kompajliranje među sloja. Među sloj efektivno omogućuje pristup funkcionalnostima video plejera Unity pogonu kao što su: inicijalizacija, start, pauziranje, zaustavljanje, premotavanje video strima itd. Među sloj je adapter koji je neophodan kako bi se omogućio pristup funkcionalnostima video plejera sa što manje poziva iz Unity C# skripti prema JVM jer su ovi pozivi veoma skupi.

2.3.6 Standardna grafička procesna linija *Unity* pogona

Grafička procesna linija (engl. *Rendering pipeline*) izvršava skup koraka koji sadržaj scene prikazuju na ekranu. Zbog toga što različite grafičke procesne linije imaju različite izlaze, performanse i karakteristike u zavisnosti od platforme bitno je odabrati onu koja se najbolje uklapa u zahteve projekta još u početnoj fazi razvoja. Postoje više opcija za grafičke procesne linije u Unity pogonu, tako da je bitno izabrati onu koja će omogućiti uspešan prikaz videa kada se uzme u obzir specifičnost reprodukcije videa na Androidu (vidi 2.4.2.). U fazi istraživanja se pokazala standardna grafička procesna linija kao najbolja zbog toga što poseduje komandne bafere za njeno proširenje¹⁰.

Standardna grafička procesna linija (engl. *Standard Render Pipeline - SRP*) je osnovna metoda za generisanje slika u Unity razvojnom okruženju. Ovo je osnovna grafička procesna linija koja je ugrađena u Unity pogon. SRP definiše proces koji se koristi za transformisanje 3D modela i scene u konačnu sliku koja se prikazuje na ekranu. Obično se ovi koraci izvršavaju u tri glavne faze koje se ponavljaju mnogo puta u sekundi¹¹:

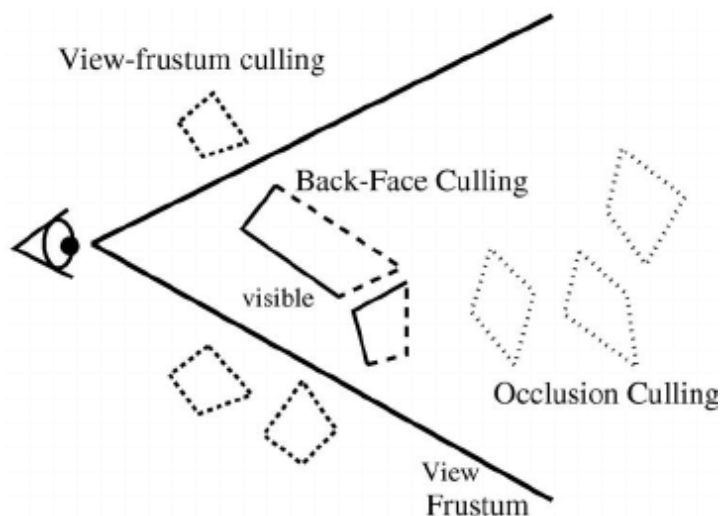
- Seckanje (engl. *Culling*) se odnosi na princip po kome se optimizuje iscrtavanje tako što se među svim objektima koji se nalaze unutar scene izdvajaju samo koji su vidljivi. Seckanje objekta se ne mora nužno odnositi na cele modele, seckaju se razni primitivni geometrijski oblici: tačke, grane, mnogouglovi i trouglovi. Najjednostavniji oblik seckanja je kada se iseckaju oni oblici koji ne upadaju u pregledni prostor (engl. *Frustum*) kamere. Pregledni prostor kamere je efektivno četvorostrana prizma sačinjena od bliske i udaljene ravni odsecanja unutar koje se svi vidljivi objekti u sceni prikazuju. Seckanje okluzije (engl. *Occlusion culling*) se radi kada je vidljivost neko objekta

⁹ <https://docs.unity3d.com/2021.3/Documentation/Manual/android-plugins-java-code-from-c-sharp.html>

¹⁰ Ostale grafičke procesne linije je takođe moguće proširiti ali zahteva naprednije metode od komandnih bafera.

¹¹ <https://docs.unity3d.com/Manual/render-pipelines-overview.html>

smanjena u preglednom prostoru zbog toga što je neki drugi objekat ispred. Za seckanje se koriste napredni algoritmi koji su hardverski optimizovani na GPU.



Slika 8 Seckanje preglednog prostora

- Iscrtavanje objekata sa odgovarajućim svojstvima materijala i osvetljenjem u piksel bafere.
- Post-procesiranje su dodatne operacije koje se vrše nad piksel baferima kako bi se, na primer, primenili gradijenti boja, isijavanje, dubina i generisao konači okvir koji će se prikazati na ekranu uređaja.

Unity pruža fleksibilnost za prilagođavanje SRP-a putem skripti, tako da programeri mogu da prilagode kako se podaci o sceni iscrtavaju i prikazuju.

2.3.6.1 Komandni baferi SRP

Komandni baferi (engl. *Command buffers*) su moćan alat u Unity razvojnom okruženju koji omogućava optimizovanje procesa iscrtavanja i prilagođavanje obrade grafičkih podataka za prikazivanje¹². Predstavljaju kolekciju naredbi koje se izvršavaju u određenom trenutku tokom iscrtavanja scene. Omogućuju grupisanje naredbi za iscrtavanje i upravljaju njihovim redosledom izvršavanja. Komandni baferi pružaju fleksibilnost u upravljanju radnjama koje se izvršavaju pre, tokom ili posle glavnog iscrtavanja scene. Na primer, može se definisati komandni bafer koji primenjuje posebne efekte, menja materijale ili manipuliše svetlostima pre nego što se scena prikaže na ekranu. Takođe pružaju mogućnost da se optimizuje radnja iscrtavanja tako što se grupišu slične naredbe u jedan bafer kako bi se smanjio broj poziva ka grafičkom uređaju, što dovodi do efikasnijeg izvršavanja. Unity pruža programerima API za rad

¹² Komandne bafere je moguće koristiti u raznim slučajevima kada je neophodna veća kontrola za iscrtavanje. U ovom radu je fokus na događaje za iscrtavanje koji se dodaju u grafički tok izvršavanja putem *CommandBuffer.IssuePluginEventAndData* metode.
<https://docs.unity3d.com/2021.3/Documentation/Manual/LowLevelNativePluginRenderingExtensions.html>

sa komandnim baferima. Kroz ovaj API, moguće je kreirati, konfigurisati i izvršavati komandne bafer-e prema svojim potrebama.

Komandni baferi se priključuju u grafički tok izvršavanja putem kamere. Svaka kamera unutar Unity scene ima svoj grafički tok, tako da je moguće proširivati grafički tok svake kamere ponaosob. Razlog zašto ovakav mehanizam postoji je zato što kamera ne mora nužno da iscrtava objekte u sceni, već je moguće i prikazati specijalnu teksturu za iscrtavanje na kameru. Teksture za iscrtavanje (engl. *Render Texture*) se mogu prikazati i na komandne bafere¹³. Kačenje tekstura za iscrtavanje na komandne bafere služi za usmeravanje iscrtavanja u određeni grafički bafer koji Unity slaže unutar scene. Teksture za iscrtavanje se primenjuju unutar scene tako što se kreira materijal koji tu teksturu sadrži. Zatim se ovaj materijal mapira na površinu ekrana. Na ovaj način će se svaki grafički događaj komandnog bafera iscrtati na ekranu video plejera.

U ovom radu komandi baferi igraju ključnu ulogu kod prikaza videa u prostoru. XR uređaji imaju dva režima koji optimizuju iscrtavanje na XR platformama. Izvršavanje koda za iscrtavanje videa unutar komandnih bafera efektivno rešava problem prikaza teksture videa, nezavisno od režima iscrtavanja, tako da se isti princip može primeniti u oba slučaja.

2.3.6.2 Režimi iscrtavanja na XR uređajima

Jednoprolazni režim za iscrtavanje (engl. *Single Pass*) i višestruki režim za iscrtavanje (engl. *Multiview*) su dva načina za optimizaciju iscrtavanja u XR Unity aplikacijama.

Jednoprolazni režim za iscrtavanje je tehnika koja omogućava da se iscrtavaju obe perspektive za levo i desno oko kamere u jednom prolazu. To znači da se objekti i efekti iscrtavaju samo jednom, čime se smanjuje opterećenje na GPU i povećavaju performanse aplikacije. Ova tehnika je posebno korisna u VR aplikacijama gde je potrebno iscrtati dve perspektive za svako oko.

Sa druge strane, višestruki režim za iscrtavanje je tehnika koja koristi GPU da efikasno iscrta obe perspektive za levo i desno oko kamere, ali u jednom pozivu za iscrtavanje. To znači da se scene iscrtavaju samo jednom, a GPU generiše oba pogleda iz tog jednog seta podataka. Ova tehnika takođe smanjuje opterećenje GPU-a i može pružiti bolje performanse u XR aplikacijama.

Razlika između jednoprolaznog režima za iscrtavanje i višestrukog režima za iscrtavanje je u načinu na koji se iscrtavaju perspektive za levo i desno oko kamere. Jednoprolazni pogon za iscrtavanje izvršava dva prolaza iscrtavanja (svako oko pojedinačno), dok višestruki režim za

13

<https://docs.unity3d.com/2021.3/Documentation/ScriptReference/Rendering.CommandBuffer.SetRenderTarget.html>

iscrtavanje izvršava jedan prolaz iscrtavanja, ali generiše oba pogleda iz tog jednog seta podataka¹⁴.

Unity pruža podršku za oba režima u svojoj XR funkcionalnosti, tako da je moguće odabrati optimalnu tehniku iscrtavanja za XR aplikacije, u skladu sa zahtevima performanse i hardvera.

Još jedna značajna podela kod režima iscrtavanja je po prostoru boja koje podržava grafički API, u ovom slučaju OpenGL ES. Unity podržava gama i linearni prostor boja.

Kada je Unity pogon konfigurisan za Android platforme na kojima je OpenGL ES podešen kao grafički API, Unity pogon podržava isključivo **linearni prostor boja**. Ovo ograničenje predstavlja veliki problem, zato što je video u gama prostoru boja.¹⁵

2.4 OpenGL ES biblioteka na Android platformama

Android koristi OpenGL ES (engl. *OpenGL for Embedded Systems*) kao standardnu biblioteku za iscrtavanje grafike na mobilnim uređajima. Ova tehnologija igra ključnu ulogu u postizanju visokokvalitetne grafike u XR iskustvima.

OpenGL ES je posebno prilagođena verzija OpenGL-a, koja je optimizovana za rad na mobilnim platformama. Koristeći ovu biblioteku, razvojni timovi mogu implementirati napredne 3D grafike, efekte svetlosti, senke, teksture i mnoge druge vizuelne elemente u XR aplikacijama.

U kontekstu XR grafike, OpenGL ES pruža podršku za iscrtavanje virtuelnih objekata, interakciju sa okruženjem i prikazivanje visoko detaljnih tekstura. On omogućava glatko prikazivanje i praćenje pokreta kako bi se korisnicima pružilo što realističnije iskustvo.

Kroz OpenGL ES, Android uređaji koriste GPU (Grafički procesor) za ubrzano izvršavanje grafike. Ovo omogućava brže iscrtavanje, veći broj poligona, bogatije efekte i realističnije animacije u XR aplikacijama. Kako bi podržali XR grafiku, Android uređaji često koriste senzore kao što su akcelerometar, žiroskop i kamere za praćenje pokreta korisnika. Ovi podaci se zatim koriste u kombinaciji sa OpenGL ES bibliotekom kako bi se virtuelni objekti pravilno pozicionirali i reagovali na korisničke akcije.

XR grafika zahteva visoku brzinu iscrtavanja i brz odziv kako bi se postigla što veća fluidnost i osećaj prisutnosti u virtuelnom okruženju. Upravo OpenGL ES i optimizacija performansi na Android platformi igraju ključnu ulogu u postizanju ovih zahteva. Pored toga, razvojni timovi mogu koristiti razne tehnike, poput optimizacija za iscrtavanje i smanjenje broja poligona, kako bi unapredili performanse XR grafike na Android uređajima. Kontinuirani razvoj OpenGL ES biblioteke i podrška razvojnih alata olakšavaju stvaranje bogatih XR iskustava na

¹⁴ <https://blog.unity.com/technology/how-to-maximize-ar-and-vr-performance-with-advanced-stereo-rendering>

¹⁵ <https://docs.unity.cn/Manual/LinearLighting.html>

Android platformi. OpenGL ES ima ključnu ulogu u postizanju visokokvalitetne i realistične grafike u XR iskustvima na Android uređajima.

U nastavku ovog poglavlja će se uvesti OpenGL ES koncepti koji su neophodni za reprodukciju video strima na Androidu.

2.4.1 OpenGL ES za prikaz videa u obliku ekrana pozicioniranog u 3D prostoru

Da bi se prikazao ekran u 3D prostoru, potrebno je razumeti sledeće koncepte:

OpenGL ES koristi kontekst *EGL* (engl. *Khronos Native Platform Graphics Interface*) koji se uspostavlja sa ciljem da komunicira sa grafičkom karticom ili drugim uređajem koji podržava OpenGL ES. Kontekst čuva sve informacije o trenutnom stanju OpenGL ES-a, kao što su podaci o teksturi, šejderima, boji i drugim parametrima. Ispod haube, kontekst je veliki automat sa konačnim brojem stanja i pozivi prema OpenGL ES drajveru u osnovi menjaju stanja u automatu kako bi pripremili objekte za iscrtavanje na ekranu.

Šejderi (engl. *Shaders*) su programi napisani u programskom jeziku *GLSL* (engl. *OpenGL Shading Language*) koji definišu kako se vrši obrada geometrije, osvetljenja i bojenja objekata. Dva glavna tipa šejdera su:

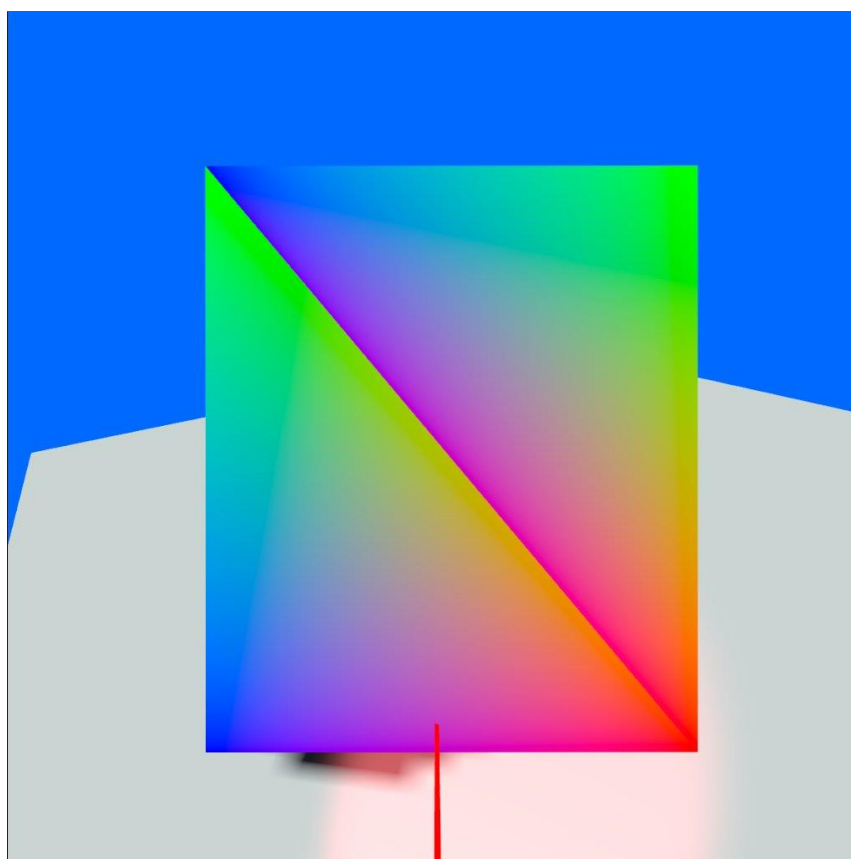
- Šejder temena obrađuje svako teme (engl. *Vertex shader*) objekta i odgovoran je za transformaciju temena u prostoru, kao i za slanje podataka fragment šejderu.
- Šejder fragmenta obrađuje svaki fragment (piksel, engl. *Fragment shader*) objekta i odgovoran je za određivanje boje piksela, osvetljenja i drugih efekata.

Da bi se prikazali objekti u 3D prostoru, koriste se matrice transformacija. Tri glavne vrste matrica transformacija su:

- Matrica modela (engl. *Model Matrix*) se koristi za transformacije objekta u prostoru, kao što su rotacija, translacija i skaliranje.
- Matrica pogleda (engl. *View Matrix*) predstavlja kameru ili perspektivu iz koje posmatramo scenu. Matrica pogleda definiše položaj, orijentaciju i perspektivu kamere.
- Matrica projekcije (engl. *Projection Matrix*): Definiše perspektivnu projekciju scene. Može biti ortografska projekcija ili perspektivna projekcija. Za prostorno modelovanje koristi se perspektivna projekcija.

Temena definišu geometriju objekta. Za prikazivanje ekrana, potrebno je specificirati četiri temena koja definišu uglove ekrana u 3D prostoru. Sledeći OpenGL ES objekti omogućuju efikasno iscrtavanje ekrana u 3D prostoru:

- VAO (engl. *Vertex Array Object*) je objekat koji sadrži definiciju atributa temena, uključujući njihovu poziciju, normalu, koordinate teksture itd. Ovaj objekat čuva podataka o temenima neophodne za iscrtavanje objekta.
- VBO (engl. *Vertex Buffer Object*) je bafer memorije vezan za VAO u GPU memoriji velike brzine dizajniran da drži informacije o temenima. Takođe može da sadrži informacije o poziciji, normali, koordinatama teksture objekta itd. Jedan VAO sadrži u sebi jedan ili više VBO objekta.
- Indeksi (engl. *Indices*) su nizovi koji omogućavaju efikasno indeksiranje temena kako bi se smanjila redundantnost podataka. Kako bi se ekran iscrtao, potrebno je iscrtati dva trougla (efektivno dijagonala ekrana razdvaja ekran na dva trougla). Za svaki trougao neophodno je definisati njegovo teme i kreirati VBO objekat sa ovim informacijama. Temena kroz koje prolazi dijagonala koja razdvaja ekran na pola se definišu po jednom za oba trougla. Kako bi se eliminisali ovi duplikati koristi se indeksiranje. IBO (engl. *Index Buffer Object*) sadrži informacije o tome na kojim indeksima u nizu temena se nalaze temena neophodna za iscrtavanje svakog trougla. Indeksiranje može efektivno da ubrza iscrtavanje.



Slika 9 Ekran iscrtan u 3D prostoru primenom šejdera interpolacije¹⁶.

¹⁶ Interpolacija se ovde odnosi na primenu koordinata za položaj temena iz *Vertex* šejdera u *Fragment* šejderu za boju piksela. Šejderi interpolacije se često koriste za proveru ispravnosti OpenGL iscrtavanja.

Kada se ovi koncepti razumeju, može se koristiti OpenGL ES za inicijalizaciju konteksta, kompiliranje i povezivanje šejdera, definisanje matrica transformacija, specificiranje temena i indeksa, primena tekstura i iscrtavanje objekta u 3D prostoru.

