# Protočni sustavi renderianja u programskom alatu Unity

**Geček, Fran**

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ORGANIZACIJE I INFORMATIKE**
**VARAŽDIN**

**Fran Geček**

# PROTOČNI SUSTAVI RENDERIANJA U PROGRAMSKOM ALATU UNITY

**ZAVRŠNI RAD**

**Varaždin, 2023.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Fran Geček**

**Matični broj: 0036519519**

**Studij: Informacijski i poslovni sustavi**

# PROTOČNI SUSTAVI RENDERIANJA U PROGRAMSKOM ALATU UNITY

**ZAVRŠNI RAD**

**:**

Doc. dr. sc. Mladen Konecki

**Varaždin, rujan 2023.**

*Fran Geček*

**Izjava o izvornosti**

Izjavljujem da je moj ZAVRŠNI RAD izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*prihvaćanjem odredbi u sustavu FOI-radovi*

---

**Sažetak**

U Unity game engineu postoje tri protočna sustava za renderiranje grafike. Ta tri protočna sustava za renderiranje grafike su: Built-in Rendering Pipeline, Universal Rendering Pipeline i High Definition Rendering Pipeline. Kako bi projekti razvoja video igara u Unityju bili uspješni, važno je upoznati mogućnosti svakog od ta tri protočna sustava za renderiranje grafike. Dodatni razlog za upoznavanje njihovih specifičnosti i tehničkih mogućnosti leži i u činjenici da je odabir sustava za renderiranje grafike nužno učiniti prije kreiranja novog projekta, odnosno u fazi planiranja izrade video igre. Cilj ovog rad je istražiti i opisati neke od osnovnih mogućnosti svakog od ta tri sustava te ih usporediti i donesti zaključke o svrsi i najboljim praksama primjene svakog od njih.

**Ključne riječi**: Unity, grafika, protočni sustav renderiranja, skriptiranje, objekt, shader, scena

# Sadržaj

# 1. Uvod

Unity je računalni software razvijen od strane kompanije Unity Technologies. Unity je software koji, prema svojoj namjeni, spada u skupinu pogona za video igre (engl. video game engine). Unity u sebi sadrži sve potrebne alate za razvoj video igara za različite platforme, odnosno za sve moderne sustave za koje se video igre trenutno razvijaju. Sam Unity editor, software za razvoj video igara, može se koristiti na Windowsu, Linuxu i macOS-u. Prva verzija Unityja izdana je 8. lipnja 2005 godine te je do danas izašlo više različitih inačica i nadogradnji.

Unity je pogon za video igre koji je tijekom godina stekao veliku popularnost među developerima video igara zbog svoje pristupačnosti i velikih mogućnosti koje pruža. Postojanje besplatnih pogona za video igre poput Unityja koje developeri mogu jednostavno preuzeti i koristiti olakšava proces izrade video igre na način da eliminira potrebu za izradom pogona za video igru od strane samih developera, već im omogućava da se fokusiraju više na kreativne aspekte developmenta video igara. Ipak, korištenje Unityja zahtjeva upoznavanje s njegovim alatima i mogućnostima ukoliko developer video igara želi svoje ideje pretvoriti u stvarni proizvod. Upravo je zato ideja ovog rada izložiti neke od Unityjevih tehničkih aspekata koje je nužno poznavati kako bi proces izrade video igre bio kvalitetan i polučio uspjeh. Konkretno, tema ovog rada su protočni sustavi renderiranja (engl. rendering pipelines) u Unityju.

Protočni sustavi renderiranja sustavi su zaslužni za iscrtavanje svih vizuala na zaslonu koji izvode niz koraka i operacija kako bi se slika iscrtala na zaslonu prema zamislima kreatora igre. U Unityju postoje tri protočna sustava renderiranja, svaki sa svojim specifičnostima, prednostima i nedostacima. Za svakog developera video igara važno je upoznati se s načinima na koji svaki od tih sustava renderiranja funkcionira kako bi pravodobno mogao donijeti odluku o tome koji bi od njih bilo najbolje primijeniti u kojoj vrsti projekta. Ideja ovog rada je u osnovnim crtama izložiti specifičnosti, prikazati sličnosti te mane i prednosti svakog od tri protočna sustava renderiranja u Unityju

Konkretno, tri protočna sustava renderiranja koje Unity developerima stavlja na raspolaganje su ugrađeni sustav renderiranja (engl. Built-in Rendering Pipeline), univerzalni sustav renderiranja (engl. Universal Rendering Pipeline, kraće URP) te sustav renderiranja visoke rezolucije (engl. High Definition Rendering Pipeline, kraće HDRP). Svaki od ta tri sustava bit će u daljnjim poglavljima predstavljen kroz pokazne scene izrađene korištenjem svakog od njih.

U daljnjem tekstu, preferirat će se korištenje engleskih naziva, odnosno terminologije iz samog Unityja u svrhu lakšeg razumijevanja teksta te podudarnosti između termina korištenih u tekstu i onih vidljivih na slikama i u programskom kodu.

# 2. Metode i tehnike rada

Kako bi se demonstrirao način funkcioniranja svakog od Unityjevih rendering pipelineova, izrađena su tri projekta, jedan pomoću svakog rendering pipelinea. U svakom projektu naglasak je stavljen na demonstraciju glavnih specifičnosti svakog pipelinea te su one demonstrirane kroz praktične primjere, odnosno njihovom primjenom prilikom izrade pokaznih scena.

Svaka od tri pokazne scene izrađena je korištenjem, u najvećoj mjeri, alata koje pruža Unity editor, a od dodatnih alata korišteni su Adobe Illustrator i Adobe Photoshop za izradu dodatnih tekstura za pojedine objekte na sceni. Za izradu C# skripti potrebnih za implementiranje određenih funkcionalnosti na scenama korišten je IDE Rider tvrtke JetBrains preuzet putem akademske licence koja omogućava njegovo besplatno korištenje. Za obradu i prilagodbu audio datoteka korištenih prilikom izrade projekata korišten je besplatni alat otvorenog koda Audacity.

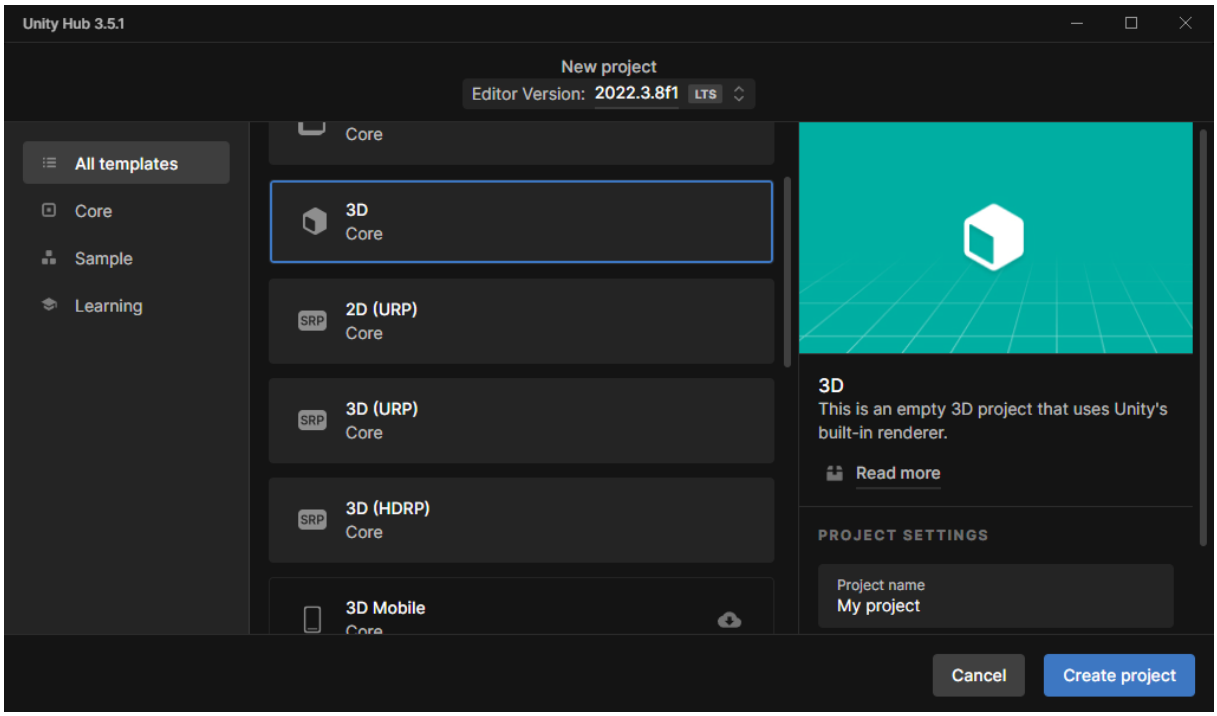
Na scenama su također korišteni i besplatni asseti preuzeti s Unity Asset Storea te asseti preuzeti s web stranice Poly Haven. Svi asseti koji nisu izrađeni od strane autora ovog rada mogu se bez bilo kakve naknade koristiti i u komercijalne i u nekomercijalne svrhe tako da ničija autorska prava nisu povrijeđena. Popis svih asseta preuzetih sa spomenutih web stranica može se naći na kraju rada.

NAPOMENA: Za izradu projekata korištena je inačica Unityja 2022.3.8f1 tako da pojedine stvari demonstrirane u ovom radu mogu funkcionirati na drugačiji način u ostalim verzijama Unityja ili će s vremenom ovdje iznesene informacije postati zastarjele.

# 3. Izrada novog projketa i odabir rendering pipelinea

Unity game engine developerima na raspolaganje stavlja tri različita rendering pipelinea, a odluku o tome koji će rendering pipeline biti korišten potrebno je donijeti prije samog kreiranja projekta. Tri rendering pipelinea između kojih je moguće birati su: Built-in Rendering Pipeline, Universal Rendering Pipeline (kraće URP) i High Definition Rendering Pipeline (kraće, HDRP).

Kako bi kreirali novi projekt potrebno je otvoriti Unity Hub te pritisnuti tipku *New project*. Pritiskom na tipku *New project* otvara se zaslon za odabir predloška prema kojem želimo izraditi novi projekt. Ti predlošci u stvari postavljaju osnovne elemente potrebne za izradu igre. Nakon odabira predloška, također je potrebno dati ime projektu te odabrati mjesto za pohranu istog. Konkretno, zaslon za kreiranje novog projekta izgleda ovako:




Slika 1: Zaslon za kreiranje novog projekta

Kao što na slici možemo uočiti, zaslon za kreiranje novog projekta nudi veliki broj predložaka za različite namjene. No, budući da je fokus ovog rada na rendering pipelineovima, pojasnimo one koji su relevantni.

Iako Unity nudi mogućnost izrade 2D projekata pomoću Built-in Rendering Pipelinea i Universal Rendering Pipelinea, prilikom izrade ovog rada fokus je bio stavljen na 3D projekte kako bi se lakše mogle demonstrirati bitne sličnosti i razlike.

Uz predloške za 3D projekte možemo uočiti oznake kao što su *URP* i *HDRP*. One označavaju koji će rendering pipeline Unity koristiti za izradu odabranog predloška. Konkretno, ako odaberemo *3D*, predložak će biti generiran korištenjem Built-in Rendering Pipelinea. Ukoliko odaberemo *3D (URP)*, generirat će se predložak koji koristi Universal Rendering Pipeline. Ako pak odaberemo *3D (HDRP)*, generirani predložak koristit će High Definition Rendering Pipeline.

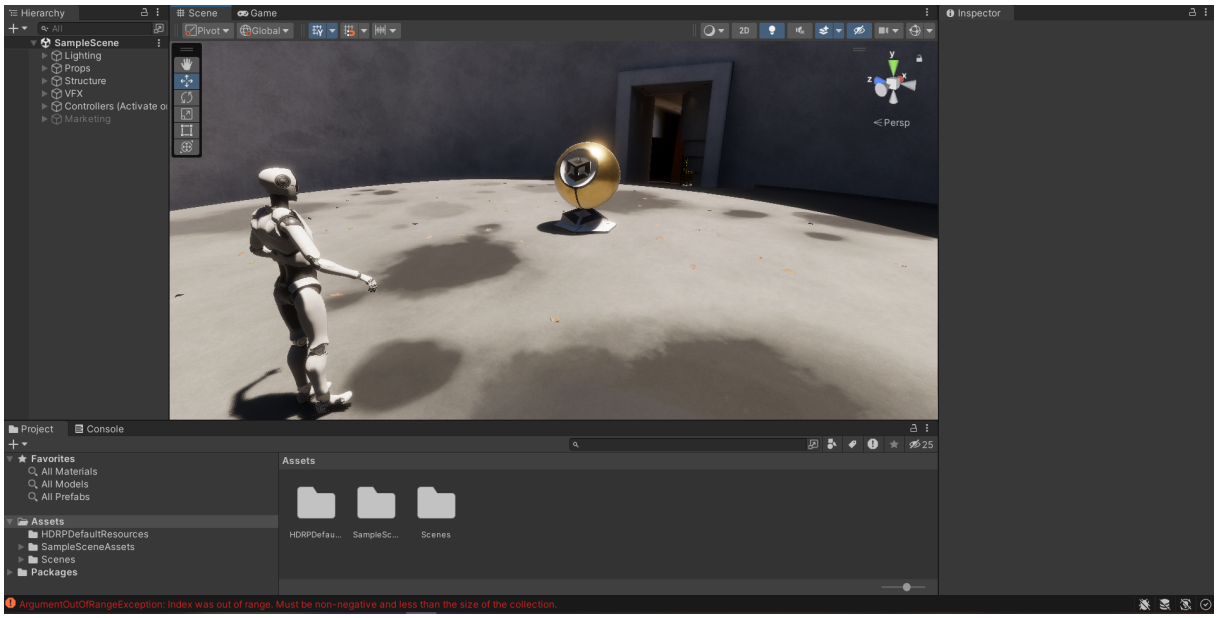
Na temelju dosad prikazanog možemo uočiti glavni razlog zašto je potrebno upoznati se mogućnostima svakog od dostupnih rendering pipelineova. Naime, odluku o tome koji će rendering pipeline biti korišten u projektu potrebno je donijeti prije samog početka rada na projektu. Ukoliko smo započeli rad na projektu, migracija prema drugom rendering pipelineu, iako u nekim slučajevima moguća, nije preporučljiva zbog brojnih tehničkih problema koji iz nje mogu proizaći.

Kao temeljni resurs za upoznavanje rendering pipelineova, najbolji je izbor Unityjeva službena dokumentacija. Na Unityjevim službenim stranicama moguće je pronaći detaljnu tablicu usporedbe rendering pipelineova prema raznim svojstvima smisleno grupiranim. Tablica usporedbe svojstava rendering pipelineova poslužila je i kao polazna točka u istraživanju za ovaj rad.