2.4.2 Teksture za prikaz video strima u Androidu

Glavna programska podrška za prikaz videa koju *Android* poseduje se zove površina (engl. *Surface*) i ova površina je proizvođač video strima. Nezavisno od API-a koji programeri koriste sve se iscrtava na površinama. Površina predstavlja stranu proizvođača bafera koji se stavljaju u red čekanja koji se često konzumira od strane površine za prikaz (engl. *Surface Flinger*). Svaki prozor koji se kreira na Android platformi je pokriven površinom. Sve vidljive površine iscrtavane se sastavljaju na ekran od strane površine za prikaz.

Glavne komponente uz pomoć kojih se površine iscrtavaju¹⁷:

Proizvođač strima slika (engl. *Image stream producer*) može biti bilo šta što proizvodi grafičke bafere za potrošače. Primeri proizvođača grafičkih bafera su: OpenGL ES, Canvas 2D i medijski serveri za dekodovanje videa (engl. *Media server video decoders*).

Potrošač toka slika je najčešće površina za prikaz, sistemski servis koji konzumira trenutno vidljive površine i sastavlja ih na ekran na osnovu informacija koje pruža menadžer prozora (engl. *Window manager*). Površina za prikaz je jedini servis koji može da modifikuje sadržaj ekrana. Površina za prikaz koristi OpenGL ES i kompozitor hardvera (engl. *Hardware compositor*) da složi grupe površina. OpenGL ES aplikacije mogu biti potrošači strima slika takođe, na primer, kao što je OpenGL ES aplikacija za prikaz strima slika sa kamere¹⁸.

Sloju hardverske apstrakcije (engl. *Hardware abstraction layer*) površina za prikaz može delegirati određene poslove vezane za slaganje kako bi se smanjila količina posla koji rade OpenGL ES i GPU. Površina za prikaz deluje kao OpenGL ES klijent. Dakle, kada površina za prikaz aktivno slaže jedan bafer preko drugog, trećeg, na primer, koristi OpenGL ES. Na taj način se smanjuje opterećenje GPU za kompoziciju i smanjuje potrošnja struje. Sloj hardverske apstrakcije diriguje drugu polovinu poslova i igra centralnu ulogu u grafičkom iscrtavanju na Androidu. HAL koristi *Gralloc* (engl. *Graphics allocator system*) sistem za alokaciju memorije na zahtev proizvođačima strima slika.

Red čekanja bafera (engl. *Buffer Queue*) predstavlja glavnu vezu između Android grafičkih komponenti. Ovi baferi su parovi redova koji omogućuju neprekidan ciklus prenosa bafera sa proizvođača do potrošača. U trenutku kada proizvođači bafera predaju bafere, površina

¹⁷ <https://source.android.com/docs/core/graphics>

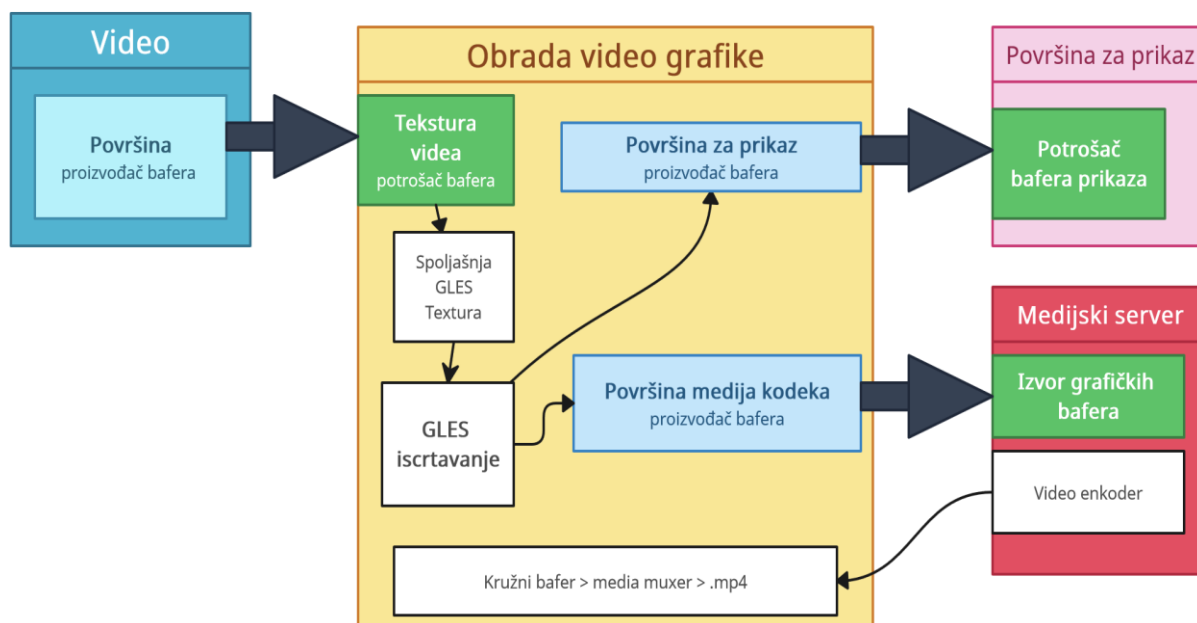
¹⁸

<https://github.com/google/grafika/blob/master/app/src/main/java/com/android/grafika/ContinuousCaptureActivity.java>

za prikaz slaže sve na ekranu. Red čekanja bafera je struktura podataka koja kombinuje bazene bafera sa redom čekanja i koristi IPC kako bi prosleđivala bafere između Android procesa. Interfejs za generisanje grafičkih bafera je sastavni deo površinske teksture (engl. *Surface Texture*).

Površinska tekstura u ovom radu služi kao potrošač bafera koji u sebi sadrži spoljnu GLES teksturu. OpenGL ES sadrži specijalni atribut teksture koji naglašava drajveru da je ovo spoljna tekstura - *GL_TEXTURE_EXTERNAL_OES*. Spoljna GLES tekstura ima tu prednost da se iscrtava direktno iz podatka reda čekanja bafera. Na osnovu ove površinske teksture može se napraviti odgovarajuća površina, proizvođač bafera, koji odgovara za pokretanje video plejera (video plejer proizvodi bafere koji su preko te namenske površine povezani sa potrošačem bafera – površinskom teksturom).

Uz pomoć adekvatnog GLSL šejder programa postiže se efikasna proizvodnja bafera u GPU memoriji koje površina za prikaz može slagati na ekranu. Naredni dijagram prikazuje kako se video iscrtava upotrebom OpenGL ES-a, GLSL šejder programa i Android površini¹⁹:

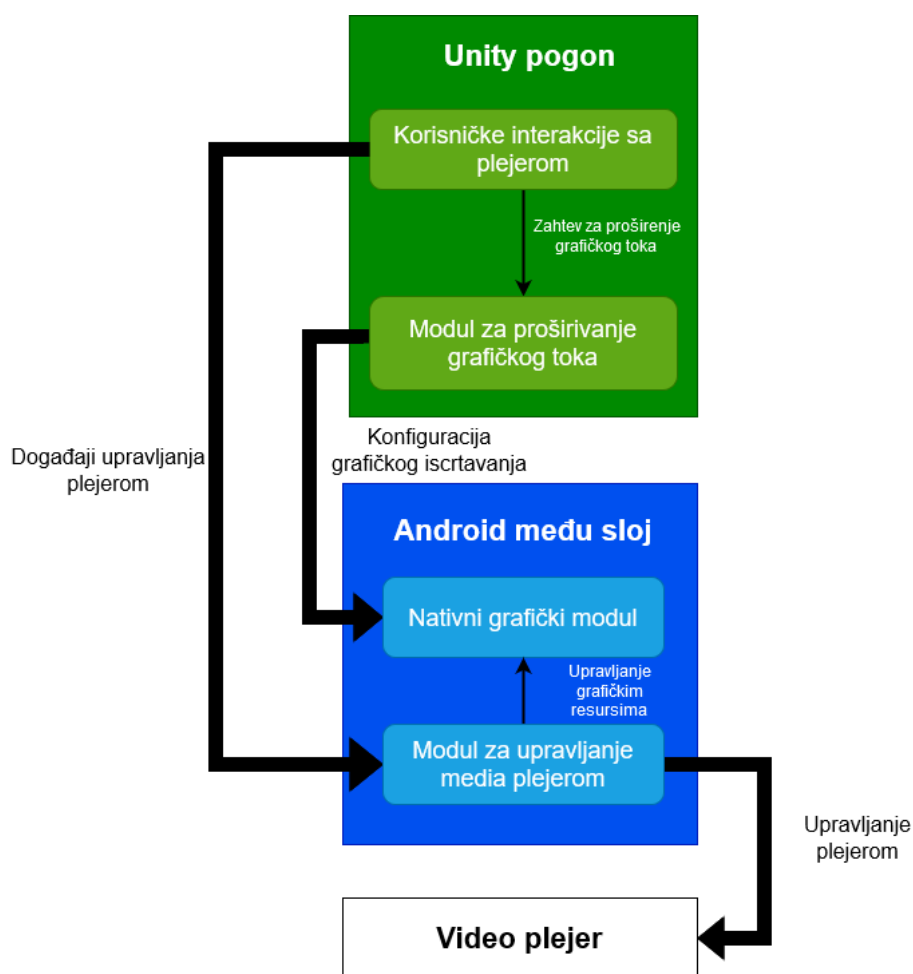


Slika 10 Android aktivnost kontinualne obrade video strima.

¹⁹ <https://source.android.com/docs/core/graphics/arch-st>

3. Koncept rešenja

U ovom poglavlju se objašnjava ideja iza samog rešenja. Koncept je podeljen blokovski, tako da su najbitniji delovi svakog modula izdvojeni i međusobna komunikacija blokova naznačena. Blokovi koji su obojeni predstavljaju glavni deo rešenja, dok je neobojenim blokom predstavljena Android arhiva video plejera:



Slika 11 Arhitektura modula koji čine rešenje

Za početak, kako bi se prikazao video sadržaj Android aplikacija mora biti u fokusu. Pošto Unity proširuje Android aktivnost, kada je Android aplikacija u fokusu to znači da je zapravo Unity scena u fokusu. Kao što je navedeno u teorijskim osnovama, *Start* metoda *MonoBehaviour* skripti se poziva neposredno pre ažuriranja okvira i prikazivanja scene na ekranu. Ova metoda je početna tačka izvršavanja nakon inicijalizacije neophodnih resursa unutar Unity scene.

Blok Unity pogona je znatno veći nego što je prikazan na arhitekturi. Pod ovim blokom se podrazumeva čitav niz paketa koji su dodati u Unity pogon kako bi aplikacija radila sa svim funkcionalnostima XR-a. Moduli koji su izdvojeni u bloku Unity pogona su najrelevantniji za koncept rešenja, oni su duboko spojeni sa Unity pogonom, koriste bazične funkcionalnosti Unity pogona ali i paketa za XR koji ga proširuju.

Modul korisničkih interakcija sa plejerom omogućuje korisnicima aplikacije da putem *OpenXR* interakcija manipulišu video plejerom unutar scene. Spolja prema korisniku, korisničke interakcije se realizuju putem kompleta alata za XR interakcije²⁰. Modul korisničke interakcije je zadužen za registrovanje događaja interakcije sa korisnikom nakon što se scena učitava i bude u fokusu.

Interakcije koje su podržane od strane ovog modula su:

- Vizuelizacija zraka koji prikazuje trenutne moguće interakcije sa scenom.
- Registracija događaja pritiska dugmeta na kontroleru koji služi za selekciju UI elementa unutar scene.
- Ako se zrak poklapa sa površinom sa kojom korisnik može da interaguje, zrak menja boju i pritisak dugmeta za selekciju okida događaj interakcije korisnika sa video plejerom koji se može implementirati putem C# Unity metoda.

UI elementi sa podrškom za XR interakcije poseduju komponente koje su sastavni deo ovog rešenja i njihovom konfiguracijom se omogućuje XR interakcija sa objektima unutar Unity scene. Na događaj interakcije se vežu metode koje upravljaju video plejerom.

Drugi element unutar Unity pogona bloka je modul za proširivanje grafičkog toka. Zadatak ovog modula je da očita teksturu za iscrtavanje koja je prikačena na ekran i mapira je na komandni bafer. Ovaj modul dinamički učitava C++ funkcije koje su neophodne za iscrtavanje videa u komandnom baferu. Pošto Android među sloj nema predstavu o tome kada korisnik napušta ili ulazi u scenu u kojoj se ekrani za reprodukciju videa nalaze, modul za proširivanje grafičkog toka vodi računa o resursima.

²⁰ Komplet alata za XR interakcije (engl. *XR interaction toolkit*) je Unity paket koji je ugrađen u šablon za kreiranje XR projekta. <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.4/manual/index.html>

Android među sloj blok je, kao što je već navedeno, Android arhiva. Dva modula su izdvojena kao značajne konceptualne celine i mogu se efektivno razdvojiti po programskim jezicima. Nativni grafički modul sadrži u sebi funkcije koje su spregnute kao dinamičke biblioteke za Unity C# skripte, funkcije koje se putem JNI vežu na Java metode koje modul za upravljanje plejerom koristi i svoje interne funkcije. Nativni grafički modul poseduje svoju strukturu podataka koja za svaku instancu video plejera čuva površinsku teksturu, JNI ekvivalent metode za konzumiranje grafičkih bafera putem površinske teksture i OpenGL ES identifikator spoljne teksture od koje je površinska tekstura napravljena. U nativnom grafičkom modulu se nalazi vektor struktura za iscrtavanje koji je popunjen trenutno aktivnim instancama video plejera. OpenGL ES resursi koji su neophodni za iscrtavanje ekrana u 3D prostoru (VAO, VBO, IBO i šejder) su deljeni resursi među video plejer instancama.

Funkcionalnosti koje se izlažu prema Unity skriptama su: specijalna metoda za iscrtavanje video strima, metoda za brisanje svih članova vektora trenutno aktivnih instanci video plejera i za oslobađanje GLES resursa. Deljeni OpenGL ES resursi za iscrtavanje ekrana se zauzimaju kada se okine događaj Unity pogona: *kUnityGfxDeviceEventInitialize*, koji se okida kada Unity podešava svoj pristup grafičkom uređaju. Drugi podskup funkcionalnosti nativnog modula je povezan sa Javom i generalno služi da sinhronizuje zauzimanje nativnih resursa sa Java omotačem za plejer. Nativni modul kreira potrošača bafera, površinsku teksturu, prosleđuje je Java omotaču koji kreira proizvođača bafera, površinu, koju prosleđuje video plejeru da bi se kreirao.

Modul za upravljanje video plejerom uključuje u sebe JNI metode video plejera i kreira omotač oko njih koji je pogodan za pozivanje iz Unity pogona. Glavna klasa kojom se Unity C# skripte služe kada koriste Java deo Android među sloja je plejer omotač (engl. *Player Wrapper*). Omotač plejera se kreira putem statičke fabričke metode koja za povratnu vrednost ima instancu omotača, ovaj način se koristi zbog ograničenja Android objekta unutar C# skripti. Iz fabričkog konstruktora omotača se poziva JNI metoda za dodavanje novog člana u vektor za iscrtavanje i površinska tekstura kreirana u nativnom kodu se prosleđuje dalje zarad kreiranja nove interne video plejer instance. Omotač pruža jednostavan API za manipulaciju plejerom, omogućuje puštanje, pauziranje, stopiranje i premotavanje iz C# skripti. Drastično smanjuje broj poziva neophodnih za rad sa video plejerom iz C# skripti i omogućuje jednostavan rad sa više video tokova koji se reprodukuju unutar scene istovremeno.

3.1 XR profili za interakciju

U OpenXR-u akcija je značajni deo interakcije: nešto što korisnik uradi. Uzimanje objekta u ruke, teleportacija su primeri nekih akcija. Akcije koje bi trebale da budu dostupne u

nekom kontekstu ili okruženju se grupišu u setove akcija (engl. *Action Sets*). Primera radi, neka su definisan setovi interakcija „meni“, „vožnja“, „igranje“. Moguće je imati jedan ili više aktivnih setova u datom trenutku, što znači da nije neophodno praviti duplikate akcija.

Profil interakcije je način na koji su setovi akcija vezani za kontrolere. Nije neophodno da svaka akcija bude vezana za kontroler, kao i svaka mogućnost na kontroleru da poseduje akciju. Vezivanje akcija za kontrolere se mora uradi za svaku vrstu kontrolera ponaosob, tj. Za svaki XR uređaj koji podržava OpenXR, neophodno je kreirati profil interakcije.

Sve opcije koje kontroleri pružaju su organizovane kao sistem direktorijuma na particiji. Vezivanje akcija na kontrolere se radi putem *XrPath* atoma, tipa koji poseduje informacije o putanjama. Tako da ako treba vezati akciju puštanja plejera na dugme za selekciju, u setu akcija nad plejerom dodala bi se nova akcija *onPlayerSelected*, na primer. U profil interakcije za uređaj *Oculus Quest 2* bismo vezali akciju selekcije plejera na putanji:

`/user/hand/right/input/trigger/value`²¹

Podešavanje akcija, seta akcija i profila interakcije se radi jednom na početku rada aplikacije koja započinje XR tok izvršavanja. Tok izvršavanja automatski mapira akcije u zavisnosti od hardvera na kome je aplikacija pokrenuta. Poslednji korak za podešavanje akcija je da se setovi akcija zakače na XR sesiju putem poziva *xrAttachSessionActionSets*. Jednom kad setovi akcija budu zakačeni za sesiju nije moguće promeniti akcije bez da se sesija poruši i ponovo pokrene.

3.1.1 Podsystem za interakciju sa plejerom

Fokus ove funkcionalnosti je vizuelni prikaz kada korisnik cilja ekran video plejera kontrolerima. Jedan kraj zraka je usidren na kontroler, dok se sa drugim krajem ciljaju objekti u sceni. U ovom radu je korišćen XR profil interakcije koji je dat kao primer iz kompleta alata za interakcije u XR. Konfiguracija interakcije između kontrolera i ostalih objekata u sceni se vrši preko inspektora.

Za početak je neophodno odabrati profil XR interakcije koji će biti aktivan unutar scene. Zatim vezati ovaj profil za menadžer akcija unosa (engl. *Input Action Manager*) koji je deo Unity scene. Sledeći menadžer je menadžer kontrolera baziran na akcijama (engl. *Action Based Controller Manager*). On interpretira vrednosti praćenja kontrolera u prostoru tako što koristi XR sistem. Primenjuje vrednost za poziciju kontrolera na položaj 3D modela ruke. Veže akcije kao što je selekcija na Unity ugrađen sistem unosa.

²¹<https://docs.unity3d.com/Packages/com.unity.xr.openxr@0.1/manual/features/oculustouchcontrollerprofile.html>

XRDirectInteractor je komponenta koja postoji na svakoj ruci ponaosob i omogućuje direktnu interakciju sa objektima u sceni sa kojima se model ruke dodiruje. Objekti sa kojim se može interagovati su interaktivni objekti (engl. *Interactable*). Ovu funkcionalnost pruža *XRSimpleInteractable* skripta koja pripada kompletu alata za XR. Detektovanje ciljanja nekog objekta u sceni od strane zraka se sprovodi putem kolizija. Objekat mora posedovati površinu za kolizije (engl. *Collider*), komponentu za detektovanje kolizija između dve ili više površina unutar scene.

Kako bi se ove interakcije mogle sprovesti na daljinu, putem zraka, dodaju se dodatne skripte na ovaj već postojeći niz skripti. Ove skripte idu na ruku koja je namenjena da bude izvor zraka. Komponenta koja iscrtava zrak je *LineRenderer* i ova komponenta je nezavisna od XR. Druga skripta koja je neophodna je *XRInteracotorLineVisual* koja dobavlja početnu i krajnju tačku zraka kako bi se iscrtao. Naposljetku, za interakcije sa zrakom koristi se *XRRayInteractor* skripta koja okida događaje sa objektima koji poseduju *XRSimpleInteractable* komponentu i njihova površina je u koliziji sa zrakom.

Komplet alata za interakciju u XR-u je omogućio da se vežu tasteri na kontrolerima na akcije koje su C# metode. Korisničko iskustvo i mogućnosti u XR aplikacijama su izuzetne, a isti sistem se može primeniti na svaku platformu koja podržava OpenXR i Unity pogon. Ovde naveden sistem je osnova korisničkih interakcija sa plejerom, klasa za plejer interakcije je krajnja tačka koja sluša događaje interakcije i obaveštava ostatak sistema u ovom radu o njima.

3.2 Tok izvršavanja aplikacije

Tok događaja do stanja u kome se video uspešno reprodukuje ide ovako: nakon što korisnik okine događaj selekcije kako bi pusto video na ekranu, modul korisničkih interakcija započinje komunikaciju sa Android arhivom među sloja putem C# interfejsa za pozivanje Java metoda. Putem Java fabričke metode stvara se nova instanca klase plejer omotača koja interno podesi plejer za reprodukciju. Android među sloj u ovoj fazi kreira grafičke objekte neophodne za konstruisanje instance plejera. Pri završetku fabričke metode, zauzimanju neophodnih grafičkih resursa i kreiranju instance plejera unutar Java omotača, modul za korisničke interakcije šalje zahtev za proširenje grafičkog toka Unity pogona. Kreira se komandni bafer kome se prosleđuje pokazivač na funkciju iz Android među sloja koja je zadužena za potrošnju grafičkih bafera teksturom za prikaz videa kao i nativni identifikator strukture za iscrtavanje asocirane sa ekranom. Ova funkcija će se okinuti svaki put kada grafička procesna linija pozove komandni bafer. Naposljetku se na komandni bafer mapira tekstura iscrtavanja koja je nalepljena na ekran za video prikaz unutar Unity scene.

Stanje iscrtavanja je specifična faza u aplikaciji koja se odvija na niti za iscrtavanje odvojeno od ostatka sistema. Prvi korak je konzumirati bafer površinske teksture. Ovo se izvodi putem JNI tako što se pozove metoda za konzumiranje bafera unutar trenutne instance JVM-a. Ako postoji više ekrana koji reprodukuju video u sceni, neophodno je pozvati metodu za konzumiranje bafera na svakom objektu tipa površinske teksture koja je inicijalizovana u nativnom modulu.

Video tokovi i ostatak objekata u sceni se ne iscrtavaju u istu grafičku zonu memorije. Komandni baferi su konfigurisani tako da se drugi namenski baferi koriste za iscrtavanje video sadržaja koji se razlikuju od osnovnih bafera koji koristi ostatak objekata unutar scene.

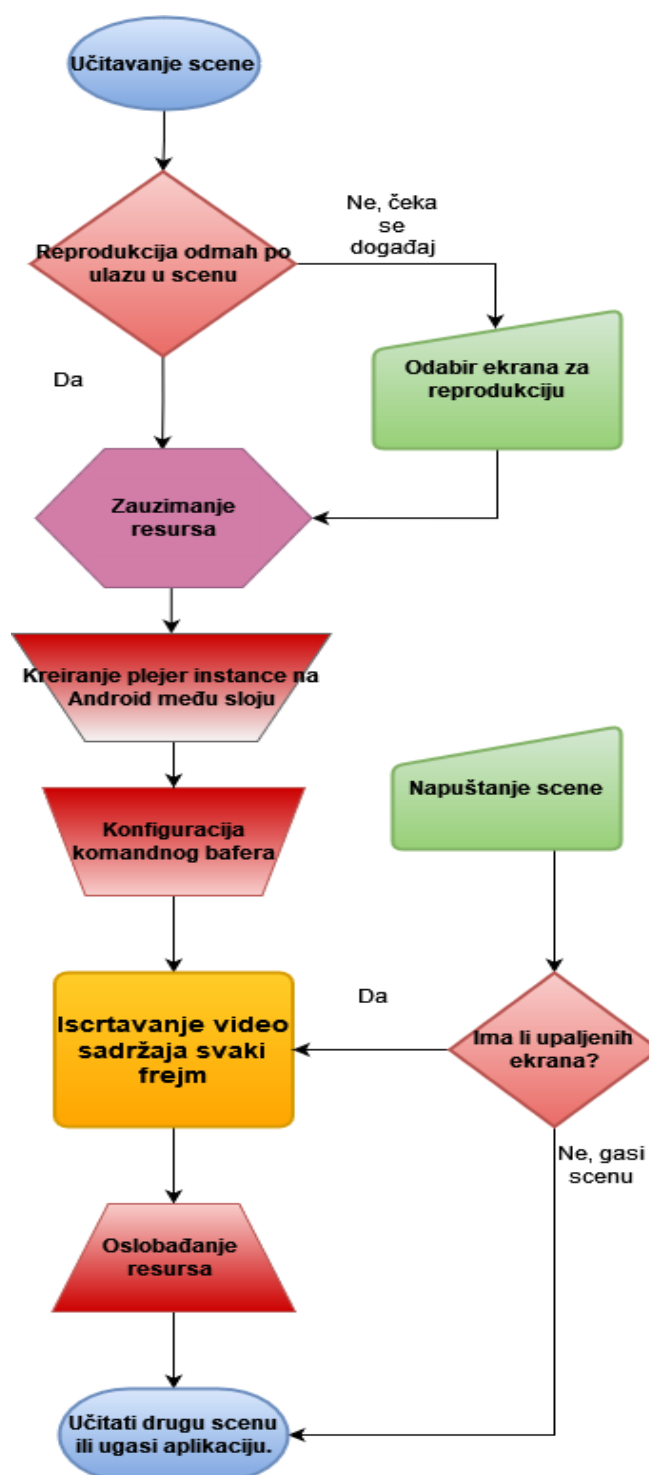
Tekstura za iscrtavanje ispod haube, kada je OpenGL ES korišćeni API, je bafer objekat okvira (engl. *Frame Buffer Object* – *FBO*). FBO je napredni koncept u OpenGL ES-u koji omogućuje da se iscrtavanje ne vrši direktno na ekranu, nego je moguće iscrtavati u pozadinske bafere i po potrebi ih prikazivati na ekran. Svaka tekstura za iscrtavanje poseduje bafer za boju i bafer za dubinu. Bafer za boju je najbitniji, jer 2D tok videa nema dubinu tako da se može zanemariti. Svaki komandni bafer koji se kači na glavnu kameru kroz koju korisnik vidi scenu oko sebe ima svoju teksturu za iscrtavanje – svaki komandni bafer ima svoj FBO.

Ako je tekstura za iscrtavanje deo scene, njen interni FBO neće biti pozadinski već će se iscrtati. Dobra analogija bi bila ekran u ekranu, teksture za iscrtavanje su prozori u koje se iscrtava video i putem položaja u sceni pozicionira u 3D prostoru. Video tok se iscrtava na celoj površini teksture, položaj teksture (ekrana) će Unity interno odraditi. U iscrtavanju se koristi šejder program koji vrši odabiranje uzoraka putem *samplerExternalOES* promenljive, koja je namenska za OpenGL ES. Ova promenljiva ima pristup baferima na potrošaču površinske teksture i može učitati vrednosti iz video bafera u fragment deo šejder programa.

Počevši od potrošnje bafera pa do iscrtavanja ekrana na celoj površini teksture preko *samplerExternalOES* odabirača, process iscrtavanja se ponavlja više puta u sekundi i njegova učestanost je vrlo često znatno veća od FPS video toka.

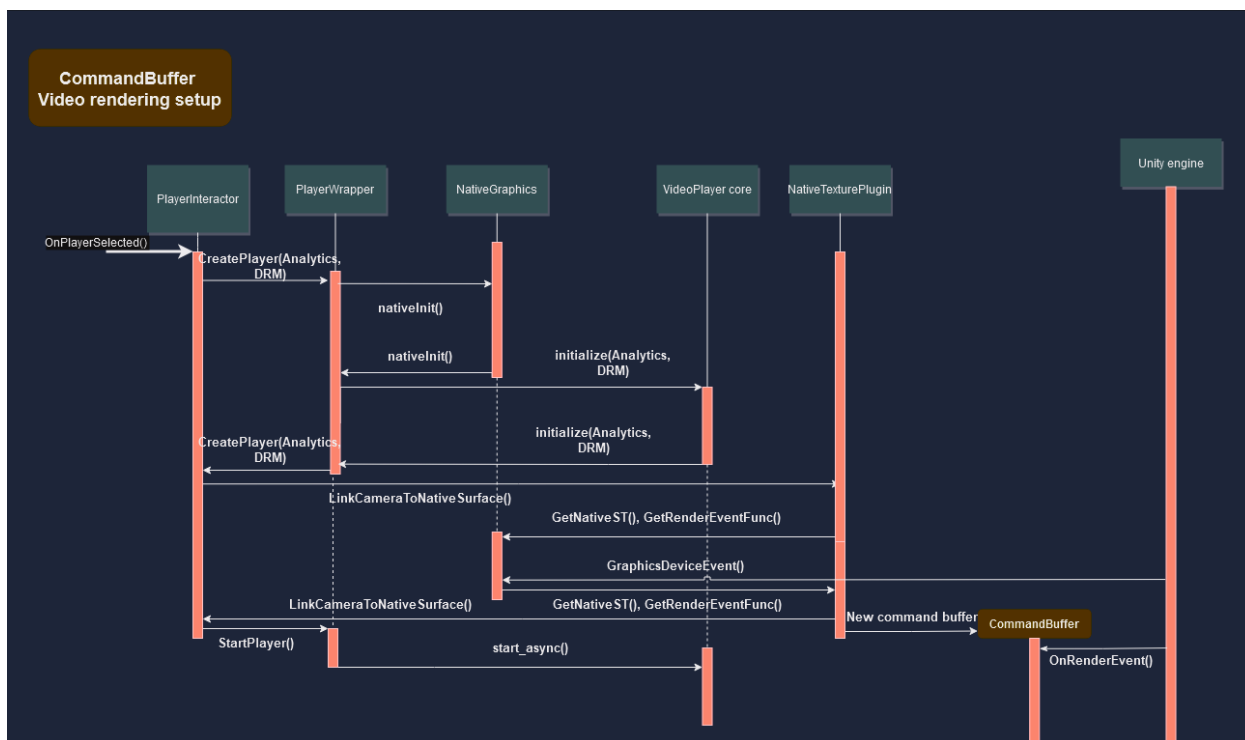
U slučaju da se aplikacija ugasila ili je korisnik izašao iz scene, neophodno je obustaviti rad komandnih bafera i osloboditi grafičke resurse koji su bili dodeljeni nativnom modulu prilikom puštanja video sadržaja. Oslobađanje memorije zatim završava Unity pogon koji oslobađa sve Android Java objekte koje je kreirao dok je korisnik bio u sceni automatski, pošto su Android Java objekti polja unutar *MonoBehaviour* klasa koje se uništavaju kada se scena ugasi.

Sledeća slika prikazuje ovaj tok događaja:



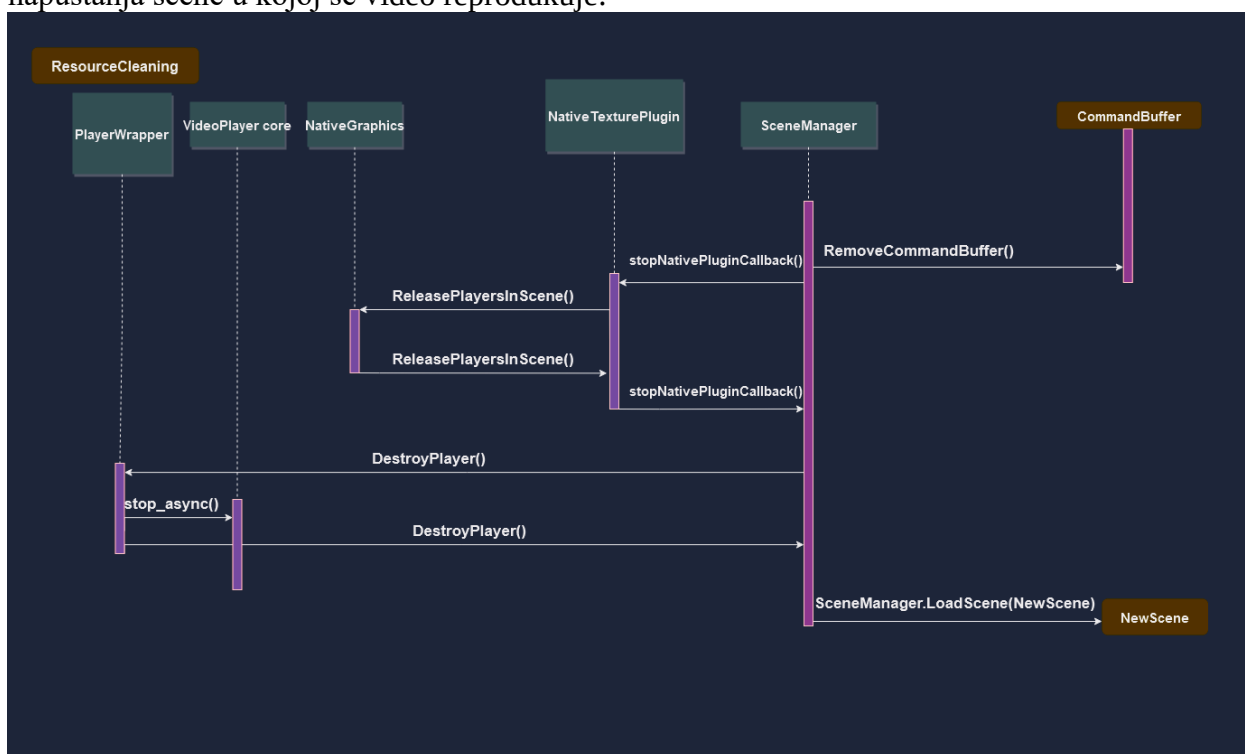
Slika 12 Dijagram toka is crtavanja videa unutar Unity scene.

Kako bi se jasnije pokazalo kako funkcioniše među komunikacija svih modula kod zauzimanja resursa, sledeći UML dijagram pokazuje tok poziva kod zauzimanja resursa:



Slika 13 Konfiguracija rešenja za iscertavanje videa u komandni bafer.

Naredni UML dijagram pokazuje tok izvršavanja oslobađanja resursa u slučaju napuštanja scene u kojoj se video reprodukuje:



Slika 14 Oslobađanje resursa iscertavanja video ekrana.

3.3 Upravljanje teksturama za prikaz videa u C++ kodu

Tekstura za prikaz je Java objekat. Potrošnja bafera se, kao što se vidi na *Slici 10*, dešava pre nego što se dogodi GLES iscertavanje. Zato što XR aplikacija ima veliki broj niti koje

paralelno rade razne poslove, ovo je teško garantovati. U ovom radu je taj problem rešen tako što se iz funkcije izvršavanja komandnog bafera vrši i potrošnja bafera teksture videa.

Android poseduje API za pristup trenutnoj JVM iz JNI funkcija²². JNI definiše dve ključne klase koje ovo omogućuju, *JavaVM* i *JNIEnv*. U C++ verziji JNI, *JavaVM* i *JNIEnv* poseduju pokazivače do tabele funkcija i metodu za svaku JNI funkciju koja se preko pokazivača nalazi u tabeli. *JavaVM* pruža interfejs za kreiranje i uništavanje JVM. U teoriji je moguće posedovati više instanci JVM po procesu, ali Android dozvoljava samo jednu.

JNIEnv u sebi sadrži većinu JNI funkcija. Prvi argument svih JNI funkcija je *JNIEnv*. U ovom radu se koristi princip deljenja *JNIEnv* sa statičkim funkcijama (engl. *Static*) unutar C++ grafičke biblioteke. Iz ovog razloga je neophodno da instanca JVM koja se kreira na početku rada aplikacije bude globalna. Kako bi kod mogao da se izvrši na JVM u niti koja nije deo JNI, *JavaVM* klasa poseduje metodu *GetEnv* koja pokušava da nađe *JNIEnv* instancu, pod uslovom da postoji jedna u aplikaciji. Ako okruženje postoji nit koja izvršava zatvorenu funkciju treba samo da se zakači na pronađen *JNIEnv* objekat. Klasa *JavaVM* poseduje metodu *AttachCurrentThread* koja pokušava da zakači trenutnu nit na JNI. Ako ova procedura bude uspešna, potrošnja bafera površinske teksture iz komandnog bafera je moguća, pod uslovom da JNI ekvivalent površinske teksture u tom zadatom kontekstu postoji.

Kako bi se zagarantovalo da će površinska tekstura postojati nakon faze inicijalizacije nativnog koda, kada se na zahtev Java omotača kreira nativno okruženje za iscrtavanje, površinska tekstura koja je kreirana mora da bude globalna.

Svaki objekat koji se napravi unutar JNI metode je lokalna referenca na dati objekat. To znači da je validna dokle god traje izvršavanje JNI metode u trenutnoj niti. Ako se pokuša korišćenje nativnog objekta nakon završetka JNI metode ili na drugoj niti referenca neće biti validna. Jedini način da se dobiju reference koje nisu lokalne je putem *JNIEnv* metoda *NewGlobalRef* i *NewWeakGlobalRef*. U ovom radu se koristi *NewGlobalRef* metoda.

Kako bi se pristupilo površinskoj teksturi iz nativnog koda postoji sledeća procedura:

1. Upotrebiti JNI za dobavljanje reference na klasu objekta upotrebom *FindClass* metode;
2. Pozvati metodu koja pretražuje trenutnu JNI okruženja u potrazi za konstruktorom površinske teksture.
3. Ako je konstruktor uspešno pronađen, moguće je kreirati novi objekat upotrebom JNI metode.
4. Neophodno je napraviti ovaj objekat globalnim.

²² Teorijske osnove ovog poglavlja se baziraju na sledećem članku:
<https://developer.android.com/training/articles/perf-jni>

5. Naposljetku pronaći metodu za potrošnju bafera unutar JNI okruženja.

Android poseduje namenske ekstenzije OpenGL ES API-a. Osim spoljnih tekstura (GL_TEXTURE_EXTERNAL_OES) drugi bitan koncept za iscrtavanje videa na Android platformama su proširenja GLSL programa. Ova proširenja ne moraju da postoje na svim Android platformama ali su širokorašprostranjena. Kako bi se okvir slike mogao učitati u fragment šejderu neophodno je da postoji *samplerExternalOES* tip promenjive unutar GLSL šejder programa. Tip *samplerExternalOES* se može pronaći u *GL_OES_EGL_image_external*²³ ili *GL_OES_EGL_image_external_essl3*²⁴ OpenGL ES proširenjima.

Izvedba prethodnih koraka kako treba garantuje da će površinska tekstura moći da potroši bafere video toka adekvatno iz komandnog bafera. Pravilna primena GLSL programa iscrtava video u 3D prostoru.