## 3.1. Pokazne scene

Unity korisnicima također nudi mogućnost preuzimanja unaprijed izgrađenih pokaznih scena (engl. sample scene) te njihovo korištenje kao predložak za izradu novog projekta. Iako kvalitetno izrađene, te pokazne scene u svojoj suštini imaju više edukativnu ulogu te omogućuju korisnicima da se upoznaju s nekim mogućnostima pojedinih rendering pipelinova. Konkretno, moguće je preuzeti pokaznu scenu za URP i HDRP. Na svakoj od tih scena nalazi se nekoliko objekata pomoću kojih se nastoji demonstrirati u manjoj mjeri što je sve moguće postići korištenjem spomenutih rendering pipelineova. Ukoliko se netko odluči na ovu varijantu kod kreiranja novog projekta, na zaslonu za izbor predložaka potrebno je odabrati opciju 3D Sample Scene (URP) ili 3D Sample Scene (HDRP). Scene koje se dobiju korištenjem ovih predložaka izgledaju ovako:


Slika 2: Pokazna scena izrađena korištenjem Universal Rendering Pipelinea



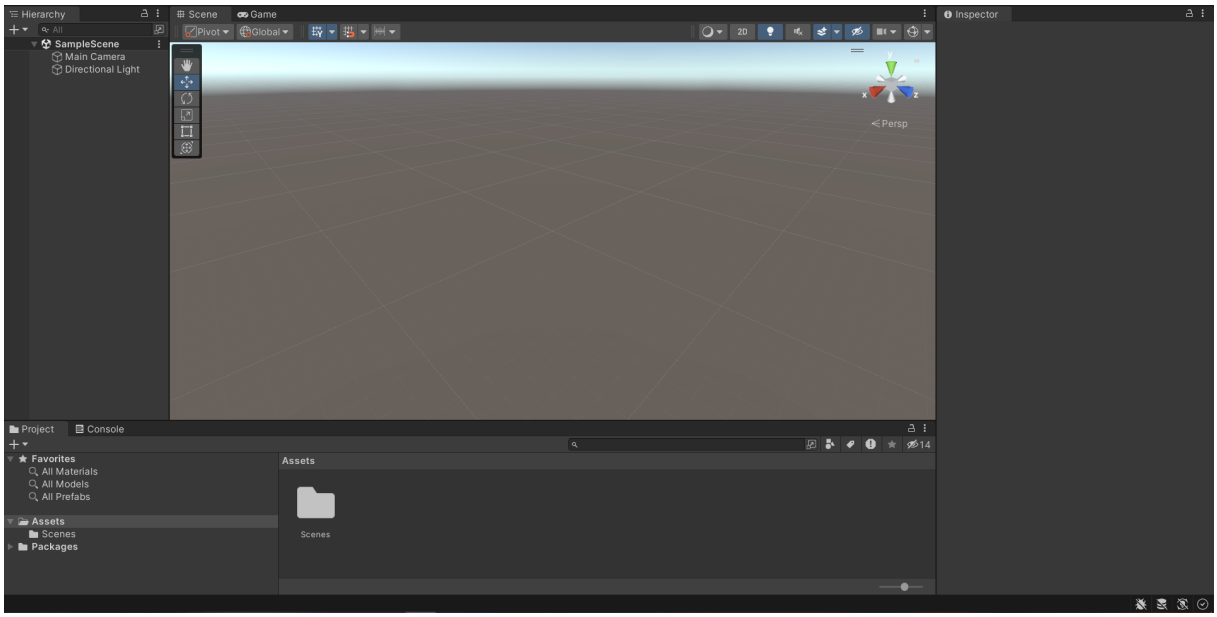

Slika 3: Pokazna scena izrađena korištenjem High Definition Rendering Pipelinea

Pokušati secirati ove scene, proučiti kako su objekti kreirani i razni efekti primijenjeni može poslužiti kao početno upoznavanje s Universal Rendering Pipelineom i High Definition Rendering Pipelineom, no ta metoda, kao što su pripreme za izradu ovog rada pokazale, imaju svoje mane. Učenje na potpuno unaprijed pripremljenoj sceni često ne daje potpuni uvid u to kako pojedine mehanike funkcioniraju i kako ih kvalitetno primijeniti pri izradi scene. Tijekom izrade projekata za ovaj rad, pokušaji rekreiranja pojedinih aspekata pokaznih scena na scenama u izradi nisu polučili najbolji uspjeh pukim proučavanjem strukture pokaznih scena, već je u gotovo svim slučajevima bilo potrebno napraviti detaljnije istraživanje o implementaciji pojedinih stvari i dublje proučiti dokumentaciju.

Ipak, iako se njihova edukativna vrijednost pokazala ograničenom, pokazne su scene i dalje zanimljiv dodatak Unityjevoj bazi predložaka te svakako mogu poslužiti kao prvo upoznavanje s određenim rendering pipelineom ili pak kao demonstracija nekog od njih, odnosno pomoći developeru u donošenju odluke o tome koji će rendering pipeline primijeniti na svom projektu.

# 4. Built-in Rendering Pipeline

Built-in rendering pipeline Unityjev je osnovni rendering pipeline za 2D i 3D projekte. Built-in Rendering Pipeline, kao takav, developerima nudi mogućnost izrade projekata bez naprednijih opcija za manipulaciju igrine grafike, ali sa svim potrebnim elementima koje igra mora sadržavati. Kreiranjem novog projekta pomoću Built-in Rendering Pipelinea, Unity stvara praznu scenu na koju postavlja dva najosnovnija objekta, kameru i izvor svijetla. Objekt kamere prema zadanim postavkama nosi naziv *Main Camera*. Izvor svijetla koji se automatski postavlja na scenu je objekt pod nazivom *Directional Light*. Koncept Directional Lighta je svojstven svim rendering pipelineovima u Unityju, a predstavlja jednu od četiri vrste osvjetljenja koju rendering pipelineovi u Unityju podržavaju. *Directional Light* jedan je od temeljnih objekata na većini scena jer po svom načinu funkcioniranja predstavlja izvor svijetla za cijelu scenu te najčešće služi za simulaciju prirodnog osvjetljenja poput sunčeve svjetlosti.




Slika 4: Built-in Rendering Pipeline – novokreirani projekt

## 4.1. Izrada scene

Kao što je u uvodu bilo najavljeno, korištenjem svakog od rendering pipelineova, izrađena je po jedna scena na kojoj se nastojalo demonstrirati koje sve mogućnosti svaki od njih pruža.

### 4.1.1. Omogućavanje kretanja

Kako bi se olakšalo kretanje kroz scenu te kako bi se projekt što je više moguće približio formi stvarne igre, prva stvar koju je bilo potrebno napraviti je omogućiti kretanje kroz scenu pomoću miša i tipkovnice. U tu je svrhu kreiran novi objekt pod nazivom *Player* na koji je dodana komponenta *Character Controller* koja omogućuje kontroliranje igračeva lika bez korištenja Ri-

gidbody simulacije fizike[3]. Nakon toga, izmijenjena je hijerarhija objekata pa je tako objekt *Main Camera* postavljen kao podređeni objekt objektu *Player*. Naposlijetku, kreirane su dvije skripte, *MouseLook.cs* i *Movement.cs* kako bi se omogućilo okretanje kamere pomoću miša i pomicanje objekta igrača pomoću input s tipkovnice.

Skripta *MouseLook.cs* koja omogućava pomicanje kamere pomoću miša implementirana je na sljedeći način:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MouseLook : MonoBehaviour
{
    [SerializeField] float sensitivity = 500f;
    [SerializeField] Transform person;
    private float viewUp = 0f;

    // Start is called before the first frame update
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
    }

    // Update is called once per frame
    void Update()
    {
        float mouseAxisX = Input.GetAxis("Mouse␣X") * sensitivity * Time.deltaTime;
        float mouseAxisY = Input.GetAxis("Mouse␣Y") * sensitivity * Time.deltaTime;
        viewUp -= mouseAxisY;
        viewUp = Mathf.Clamp(viewUp,-90f, 90f);

        person.Rotate(Vector3.up * mouseAxisX);
        transform.localRotation = Quaternion.Euler(viewUp, 0f, 0f);
    }
}
```

Nakon toga, skripta je nadodana kao komponenta objektu *Main Camera*. Zatim je izrađena skripta *Movement.cs* koja omogućava kretanje objekta igrača po sceni. Skripta je izrađena ovako:

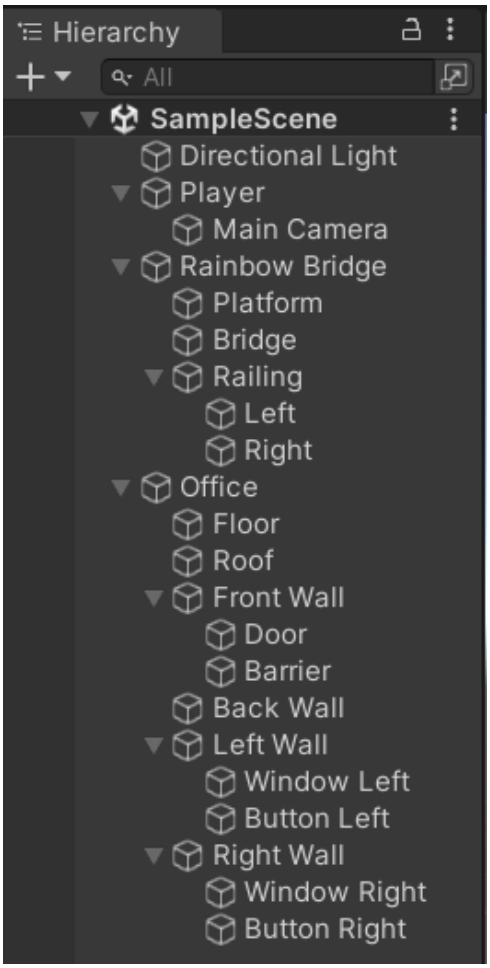
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Movement : MonoBehaviour
{
    [SerializeField] CharacterController person;
    [SerializeField] float movementSpeed = 5f;
    private float gravitationalAcceleration = 9.8f;
    private Vector3 velocity;

    // Start is called before the first frame update
```

```
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        float axisX = Input.GetAxis("Horizontal");
        float axisZ = Input.GetAxis("Vertical");
        Vector3 move = transform.right * axisX + transform.forward * axisZ;

        person.Move(move * movementSpeed * Time.deltaTime);

        velocity.y = -gravitationalAcceleration * Time.deltaTime;
        person.Move(velocity);

    }
}
```

Skripta je potom dodana kao komponenta objektu *Player*. Ove dvije skripte omogućile su jednostavan, ali efikasan sustav kretanja. Sustav kretanja u ostala je dva projekta izrađena za potrebe demonstracije drugih rendering pipelineova identičan ovome tako da u narednim poglavljima ne će biti ponovo objašnjavan.

### 4.1.2. Postavljanje osnovnih objekata na scenu

Izgled pokazne scene zamišljen je kao jednostavna platforma koju sa zatvorenom prostorijom nasuprot nje povezuje most. Ovakav raspored objekata na sceni omogućio je prikaz nekih od glavnih značajki projekta izrađenog pomoću Built-in Rendering Pipelinea. Svi objekti na sceni kreirani su pomoću alata za kreiranje 3D objekata unutar Unity editora. Iako su prva dva objekta, platforma na kojoj igra započinje i most kreirani Unityjevom standardnom metodom za kreiranje 3D objekata (u prozoru *Hierarchy* pritiskom na desnu tipku miša i odabir opcije *3D Object*), za potrebe izrade prostorije na kraju mosta, iskorišten je alat ProBuilder. ProBuilder je jedan od Unityjevih alata koji omogućava naprednije tehnike za izradu i manipulaciju 3D objektima na sceni. ProBuilder nije prema inicijalnim postavkama uključen u projekt, već ga je potrebno u projekt uključiti putem *Package Managera*. Alat se pokazao izuzetno pogodnim za oblikovanje geometrije prostora te je omogućio da se, primjerice, na zidove dodaju otvori koji predstavljaju prozore i vrata bez korištenja vanjskih alata za 3D modeliranje. Nakon postavljanja ovih osnovnih objekata, dobivena je scena sa svim potrebnim objektima s kojima će igrač vršiti interakciju i prostorima kojima će se igrač kretati.

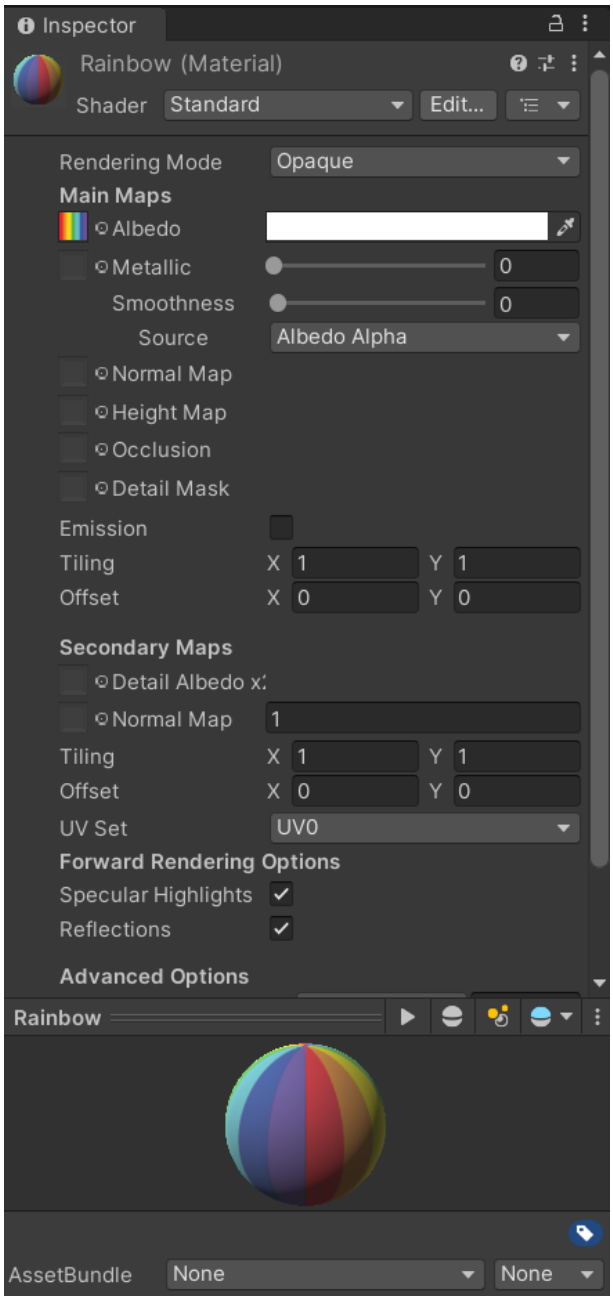



Slika 5: Hijerarhija objekata na sceni

### 4.1.3. Definiranje vizualnog identiteta scene

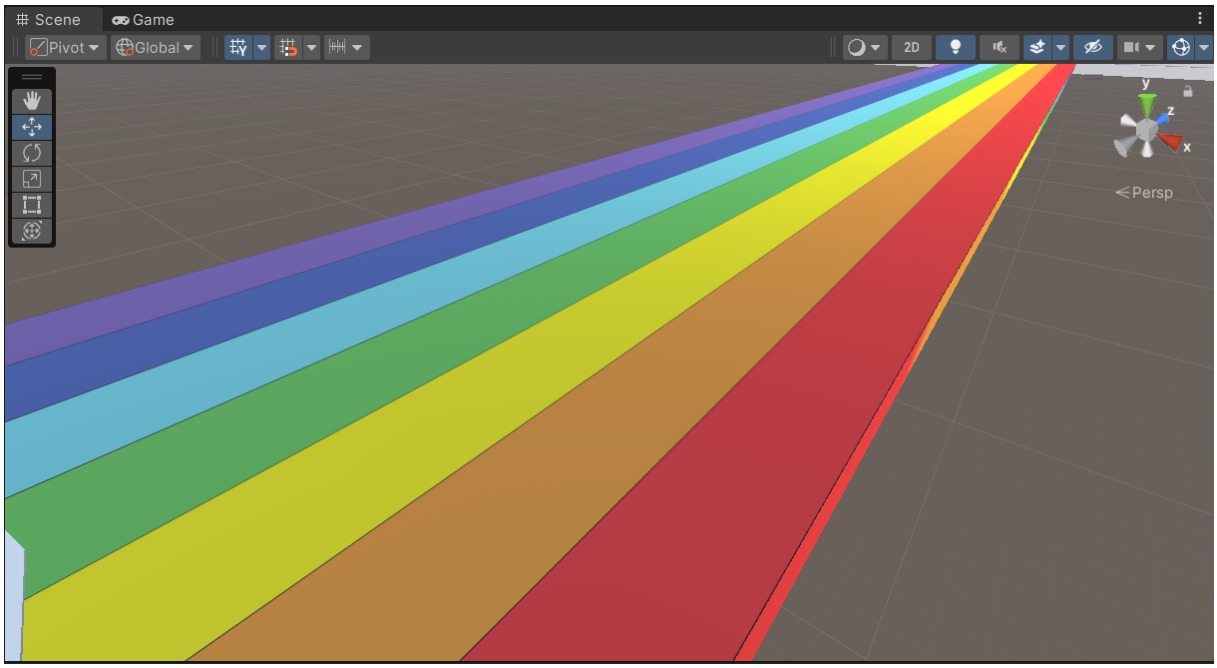
Pošto su kreirani svi potrebni objekti na sceni, sljedeći korak bilo je davanja vizualnog identiteta sceni. Za tu je svrhu pomoću Adobe Illustratora i Adobe Photoshopa kreiran niz slika koje predstavljaju teksture i površine objekata na sceni. Te je slike bilo potrebno uvesti u projekt te ih pretvoriti u materijale koji se mogu primijeniti na objekte na sceni. Sam proces kreiranja materijala ne varira ovisno o tome koji se rendering pipeline koristi, ali mogućnosti oblikovanja materijala značajno su drugačije zavisno o odabranom rendering pipelineu.