3.4 Uključivanje grafičkih biblioteka u C++ biblioteku

Kao što je u teorijskim osnovama navedeno, alat za generisanje konfiguracije za kompajliranje je *CMake*. Biblioteke se uključuju u kompajliranje sa *add_library* funkcijom. Grafička biblioteka za iscrtavanje videa se definiše u ovoj funkciji. Njeno ime, status uvezivanja u Android arhivu i datoteke izvornog koda koje je sačinjavaju se navode kao parametri. Dinamičke biblioteke za ciljanu platformu neophodne za integraciju OpenGL ES API u *cppgraphics* biblioteku se nalaze u NDK alatima. Android Studio automatski pretražuje i uvezuje ove biblioteke funkcijom *find_library* i *target_link_libraries*. Sledi spisak svih biblioteka koje se uključuju u *cppgraphics* biblioteku:

- **app-glue** – spaja *cppgraphics* biblioteku sa Android aplikacijom.
- **log-lib** – omogućuje ispile na ADB radi otklanjanja greški.
- **android-lib** – Android JNI funkcionalnosti.
- **EGL-lib** – nativni grafički interfejs.
- **GLSv1-CM-lib** – OpenGL ES API v1
- **GLSv3-lib** – OpenGL ES API v3

Osim ovih biblioteka u C++ biblioteku se uključuju i Unity zaglavlja koja omogućuju proširenje pogona i sa komandnim baferima.

3.5 Problem linearnog prostora boja

²³ Za starije verzije OpenGL ES API se može pronaći ovde:
https://registry.khronos.org/OpenGL/extensions/OES/OES_EGL_image_external.txt

²⁴ Za verziju OpenGL ES 3.0 i veću ekstenzija se može pronaći ovde:
https://registry.khronos.org/OpenGL/extensions/OES/OES_EGL_image_external_essl3.txt

Prikazivanje boja unutar videa predstavljaju specifičan problem koji proizilazi iz izbora Unity pogona i nemogućnosti da se sa OpenGL ES-om koristi gama prostor boja. Naime, video je eksplicitno enkodiran u gama prostoru boja što znači da će prikaz videa sa bilo kojim šejderom unutar scene koji koristi linearni prostor boja proizvesti video lošijeg kvaliteta.

Rešenje koje se koristi u ovom radu je aproksimacija gama vrednosti za RGB piksela u linearnom prostoru:

```
fragColor = vec4(sRGB * (sRGB * (sRGB * 0.305306011 + 0.682171111) + 0.012522878),  
1.0);
```

Ovaj izraz je izlaz iz fragment šejdera koji iscrtava video. *sRGB* promenljiva je vrednost fragmenta videa koja se odabira. Izraz je napisan u GLSL sintaksi i neophodan je kompromis kako bi se kvalitet videa popravio.²⁵

²⁵ Vrednosti koje se nalaze u šejder programu su uzete iz izvornog koda Unity ugrađenih šejdera i izraz preveden u GLSL, pogledati funkciju *GammaToLinearSpace*: <https://github.com/TwoTailsGames/Unity-Built-in-Shaders/blob/master/CGIncludes/UnityCG.cginc>

4. Programsko rešenje

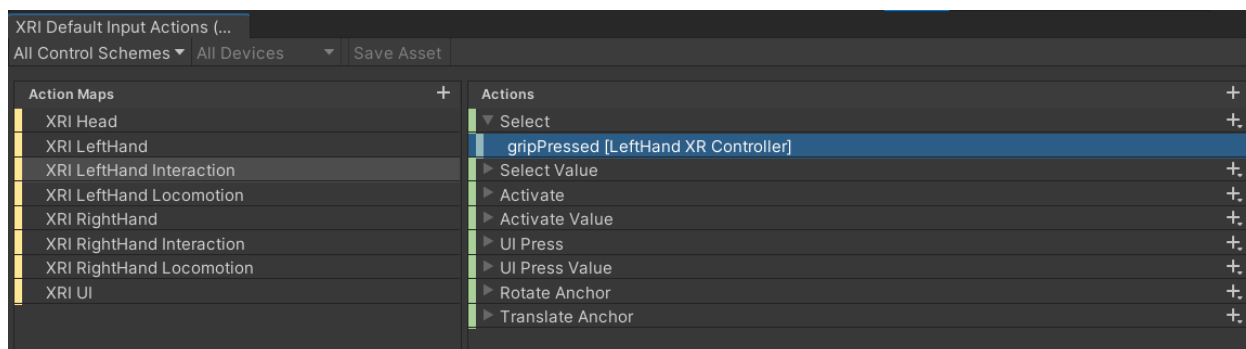
U daljem tekstu će biti opisana programska realizacija koncepta rešenja. Programsko rešenje je, kao što je već rečeno u teorijskim osnovama, realizovano kao Android aplikacija za XR. Za realizaciju su korišćena četiri programska jezika, C++, Java, C# i GLSL. Java, C++ i GLSL. Java, C++ i GLSL programi su sastavni deo Android arhive koja se kao dodatak uključuje u Unity pogon. Unity C# skripte su integrisane u pogon i komuniciraju sa Android među slojem. Dokumentacija rešenja je podeljena po programskom jeziku i detalji veza između programskih jezika navedeni.

4.1 Unity skripte

U ovom poglavlju će biti opisane komponente Unity pogona koje su sastavni deo rešenja. Dalje izlaganje će biti podeljeno u dve celine. Prva se odnosi na komplet alata za XR i konfiguraciju Unity radnog okruženja kako bi interakcije sa video bile omogućene u aplikaciji. Druga se odnosi na skripte napisane da integrišu video plejer u scenu tako se video reprodukuje i korisnik ima kontrolu.

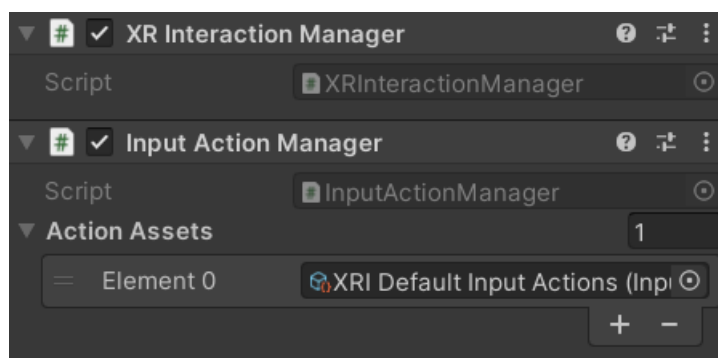
4.1.1 Podešavanje Unity radnog okruženja za XR interakciju

Kao što je navedeno u konceptu rešenja, komponente za XR interakciju su deo kompleta alata za XR interakcije. U ovom radu je korišćen osnovni set akcija za interakciju koji dolazi sa kompletom radi jednostavnosti.



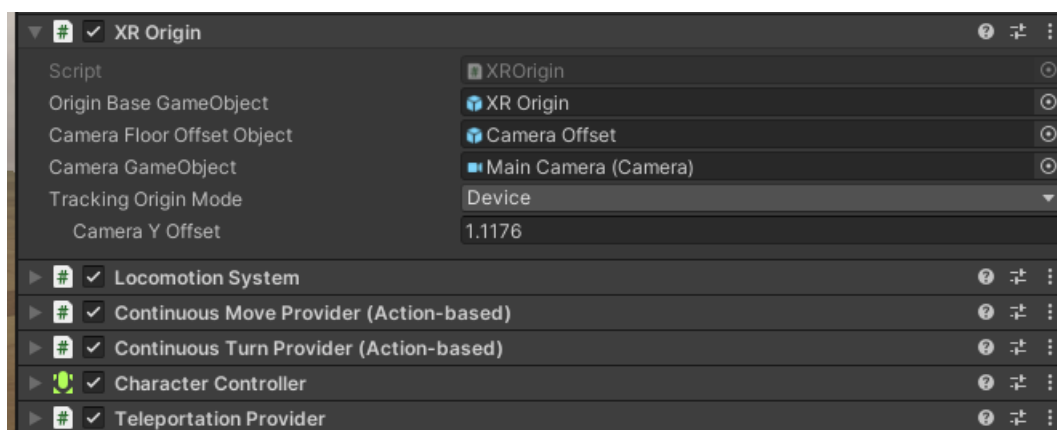
Slika 15 Osnovni set XR interakcija.

Unutar scene mora da postoji menadžer XR interakcija u kome se postavljaju akcije koje se koriste u sceni. Upotrebom menadžera XR interakcija, vrednosti za položaj glave i kontrolera očitane na XR uređaju se dobavljaju i ažuriraju u sceni. Akcije se vežu za događaje na XR uređaju preko menadžera XR interakcija. Podešavanje komponente menadžera XR interakcija:



Slika 16 Menadžer XR interakcija.

Za razliku od Unity aplikacija napravljenih za desktop računare, aplikacije za XR imaju dodatni objekat unutar scene koji je koordinatni početak 3D scene, *XROrigin*. Namena *XROrigin* objekta je da primeni transformacije položaja, orijentacije i skaliranja na objekte koji imaju mogućnost praćenja promena vrednosti položaja ili orijentacije na XR uređaju.

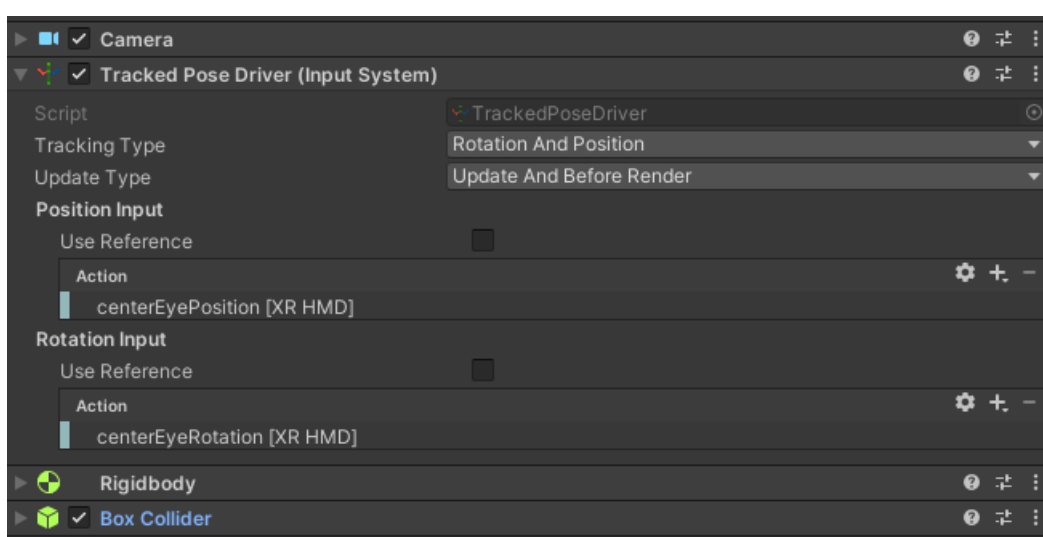
Slika 17 Podešavanje *XROrigin* objekta u inspektoru.

Unutar Unity scene, *XROrigin* je roditelj objektima koji predstavljaju ruke i glavu.



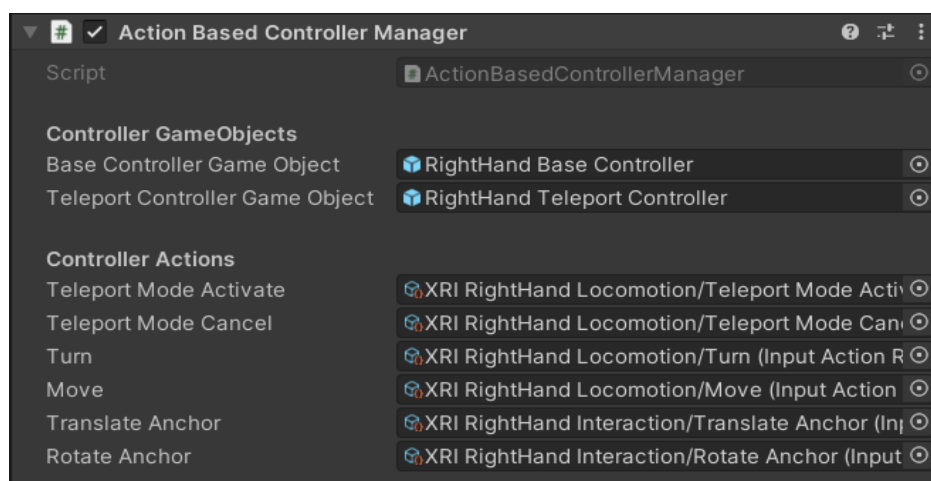
Slika 18 XROrigin objekat i njegova deca unutar scene.

Kamera je standardna za bilo koji tip aplikacije 3D (*Linux, Windows, IOS* itd.). Glavna komponenta koja izdvaja kameru za XR je *TrackedPoseDriver* koji prati pokrete korisnikove glave putem OpenXR toka izvršavanja i ažurira položaj kamere unutar scene.



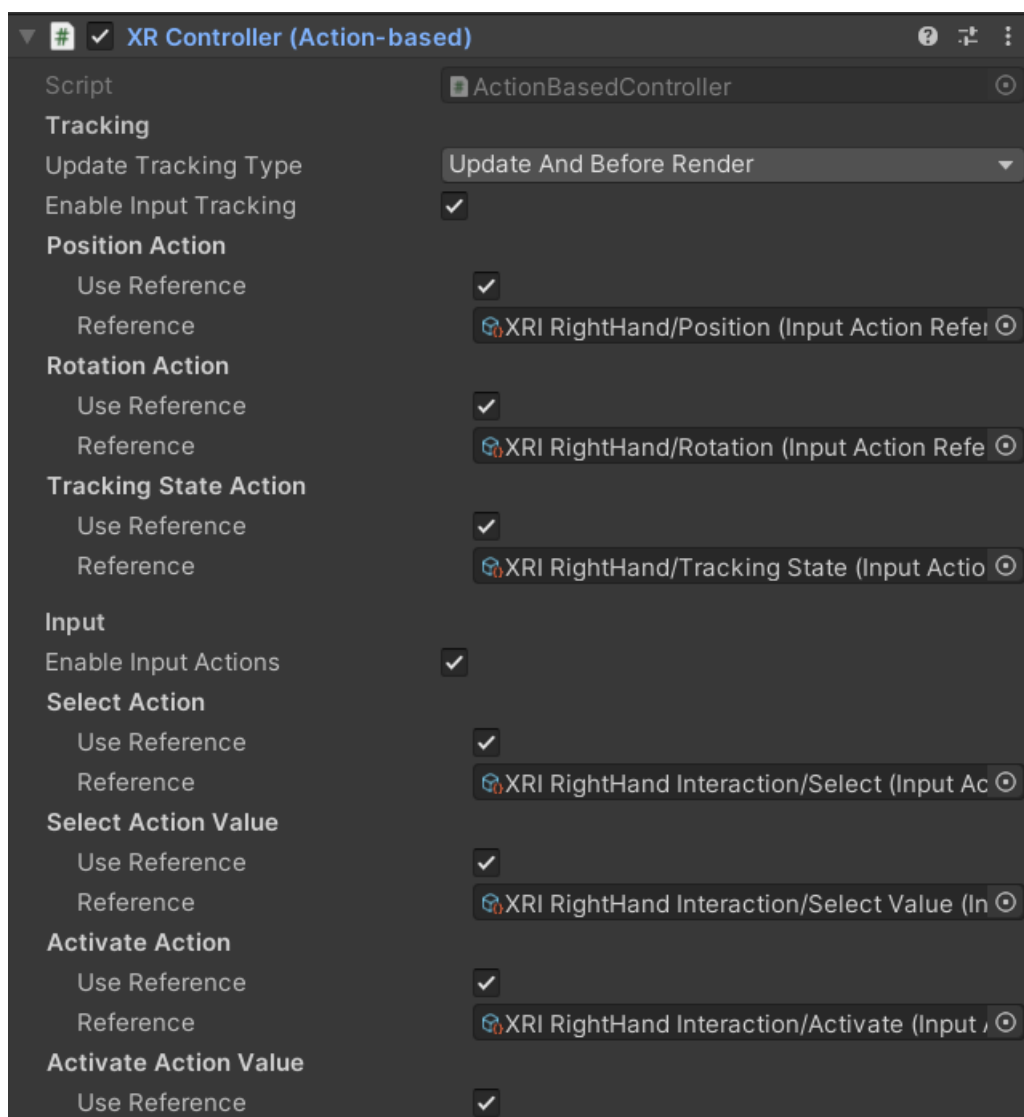
Slika 19 Konfiguracija kamera objekta.

Ruke se konfigurišu tako što se akcije iz osnovnog seta postavljaju na *ActionBasedControllerManger*. 3D model koji se prikazuje na mestu gde se nalaze kontroleri se takođe podešava na *ActionBasedControllerManager* komponenti.



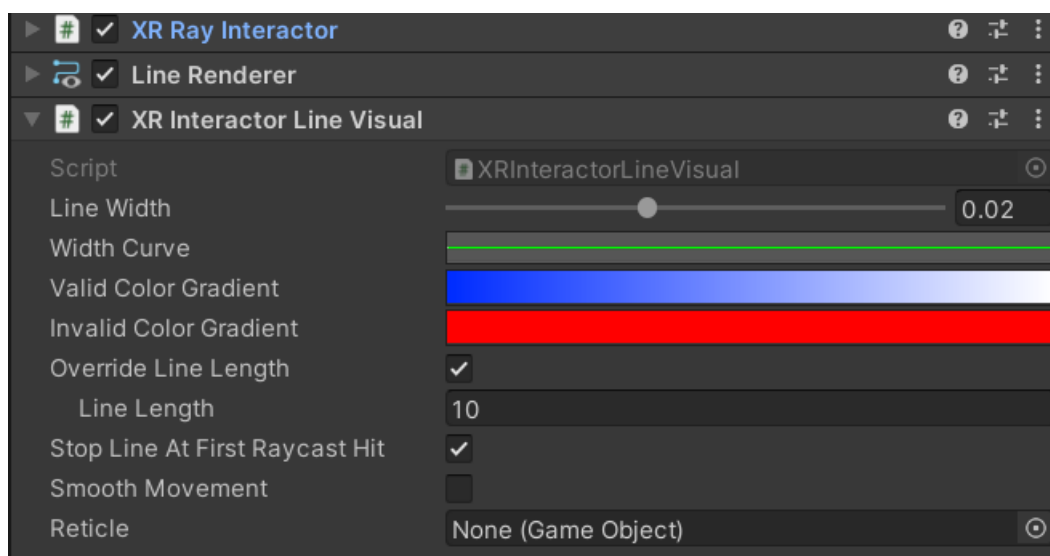
Slika 20 Menadžer kontrolera baziran na akcijama.

Svaka ruka poseduje dve objekta: osnovni kontroler i kontroler za teleportaciju. Osnovni kontroler služi da učitava sve akcije i prati kretanje kontrolera levog ili desnog. Poseduje komponentu za direktne interakcije (hvatanje objekta npr.).



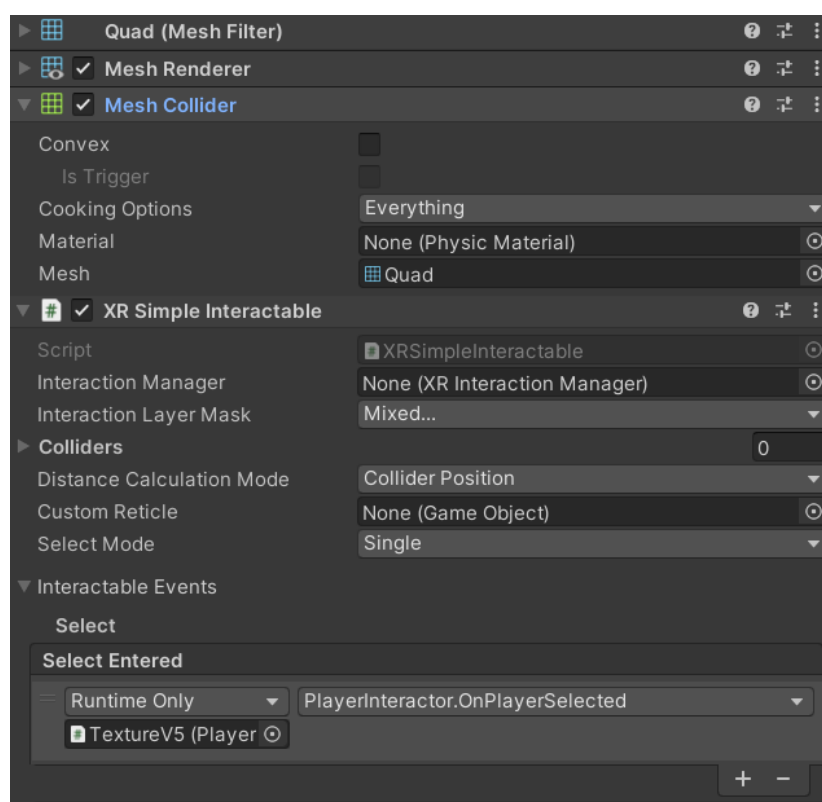
Slika 21 Osnovna komponenta kontrolera.