Za dobivanje novog materijala potrebno je među assetima projekta u željenoj mapi kreirati novu datoteku tipa materijal (ekstenzija .mat). Nakon što je novi materijal kreiran, u prozoru *Inspector* možemo otvoriti njegove postavke te ga modificirati. Najprije je potrebno kreirati glavnu mapu materijala (engl. *Main Map*). Prvi je korak željenu teksturu, odnosno jednu od prethodno uvezenih slika, dodati u polje *Albedo*. Ukoliko želimo detaljnije definiranu teksturu, možemo dodati i stvari kao što su mapa normala (engl. *Normal Map*), visinska mapa (engl. *Height Map*) i slično. Također možemo mijenjati ili uključivati i isključivati razne parametre kao što su način renderiranja (engl. *Rendering Mode*), odraz (engl. *Reflection*) i slično.

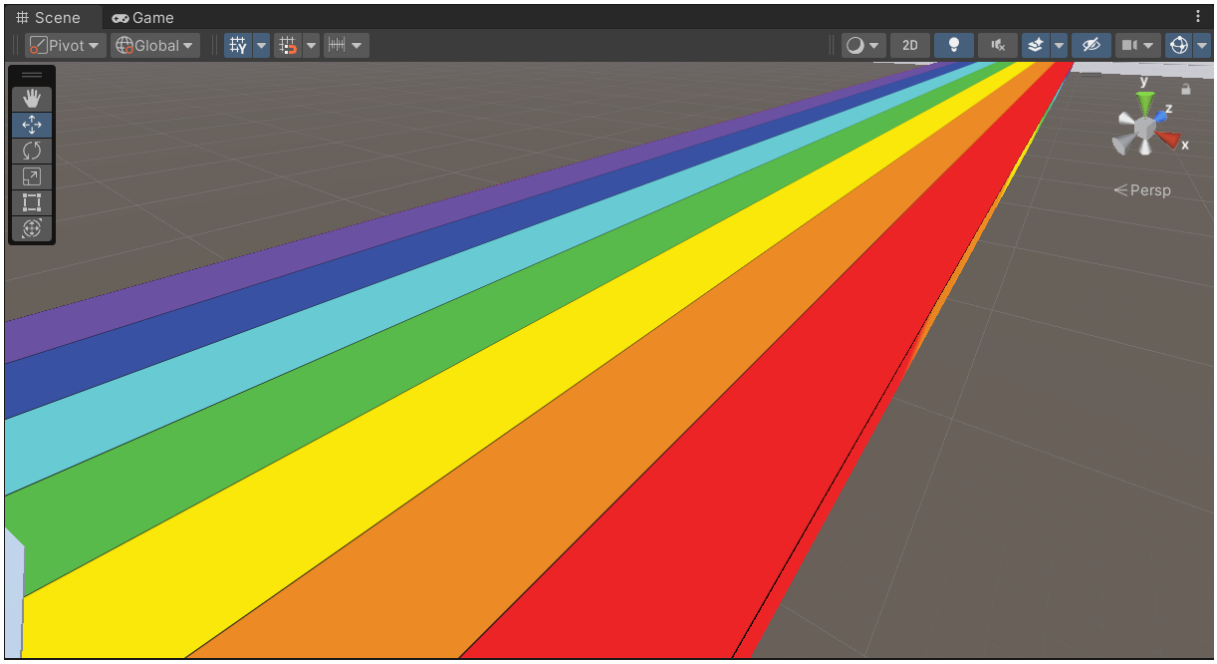



Slika 6: Prikaz materijala *Rainbow* u prozoru *Inspector*

Aspekt u kojem se razlike između različitih rendering pipelineova najviše očituju prilikom oblikovanja materijala su mogućnosti odabira shadera. Iako je moguće i kreiranje vlastitih shadera, svaki rendering pipeline nudi nekoliko različitih vrsta unaprijed definiranih shadera koji iscrtavaju teksturu materijala na različiti način. Konkretno, za sve materijale izrađene za potrebe scene korišten je *Standard* shader iako Built-in Rendering Pipeline nudi veći broj shadera. Neki od shadera koje Built-in Rendering Pipeline nudi su: *Standard*, *Standard (Specular Setup)*, nekoliko vrsta *Unlit* shadera, nekoliko vrsta *Sprite* shadera i slično.

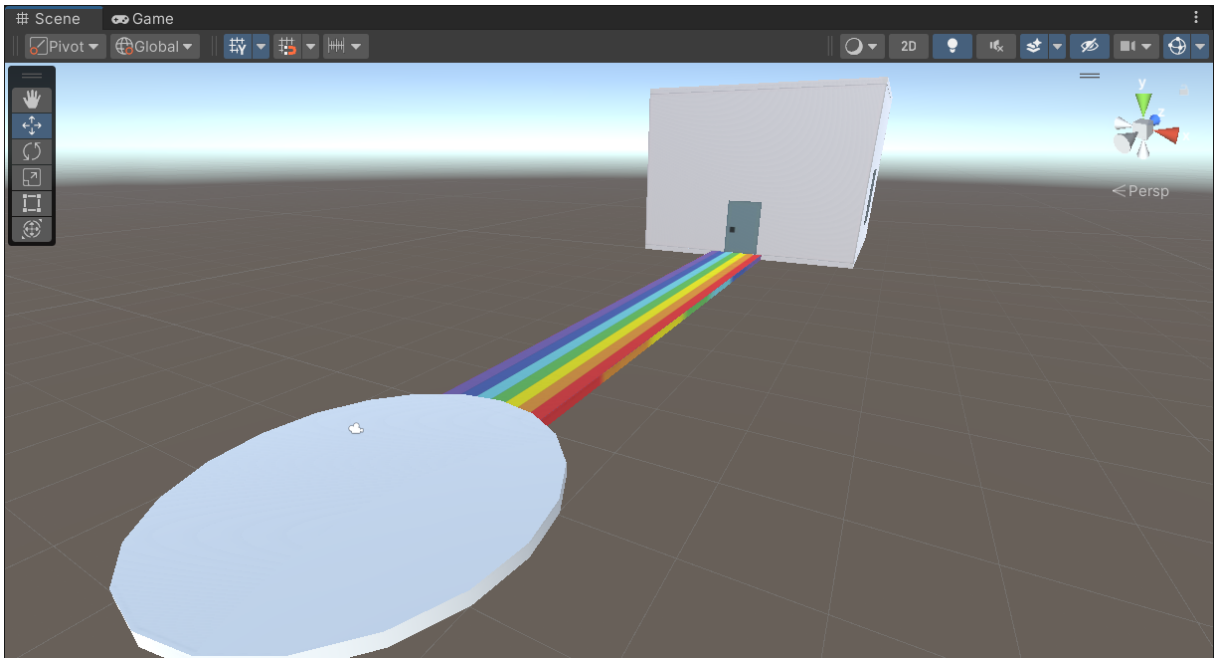



Slika 7: Materijal *Rainbow* (*Standard* shader)

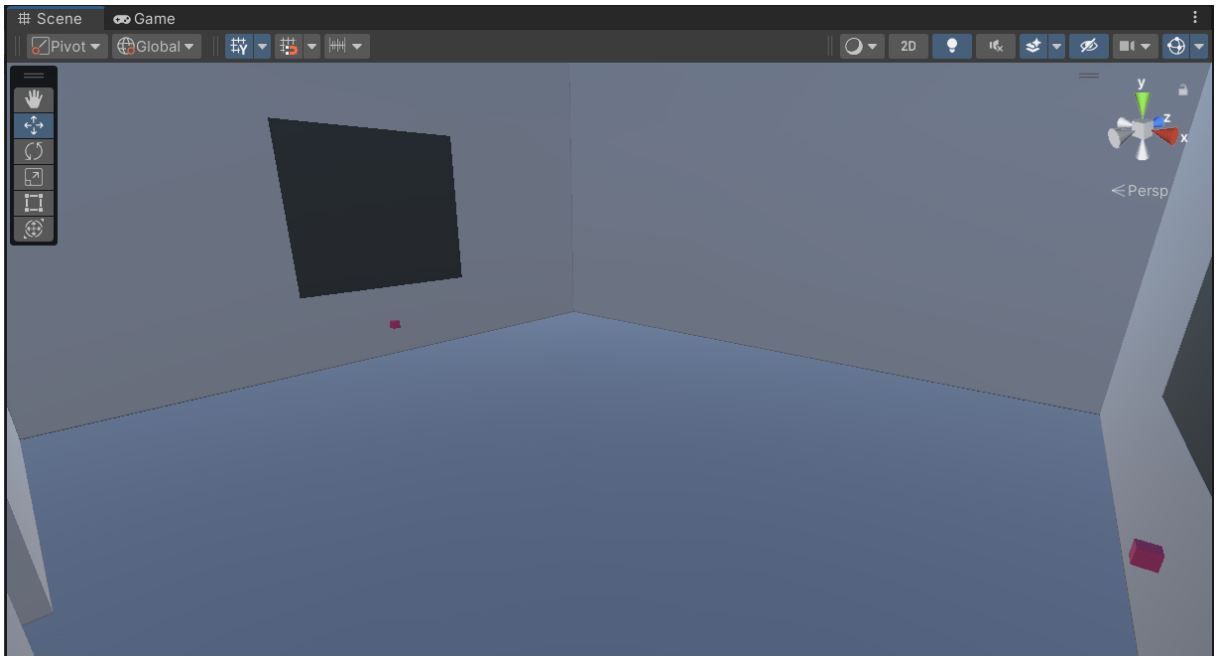



Slika 8: Materijal *Rainbow* (*Unlit/Texture* shader)

Nakon izrade potrebnih materijala i dodavanja istih objektima na sceni dobiven je sljedeći rezultat.




Slika 9: Pregled scene




Slika 10: Pregled scene (interijer)

### 4.1.4. Postavljanje neba i skriptiranje kamere

Posljednji detalj koji je bilo potrebno dodati sceni bilo je nebo. Kao idejno rješenje nametnulo se postaviti kompletno monokromno okruženje na putu od platforme do interijera, to jest obojati čitavo nebo oko scene u crnu boju, a zatim, pošto igrač uđe u prostoriju, nebo oko scene pretvoriti u nešto sasvim drugačije, odnosno primijeniti kubnu mapu (engl. cubemap, vrsta materijala koji se koristi za izradu neba u Unity projektima) izrađenu na temelju ilustracije

izrađene, poput i ostalih tekstura, u Adobe Illustratoru. Sve takve promijene na sceni izrađenoj pomoću Built-in Rendering Pipelinea rade se modifikacijom objekta kamere. Postići ovakav efekt daleko je lakše ukoliko se koristi High Definition Rendering Pipeline zbog volume objekata o kojima će više riječi biti kasnije. Za potrebe ovog projekta bilo je potrebno izraditi nekoliko skripti koje manipuliraju kamerom te mijenjaju i prate položaj igrača na sceni.

Budući da se ulazak u prostoriju odvija igračevom interakcijom s vratima prostorije, to jest pritiskom lijeve tipke miša na vrata, najprije je bilo potrebno implementirati skriptu koja igrača, pošto je lijeva tipka miša pritisnuta, uvodi u prostoriju. Za to je izrađena skripta *Door.cs* koja je potom dodana kao komponenta objektu *Door*.

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class Door : MonoBehaviour
{
    private GameObject player;
    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.Find("Player");
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            RaycastHit rayCastHit;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out rayCastHit, 5f))
            {
                if (rayCastHit.transform.gameObject.name == "Door")
                {
                    player.SetActive(false);
                    player.transform.position += new Vector3(0, 0, 10);
                    Position position = player.GetComponent<Position>();
                    player.SetActive(true);
                    position.office = true;
                }
            }
        }
    }
}
```

Zatim je, kako bi se pozicija igrača na mapi mogla pratiti, odnosno znati je li igrač ušao u prostoriju ili napravljena jedna jednostavna skripta *Position.cs* i dodana objektu *Player* (pristupanje varijabli definiranoj u skripti *Position.cs* može se vidjeti u iznad prikazanom kodu skripte *Door.cs*).

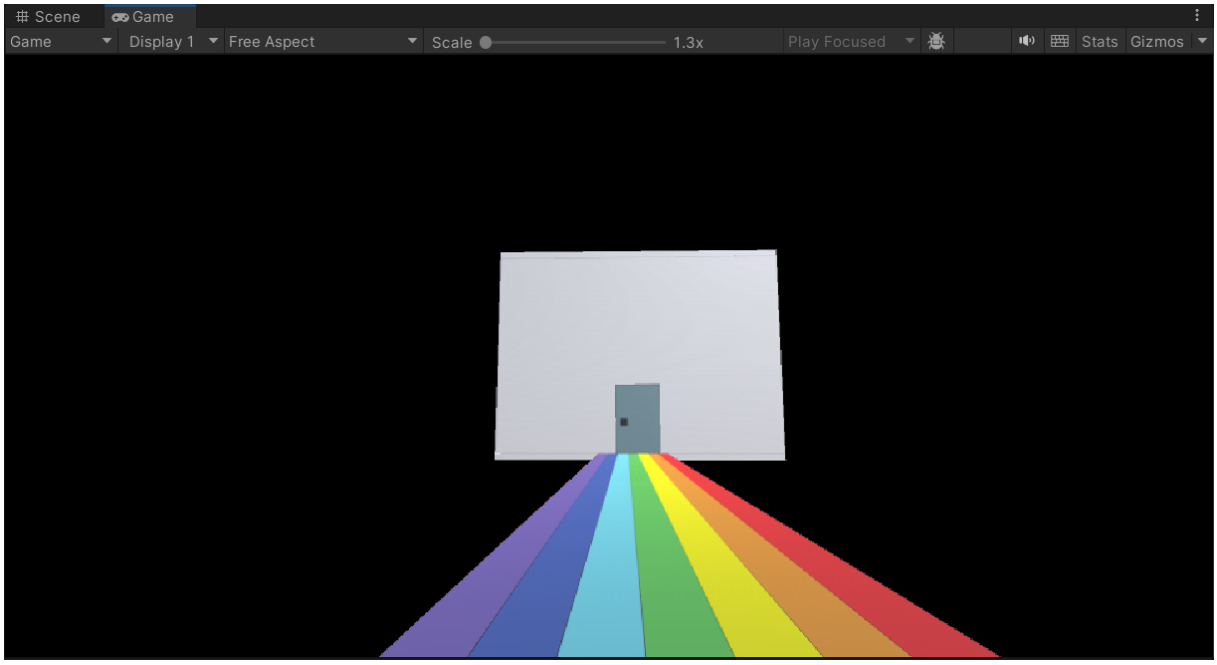
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Position : MonoBehaviour
{
    [SerializeField] public bool office;
    // Start is called before the first frame update
    void Start()
    {
        office = false;
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Naposlijetku je izrađena skripta *CameraSkybox.cs* koja prati skriptu *Position.cs* te na temelju činjenice je li igrač ušao u prostoriju ili ne prilagođava nebo oko scene. Skripta *CameraSkybox.cs* dodana je kao komponenta objektu *Main Camera*.

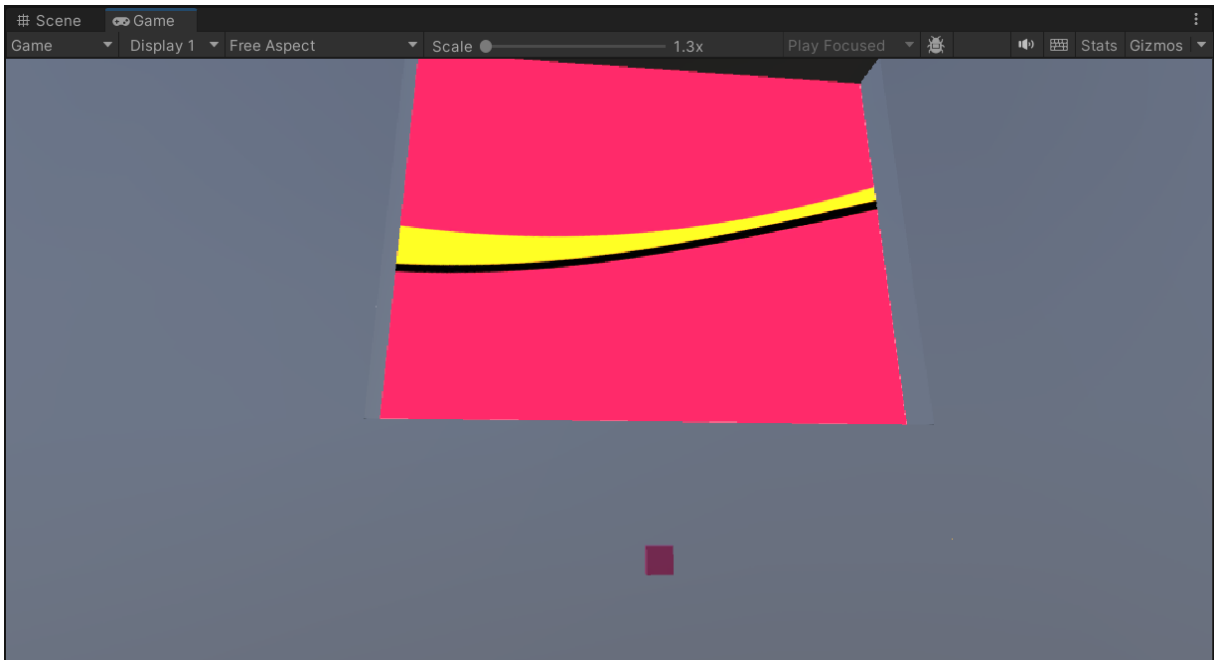
```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class CameraSkybox : MonoBehaviour
{
    private Camera mainCamera;
    [SerializeField] public Material material;
    private Skybox skybox;
    private GameObject player;
    private Position position;

    // Start is called before the first frame update
    void Start()
    {
        mainCamera = gameObject.GetComponent<Camera>();
        mainCamera.clearFlags = CameraClearFlags.SolidColor;
        mainCamera.backgroundColor = Color.black;
        player = GameObject.Find("Player");
        position = player.GetComponent<Position>();
    }

    // Update is called once per frame
    void Update()
    {
        if (position.office)
        {
            mainCamera.clearFlags = CameraClearFlags.Skybox;
```

```
            mainCamera.AddComponent<Skybox>();
            skybox = mainCamera.GetComponent<Skybox>();
            skybox.material = material;
        }
    }
}
```

Na slikama zaslona ispod može se vidjeti da je ova ideja doista uspjela te da se nebo oko scene doista mijenja kad igrač uđe u prostoriju.




Slika 11: Nebo oko scene na putu prema građevini na kraju mosta




Slika 12: Pogled na nebo oko scene iz interijera

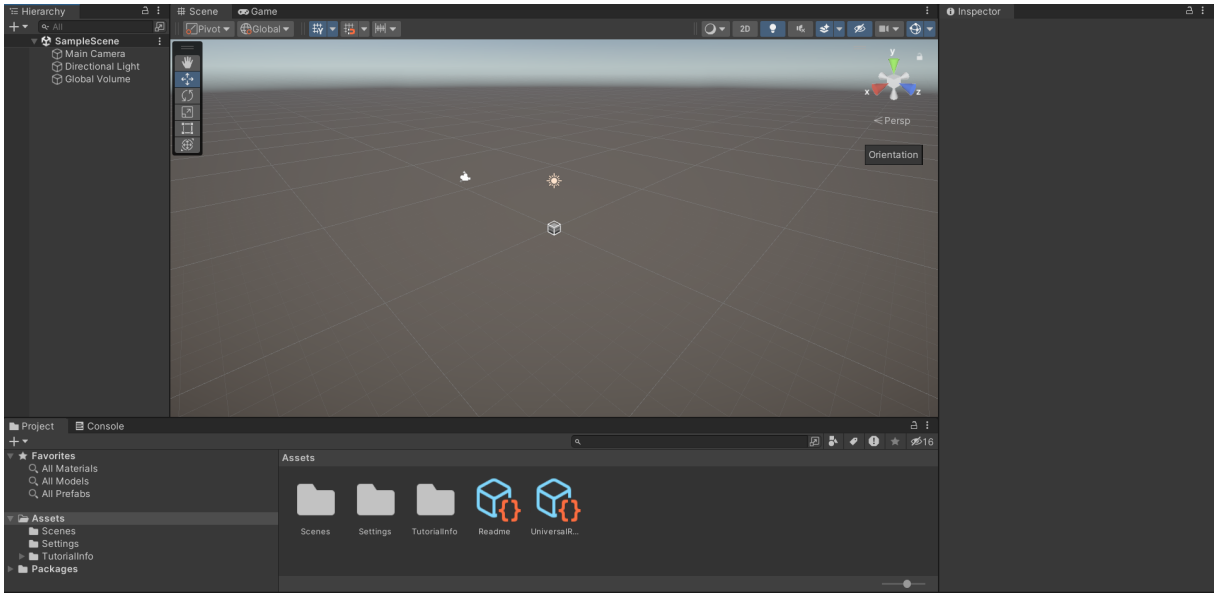
Na kraju je, čisto radi dodavanja interaktivnosti scene i efektivnosti, omogućeno otvaranje i zatvaranje otvora kroz kojih je moguće pogledati nebo oko scene iz unutrašnjosti prostorije. Za tu su potrebu izrađene dvije skripte, po jedna za gumb na svakom zidu. Skripte o kojima se radi su *WindowButtonLeft.cs* i *WindowButtonRight.cs*. Kod prikazan ispod je kod skripte *WindowButtonLeft.cs*. Kod skripte *WindowButtonRight.cs* gotovo je identičan, uz tek male promijene pojedinih parametara.

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class WindowButtons : MonoBehaviour
{
    private GameObject leftButton;
    private GameObject leftWindow;
    private bool leftButtonPressed;

    // Start is called before the first frame update
    void Start()
    {
        leftButton = GameObject.Find("Button␣Left");
        leftWindow = GameObject.Find("Window␣Left");
        leftButtonPressed = false;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            RaycastHit rayCastHit;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out rayCastHit, 5f))
            {
                if (rayCastHit.transform.gameObject.name == "Button␣Left")
                {
                    if (!leftButtonPressed)
                        {
                            leftButton.transform.localScale += new Vector3(-0.3f, 0f
                                , 0f);
                            leftButton.transform.position += new Vector3(-0.15f, 0f,
                                 0f);
                            leftWindow.transform.localScale += new Vector3(0f,
                                -9.3068384f, 0f);
                            leftWindow.transform.position += new Vector3(0f,
                                9.3068384f, 0f);
                            leftButtonPressed = true;
                        }
                        else
                        {
                            leftButton.transform.localScale += new Vector3(0.3f, 0f,
```

```
                    0f);
                leftButton.transform.position += new Vector3(0.15f, 0f,
                    0f);
                leftWindow.transform.localScale += new Vector3(0f,
                    9.3068384f, 0f);
                leftWindow.transform.position += new Vector3(0f,
                    -9.3068384f, 0f);
                leftButtonPressed = false;

            }
        }
    }
}
}
}
```

# 5. Universal Rendering Pipeline

Universal Rendering Pipeline napredniji je rendering pipeline od Built-in Rendering Pipelinea. Osim što sam po sebi nudi veće grafičke mogućnosti od rendering pipelinea opisanog u prethodnom poglavlju, ono što ga čini doista posebnim u odnosu na njega je činjenica da, kao i High Definition Rendering Pipeline, spada u skupinu skriptabilnih cjevovoda renderiranja (engl. scriptable rendering pipeline, kraće SRP). To znači da korisnik može utjecati na način na koji Universal Rendering Pipeline iscrtava sliku na zaslonu pisanjem vlastitih C# skripti koje manipuliraju načinom na koji se to iscrtavanje izvodi. To korisniku daje veći stupanj slobode i mogućnost prilagodbe rendering pipelinea svojim potrebama. Kako bi se demonstrirale mogućnosti Universal Rendering Pipeline, izrađen je novi projekt unutar kojeg je postavljena nova scena.




Slika 13: Universal Rendering Pipeline – novokreirani projekt

Prazna scena izgleda gotovo identično praznoj sceni dobivenoj nakon kreiranja novog projekta pomoću Built-in Rendering Pipelinea. Jedina uočljiva razlika je treći objekt pod nazivom *Global Volume*. Koncept volume objekata ne postoji u Built-in Rendering Pipelineu. Volume objekti su, jednostavno rečeno, objekti koji određuju kako će vizualni identitet pojedinih dijelova scene biti obogaćen raznim efektima. Unutar svakog volumea moguće je postaviti niz post-processing efekata. Svaki volume može biti ili globalan ili lokalan. Ukoliko je globalan, efekti postavljeni unutar njega primjenjuju se na čitavoj sceni, a lokalni volume definira efekte koji se primjenjuju samo na označenom području. Kako bi se primjena lokalnih volumeova olakšala, Unity nudi mogućnost kreiranja kockastih volumeova (engl. Box Volume), sfernih volumeova (engl. Sphere Volume) te konveksnih volumeova (engl. Convex Mesh Volume). Ovisno o vrsti volumea koji izaberemo, kreirat će se objekt s colliderom u željenom obliku te će se svi zadani efekti primjenjivati unutar granica tog collidera. Ukoliko postoji preklapanje između granica dvaju ili više volumea, vrijedit će pravila onog volumea koji ima veći broj postavljen kao parametar *Priority*. Objekt *Global Volume* parametar *Priority* ima postavljen s obzirom na to

da predstavlja temeljni volume na sceni te da će ga najvjerojatnije tijekom uređivanja scene biti potrebno nadjačati jednim ili više lokalnih volumeova.

## 5.1. Postavljanje scene

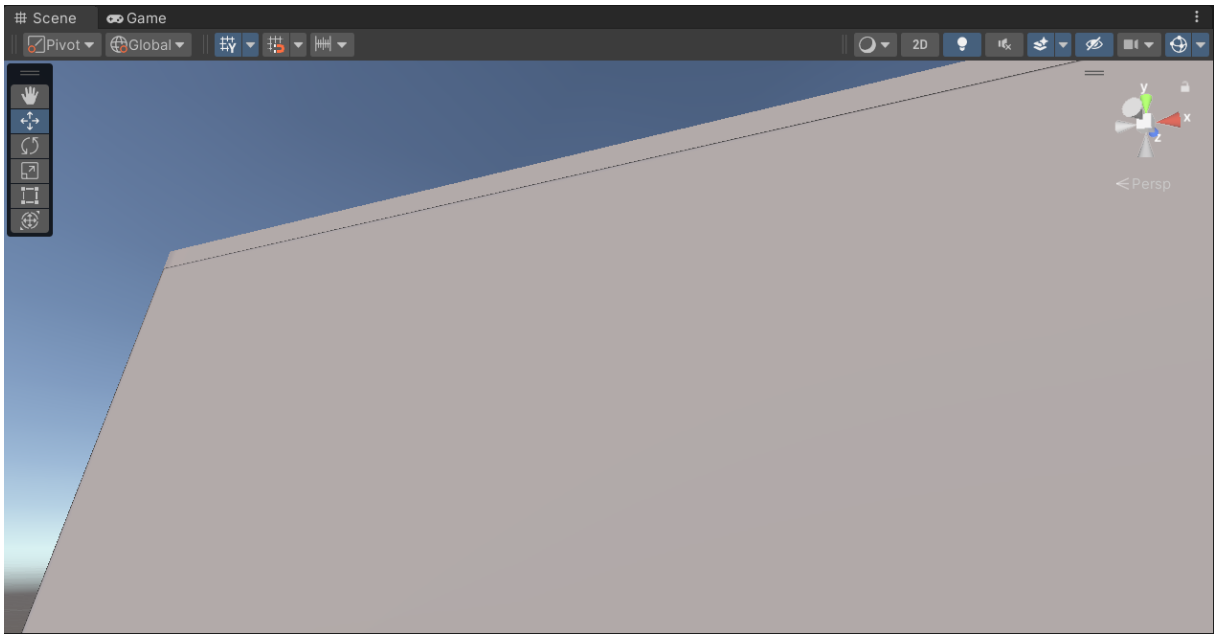
Kako bi se demonstrirale prednosti Universal Rendering Pipelinea u odnosu na Built-in Rendering Pipeline, ali i istaknule sličnosti, rekreirana je scena identična onoj izrađenoj u prethodnom poglavlju. Na taj je način moguće lakše prikazati dodatne mogućnosti koje Universal Rendering Pipeline pruža, to jest pokazati kako je moguće scenu izrađenu u Built-in Rendering Pipelineu poboljšati dodatnim mogućnostima koje URP pruža. Procedura postavljanja scene ovoga je puta bila mnogo jednostavnija. Svi objekti postavljeni na sceni kao i svi asseti izvezeni su iz prethodnog projekta u obliku Unity paketa. Taj je paket potom uvezen u novi projekt.