Interakcija zraka sa objektima u sceni se sprovodi preko *XRRayInteractor* komponente. Kako bi se zrak iscrtao neophodno je imati i *LineRenderer* komponentu. Naposljetku *XRIInteractorLineVisual* fiksira jedan kraj zraka na 3D model ruke, dok se sa drugim krajem cilja u sceni.



Slika 22 Komponente za interakciju zrakom.

Do sada su opisivane komponente kompleta za interakciju koje omogućuju korisniku da interaguje sa scenom. U nastavku će biti opisane komponente koje objekat u sceni prave interaktivnim. Fokus je na ekran za reprodukciju, pokazaće se kako se omogućuje interakcija putem ciljanja zrakom na ekran i kako pritisak dugmeta za selekciju na kontroleru okida događaj interakcije. U ovu svrhu dovoljno je dodati *MeshCollider* komponentu, podesiti je da odgovara obliku ekrana kako bi se detektovalo preklapanje zraka sa ekranom. Zatim u *XRSimpleInteractor* skripti registrovati događaj interakcije selekcijom kao na sledećoj slici:



Slika 23 Komponente interaktivnog ekrana u sceni.

4.1.2 Realizacija upravljanja plejerom unutar Unity scene upotrebom **PlayerInteractor** klase

Najviši nivo Unity plejer dodatka predstavlja klasa *PlayerInteractor*. Metoda *OnPlayerSelected* je glavna metoda koja spaja video plejer sa kompletom za XR interakcije i OpenXR tokom izvršavanja. Za osnovne operacije inicijalizacije, puštanja, pauziranja i stopiranja plejera definisan je automat koji prati stanje plejera:

Vrednost	Opis
Idle	Inicijalno stanje svakog plejera.
Started	Plejer je započeo reprodukciju.
Paused	Reprodukcija je pauzirana.
Resumed	Reprodukcija je nakon pauziranja nastavljena.
Stopped	Reprodukcija je zaustavljena.

Tabela 2 Stanja plejera.

Polja sa [SerializeField] atributom su namenjena da se menjaju iz Unity radnog okruženja preko inspektor prozora. Glavno polje ove klase je *playerWrapper* koje je referenca na Java objekat za upravljanje plejerom.

Polje	Opis
<code>private IDictionary<int, PlayerState> PlayerStates;</code>	Rečnik koji čuva trenutno stanje plejera.
<code>[SerializeField] private bool playOnSceneLoad;</code>	Ako je iskaz tačan pokreće reprodukciju videa odmah po učitavanju scene.
<code>[SerializeField] private bool Analytics;</code>	Ako je iskaz tačan Java omotač će slati meta podatke o videu koji se reprodukuje u <i>PlayerInteractor</i> .
<code>string[] contentUri</code>	Niz linkova ka videima za reprodukciju.
<code>[SerializeField] private int track;</code>	Indeks trenutnog videa unutar <i>contentUri</i> liste videa za reprodukciju.
<code>public AndroidJavaObject playerWrapper;</code>	Java objekat koji rukovodi nativnim slojem plejera.

Tabela 3 Polja klase *PlayerInteractor*.

Metode *PlejerInteractor* klase se efektivno mogu podeliti na nasleđene *MonoBehaviour* metode, metode koje osluškuju događaje interakcije sa korisnikom i osnovne metode upravljanja video plejerom. Na sledećoj tabeli se nalaze sve metode *PlejerInteractor* klase:

Metoda	Opis
<code>private void Awake ()</code>	Inicijalizuje promenljive pre iscrtavanja scene.
<code>void Start ()</code>	Pod uslovom da je <i>playOnSceneLoad</i> iskaz tačan, plejer se inicijalizuje i počinje sa reprodukcijom.
<code>public void OnPlayerSelected()</code>	Događaj odabira plejera. Započinje reprodukciju videa ako je plejer u <i>Idle</i> ili <i>Stopped</i> stanju.
<code>public void InitializePlayer()</code>	Poziva fabričku Java metodu koju kreira instancu plejer omotača.
<code>public void ParseAndroidMessage(string message)</code>	Pod uslovom da je <i>Analytics</i> iskaz tačan, Java omotač će prosleđivati poruke sa meta podacima pozivanjem ove metode.
<code>public void StartPlayer(string uri)</code>	Započinje reprodukciju videa pozivanjem ekvivalentne metode iz Java omotača.
<code>public void StartPlayerOffset(string uri, float offset)</code>	Započinje reprodukciju videa od nekog proizvoljnog trenutka u toku.
<code>public void PausePlayer()</code>	Pauzira videa ako je započeo reprodukciju.
<code>public void ResumePlayer ()</code>	Nastavlja reprodukciju videa ako je video u tom trenutku bio u stanju <i>Paused</i> .
<code>public void StopPlayer()</code>	Zaustavlja video pod uslovom da je video bio u stanju reprodukcije.
<code>private void SeekBarUpdate()</code>	Zahteva od Java omotača trenutno vreme do kog je video stigao sa reprodukcijom.
<code>public void SeekPlayerToTimestamp(string timestamp)</code>	Pod uslovom da je video u stanju reprodukcije, ovom metodom se zahteva od Java omotača da premota video.

Tabela 4 Metode *PlayerInteractor* klase.

4.1.3 Realizacija proširivanja grafičkog toka SRP upotrebom *NativeTexturePlugin* klase

Proširivanje grafičkog toka se realizuje tako što *PlayerInteractor* uhvati događaj interakcije sa ekranom unutar scene i napravi zahtev prema *NativeTexturePlugin* klasi da proširi grafički tok. *PlayerInteractor* zahteva postojanje *NativeTexturePlugin* komponente na istom objektu (u ovom slučaju ekranu) kako bi pozvao metodu *LinkCameraToNativeSurface*. *NativeTexturePlugin*, za razliku od *PlayerInteractor* klase koja komunicira sa Javom, dinamički uvezuje funkcije iz C++ grafičke biblioteke. *StopNativePluginCallback* metoda služi

za upravljanje resursima, u slučaju kada se napušta scena ili se gasi aplikacija. Skripte koje uhvate ovaj događaj mogu za svaki ekran osloboditi native resurse.

Polje	Opis
<code>public static List<NativeTexturePlugin> nativeLinks;</code>	Lista svih <i>NativeTexturePlugin</i> objekta. Služi za upravljanje resursima, kada je potrebno osloboditi sve native resurse vezane za <i>NativeTexturePlugin</i> objekte.
<code>[SerializeField] private RenderTexture renderTex;</code>	Tekstura za iscrtavanje koja se nalazi na ekranu. Jedan je od parametara za konfiguraciju komandnog bafera.
<code>private CommandBuffer cb;</code>	Komandni bafer koji se kači na kameru kako bi se proširio grafički tok. Svaki <i>NativeTexturePlugin</i> poseduje jednu instancu, tako da se za svaki video dodaje po jedan komandi bafer u SRP.

Tabela 5 Polja *NativeTexturePlugin* klase.

Metoda	Opis
<code>private void Awake()</code>	Prebrojava sve instance <i>NativeTexturePlugin</i> i dodaje ih u <i>nativeLinks</i> listu.
<code>public void LinkCameraToNativeSurface()</code>	Metoda kreira i kači komandni bafer na SRP.
<code>public static void stopNativePluginCallback()</code>	Oslobađa native resurse na zahtev skripte koja prepoznaje promenu scene.

Tabela 6 Metode *NativeTexturePlugin* klase.

Osim ovih metoda *NativeTexturePlugin* uključuje i dinamički uvezane C++ funkcije:

- `[DllImport("cppgraphics")] private static extern IntPtr GetRenderEventFunc();`
- `[DllImport("cppgraphics")] private static extern void ReleasePlayersInScene();`
- `[DllImport("cppgraphics")] private static extern long GetNativeSt();`

Ove funkcije će biti opisane u 4.3 kada se govori o C++ biblioteci za iscrtavanje videa.

4.1.4 Realizacija *SceneReload* klase

Klasa *SceneReload* služi da demonstrira upravljanje resursima koje je aplikacija zauzela za reprodukciju video sadržaja. Njena jedina funkcija unutar aplikacije je da ponovo učitati istu scenu. Ovaj metod je izabran zato što je jednostavan za ponavljanje. Učitavanje neke proizvoljne

Unity scene koja je uključena u aplikaciju se radi preko *UnityEngine.SceneManagement* ugrađene biblioteke. Osvežavanje scene se vrši sledećim izrazom:

```
SceneManager.LoadScene(SceneManager.GetActiveScene().name);
```

Ova klasa poseduje samo jednu značajnu metodu:

Metoda	Opis
<code>public void OnReloadScene()</code>	Osvežava scenu, tako što je pokreće iznova. Korisna ako se registruje kao događaj za neku interakciju.

Tabela 7 Metode *SceneReload* klase.

4.2 Realizacija Java omotača za upravljanje plejerom

Java omotač služi da smanji broj pristupa neophodnih za upravljanje plejerima unutar Unity pogona. Ovaj modul je omotač zato što pristupa JNI metodama video plejer Android arhive na zahtev Unity skripti.

Android arhiva je proširena Java bibliotekom Unity pogona. Ova biblioteka u sebi sadrži Java klase koje su neophodne da bi se Unity pogon integrisao sa Android tokom izvršavanja. Klase koje se koriste da se omogući integracija sa Androidom su *UnityPlayer* i *UnityPlayerActivity*.

Glavna aktivnost Android aplikacije nasleđuje od Unity plejer aktivnosti i nju stavlja u fokus. Klasa *MainActivity* ima metodu za uništavanje *OnDestroy* koja se poziva kada se aplikacija ugasi. Metoda *OnDestroy* poziva JNI metodu iz C++ grafičke biblioteke *nativeRelease* koja oslobađa sve dinamički alocirane objekte u biblioteci. Ovo je čist način za obustavljanje rada grafičkog iscrtavanja. Sledi opis Java klasi koje se koriste u ovom radu.

4.2.1 Realizacija *PlayerWrapper* klase

Na sloju Unity pogona glavna klasa za upravljanje plejerom je *PlayerInteractor*, analogna klasa na Android sloju je *PlayerWrapper*. Početna tačka pristupa Java omotaču je fabrička metoda *CreatePlayer* koja je zadužena da na zahtev Unity pogona stvori novi omotač plejera i vraća referencu na Java objekat u Unity scenu. Interno, ova metoda poziva konstruktor *PlayerWrapper* klase koji prvo obavesti nativni sloj da se kreira novi plejer videa. Zatim zahteva od nativnog sloja da napravi novu površinsku teksturu (potrošač bafera). Površinsku teksturu napravljenu u prethodnom koraku koristi za pravljenje površine (proizvođač bafera). Površinu prosleđuje metodi *initialize* koja putem JNI metoda video plejer biblioteke napravi jezgro plejera. Jezgro plejera je asinhrono i zahteva da se svi pozivi njegovih metoda izvršavaju na istoj niti. Nit koja je vezana za asinhrono jezgro plejera je UI nit trenutne aktivnosti. Efektivni pozivi plejer metoda se sprovode tako što se pozovu unutar Android *Runnable* metode:

```

UnityPlayer.currentActivity.runOnUiThread(new Runnable() {
    @Override
    public void run() {
        //Call any async player JNI method.
    }
});

```

Kada je ova inicijalizacija završena, dalje upravljanje plejerom je moguće. Nema više potrebe za statičkim pristupom *PlayerWrapper* klasi, upravljanje se vrši preko *PlayerWrapper* objekta. Svaki *PlayerWrapper* objekat poseduje *PlayerListener* polje koje, ako je analitika upaljena (ako Unity pogon ima potrebu za meta podacima o videu), *PlayerWrapper* veže za jezgro plejera. Naredna tabela pokazuje polja koja *PlayerWrapper* poseduje:

Polje	Opis
<code>private IAsync_player mPlayer</code>	Jezgro plejera iz video plejer Android arhive. Svi pozivi za upravljanje iz Unity scene na kraju završavaju sa pozivom asinhronne metode koju poseduje <i>mPlayer</i> .
<code>private Surface mSurface</code>	Proizvođač bafera koji se prosleđuje u <i>mPlayer</i> konstruktor.
<code>private SurfaceTexture mSurfaceTexture</code>	Potrošač bafera koji C++ biblioteka pravi.
<code>private PlayerListener mListener</code>	Klasa koja registruje događaje slanja meta podatka u Unity scenu.
<code>private String gameObjectHandle</code>	Ime objekta unutar Unity scene koji je napravio zahtev za stvaranje instance <i>PlayerWrapper</i> objekta.
<code>private final String TAG = "PlayerWrapper"</code>	Koristi se za ispisivanje logova i otklanjanje greški.

Tabela 8 Polja *PlayerWrapper* klase.

Metoda	Opis
<code>public static PlayerWrapper CreatePlayer(String handle, boolean Analytics, boolean DRM)</code>	Na zahtev kreira novu instancu <i>PlayerWrapper</i> klase. Poziva konstruktor.
<code>public PlayerWrapper(String handle, boolean Analytics, boolean DRM)</code>	Glavni konstruktor. Poziva <i>nativeInit</i> , <i>createNativeSurfaceTexture</i> i <i>initialize</i> metode. Nakon ove procedure video je spreman za reprodukciju.
<code>public native void nativeInit();</code>	JNI metoda koja registruje novi <i>PlayerWrapper</i> objekat u C++ biblioteku i omogućava dalja podešavanja.
<code>public native SurfaceTexture</code>	JNI metoda koja kreira površinsku teksturu u

<code>createNativeSurfaceTexture()</code>	C++ biblioteci. Povratna vrednost je površinska tekstura asocirana sa <i>PlayerWrapper</i> objektom.
<code>private void initialize (boolean Analytics, boolean DRM)</code>	Kreira jezgro plejera <i>mPlayer</i> u zavisnosti od <i>Analytics</i> i <i>DRM</i> argumenta.
<code>public void startPlayer(String uri)</code>	Na UI niti šalje asinhroni zahtev za reprodukciju videa tako što prosleđuje URI videa <i>mPlayer</i> objektu.
<code>public void startPlayerOffset(String uri, float offset)</code>	Na UI niti šalje asinhroni zahtev za reprodukciju videa sa premotavanjem tako što prosleđuje URI i trenutak u videu na koji se premotava video <i>mPlayer</i> objektu.
<code>public void pausePlayer()</code>	Na UI niti šalje asinhroni zahtev za pauziranje reprodukcije videa <i>mPlayer</i> objektu.
<code>public void resumePlayer()</code>	Na UI niti šalje asinhroni zahtev za nastavak reprodukcije videa <i>mPlayer</i> objektu.
<code>public void stopPlayer()</code>	Na UI niti šalje asinhroni zahtev za zaustavljanje reprodukcije videa <i>mPlayer</i> objektu.
<code>public void destroyPlayer()</code>	Na UI niti šalje asinhroni zahtev za uništavanje instance <i>mPlayer</i> objekta.
<code>public void mutePlayer(boolean mute)</code>	Na UI niti šalje asinhroni zahtev za utišavanje zvuka videa na <i>mPlayer</i> objektu.
<code>public String unitySeekBarUpdateCallback()</code>	Na UI niti šalje asinhroni zahtev za trenutno vreme dokle je video stigao sa reprodukcijom <i>mPlayer</i> objektu.
<code>public void unityPlaybackUpdateProgress(String time)</code>	Na UI niti šalje asinhroni zahtev za premotavanje videa <i>mPlayer</i> objektu.
<code>@Override public void onFrameAvailable(SurfaceTexture surfaceTexture)</code>	Događaj koji okida površinska tekstura svaki put kada je novi bafer spreman da se potroši.

Tabela 9 Metode *PlayerWrapper* klase.

4.2.2 Realizacija *PlayerListener* klase

Klasa *PlayerListener* služi za analitičke događaje koji se okidaju unutar jezgra plejera. Izvodi to tako što nasleđuje *IAalytics_listener* klasu iz video plejer Android biblioteke. Sa *PlayerListener* klasom je moguće saznati:

- Trenutak do kog je video stigao u reprodukciji;
- Kada se prvi okvir slike dekodirao;
- Informaciju o trenutnom stanju (reprodukuje, pauziran, stopiran itd.)
- Video kodeku koji se koristi za dekodiranje videa;
- Rezoluciju videa;

- Kraj videa;
- Promene u procenama propusnog opsega koje daju informacije o kvalitetu veze ako se video dobavlja preko interneta;
- Pod uslovom da je GPU na uređaju preopterećen, naglašava koji se okviri slike nisu mogli iscrtati;

Polje	Opis
<code>private String TAG</code>	Koristi se za logovanje događaja i otklanjanje greški.
<code>private String listeningObject</code>	Ime objekta unutar Unity scene koji sluša događaje.

Tabela 10 Polja *PlayerListener* klase.

Sve metode *PlayerListener* klase koriste *UnityMessenger* klasu za slanje poruka *PlayerInteractor* skripti. U nastavku su navedene te metode:

Metoda	Opis
<code>public PlayerListener(String gameObjectName)</code>	Konstruktor koji postavlja ime C# skripte kojoj treba slati poruke.
<code>public void on_state_changed(Player_state old_state, Player_state new_state)</code>	Šalje informacije o promeni stanja video plejera (pokrenut, pauziran itd.) C# skripti.
<code>public void on_bandwidth_estimation_changed(int bps)</code>	Procenjuje trenutni kvalitet veze sa internetom.
<code>public void on_low_bandwidth()</code>	Događaj koji obaveštava da je kvalitet veze sa internetom loš.
<code>public void on_input_format_changed(Media_format format)</code>	Kada medijski kodek interno parsira koji je format videa prosleđen plejeru, ova metoda šalje informacije C# skript o formatu, veličini, brzini protoka, rezoluciji i kodeku videa.
<code>public void on_frames_dropped(int num_frames)</code>	Kada je uređaj preopterećen i ne može da postigne da iscrtava svaki okvir slike, ova metoda obaveštava C# skriptu da su neki okviri odbačeni.
<code>public void on_output_format_changed(Media_format format)</code>	Obaveštava da li se dogodila promena unutar plejera prelaza sa video na audio format ili obrnuto.
<code>public void on_first_frame_rendered(int track_id)</code>	Obaveštava da se prvi okvir slike iscrtao na ekran.
<code>public void on_end_of_stream(IPlayer_listener.Callback_data data)</code>	Obaveštava da je došao kraj video toka.
<code>public void on_stall_start(IPlayer_listener.Stall_start_data stall_start)</code>	Obaveštava da postoji kašnjenje u početku reprodukcije i navodi razlog kašnjenja.

<pre>public void on_stall_stop(IPlayer_listener.Callback_data stall_stop)</pre>	Obaveštava da postoji kašnjenje u zaustavljanju reprodukcije i navodi razlog kašnjenja.
---	---

Tabela 11 Metode *PlayerListener* klase.

4.2.3 Realizacija *UnityMessenger* klase

Klasa *UnityMessenger* služi za slanje poruka iz Java klase u C# klase. Kada *PlayerInteractor* pozove metodu *CreatePlayer* ako je iskaz *Analytics* tačan to će proslediti novi *PlayerListener* asinhronom konstruktoru plejera. Svaki događaj o kome treba obavestiti C# klase se sprovodi preko *UnityMessenger* klase. Unity Java biblioteka poseduje statičku metodu unutar *UnityPlayer* klase koja omogućuje ovaj mehanizam:

```
public static void UnitySendMessage(String var0, String var1, String var2)
```

- Var0 je ime objekta u sceni kome je poruka namenjena;
- Var1 je ime metode koja očekuje poruku za slanje;
- Var2 je poruka koja se šalje.

Primetno je ograničenje da se preko ovog mehanizma šalju isključivo nizovi karaktera. Argument *var1* se ne koristi, zato što u programskom rešenju postoji samo jedna metoda za parsiranje poruka iz Androida. Poruke se parsiraju tako što se podele na komponentu ili metodu koja treba da reaguje i vrednost. U programskom rešenju, parsiranje obavlja isključivo *PlayerInteractor.ParseAndroidMessage* metoda. Format poruka koje *UnityMessenger* prosleđuje je:

IME_KOMPONENTE:VREDNOST

Sledeća tabela pokazuje koje metode *UnityMessenger* poseduje za slanje poruka:

Metoda	Opis
<pre>public static void sendMessage(String gameObjectName, String message)</pre>	Šalje jednu formatiranu poruku <i>ParseAndroidMessage</i> metodi.
<pre>public static void sendCompiledMessage(String gameObjectName, ArrayList<String> messages)</pre>	Šalje više formatiranih poruka <i>ParseAndroidMessage</i> metodi.

Tabela 12 Metode *UnityMessenger* klase.

4.3 Realizacija OpenGL ES biblioteke za iscrtavanje videa

Namena ove programske celine je da generiše dinamičku biblioteku *cppgraphics.so*. Ovo čini upotrebom *CMake* alata. Biblioteka za grafičko iscrtavanje je napisana u C++ jeziku i sastoji se iz sledećih modula:

Moduli	Datoteke izvornog koda
Dodaci za iscrtavanje videa	<code>RenderingPlugin.cpp</code> <code>RenderingPlugin.h</code>
Pomoćni funkcije za otklanjanje GL greški	<code>GLHelpers.cpp</code> <code>GLHelpers.h</code>
Pomoćni modul za učitavanje i upotrebu šejdera	<code>ShaderHelper.cpp</code> <code>ShaderHelper.h</code>

Tabela 13 Datoteke unutar C++ grafičke biblioteke.

4.3.1 Realizacija `RenderingPlugin` modula

Datoteka *RenderingPlugin* je glavni deo *cppgraphics* biblioteke. Podeljena je u nekoliko delova po funkcionalnostima koje pruža:

- Funkcije koje se povezuju u Unity pogon;
- Funkcije koje se povezuju preko JNI sa Java klasama;
- Spoljne funkcije koje se dinamički uvezuju u C# klase;
- Interne funkcije koje obavljaju OpenGL iscrtavanje i potrošnju grafičkih bafera.

Glavna struktura podataka koju ovaj modul koristi je *JNIRendering*. *JNIRendering* struktura u sebi sadrži globalne reference na JNI objekte koji su neophodni za iscrtavanje videa. Sledeća tabela navodi sva polja *JNIRendering* strukture:

Polje	Opis
<code>int isReady</code>	Nakon što je inicijalizacija svih drugih polja unutar strukture uspešna, postavlja se na 1.
<code>jobject gCallbackObject</code>	Globalna referenca na <i>PlayerWrapper</i> objekat koji je napravio zahtev za inicijalizaciju grafičkih resursa.
<code>jmethodID surfaceTextureUpdateTexImageMethodId</code>	Referenca na metodu za potrošnju bafera generisanih od video plejera.
<code>jobject surfaceTexture</code>	Globalna referenca na površinsku teksturu.
<code>GLuint surfaceTexId</code>	Identifikator OpenGL spoljne teksture.

Tabela 14 Polja *JNIRendering* strukture.

Osim navedenih funkcionalnosti *RenderingPlugin* modul poseduje i globalne promenljive koje su dostupne svim funkcijama unutar modula:

Promenljiva	Opis
<code>static JVM* gJVM</code>	Pokazivač na JVM.
<code>GLuint VAO, VBO, IBO</code>	Identifikatori OpenGL promenljivih za iscrtavanje ekrana.
<code>ShaderHelper* PlayerShader</code>	Objekat pomoćne klase za primenu šejdera u nativnom kodu.
<code>static std::vector<JNIRendering*> renderer</code>	Vektor koji sadrži u sebi sve instance plejera za koje treba da se iscrta video.
<code>static IUnityInterfaces* s_UnityInterfaces</code>	Početna tačka integracije sa Unity pogonom. Omogućuje dalji pristup Unity nativnom API.
<code>static IUnityGraphics* s_Graphics</code>	Pruža generičke grafičke funkcionalnosti biblioteci.
<code>static UnityGfxRenderer s_DeviceType</code>	Tip grafičke biblioteke koji aplikacija koristi za iscrtavanje na ekranu.

Tabela 15 Globalne promenljive unutar *RenderingPlugin* datoteke.

4.3.1.1 Realizacija funkcija koje se povezuju u Unity pogon

Unutar *cppgraphics* biblioteke je uključena biblioteka Unity pogona koja pruža mogućnosti za njegovo proširenje. Funkcije koje proširuju Unity pogon moraju u svojoj definiciji imati *UNITY_INTERFACE_API* i *UNITY_INTERFACE_EXPORT* makroe ispred imena funkcije. Ovakav makro je specifikacija za dinamičko povezivanje koji se koristi da naznači kompajleru sa kojom deklaracijom funkcije iz Unity biblioteke za proširivanje pogona da veže definiciju iz druge biblioteke (u ovom slučaju *cppgraphics*). Ovi makroi su definisani ovako:

```
#define UNITY_INTERFACE_API
#define UNITY_INTERFACE_EXPORT __attribute__((visibility("default")))
```

Još jedna bitna stavka kod dinamičkog vezivanja sa spoljnim funkcijama je da Unity pogon očekuje da sve funkcije koje ga proširuju koriste vezivanje po C standardu. C++ funkcijama se imena pokvare kada se kompajliraju zato što je moguće imati više funkcija istog imena sa različitim argumentima, dok u C jeziku ovo nije moguće. Vezivanje po C standardu u C++ jeziku je moguće tako što se posle *extern* ključne reči za spoljne funkcije doda "C" iskaz.

Funkcija	Opis
extern "C" void UNITY_INTERFACE_EXPORT UNITY_INTERFACE_API UnityPluginLoad(IUnityInterfaces* unityInterfaces)	Kada Unity pogon završi sa podešavanjem grafičkog API-a za njegov rad, obaveštava <i>cppgraphics</i> biblioteku.
extern "C" void UNITY_INTERFACE_EXPORT UNITY_INTERFACE_API UnityPluginUnload()	Kada Unity pogon završi sa radom obaveštava <i>cppgraphics</i> biblioteku da se gasi.
static void UNITY_INTERFACE_API OnGraphicsDeviceEvent(UnityGfxDeviceEventType eventType)	Unity jednom pred početak iscrtavanja okida ovaj događaj kako bi omogućio <i>cppgraphics</i> biblioteci da zauzme resurse za iscrtavanje videa na niti za iscrtavanje. Ovde se inicijalizuju VAO, VBO, IBO i <i>ShaderHelper</i> .

Tabela 16 Funkcije koje se povezuju sa Unity pogonom.

4.3.1.2 Realizacija JNI funkcija RenderingPlugin modula

Funkcije koje su navedene ovde se putem JNI spajaju sa Java klasama. Naredna tabela navodi sve JNI funkcije koje *RenderingPlugin* datoteka poseduje:

Funkcija	Opis
jint JNI_OnLoad(JavaVM *vm, void*)	Učitava JVM u <i>cppgraphics</i> biblioteku.
extern "C" JNIEXPORT void JNICALL PlayerWrapper.nativeInit (JNIEnv *env, jobject obj)	Kreira novi <i>JNIRendering</i> objekat za grafičko iscrtavanje.
extern "C" JNIEXPORT void JNICALL unitypluginMainActivity.nativeRelease (JNIEnv *env, jobject obj)	Oslobađa sve resurse <i>cppgraphics</i> biblioteke po završetku aplikacije.
extern "C" JNIEXPORT jobject JNICALL PlayerWrapper.createNativeSurfaceTexture (JNIEnv *env, jobject obj)	Kreira spoljnu teksturu i površinsku teksturu. Povratna vrednost je površinska tekstura.

Tabela 17 JNI metode unutar *RenderingPlugin* datoteke²⁶.

4.3.1.3 Funkcije koje se dinamički uvezuju u NativeTexturePlugin C# klasu

Kao što je navedeno u teorijskim osnovama, metode unutar C# klase koje se dinamički vezuju za C++ funkcije poseduju `[DllImport("cppgraphics")]` atribut ispred svog imena. Bitno

²⁶ Ovo nisu puna imena JNI funkcija, skraćena su radi preglednosti tabele.

je da metoda u C# klasi i C++ funkcija imaju isto ime, inače će u suprotnom vezivanje biti neuspešno. Zato se i ove funkcije vezuju po C standardu. Svaka od ovih funkcija je dostupna za poziv unutar *NativeTexturePlugin* klase.

Funkcija	Opis
<code>extern "C" long UNITY_INTERFACE_EXPORT UNITY_INTERFACE_API GetNativeSt()</code>	Vraća trenutnu veličinu <i>renderer</i> vektora. Služi za sinhronizaciju komandnog bafera sa grafičkom bibliotekom.
<code>extern "C" UnityRenderingEvent UNITY_INTERFACE_EXPORT UNITY_INTERFACE_API GetRenderEventFunc()</code>	Komandni bafer veže na sebe ovu funkciju. Njena povratna vrednost je funkcija koju komandni bafer poziva za iscrtavanje videa na ekranu u sceni.
<code>static void UNITY_INTERFACE_API OnRenderEvent(int eventID)</code>	Funkcija koja se poziva na niti za iscrtavanje u komandnom baferu za iscrtavanje videa.
<code>extern "C" void UNITY_INTERFACE_EXPORT UNITY_INTERFACE_API ReleasePlayersInScene()</code>	Poziva se kada korisnik napušta scenu u kojoj su bili pokrenuti plejeri. Oslobađa native resurse.

Tabela 18 Funkcije koje se dinamički uvezuju sa *NativeTexturePlugin* klasom.

4.3.1.4 Realizacija internih funkcija

Unutar *RenderingPlugin* datoteke, interne funkcije su zadužene za OpenGL ES grafičke pozive. Navode se u sledećoj tabeli:

Funkcija	Opis
<code>static void SetupResources();</code>	Unutar <i>OnGraphicsDeviceEvent</i> funkcije se poziva kako bi napravila sve globalne OpenGL promenljive za iscrtavanje ekrana.
<code>static void UpdateTexture(int index)</code>	Služi za potrošnju bafera okvira slike unutar komandnog bafera.
<code>static void DrawQuad(int id)</code>	Iscrtava ekran unutar komandnog bafera.
<code>static void CreateSurfaceTextureNative(JNIRendering* rend, JNIEnv* myEnv)</code>	Stvara globalnu referencu na površinsku teksturu unutar JNI.

Tabela 19 Interne funkcije *RenderingPlugin* datoteke.

4.3.2 Realizacija GLHelpers modula

Namena ovog modula je da uključi OpenGL ES zaglavlja u projekat, sadrži u sebi funkcije za proveravanje greški kada se pozovu GL funkcije. U *GLHelpers* modulu se nalazi izvorni kod šejdera koji se koriste za iscertavanje videa. Za početak, *GLHelpers* datoteka poseduje u sebi makro definicije za ispisivanje logova u ADB:

```
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR, LOG_TAG, __VA_ARGS__)
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG, LOG_TAG, __VA_ARGS__)
#define ALOGV(...) __android_log_print(ANDROID_LOG_VERBOSE, LOG_TAG, __VA_ARGS__ )
```

Na ovaj način je moguće ispisati poruke na respektivnim nivoima:

- Greške ovo su poruke visokog prioriteta;
- Ispisa u režimu ispitivanja (engl. *Debug*) u cilju otklanjanja greški;
- Režim proširenog ispisa koji je koristan kada korisnik želi da vidi više detalja o radu programa, ovo je poruka najnižeg prioriteta;

Ovaj modul u sebi sadrži izvorni kod šejdera za reprodukciju videa. Šejder kod je u ovom radu definisan kao niz karaktera u *GLHelpers.cpp* datoteci, radi lakšeg učitavanja šejder koda u grafičko okruženje. U suprotnom bi morao da se skladišti u fajl. Globalne promenljive koje *GLHelpers* modul poseduje su:

Promenljiva	Opis
<code>const char* gVertexCodeDebug</code>	Šejder kod temena za otklanjanje greški ²⁷ .
<code>const char* gFragmentCodeDebug</code>	Šejder kod fragmenta za otklanjanje greški.
<code>const char* gVertexCodeVideo</code>	Šejder kod temena za iscertavanje videa.
<code>const char* gFragmentCodeVideo</code>	Šejder kod fragmenta za iscertavanje videa.
<code>const char* gFragmentCodeVideoGammaApproximated</code>	Šejder kod fragmenta za iscertavanje videa sa gama aproksimacijom.

Tabela 20 Promenljive *GLHelpers* modula.

Funkcije za ispisivanje greški imaju tri namene: ispis greške u OpenGL pozivu, ispis greške u podešavanju EGL grafičkog interfejsa i ispis greške u sintaksi šejder programa. U ove svrhe su definisane sledeće funkcije:

Funkcija	Opis
<code>#define GL(func)</code>	Makro izraz koji izvršava OpenGL funkciju i

²⁷ Koristan za upotrebu sa *RenderDoc* alatom za snimanje okvira slike.

<code>func; GLCheckErrors(__FILE__, __LINE__);</code>	odmah zatim proverava da li je došlo do greške.
<code>static const char *EglErrorString(const EGLint error)</code>	Parsira status EGL konteksta i ispisuje na osnovu broja statusa detaljniji opis greške.
<code>static const char * GLCheckErrors(const char *file, int line);</code>	Poziva <i>glGetError</i> funkciju i parsira njen odgovor kako bi detaljnije greška bila opisana.
<code>static const char * GLErrorString(GLenum error)</code>	Ispisuje značenje svih značajnih greški na osnovu njihovog identifikatora.

Tabela 21 Funkcije *GLHelpers* modula.

4.3.3 Realizacija ShaderHelper klase

Klasa *ShaderHelper* je namenjena da olakša upotrebu šejdera u nativnom kodu. Koristi OpenGL da kompajlira šejder iz oblika niza karaktera u binarni oblik i učitava ga u OpenGL ES drajver. Preko nje se koristi šejder program kada se u komandnom baferu iscrtava na ekran u prostoru. *ShaderHelper* objektu se poziva konstruktor iz *OnGraphicsDeviceEvent* funkcije zbog toga što se šejder kompajlira i učitava u OpenGL ES drajver na niti za iscrtavanje. Postoji jedna globalna promenljiva šejdera u *RenderingPlugin* modulu za iscrtavanja svih videa. Sledeće tabele opisuju polja i metode koje ova klasa poseduje:

Polje	Opis
<code>GLuint programID</code>	Identifikator šejder programa za iscrtavanje.
<code>GLuint uniformMVP</code>	Identifikator matrice proizvoda matrica modela, prikaza i projekcije. Koristi se ako se objekat pozicionira u 3D prostoru ručno.

Tabela 22 Polja *ShaderHelper* klase.

Metoda	Opis
<code>ShaderHelper()</code>	Konstruktor postavlja <i>programID</i> i <i>uniformMVP</i> na nulu.
<code>~ShaderHelper()</code>	Destruktor koji čisti šejder program iz GPU memorije.
<code>static ShaderHelper* CreateFromString(const char* vertexCode, const char* fragmentCode)</code>	Poziva konstruktor za <i>ShaderHelper</i> i učitava u njega šejder program na osnovu argumenta.
<code>void UseShader()</code>	Pri iscrtavanju ova metoda govori OpenGL-u da koristi šejder <i>programID</i> program.
<code>void ClearShader()</code>	Briše šejder program iz GPU memorije.

Tabela 23 Javne metode *ShaderHelper* klase.

Metoda	Opis
<code>GLuint loadShader(const char* code, GLenum shaderType)</code>	Kompajlira šejder temena ili šejder fragmenta.
<code>void loadProgram(const char* vsCode, const char* fsCode)</code>	Poziva prvo <i>loadShader</i> kako bi kompajlirao šejdere, zatim vezuje zajedno u jedan program šejder temena i fragmenta. Izlaz se smešta u <i>programID</i> .

Tabela 24 Privatne metode *ShaderHelper* klase.

5. Evaluacija

Uređaji na kojima će se izvesti ispitivanje su Android tipa sa ARM64 procesorskom arhitekturom. Svi ispitni slučajevi su sprovedeni sa istim Android Studio i Unity projektom. Ispitivanje nastoji da pokaže:

- Svaki komandni bafer koji iscrtava svaku instancu ekrana u sceni;
- Ispis na terminali koji verifikuje tok izvršavanja aplikacije koji se nalazi u konceptu rešenja;
- Upravljanje grafičkim i Android resursima unutar Unity pogona. Fokus je na zauzimanju resursa putem korisničkih interakcija i oslobađanju resursa kada se scena ili aplikacija gasi.
- Gama aproksimacije boja u linearnom prostoru.

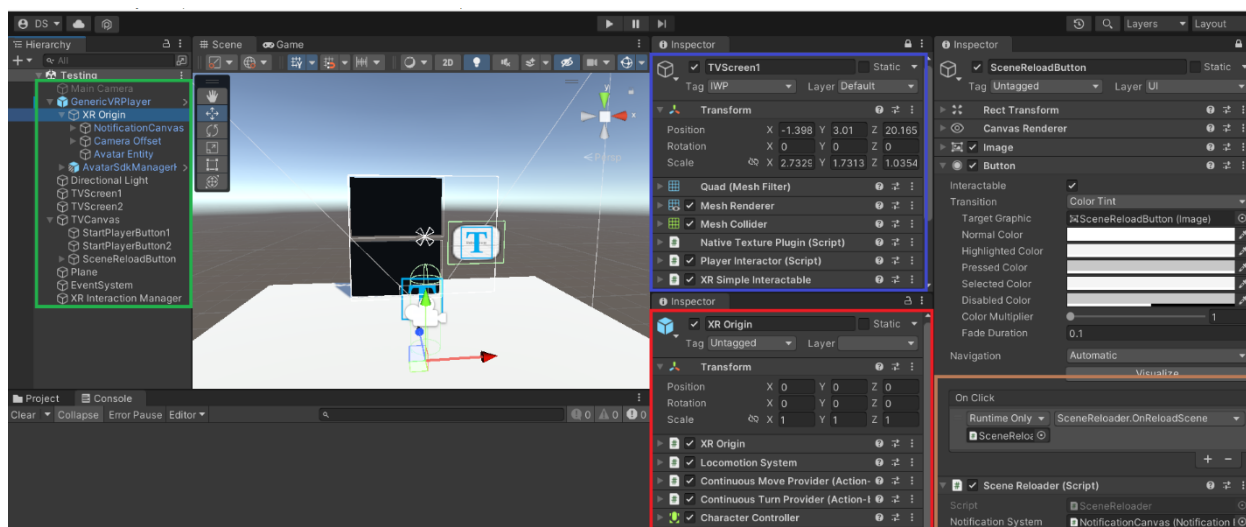
Android uređaji koji će se koristiti u ispitivanju su: *Oculus Quest 2*, *Poco X3 NFC* i *Samsung S22* na koji su povezane *Nreal AIR* AR naočare.