## 5.2. Shaderi

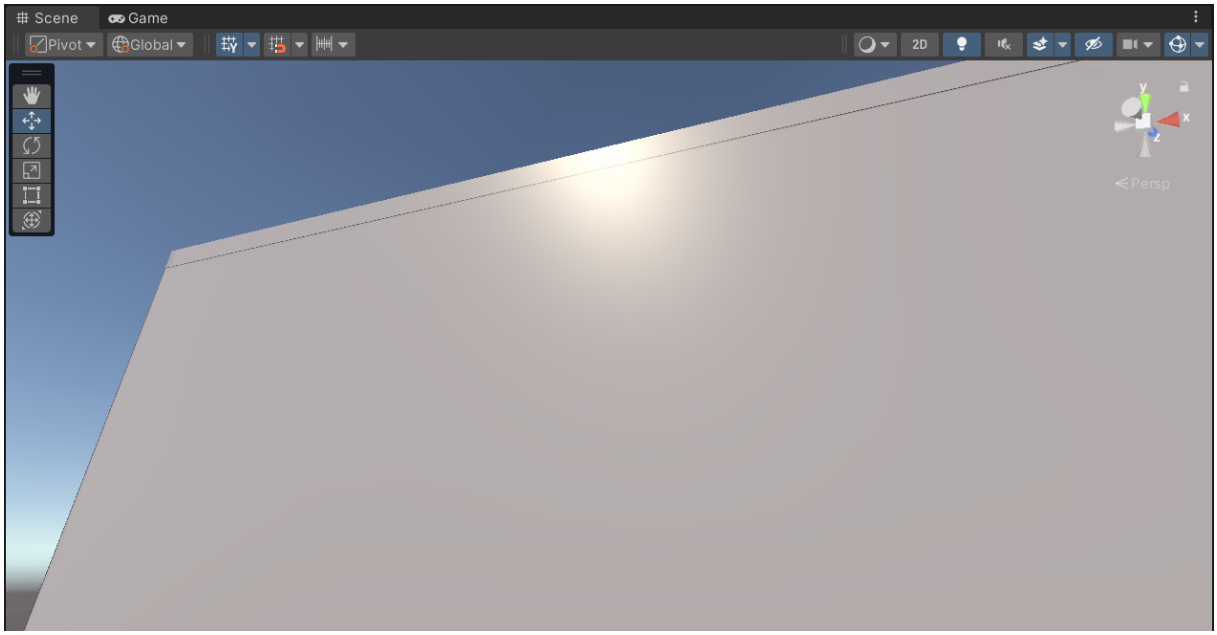
Prva opservacija nakon uvoženja Unity paketa s assetima iz ranije izrađenog projekta bila je nedostatak tekstura na objektima scene. Materijali izrađeni korištenjem Standard shadera u Built-in Rendering Pipelineu nisu sadržavali ranije zadane teksture i boje pa su, stoga, promijenjeni shaderi svakog od materijala. Kada otvorimo bilo koji materijal u prozoru *Inspector* te otvorimo padajući izbornik s popisom dostupnih shadera, možemo uočiti, ako ga usporedimo s istim izbornikom u Built-in Rendering Pipelineu, da su mogućnosti ovdje veće. Konkretno, možemo primijetiti da postoji čitava nova skupina shadera grupiranih pod *Universal Render Pipeline*. U toj su skupini izdvojeni svi shaderi specifični za URP, oni nedostupni u Built-in Rendering Pipelineu. Od novih shadera čije korištenje na sceni URP omogućuje, svakako su najzanimljiviji *Lit* shaderi.

*Lit* shaderi, naprednija su skupina shadera koji funkcioniraju na način da za svaku teksturu koja se iscrtava uz njihovu primjenu izračunavaju količinu svijetla koja se odbija od površine. Metode koje *Lit* shaderi koriste za izračunavanje količine svijetla koja se odbija od površine objekata troše mnogo računalnih resursa tako da njihova primjena nije uvijek preporučljiva[9]. Uglavnom se koriste za kvalitetnije prikaze materijala iz stvarnog svijeta poput kamenih i drvenih površina. U Universal Rendering Pipelineu postoji nekoliko vrsta *Lit* shadera, *Lit*, *Complex Lit*, *Simple Lit* i *Baked Lit*. Osnovni *Lit* shader primijenjen je na sve materijale u projektu.

Primjena *Lit* shadera na sve materijale u projektu u pogledu optimizacije nije baš najbolja ideja, no ovdje se ne radi o nekom odveć kompleksnom projektu tako da malo dodatnog procesorskog opterećenja nije naškodilo funkcionalnosti demonstrativne scene. Štoviše, doista je pomoglo u percipiranju načina na koji *Lit* shaderi funkcioniraju, čak i uz nesavršene teksture objekata na sceni. Za kreirane materijale nisu izrađene mape normala niti visinske mape, ali se podešavanjem parametara kao što su *Smoothness* (glatkoća površine) doista mogu vidjeti učinci primjene *Lit* shadera.




Slika 14: Tekstura zida dobivnea korištenjem *Lit* shadera uz parametar *Smoothness* postavljen na 0




Slika 15: Tekstura zida dobivnea korištenjem *Lit* shadera uz parametar *Smoothness* postavljen na 0.63

## 5.3. Post-processing efekti

Dodavanje post-processing efekata uvelike je olakšano u Universal Rendering Pipelineu uvođenjem volume objekata. Volume objekti omogućuju da se, kao što je ranije u tekstu opisano, na jednostavan način definira područje unutar kojeg se primjenjuju post-processing efekti. Premda je dodavanje post-processing efekata moguće i u Built-in Rendering Pipelineu,

u njemu se izvodi na drugi način. Potrebno je najprije pomoću *Package Managera* u projekt uključiti paket *Post Processing*, a zatim primijeniti efekte dodavanjem komponente *Post-process Layer* na objekt kamere. Premda je ova metoda dostupna i u Universal Rendering Pipelineu, favorizira se dodavanje post-processing efekata kreiranjem volume objekata.

Dodavanje post-processing efekata na scenu korištenjem volume objekata intuitivan je i jednostavan način za obogaćivanje scene vizualnim efektima. Postupak se sastoji od sljedećih koraka. Najprije je potrebno kreirati novi objekt tipa *Volume*. Prilikom kreiranja objekta potrebno je odabrati hoće li novi objekt biti globalni volume (primijenjen na čitavoj sceni) ili lokalni te, ukoliko se odabere lokalni volume, odabrati oblik collidera pomoću kojeg će se označiti područje djelovanja volumea. Konkretno, na ovoj je sceni kreiran lokalni kockasti volume (engl. *Box Volume*). Nakon što je novi objekt kreiran pod imenom *Post-processing*, potrebno je bilo alatima za pomicanje, skaliranje i rotaciju proširiti collider objekta na čitavo područje na kojem želimo da volume djeluje. Volume *Post-processing* na ovoj sceni obuhvaća područje od početne platforme do kraja mosta. Nakon što je objekt kreiran, imenovan i oblikovan, potrebno je volume objektu dodati novi profil ili primijeniti neki od postojećih ukoliko smo ranije kreirali profil sa željenim postavkama. Profili volume objekata sadrže sve informacije o efektima koje želimo primijeniti unutar volumea. Profili sadrže same efekte koji se dodaju u obliku overrideova te podatke o parametrima za svaki override, odnosno efekt. Profil se volume objektu dodaje unutar prozora *Inspector*. U prozoru *Inspector* u polju *Profile* kreiramo novi profil pritiskom na tipku *New* ili pak odabiremo jedan od postojećih. Kako profil u ovom projektu nije bio ranije kreiran, pritiskom na tipku *New* kreiran je novi profil koji je automatski dobio ime *Post-processing Profile*. Novi profil u sebi nije sadržavao nikakve overrideove, odnosno efekte pa ih je, stoga, bilo potrebno dodati. U profil su dodana dva post-processing efekta, *Bloom* i *Chromatic Aberration*. Treba obratiti pozornost i na objekt kamere (u slučaju ove scene objekt *Main Camera*) na kojem je potrebno omogućiti renderiranje post-processing efekata označavanjem parametra *Post Processing* u prozoru *Inspector*. Ukoliko se renderiranje post-processing efekata ne omogući na objektu kamere, bez obzira na postavke volumea, nikakvi efekti ne će biti vidljivi na zaslonu.

S dodavanjem i oblikovanjem post-processing efekata možemo otići i korak dalje. Budući da se dodavanje i uređivanje post-processing efekata u Universal Rendering Pipelineu obavlja jednostavnim manipulacijama s objektom, nije teško uočiti potencijal za podizanje korištenja post-processing efekata na višu razinu. Primjerice, ako želimo na sceni primijeniti post-processing efekte *Bloom* i *Chromatic Aberration* na način da se njihov intenzitet automatski pojačava i smanjuje nesinkronizirano bez inputa od strane igrača, to možemo vrlo jednostavno realizirati izradom C# skripte koja radi upravo to, automatizira promjene postavki post-processing efekata unutar *Post-processing* volumea. Za tu je svrhu kreirana skripta pod nazivom *PostProcessing.cs* koja izgleda ovako:

```
using System.Collections;
using System.Collections.Generic;
using JetBrains.Annotations;
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.Universal;
```

```
public class PostProcessing : MonoBehaviour
{
    [SerializeField] private VolumeProfile volumeProfile;
    private Bloom bloom;
    private ChromaticAberration chromaticAberration;
    private float bloomIntensityIncrement;
    private float chromaticAberrationIntensityIncrement;
    private bool raisingBloom;

    // Start is called before the first frame update
    void Start()
    {
        if (volumeProfile.TryGet(out bloom))
        {
            bloom.intensity.value = 0f;
        }

        if (volumeProfile.TryGet(out chromaticAberration))
        {
            chromaticAberration.intensity.value = 1000f;
        }
        raisingBloom = true;
        bloomIntensityIncrement = 0.1f;
        chromaticAberrationIntensityIncrement = 0.0001f;

    }

    // Update is called once per frame
    void Update()
    {
        if (raisingBloom)
        {
            if (volumeProfile.TryGet(out bloom))
            {
                bloom.intensity.value += bloomIntensityIncrement;
                if (bloom.intensity.value >= 1000) raisingBloom = false;
            }

            if (volumeProfile.TryGet(out chromaticAberration))
            {
                chromaticAberration.intensity.value -=
                    chromaticAberrationIntensityIncrement;
            }
        }
        else
        {
            if (volumeProfile.TryGet(out bloom))
            {
                bloom.intensity.value -= bloomIntensityIncrement;
                if (bloom.intensity.value <= 0) raisingBloom = true;
            }
            if (volumeProfile.TryGet(out chromaticAberration))
            {
```

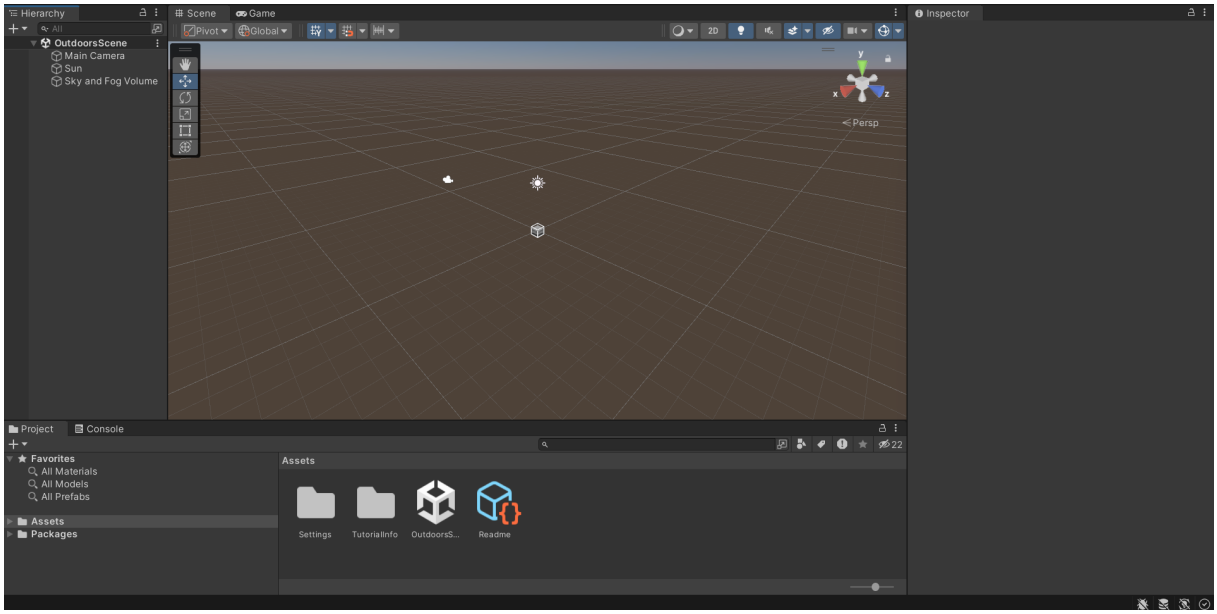
```
                chromaticAberration.intensity.value +=
                    chromaticAberrationIntensityIncrement;
            }
        }
    }
}
```

# 6. High Definition Rendering Pipeline

High Definition Rendering Pipeline, kao i Universal Rendering Pipeline, spada u skupinu skriptabilnih rendering pipelineova. Kao što je u ranijem poglavlju bilo pojašnjeno, to znači da korisniku omogućuje manipulaciju sustavom renderiranja izradom vlastitih skripti u kojima se definira na koji način grafika igre treba biti iscrtana. Iako je u tom aspektu sličan Universal Rendering Pipelineu, High Definition Rendering Pipeline od ostalih se Unityjevih rendering pipelineova izdvaja po tome što nudi mogućnost iscrtavanja gotovo fotorealistične grafike na sceni i izuzetno naprednim metodama osvjetljenja. Fotorealistična grafika i napredno osvjetljenje samo su neke od brojnih specifičnosti High Definition Rendering Pipelinea. On kao takav također korisniku na raspolaganje stavlja brojne alate nedostupne u ostalim rendering pipelineovima koji, u nekim slučajevima olakšavaju postavljanje pojedinih aspekata scene, a u nekim slučajevima posve mijenjaju način na koji se neki elementi scene postavljaju. U nastavku poglavlja bit će pojašnjeno kako je pokazna scena izrađena korištenjem alata koje High Definition Rendering Pipeline pruža.

## 6.1. Postavljanje scene

Kao što je to bio slučaj i sa prethodna dva rendering pipelinea, ponovo je kreiran novi projekt korištenjem *3D (HDRP)* predloška.




Slika 16: High Definition Rendering Pipeline – novokreirani projekt

Na praznom projektu kreiranom iz *3D (HDRP)* predloška možemo uočiti tri osnovna objekta. Ta tri objekta su: *Main Camera*, *Sun* i *Sky and Fog Volume*. *Main Camera*, kao i u prethodnim slučajevima, predstavlja glavnu kameru na sceni. Objekt *Sun* predstavlja glavni izvor svijetla na sceni. To je svjetlosni objekt tipa *Directional Light* čija je ideja i korištenje bilo opisano u jednom od ranijih poglavlja. Objekt je automatski intuitivno nazvan *Sun* budući da

predstavlja glavni izvor osvjetljenja na sceni, to jest simulira sunce. *Sky and Fog Volume* je objekt tipa *Volume*, globalni volume koji u sebi sadrži temeljne postavke za simulaciju prirodne okoline. Jedan novitet koji odmah možemo uočiti u usporedbi s Universal Rendering Pipelineom je činjenica da objekti tipa *Volume* više ne služe samo za postavljanje post-processing efekata, već se radom s njima postavlja cijeli vizualni identitet scene, odnosno mnogobrojni elementi iz prirodne okoline kao što su nebo, magla, vodene površine i slično.

Budući da je temeljna namjena High Definition Rendering Pipelinea poprilično različita od dva prethodno opisana rendering pipelinea, u ovom projektu nisu korišteni nikakvi asseti iz ranije izrađenih projekata već je čitava scena izgrađena ispočetka.