5.1 Opis okruženja za ispitivanje

Ispitivanje ispravnosti programskog rešenja se bazira na *Testing* Unity sceni. Ova scena je konfigurisana tako da se koristi za sve uređaje na kojima će se testirati programsko rešenje uz minimalne promene. Napravljena je sa jednostavnošću na umu kako bi fokus bio na testiranju reprodukcije videa u 3D prostoru.

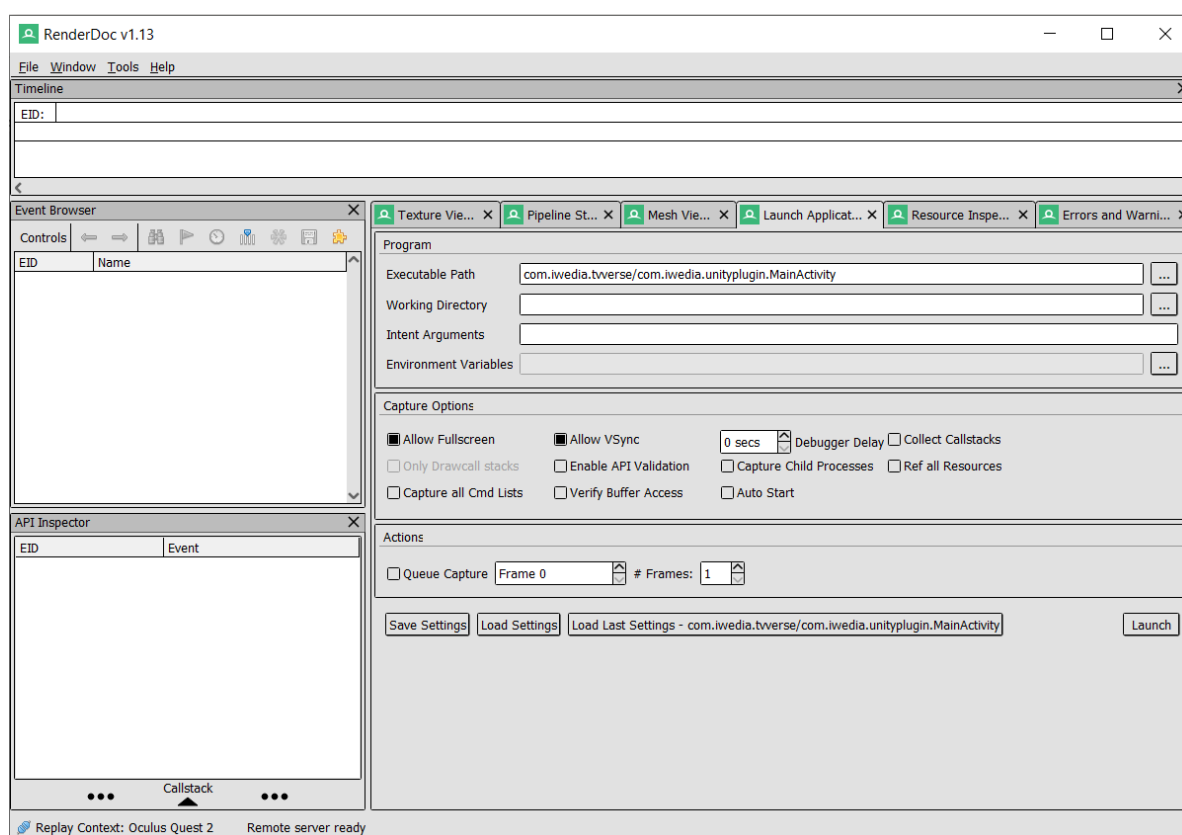
Na sledećoj slici je prikazano Unity radno okruženje. U njemu je otvorena *Testing* scena. Zelenim pravougaonikom je obeležena hijerarhija unutar scene, a jedini neaktivan element je *MainCamera*. Ova kamera je namenjena za testiranje na Android telefonima. U slučaju testa na Android telefonu, ova kamera se aktivira dok se *GenericVRPlayer* objekat gasi zato što je on namenjen za VR. Plavom bojom su prikazane sve komponente jednog ekrana. Plavim pravougaonikom je obeležen jedan ekran za reprodukciju i sve komponente na njemu. Crvenim

pravougaonikom je obeležen glavni objekat XR aplikacija – *XROrigin* i njegove komponente. Braon bojom je obeležen događaj pritiska dugmeta i skripta za osvežavanje scene.



Slika 24 Prikaz Unity scene za ispitivanje rešenja.

Prikaz ispravnosti rešenja iscrtavanja sa komandnim baferima je baziran na RenderDoc alatu. Uz pomoć RenderDoc-a će biti uhvaćeni okviri iscrtavanja i biti analizirani na više Android platforma²⁸. Ispitaće se da li je rešenje uspešno u oba režima iscrtavanja, sa jednim prolazom i višestrukim iscrtavanjem.

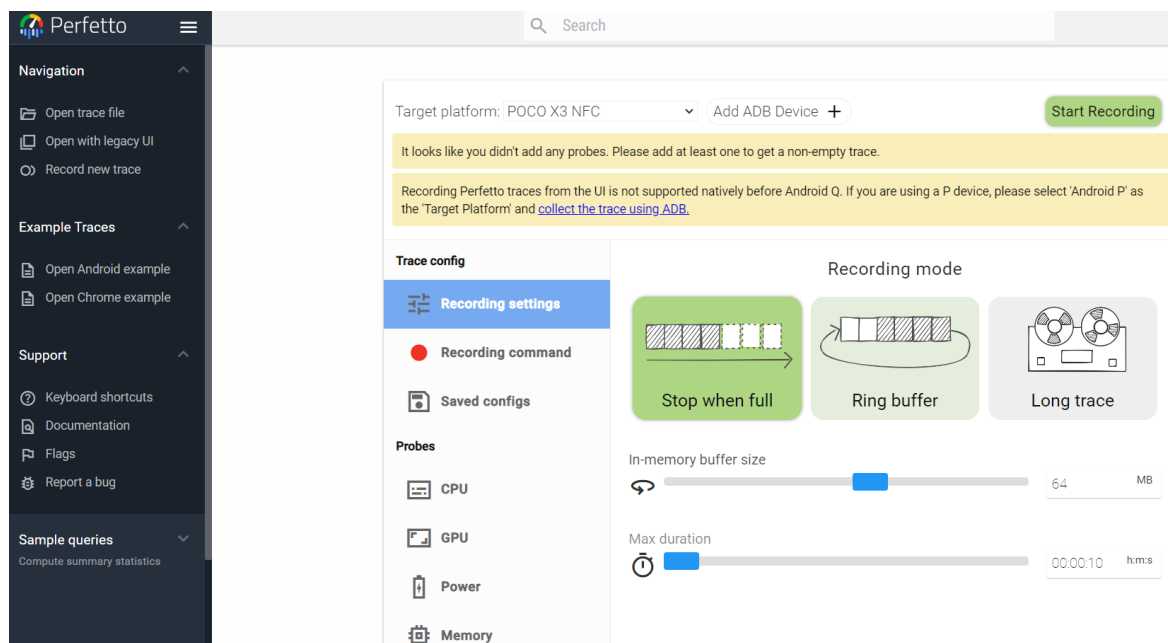


Slika 25 Prozor *RenderDoc* alata kada je zakačen na *Oculus Quest 2*.

²⁸ https://renderdoc.org/docs/how/how_android_capture.html

Ispitaće se upravljanje resursima video plejera na svakom sloju arhitekture tako što će se scena sa jednim ili više video ekrana učitati više puta. Rezultati ovog ispitivanja će se biti ispis na uređaju dobavljen sa ADB komandom.

Upoređiće se kvalitet slike sa i bez aproksimiranih vrednosti za gama prostor u linearnom. Uz pomoć alata za profilisanje Android aplikacija pokazaće se gornje granice mogućnosti rešenja, tj. koliko ekrana je moguće prikazati na uređaju istovremeno pre nego što brzina smene okvira unutar aplikacije krene drastično da opada.



Slika 26 *Perfetto* alat za analizu Android/Linux aplikacija.

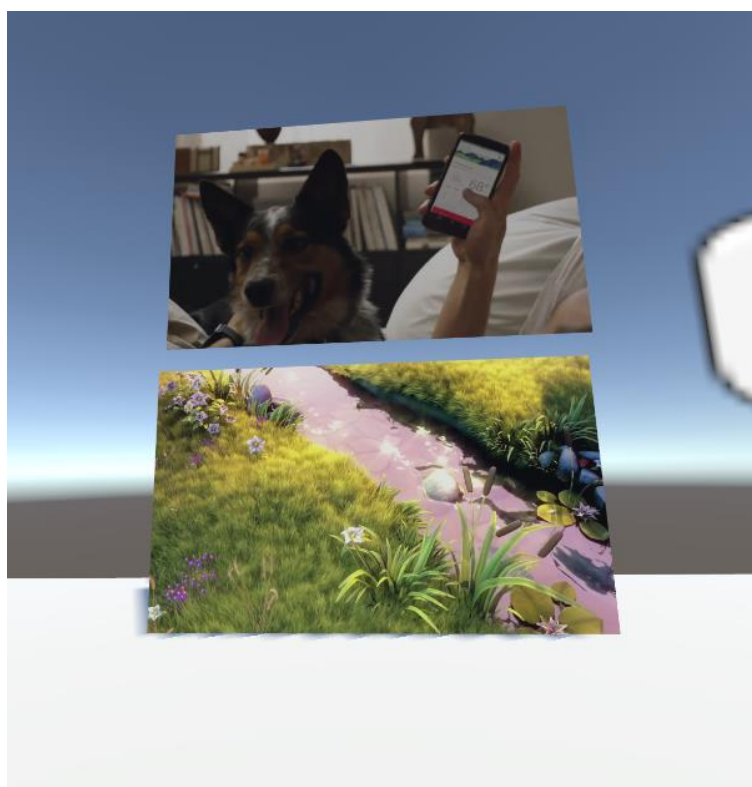
Za kraj će se diskutovati o rezultatima evaluacije.

5.2 Poređenje kvaliteta slike sa i bez gama aproksimacije

Naredne dve slike su sa i bez gama aproksimacije. U ostatku ispitivanja se koristio šejder sa aproksimacijom.



Slika 27 Video iscrtan šejderom bez gama aproksimacije.



Slika 28 Video iscrtan šejderom sa gama aproksimacijom.

5.3 Ispitivanje komandnih bafera

Ispitivanje je vršeno tako što je aplikacija sa podešeni režimom za iscertavanje (single/multi pass) instalirana na ciljanu platformu. Zatim je *RenderDoc* vezan za uređaj.

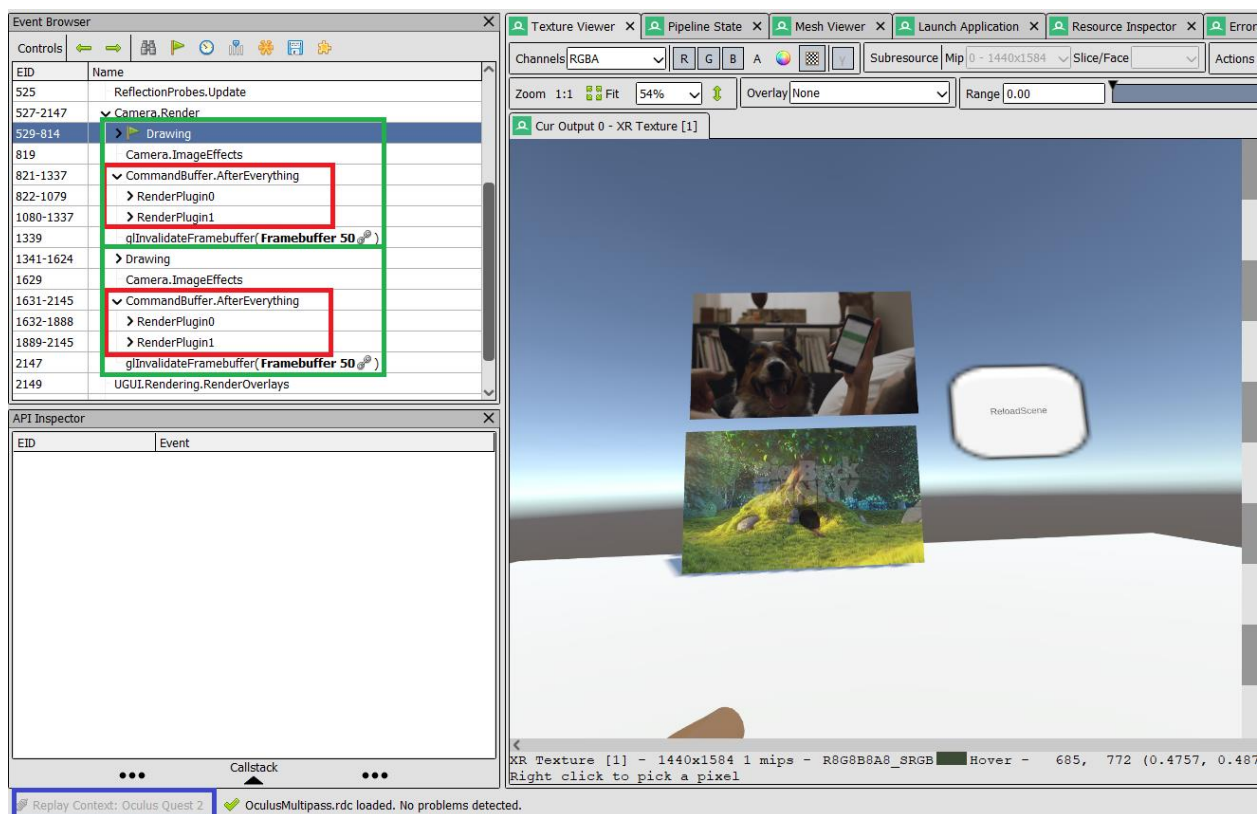
Verifikacija da je *RenderDoc* uspešno zakačen se može videti na uređaju tako što se automatski na uređaju pokrene *RenderDocCMD* aplikacija. Iz *RenderDoc* alata se pokreće aplikacija na ciljanom uređaju, dok je *RenderDocCMD* aplikacija pokrenuta. Nakon ovoga je moguće uhvatiti okvir slike iscrtavanja u proizvoljnom radu aplikacije. Ako su komandi baferi adekvatno registrovani unutar grafičkog toka, oni će se pojaviti u *EventBrowser* prozoru. Prozor za prikaz tekstura će pokazati video i ostatak scene ako je adekvatni događaj unutar *EventBrowser* prozora odabran.

Slike koje slede imaju obojene pravougaonike na sebi koji označavaju:

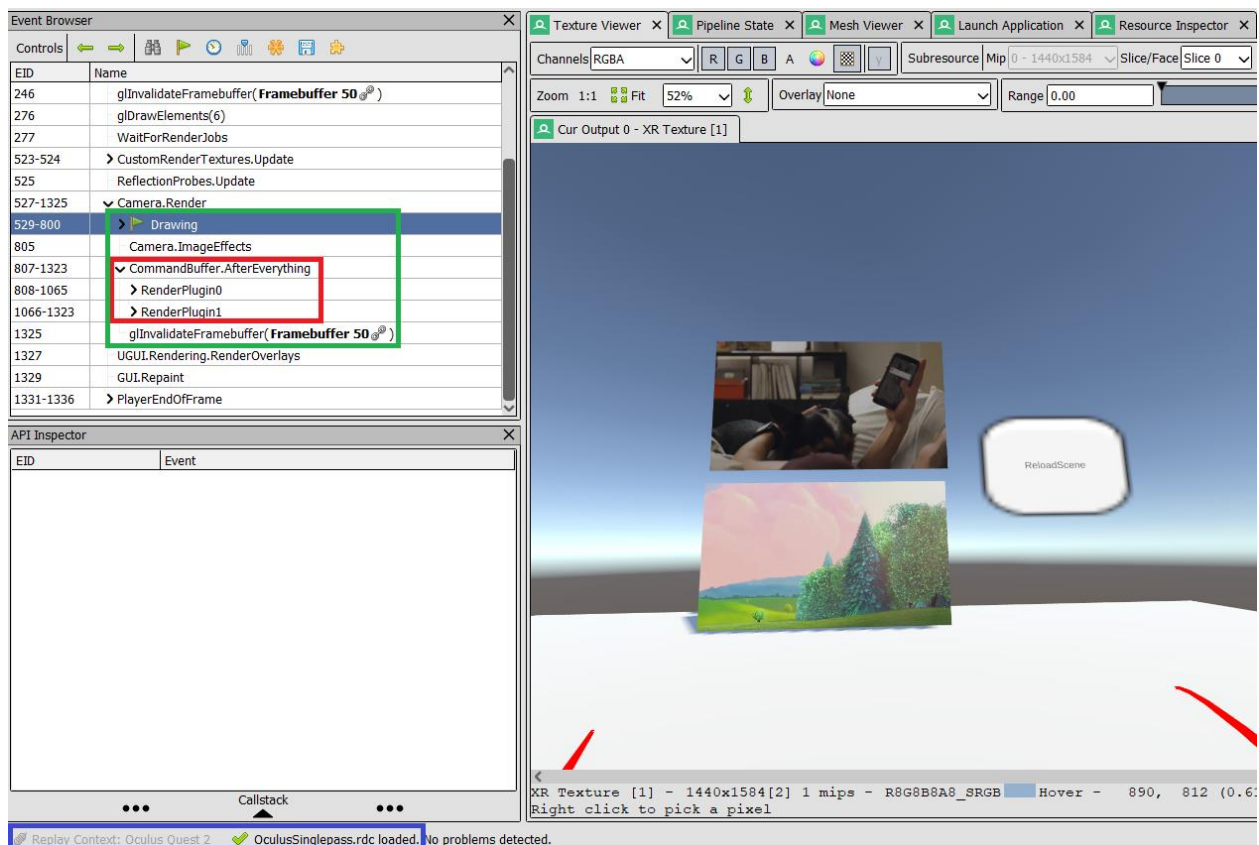
- Crvenom pravougaonikom su naznačeni komandni baferi;
- Plavim pravougaonikom je naznačena platforma na kojoj se testira i režim iscrtavanja;
- Zelenim pravougaonikom su naznačeni grafički događaji iscrtavanja.

5.3.1 Ispitivanje komandnih bafera na *Oculus Quest 2*

Naredne slike su rezultati zakačenog okvira slike unutar *RenderDoc* okruženja za *Oculus Quest 2* VR uređaju.