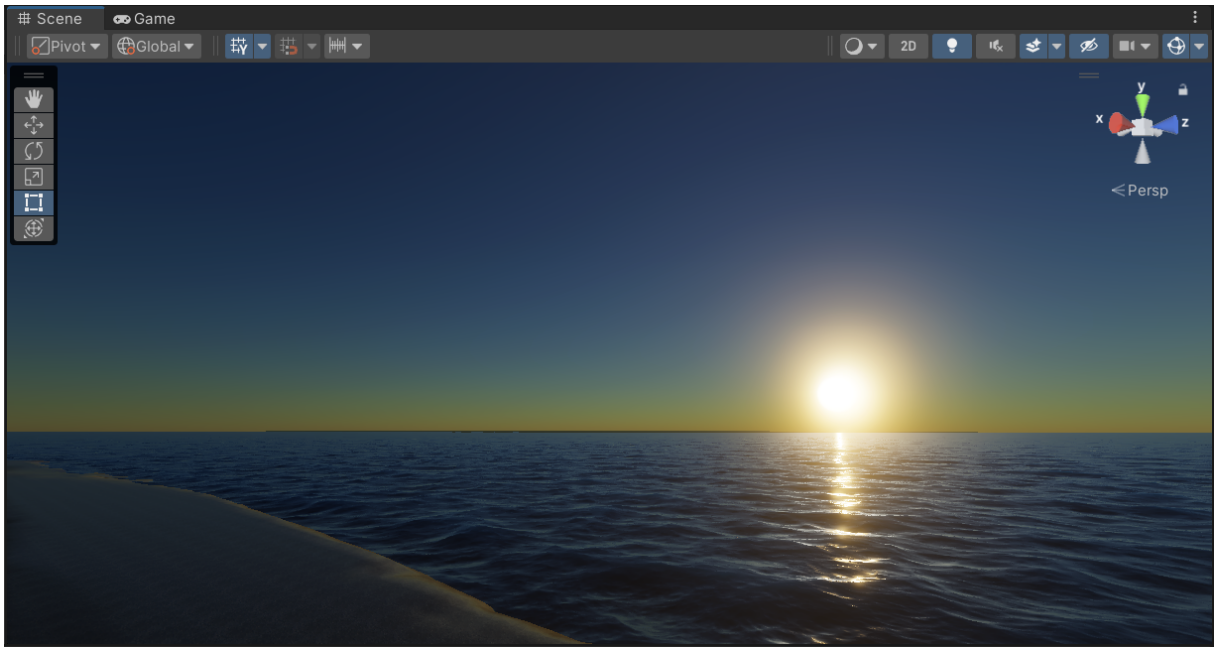
### 6.1.1. Izrada terena

Kako je ideja za kreaciju demonstrativne scene bila izrada obale i planina, za postavljanje osnovne strukture terena iskorišteni su Unityjevi alati za postavljanje terena. Konkretno kreiran je novi 3D objekt tipa *Terrain* koji je potom oblikovan alatima za podizanje, udubljivanje i bojanje terena. Kako bi se stvorio efekt šljunčane obale, s Unity Asset Storea preuzeta je besplatna tekstura šljunka koja je potom primijenjena na početni rub terena. Nakon toga, preuzeta je nova tekstura, također besplatna preuzeta s Unity Asset Storea, tekstura trave koja se nastavlja na šljunčani dio i prekriva ostatak terena. Preuzeta je i još jedna tekstura koja je primijenjena na planine koje se uzdižu iznad scene.

### 6.1.2. Postavljanje neba

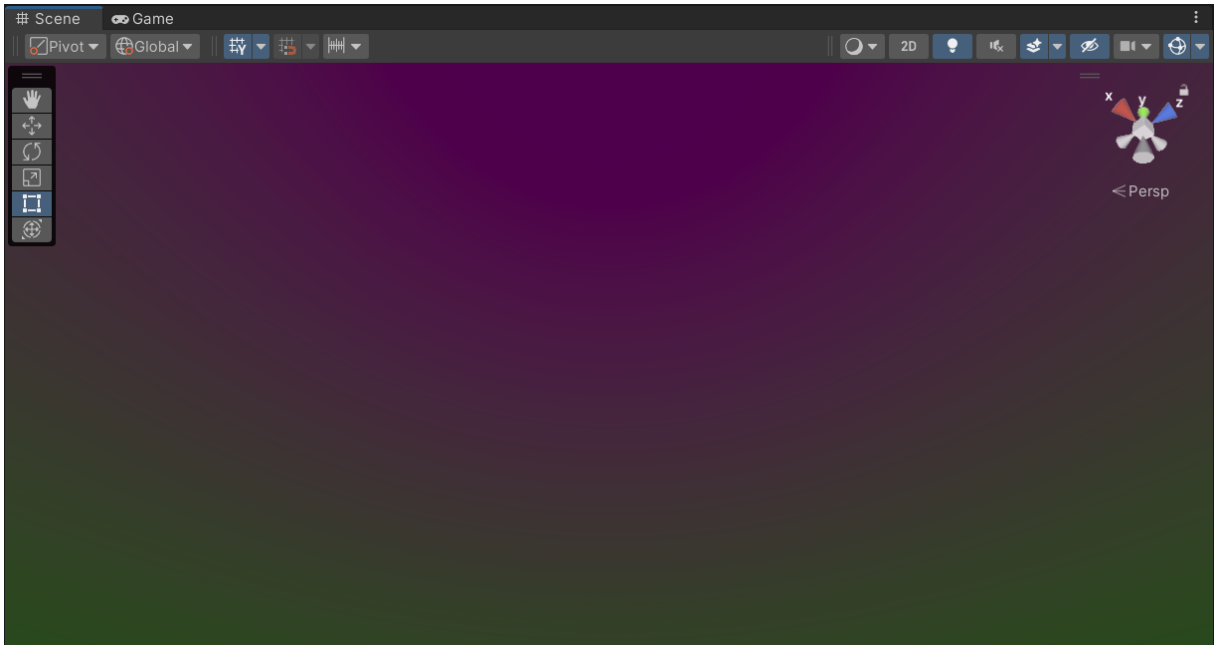
Postavljanje neba iznad scene u High Definition Rendering Pipelineu izvodi se na drugačiji način u odnosu na prethodna dva projekta. HDRP korisniku omogućuje postavljanje neba podešavanjem objekata tipa *Volume*. Kako bismo podesili glavno nebo iznad scene, možemo iskoristiti objekt *Sky and Fog Volume* unaprijed kreiran prilikom postavljanja projekta. Ne samo da High Definition Rendering Pipeline nudi mogućnost postavljanja neba korištenjem *Volume* objekata, već nudi i tri tipa oblikovanja neba koji nisu prisutni u dva ranije opisana rendering pipelinea. Tri vrste neba koje HDRP korisniku stavlja na raspolaganje su *Physically Based Sky*, *Gradient Sky* te *HDRI Sky*. Možemo uočiti nedostatak *Skyboxa*, no kako će u nastavku teksta biti pojašnjeno, *HDRI Sky* u neku ruku zamjenjuje *Skybox*.

*Physically Based Sky* metoda je renderiranja neba koja se temelji na tome da se nebo renderira na temelju raznih parametara koji opisuju uvjete u atmosferi u stvarnom svijetu koji utječu na izgled neba. *Physically Based Sky* se zapravo nastoji približiti što je više moguće stvarnom nebu. Izgled *Physically Based Skya* možemo podesiti na temelju raznih parametara kao što su količina aerosola u zraku, promjene u boji koje se dešavaju zbog reflektiranja tla, zakrivljenost zemlje i slično. Budući da objekt *Sun*, *Directional Light* simulira sunce na sceni, podešavanje pozicije i jačine svijetla koje baca objekt Sun također utječe na izgled *Physically Based Skya* baš kao što i u prirodi sunce utječe na izgled nebeskoga svoda.




Slika 17: *Physically Based Sky*

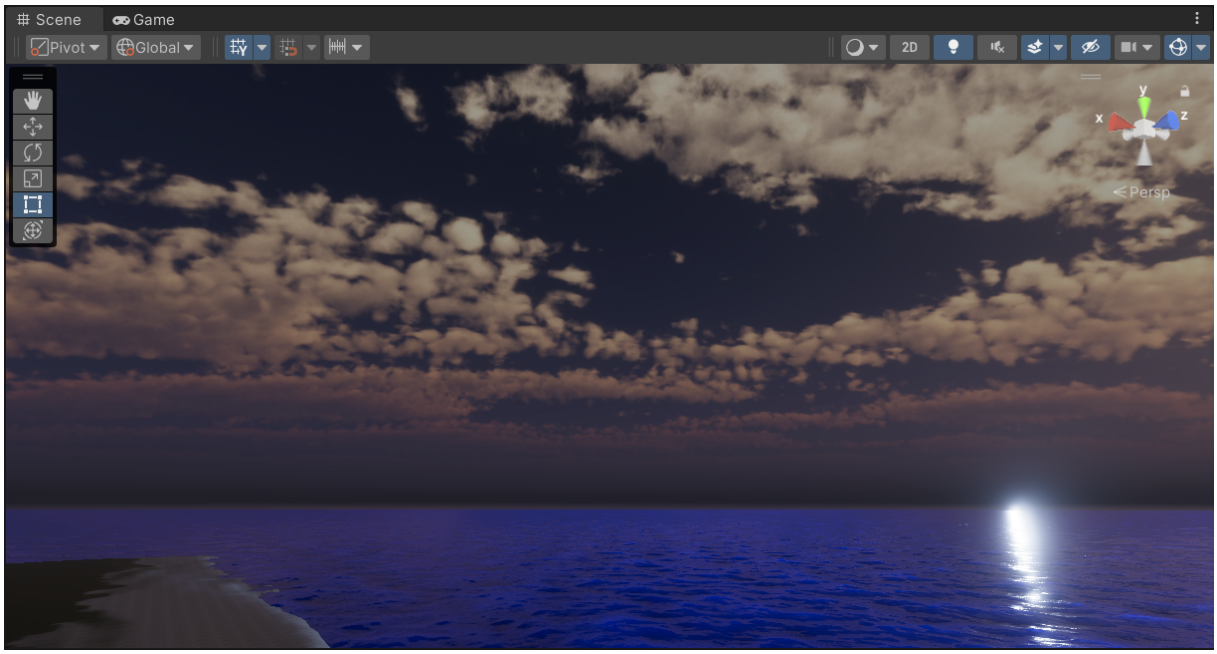
Ideja i funkcioniranje *Gradient Skya* poprilično su jasni već iz samog imena koncepta. *Gradient Sky* korisniku omogućuje da nebo iznad scene postavi izradom gradijenta od tri boje. Korištenjem *Gradient Skya* moguće je dobiti neke doista neprirodne boje na nebu.




Slika 18: *Gradient Sky*

*HDRI Sky* zapravo je, u svojoj suštini, vrlo sličan konceptu *Skyboxa* upoznatog u prethodnim poglavljima. *HDRI Sky* opcija renderira izgled neba na temelju kubne mape izrađene pomoću slike neba visoke rezolucije. Kako bi se nebo moglo renderirati potrebno je u projekt uvesti konkretnu datoteku kubne mape prema kojoj želimo da nebo bude kreirano. Pokazna

kubna mapa neba ponovo je preuzeta s Unity Asset Storea.




Slika 19: *HDRI Sky*

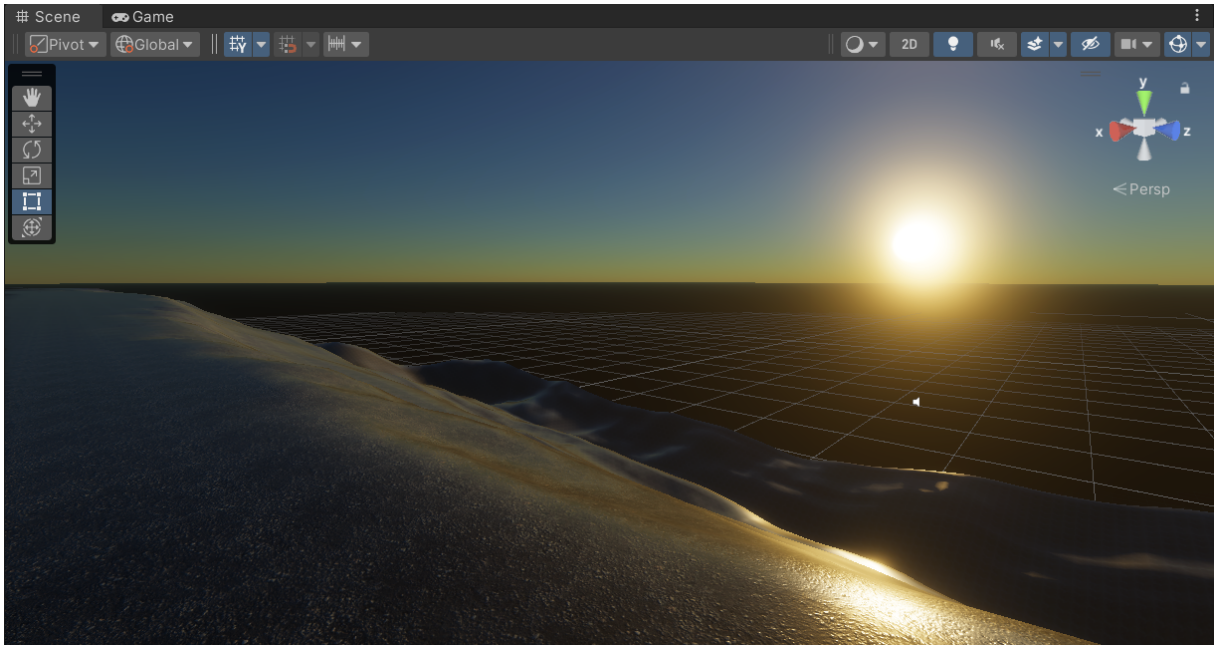
Još jedna od mogućnosti koje kod postavljanja neba High Definition Rendering Pipeline nudi je i uključivanje voluminoznih oblaka (engl. *Volumetric Clouds*). Oni se također jednostavno dodaju kao override na *Volume* koji je korišten za postavljanje neba. Osim voluminoznih oblaka, High Definition Rendering Pipeline također nudi mogućnost postavljanja voluminozne magle (engl. *Volumetric Fog*) na scenu. Magla se uključuje na sličan način kao i oblaci, to jest kao override na *Volume* objekt.

### 6.1.3. Postavljanje vodene površine na scenu

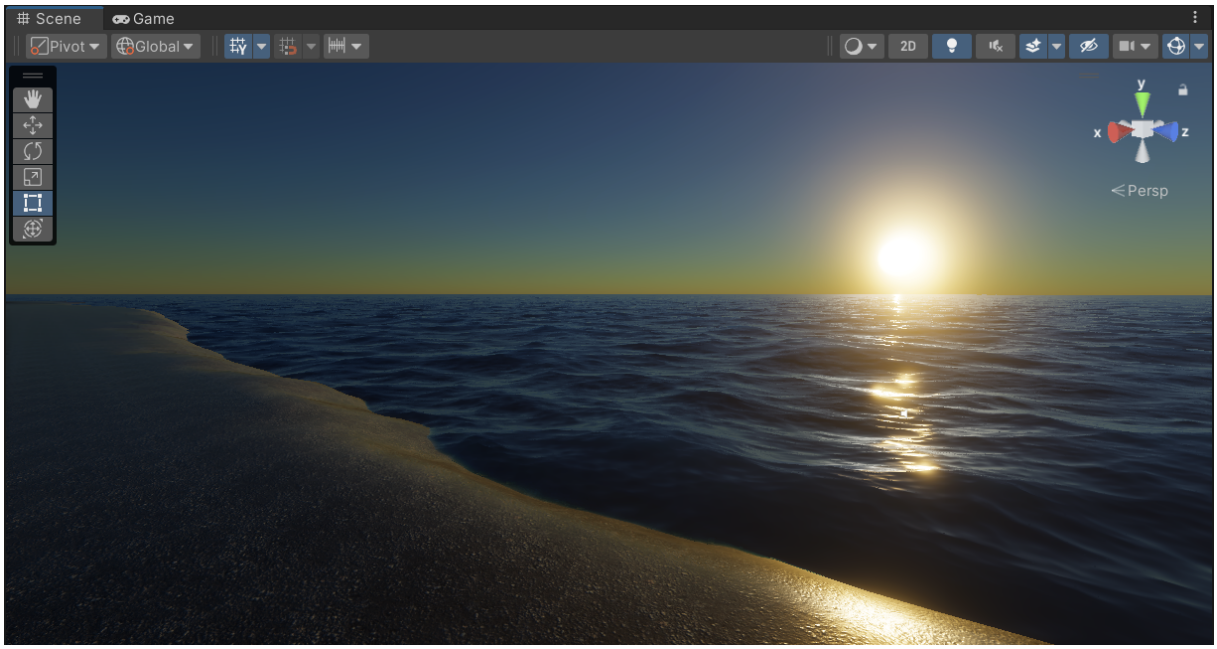
High Definition Rendering Pipeline ima također jednu zanimljivu specifičnost, a to je da uvelike olakšava postavljanje vodenih površina na scenu. Ta mogućnost nije podržana u prethodno opisanim rendering pipelineovima. Kako je u ostalim rendering pipelineovima da bi se na mapi dobila vodena površina bilo potrebno uložiti mnogo više truda i poslužiti se s nekoliko različitih metoda, u High Definition Rendering Pipelineu je čitava stvar uvelike olakšana dodavanjem *Water Surface* objekata. Sustav dodavanja i oblikovanja vodenih površina u High Definition Rendering Pipelineu je, u suštini, sprega rada dvaju objekata, jedan pomoću kojeg se vodena površina na sceni iscrtava te jednog pomoću kojeg se izgled te vodene površine kontrolira.