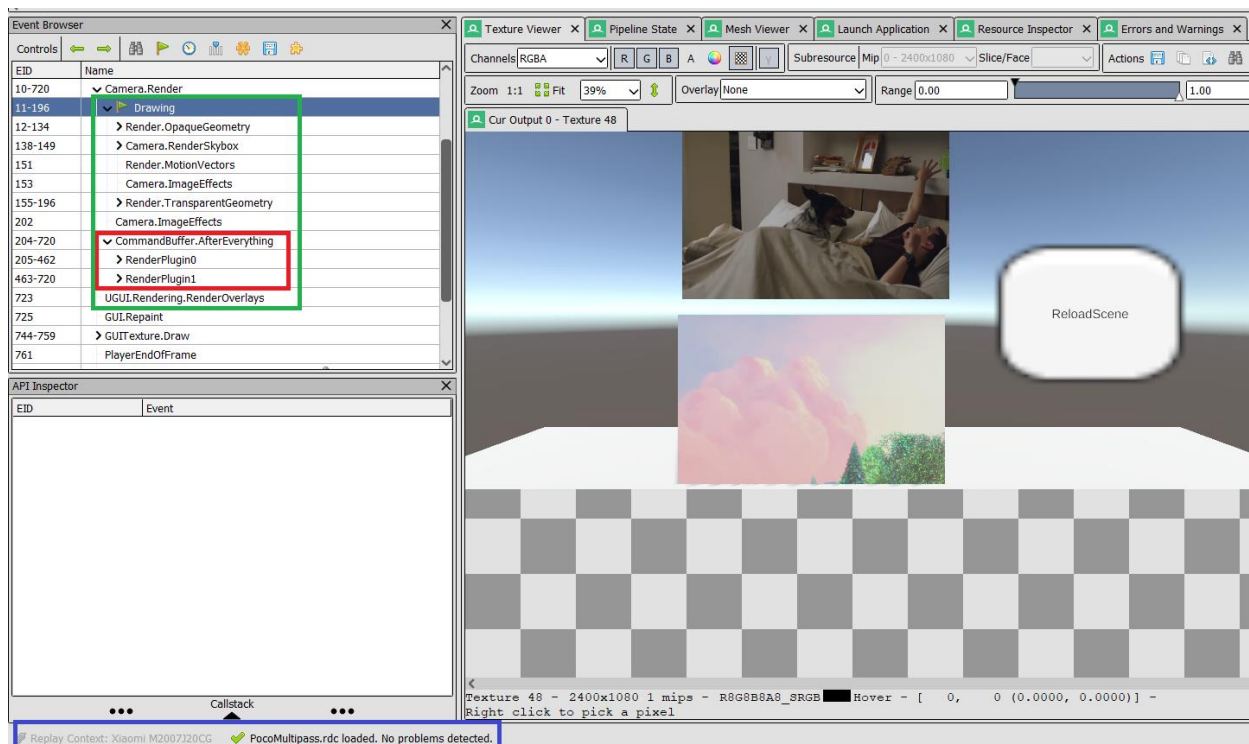
Slika 29 Okvir slike uhvaćen RenderDoc alatom na Oculus Quest 2 u višestrukom režimu.



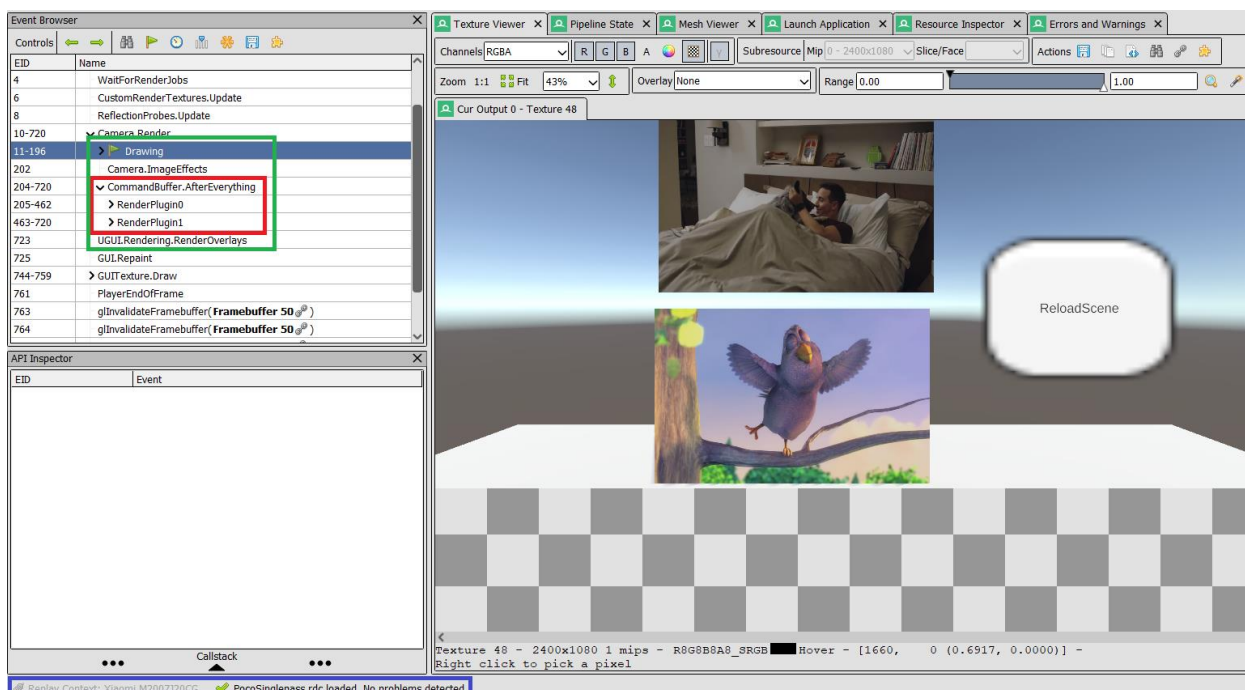
Slika 30 Okvir slike uhvaćen RenderDoc alatom na Oculus Quest 2 u jednoprolaznom režimu.

5.3.2 Ispitivanje komandnih bafera na Poco X3 NFC telefonu

Naredne slike su rezultati zakaćenog okvira slike unutar *RenderDoc* okruženja za *Poco X3 NFC* telefonu.



Slika 31 Okvir slike uhvaćen RenderDoc alatom na Poco X3 NFC u višestrukom režimu.



Slika 32 Okvir slike uhvaćen RenderDoc alatom na Poco X3 NFC u jednoprolaznom režimu.

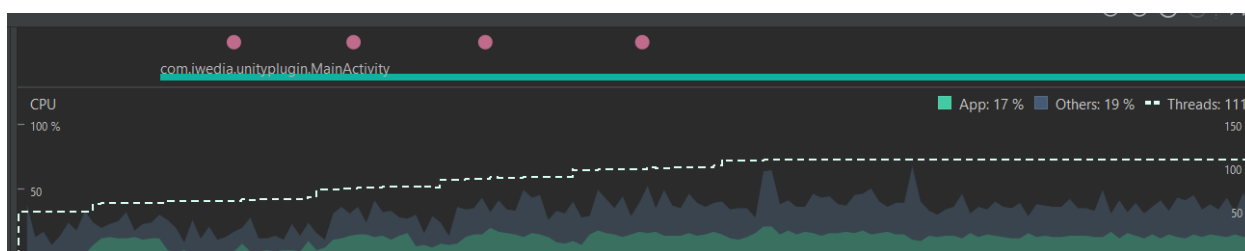
5.3.3 Ispitivanje komandnih bafera na NReal Air naočarima

Nažalost, dostavljanje okvira slike uhvaćenog sa *RenderDoc* alatom nije bilo moguće. Za hvatanje okvira slike namjera je bila da se upotrebi *Samsung Galaxy S22* Android telefon koji sa kojim *RenderDoc* nije mogao da uspostavi bežičnu vezu. Pošto se *Nreal Air* AR naočare vežu za telefon putem USB-C kabla, nije postojao port za povezivanje kablom telefona sa *RenderDoc* alatom na računaru.

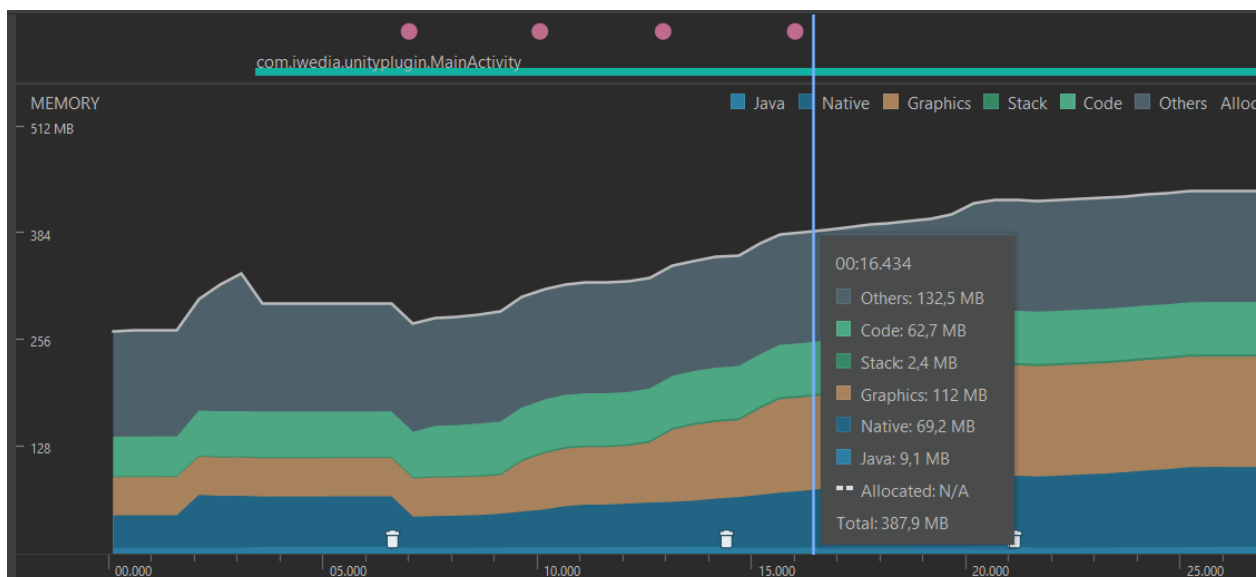
AR aplikacije se pokreću iz *Nebula NReal* softvera. Ovaj softver ograničava mogućnosti slikanja ekrana koji se prikazuje na naočarima. Aplikacija nije imala dozvole za slikanje ekrana tako da će u diskusiji rezultata biti spomenuto samo kako se iscrtavanje videa ponašalo u *Nreal Air* aplikaciji.

5.4 Ispitivanje performansi Android aplikacije

Na sledećim slikama su prikazani grafici reprodukcije 4 videa istovremeno. Crvene tačke su trenuci u kojima se novi video plejer instancira, ubrzo nakon toga započinje iscrtavanje videa.



Slika 33 Grafik CPU performansi Android aplikacije sa četiri video ekrana na Poco X3 NFC.



Slika 34 Grafik memorijskih performansi Android aplikacije sa četiri video ekrana na Poco X3 NFC.

5.5 Ispitivanje toka izvršavanja i upravljanja resursima

U ovom ispitnom slučaju je pokrenuta aplikacija u cilju analize ispisa na uređaju upotrebom ADB alata. Slučaj koji se ispituje je: pokretanje aplikacije, aktiviranje prvog ekrana, aktiviranje drugog ekrana, osvežavanje scene, paljenje proizvoljnog ekrana i gašenje aplikacije. ADB komanda koja se koristila za filtriranje²⁹ poruka je:

```
adb logcat -c && adb logcat | Tee-Object -FilePath ~\log.txt | Select-String
-Pattern "SceneReloaderV", "RENDERING_PLUGIN", "PlayerInteractorV", "MainActivityV"
```

```
MainActivityV: onCreate done
Unity   : PlayerInteractorV: Screen TVScreen1 selected.
RENDERING_PLUGIN: ##### New PlayerWrapper registered.
RENDERING_PLUGIN: GRAPHICS DEVICE EVENT
RENDERING_PLUGIN: GRAPHICS DEVICE EVENT INITIALIZE
RENDERING_PLUGIN: New programID 16
RENDERING_PLUGIN: Done VAO: 3 VBO 76
Unity   : RENDERING_PLUGIN: New command buffer registered. Player number 1
Unity   : PlayerInteractorV: Screen TVScreen2 selected.
RENDERING_PLUGIN: ##### New PlayerWrapper registered.
Unity   : RENDERING_PLUGIN: New command buffer registered. Player number 2
Unity   : SceneReloaderV: Reload scene now.
Unity   : RENDERING_PLUGIN: Clearing CommandBuffers.
Unity   : RENDERING_PLUGIN: Clearing CommandBuffers.
RENDERING_PLUGIN: ReleasePlayersInScene: 0
Unity   : SceneReloaderV: Scene reload complete.
```

²⁹ Ovaj izraz je namenjen za Powershell. Ispis je dodatno formatiran radi konciznosti.

```

RENDERING_PLUGIN: No videos attached to native plugin, aborting!
RENDERING_PLUGIN: No videos attached to native plugin, aborting!
Unity   : PlayerInteractorV: Screen TVScreen1 selected.
RENDERING_PLUGIN: ##### New PlayerWrapper registered.
Unity   : RENDERING_PLUGIN: New command buffer registered. Player number 1
Unity   : PlayerInteractorV: Screen TVScreen2 selected.
RENDERING_PLUGIN: ##### New PlayerWrapper registered.
Unity   : RENDERING_PLUGIN: New command buffer registered. Player number 2
Unity   : SceneReloaderV: Reload scene now.
Unity   : RENDERING_PLUGIN: Clearing CommandBuffers.
Unity   : RENDERING_PLUGIN: Clearing CommandBuffers.
RENDERING_PLUGIN: ReleasePlayersInScene: 0
Unity   : SceneReloaderV: Scene reload complete.
RENDERING_PLUGIN: No videos attached to native plugin, aborting!
RENDERING_PLUGIN: No videos attached to native plugin, aborting!
Unity   : PlayerInteractorV: Screen TVScreen1 selected.
RENDERING_PLUGIN: ##### New PlayerWrapper registered.
Unity   : RENDERING_PLUGIN: New command buffer registered. Player number 1
Unity   : SceneReloaderV: Reload scene now.
Unity   : RENDERING_PLUGIN: Clearing CommandBuffers.
RENDERING_PLUGIN: ReleasePlayersInScene: 0
Unity   : SceneReloaderV: Scene reload complete.
RENDERING_PLUGIN: No videos attached to native plugin, aborting!
MainActivityV: OnDestroy: finishing, releasing native global references.

```

5.6 Diskusija rezultata

Postignut je očekivani vizuelni utisak videa u XR aplikacijama. Interakcije sa plejerom unutar Unity pogona rade na zadovoljavajući način. Gama aproksimirane boje izgledaju verodostojnije originalnom snimku. Komandni baferi se pojavljuju na očekivanom mestu u grafičkoj procesnoj liniji. Pokazano je putem ADB da se tok života XR plejera ponaša na očekivan način unutar scene.

Rezultati profilisanja aplikacije na Poco X3 NFC telefonu su zadovoljavajući. Sa četiri video ekrana koji is crtavaju video istovremeno jasno se vidi da je CPU opterećenje malo (u vrhuncu oko 17% je odlazilo na procese unutar aplikacije). Radna memorija koju aplikacija zauzima nije premašila 512MB. GPU je na ovom uređaju integrisan u SOC.

Ispostavilo se da na Android telefonu višestruki režim ne postoji, time što su za jednoprolazni i višestruki uhvaćeni okviri slike identični. Kod poređenja režima is crtavanja na *Oculus Quest 2*, uređaju koji poseduje hardversku podršku za višestruki režim, očigledna je razlika u broju is crtavanja. Na *NReal Air* naočarima is crtavanje videa je radilo isključivo u

jednoprolaznom režimu. Ovaj rezultat ima smisla kad se nadoveže na nalaze sa Android telefona: oba režima iscertavanja izgledaju kao jednoprolazni, tako da ima smisla da *NReal Air* naočare podržavaju samo jednoprolazni režim iscertavanja³⁰.

³⁰ Svo grafičko iscertavanje se vrši na telefonu, naočare su tu samo kao periferni ekran za prikaz grafike u AR.

6. Zaključak

Zadatak rada je bio integrisati Android arhivu video plejera u XR aplikaciju. U poređnom analizom načina za izradu XR Android aplikacija izdvojio se Unity kao najbolji. Posедуje najveću zajednicu koja razvija XR aplikacije i mogući su raznih načini za implementaciju interaktivnosti u XR aplikaciji.

Interakcije unutar Unity scene su omogućene upotrebom OpenXR API za XR aplikacije. Upotrebom kompleta alata za XR, koji je u Unity radnom okruženju ugrađen paket, OpenXR interakcije su efikasno uključene u Unity scenu. Unity pogon je proširen sa Android arhivom koja je omotač oko video plejera. Arhiva u sebi sadrži C++ grafičku biblioteku i Java klase za upravljanje plejerom. Grafički tok Unity pogona je proširen komandnim baferima koji izvršavaju funkciju iz C++ grafičke biblioteke za iscrtavanje ekrana u prostoru koji reprodukuje video. Komandni baferi igraju ključnu ulogu u sistemu jer omogućuju da se video iscrta nezavisno od platforme i režima iscrtavanja, bio on jednoprolazni ili višestruki. Upravljanje resursima aplikacije je realizovano na svakom sloju rešenja C#, Java i C++. Svi slojevi rešenja su sinhronizovani i komuniciraju međusobno. Pronađeno je rešenje za ograničenja linearnog prostora boja, u video šejder program su uvedene aproksimacije vrednosti boja u gama prostoru.

Ispostavilo se da je rešenje vrlo prenosivo, radilo je na svim platformama na kojima je testirano. Pokazalo se da je moguće imati proizvoljan broj videa koji se reprodukuju istovremeno. Izbor Unity pogona kao razvojnog okruženja za XR aplikaciju je omogućilo korisniku da interaguje sa videom na razne načine.

Evaluacijom je pokazano da je koncept rešenja valjan i da dodatak za video reprodukciju u Unity pogonu radi stabilno. Kvalitet videa je zadovoljavajući i korisničke interakcije sa videom imaju brz odziv.

Problem koji ovde nije rešen je reprodukcija hardverski enkriptovanog videa (DRM). Buduća verzija Android arhive omotača može uključiti OpenXR biblioteke u grafičku biblioteku koje poseduju funkcije za iscrtavanje hardverski enkriptovanih tekstura.

Rešenje koje je izloženo ovde se snažno oslanja na komandne bafere. Igrom slučaja, oni su skupi za potrošnju i svaki dodatni koji proširuje grafički tok smanjuje performanse aplikacije. Jedno moguće unapređenje bi bilo ako bi se svi video ekrani iscrtavali preko jednog komandnog bafera.

Unitar Unity pogona SRP nije jedina grafička procesna linija koja postoji. Inače je za mobilne aplikacije preporučeno da se koristi URP grafička procesna linija zbog toga što je grafika lepša, a i aplikacija performantnija. URP nije korišćen u ovom radu zbog toga što ima potpuno drugačiji pristup za programiranje šejdera³¹. Nažalost koncept komandnog bafera nema direktnog rođaka u URP grafičkoj procesnoj liniji.

³¹ U URP grafičkoj procesnoj liniji, šejderi se programiraju preko blok dijagrama.

7. Literatura

- [1] Šićanin, P., Lazić, M.: *Virtuelna i proširena realnost: koncepti, tehnike i primene* Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2018
- [2] Jon Leech, OpenGL R[®] ES Version 3.1 (November 3, 2016)
https://registry.khronos.org/OpenGL/specs/es/3.1/es_spec_3.1.withchanges.pdf
- [3] The OpenXR™ Specification
<https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html>
- [4] Unity User Manual 2021.3(LTS)
<https://docs.unity3d.com/2021.3/Documentation/Manual/index.html>
- [5] Android open source project: <https://source.android.com/>
- [6] Dokumentacija RenderDoc alata: <https://renderdoc.org/docs/index.html>