Kako bi dobili vodenu površinu na sceni, konkretno kako bi dodali more koje oplakuje teren, potrebno je najprije kreirati novi objekt tipa *Water Surface*. Možemo primijetiti da se samim dodavanjem objekta ili mijenjanjem njegovih postavki unutar prozora *Inspector* ne događa ništa. To je zato jer je potrebno unutar *Volume* objekta uključiti override *Water Rendering*. Override *Water Rendering* je uključen na objektu *Sky and Fog Volume* budući da je on globa-

lan, a vodu želimo učiniti vidljivom na čitavoj sceni. Nakon što smo uključili *Water Rendering* na objektu *Sky and Fog Volume*, možemo primijetiti da se voda doista pojavila na sceni.

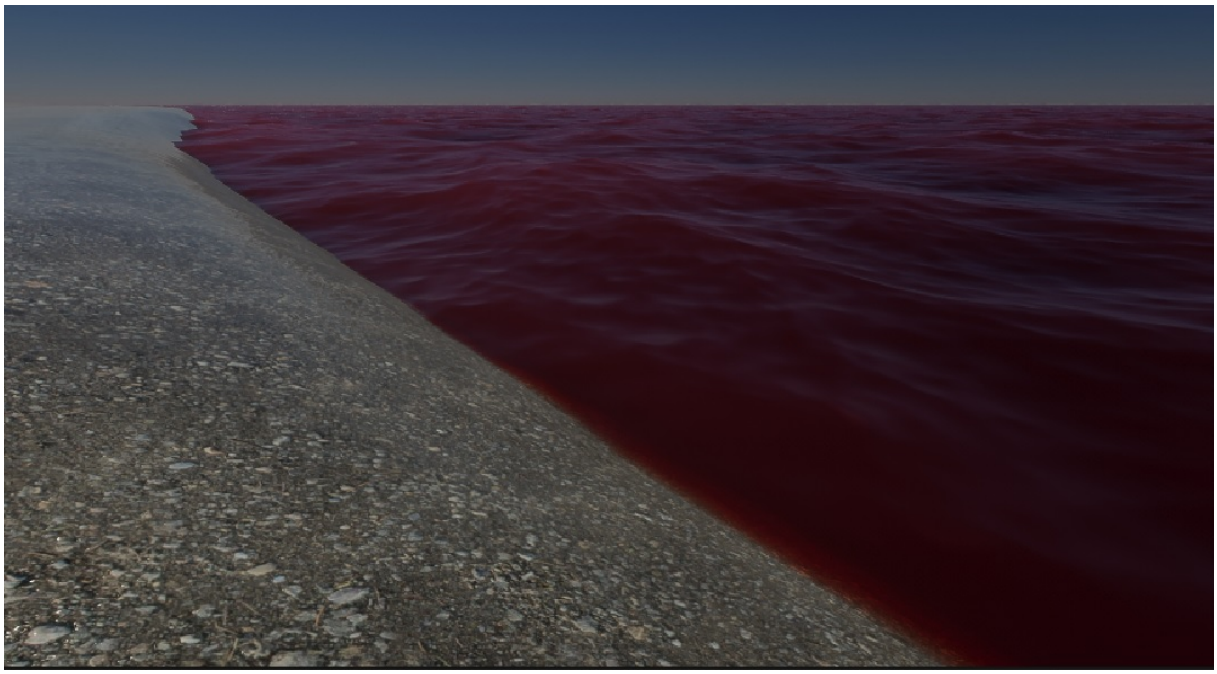



Slika 20: Scena prije dodavanja overridea *Water Rendering* objektu *Sky and Fog Volume*




Slika 21: Scena nakon dodavanja overridea *Water Rendering* objektu *Sky and Fog Volume*

Sve daljnje manipulacije vodenom površinom odvijaju se preko objekta tipa *Water Surface*. Ukoliko, primjerice, želimo povisiti razinu vode, to možemo učiniti pomicanjem objekta *Water Surface* po Y-osi u pozitivnom smjeru. Također možemo i, primjerice, promijeniti boju vode.



Slika 22: Vodena površina nakon mijenjanja boje objekta u crvenu

Također, ukoliko želimo da se simulira podvodni prikaz ako se kamera nađe ispod površine, potrebno je uključiti opciju *Under Water* na objektu *Water Surface*.

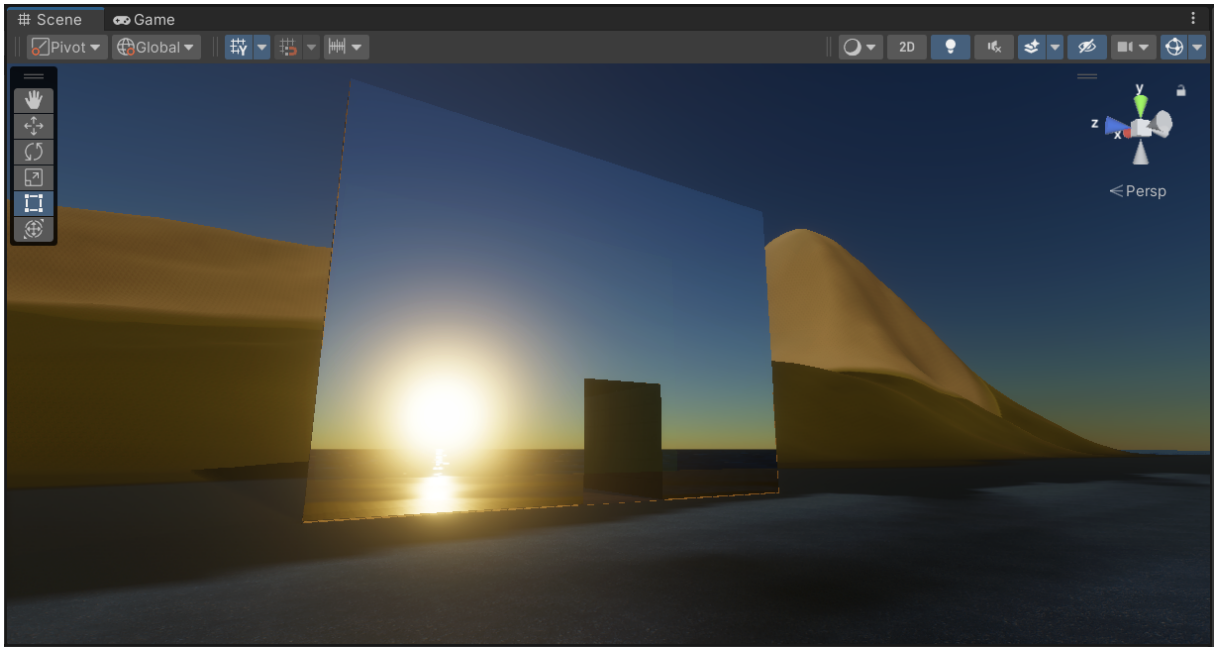
### 6.1.4. Osvjetljenje

High Definition Rendering Pipeline nudi najnaprednije mogućnosti osvjetljenja u odnosu na ostale rendering pipelineove. Postavljanje osvjetljenja na sceni započelo je s postavljanjem glavnog izvora svijetla. Glavni izvor svijetla na sceni je *Light* objekt *Sun*. Objekt *Sun* je postavljen kao svjetlosni objekt tipa *Directional Light*. Kada neki svjetlosni objekt postavimo kao *Directional Light*, možemo odmah uočiti neke promjene u odnosu na prethodne rendering pipelineove, usprkos tome što je *Directional Light* kao vrsta osvjetljenja postojala i u Built-in Rendering Pipelineu i u Universal Rendering Pipelineu. High Definition Rendering Pipeline teži fotorealizmu i simulaciji stvarnosti kao što to, naprimjer, ranije opisani *Physically Based Sky* indicira. Tako je i sve vrsta osvjetljenja moguće podešavati prema lako razumljivim parametrima i mjernim jedinicama koje se koriste u stvarnosti. Pojasnimo. Intenzitet svijetla može se podešavati na način da se njegova vrijednost zada u mjernoj jedinici lux koja je standardna mjerna jedinica za intenzitet svijetla. Prema tome, kako bi se dobila scena koja odgovara stvarnim uvjetima za vedrog dana, intenzitet svijetla je jednostavno postavljen na vrijednost od 20000 lux. Također, temperatura svijetla se izražava u mjernoj jedinici Kelvin koja je standardna mjerna jedinica za izražavanje parametra topline svjetala u stvarnom svijetu.

U pogledu osvjetljenja, još neke od novih mogućnosti koje High Definition Rendering Pipeline pruža su dodatni oblici svijetla ukoliko koristimo *Spot* ili *Area* tip svijetla. Također jedan od noviteta u odnosu na prethodna dva rendering pipelinea je i mogućnost korištenja IES profila za pojedine izvore svijetla. IES profili su zapravo datoteke koje sadrže definicije raznih

parametara svjetala kao što su temperatura i intenzitet. IES profile ne može se koristiti na *Light* objektima koji su postavljeni kao *Directional Light*. IES profile moguće je kreirati vlastoručno ili ih je moguće preuzeti iz raznih izvora. Brojni proizvođači svjetala nude na svojim stranicama IES profile svojih proizvoda. Konkretno, u ovom je projektu eksperimentirano s IES profilima baziranima na svijetlima proizvedenim od strane tvrtke Philips koji su preuzeti sa stranice tvrtke.

Kako bi se dodatno demonstriale neke napredne mogućnosti osvjetljenja koje High Definition Rendering Pipeline pruža, na sceni je kreirana građevina u čijem će interijeru te mogućnosti biti demonstrirane.




Slika 23: Građevina postavljena za potrebe demonstracije osvjetljenja u interijeru

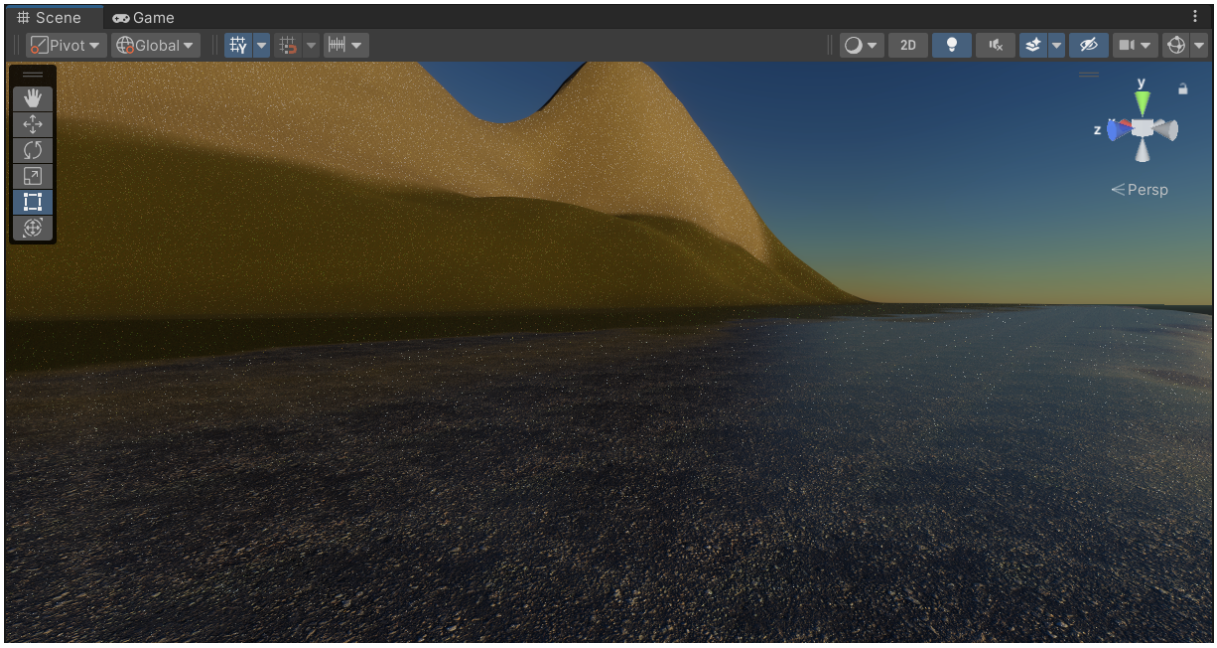
## 6.2. Ray tracing

Svakako najnaprednija grafička mogućnost koju High Definition Rendering Pipeline pruža su razne metode ray tracinga. Ray tracing je skupni naziv za više različitih metoda praćenja putanja zraka svijetla na sceni i njihovih interakcija s okolinom. Konkretno, u ovom će poglavlju biti opisani pokušaji implementacije path tracinga i ray tracinga kod ambijentalne okluzije, kao i njihove prednosti i nedostaci.

### 6.2.1. Path tracing

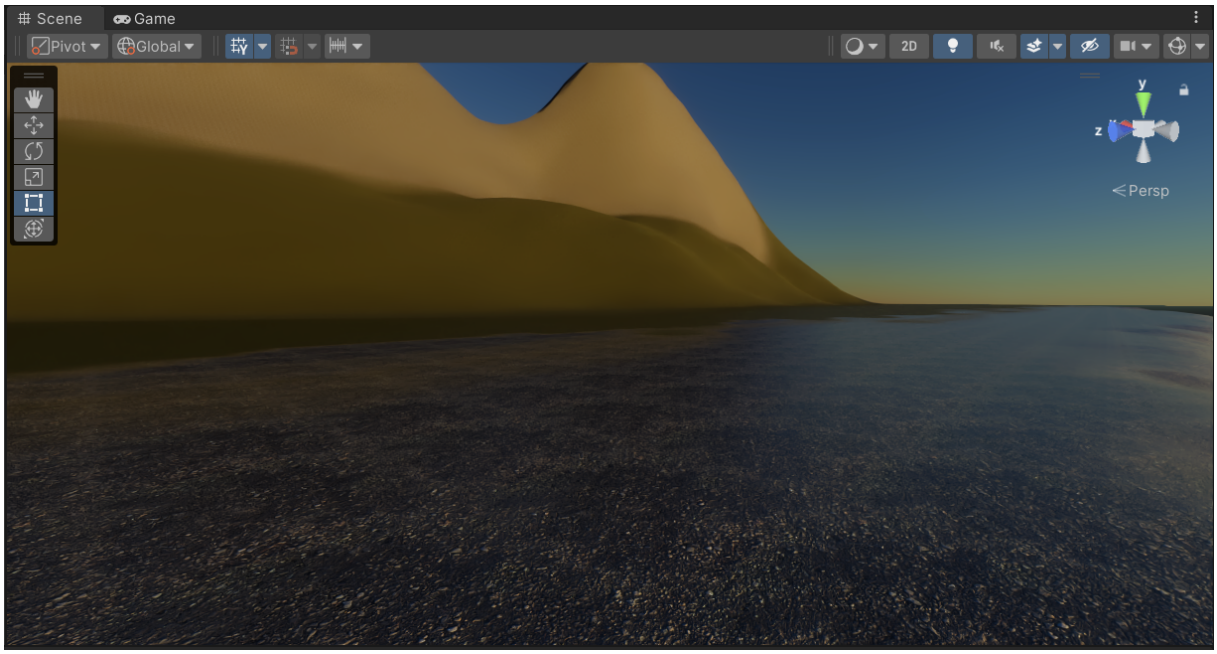
Prva ray tracing metoda koja je isprobana na sceni bio je path tracing. Path tracing je, pojednostavljeno, metoda za preciznije izračunavanje načina na koji se zrake svijetla odbijaju od površina[6]. Path tracing se, kao i svi ostali ray tracing efekti na sceni, dodaje u obliku overrideova na *Volume* objekte. Path tracing je primijenjen na objektu *Sky and Fog Volume*. Kad uključimo *Path Tracing* override na objekt *Sky and Fog Volume*, možemo primijetiti niz parametara pomoću kojih možemo modificirati primijenu tog efekta na sceni. Path tracing funkcionira

na principu prikupljanja uzoraka scene, odnosno na principu izračunavanja putanji više različitih zraka svijetla te renderiranja finalne slike na temelju prikupljenih uzoraka. Jedan od parametara koje možemo podesiti je broj uzoraka. Zaključak koji se nameće i bez dodatnih testiranja je da će slika biti kvalitetnija ovisno o tome koliki broj uzoraka uzmemo. Podešavanja ovog parametra pokazala su da je optimalni broj uzoraka za dobivanje kvalitetne slike oko 250. Ukoliko se uzme premali broj uzoraka, slika će biti mutna. Kao manu primjene ovog efekta možemo izdvojiti rast vremena renderiranja s povećanjem broja uzoraka. Također jedan od značajnijih parametara koje možemo izdvojiti je i broj odbijanja koja se uzimaju u kalkulaciju za svaku zraku svijetla.




Slika 24: Slika renderirana uz primjenu path tracinga (256 uzoraka)

Na dobivenoj slici možemo primijetiti značajan broj šumova. Šumovi nastaju zbog činjenice da se slika renderira na temelju više različitih uzoraka. Šumove je moguće ukloniti korištenjem denoising metoda. U High Definition Rendering Pipelineu dostupne su dvije denoising metode, Intel Open Image Denoise i Nvidia Optix Denoise.



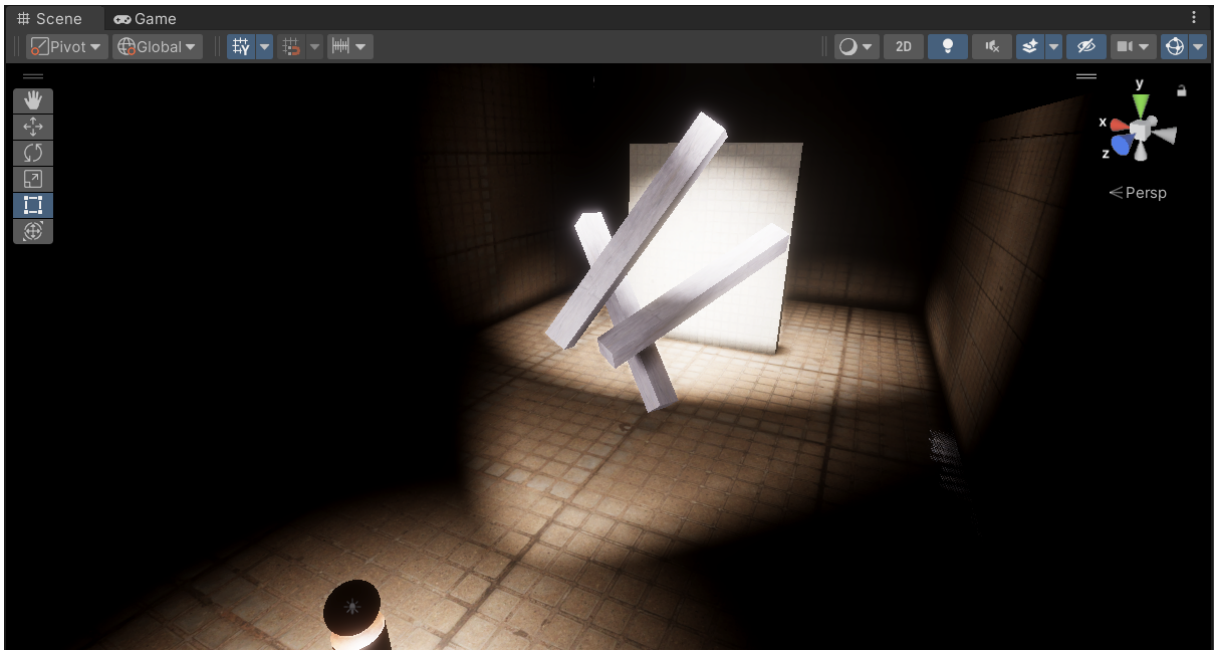

Slika 25: Slika renderirana uz primjenu path tracinga i denoisinga (256 uzoraka)

Path tracing je procesorski zahtjevna metoda i njena je primjena na scenama u većini slučajeva neoptimalna. S obzirom na veliki broj izračuna koje je potrebno napraviti za renderiranje svake slike, njena primjena bez žrtvovanja performansi izuzetno je zahtjevna. Preporučljivo je ne primjenjivati ju na čitavoj sceni, već je primjenjivati na statičnim slikama ili u slučajevima u kojima je potrebne vizuale moguće unaprijed renderirati, primjerice kod animacija u igri.

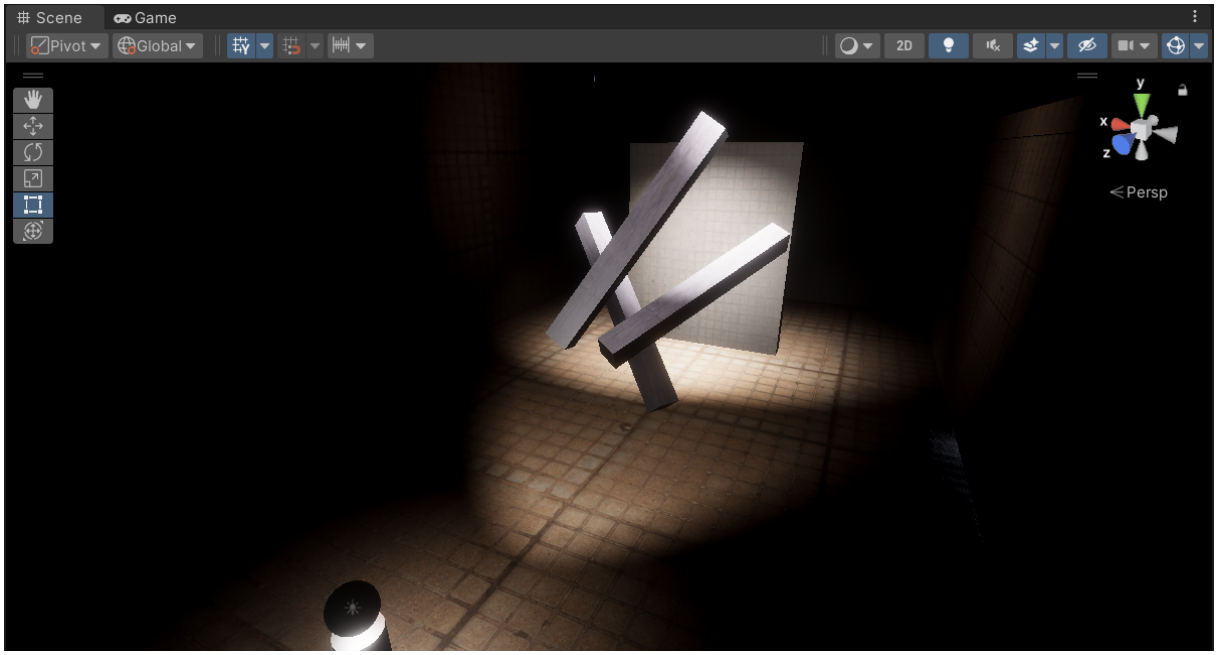
### 6.2.2. Ray tracing kod ambijentalne okluzije

Metode praćenja zraka svijetla također mogu pomoći i u kreiranju realističnije ambijentalne okluzije. Praćenje zraka svijetla i precizno izračunavanje njihove interakcije s objektima na sceni omogućuje kreiranje kvalitetnijeg ambijenta s osvjetljenim i osjenčanim prostorima. Kako bi se demonstrirale ove mogućnosti, u interijeru građevine postavljene na sceni kreiran je novi *Volume* objekt pod nazivom *Beach House Volume*. U tom je volumeu isključeno djelovanje glavnog izvora svijetla kako bi svo osvjetljenje dolazilo od nekoliko izvora svijetla tipa *Point* i *Spot*. U interijer građevine postavljeno je i par objekata pomoću kojih možemo pratiti interakciju svijetla s objektima pošto je ray tracing uključen.

Kako bi dobili ray traced ambijentalnu okluziju, odabranom volumeu potrebno je pridodati override *Screen Space Ambient Occlusion*. Unutar tog overridea možemo podešavati kvalitetu ray tracing efekta preko raznih parametara kao što su intenzitet i duljina zrake svijetla. Također, i ovaj tip ray tracinga funkcionira na principu prikupljanja uzoraka slike pa je tako i ovdje moguće podešavati broj uzoraka koji će prije svakog renderiranja biti prikupljeni, no uobičajeni brojevi uzoraka koje je potrebno prikupiti za kvalitetnu sliku je ipak nešto manji nego kod path tracinga. I ovdje možemo uočiti šumove na slici ukoliko se ne uključi opcija *Denoise*.




Slika 26: Interijer građevine bez primjene ray tracinga




Slika 27: Interijer građevine uz primjenu ray tracinga

# 7. Zaključak

Na temelju svega izrađenog i opisanog možemo zaključiti da sustav od tri rendering pipelinea s kojima je moguće raditi u Unity game engineu ima svoje prednosti i nedostatke. Sa samog tehničkog aspekta, mogućnosti koje pruža svaki od rendering pipelineova su zadovoljavajuće i svaki rendering pipeline nesumnjivo služi svrsi. Kao nedostatak takvog sustava mogli bismo navesti činjenicu da se developeri video igara moraju temeljito upoznati sa svakim od njih kako bi mogli procijeniti koji je rendering pipeline najbolje primijeniti s obzirom na prirodu projekta na kojem žele raditi. Također, upoznavanje s mogućnostima svakog od njih nužno je i zbog činjenice da je odluku o tome koji će rendering pipelinea biti primijenjen na projektu potrebno donesti prije samog kreiranja novog projekta te da je teško ili čak nemoguće promijeniti rendering pipeline tijekom rada na projektu.

Zaključak samog tehničkog osvrta na rendering pipelineove u Unityju bio bi da je svaki od njih dovoljno različit po mogućnostima koje pruža od ostalih kako bi opravdao svoju svrhu.

Možemo tako donijeti neki općeniti zaključak da je Built-in Rendering Pipeline najpogodniji za neke jednostavnije projekte u kojima nije potrebno koristiti naprednije grafičke mogućnosti, a potrebe za prilagodbom sustava renderiranja nisu velike.

Universal Rendering Pipeline mogli bismo smatrati rendering pipelineom općenite namjene jer je prema svojim mogućnostima primjenjiv na najveći spektar projekata. URP developerima nudi veliki broj alata i mogućnosti kako bi mogli kreirati projekte veće ili manje složenosti, a uz to imati i gotovo potpunu kreativnu slobodu jer nisu zakinuti tehničkim ograničenjima kao u Built-in Rendering Pipelineu budući da je URP skriptabilni rendering pipeline u kojem je sustav renderiranja moguće posve prilagoditi vlastitim potrebama.

High Definition Rendering Pipeline najkompleksniji je od tri rendering pipelinea, ali svakako i najnapredniji. Najbolje ga je primijeniti u projektima u koji teže fotorealističnoj grafici i simulaciji stvarnosti. Budući da pruža najviše, period učenja rada s High Definition Rendering Pipelineom je i najduži. Ipak, u suvremenom kontekstu, s obzirom na stanje industrije video igara i neprestanim nastojanjima da igre izgledaju bolje i realističnije, isplati se uložiti trud u upoznavanje i savladavanje ove tehnologije kako bi proizvodi koje radimo bili konkurentni i primamljivi igračima.

# Popis literature


[1] Unity Technologies. „Render pipelines introduction.” (Godina nije navedena.), adresa: https://docs.unity3d.com/Manual/render-pipelines-overview.html (pogledano 14. 8. 2023.).

[2] Unity Technologies. „Render pipeline feature comparison.” (Godina nije navedena.), adresa: https://docs.unity3d.com/2022.3/Documentation/Manual/render-pipelines-feature-comparison.html (pogledano 14. 8. 2023.).

[3] Unity Technologies. „CharacterController.” (Godina nije navedena.), adresa: https://docs.unity3d.com/ScriptReference/CharacterController.html (pogledano 17. 8. 2023.).

[4] Unity Technologies. „High Definition Render Pipeline overview.” (Godina nije navedena.), adresa: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@8.1/manual/index.html (pogledano 17. 8. 2023.).

[5] Unity Technologies. „Getting started with ray tracing.” (Godina nije navedena.), adresa: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@8.1/manual/Ray-Tracing-Getting-Started.html (pogledano 18. 8. 2023.).

[6] B. Caulfield. „What Is Path Tracing?” (23.3.2022.), adresa: https://blogs.nvidia.com/blog/2022/03/23/what-is-path-tracing/ (pogledano 18. 8. 2023.).

[7] Unity Technologies. „Using the Built-in Render Pipeline.” (Godina nije navedena.), adresa: https://docs.unity3d.com/Manual/built-in-render-pipeline.html (pogledano 2. 9. 2023.).

[8] Unity Technologies. „Universal Render Pipeline overview.” (Godina nije navedena.), adresa: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@17.0/manual/index.html (pogledano 3. 9. 2023.).

[9] Unity Technologies. „Lit Shader.” (Godina nije navedena.), adresa: https://docs.unity3d.com/Packages/com.unity.render-pipelines.lightweight@5.10/manual/lit-shader.html (pogledano 3. 9. 2023.).

[10] Unity Technologies. „Example: How to create a custom rendering effect using the Render Objects Renderer Feature.” (Godina nije navedena.), adresa: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@17.0/manual/containers/how-to-custom-effect-render-objects.html (pogledano 4. 9. 2023.).

[11] Unity Technologies. „Scriptable Render Pipeline introduction.” (Godina nije navedena.), adresa: https://docs.unity3d.com/Manual/scriptable-render-pipeline-introduction.html (pogledano 6. 9. 2023.).

# Popis slika

# 1. Prilozi

Unity pakete izrađenih pokaznih scena moguće je preuzeti na linku: https://drive.google.com/drive/folders/10pCb_C5mzZ5nV2wVgIxvjIbXcXipZGUj?usp=sharing