Tehnička škola Čakovec

ELABORAT ZAVRŠNOG RADA

POST PROCESSING SHADERI

Mentor:

Krešimir Kočiš, dipl. ing. računarstva

Učenik:

Jakov Biškup, 4.RT

Čakovec, svibanj 2024.

Tehnička škola Čakovec

Prosudbeni odbor za završni rad

Učenik: **Jakov Biškup**

Razred: **4. RT**

Školska godina: **2023./2024.**

Obrazovno područje: **računalstvo**

Zanimanje: **tehničar za računalstvo**

Naziv zadatka: **POST PROCESSING SHADERI**

Opis zadatka: Zadatak je na *Unity* scenu dodati veći broj *post processing* efekata kako bi se postigla realnija ili stilizirana slika. *Post processing* efekti primjenjuju se na cijeli zaslon nakon što je slika već iscrtana. Kako bi efekti radili u stvarnom vremenu, potrebno ih je implementirati u obliku *shader* programa koji se izvode na grafičkoj kartici.

Učenik će se za konzultacije obratiti svom mentoru.

Zadatak zadan:	Rok predaje pisanog rada:	Predviđeni datum obrane:
10. listopada 2023.	17. svibnja 2024.	13.-18. lipnja 2024.

Mentor:

Krešimir Kočiš, dipl. ing. računarstva

SADRŽAJ

1. UVOD

Kada *3D engine* završi s renderiranjem slike, ona često izgleda neupečatljivo i nerealistično te ne dopušta umjetnički izražaj. Kako bi se postigao realističniji izgled ili željeni stil, često se koriste *post processing* efekti. To su efekti koji se primjenjuju na cijelu sliku nakon što je renderirana, ali prije nego što se prikaže na zaslonu. Oni manipuliraju sliku kako bi promijenili izgled scene (npr. dodavanje magle), replicirali izgled fotografije (npr. simuliranje dubinske oštrine[1]) ili promijenili izgled boja (npr. veća ili manja zasićenost boja). Kako bi to postigli, moraju izračunavati velik broj operacija za svaki piksel, što je neizvedivo u stvarnom vremenu ako se koristi procesor, odnosno *CPU* (za *Full HD* rezoluciju to je ukupno $1920 \cdot 1080 = 2073600$ piksela, odnosno minimalno 2073600 operacija svakih nekoliko milisekundi). Zbog toga se *post processing* efekti implementiraju kao *shader* programi koji se izvode na grafičkoj kartici. Ona je specijalizirana za operacije nad slikama na način da se koristi *SPMD*[2] pristupom gdje se ista *shader* funkcija izvodi za svaki piksel u nekoj teksturi. To omogućuje milijune izračuna u djeliću sekunde te time i korištenje *post processing* efekata u stvarnom vremenu za videoigre i slične medije. Ipak, iako grafička kartica može napraviti velik broj izračuna u kratkom vremenu, za razliku od procesora, znatno je usporavaju neke jednostavne operacije kao što su grananje i petlje te vrijeme potrebno za prijenos podataka od procesora do grafičke kartice. Zbog toga je potrebno optimizirati *shadere* kako bi minimalno povećali vrijeme renderiranja slike.

[1] zamućenost dijelova fotografije koji nisu u fokusu
[2] *single program, multiple data*: model izračunavanja koji paralelno izvodi jednu operaciju nad više različitih podataka

Cilj je rada istražiti i implementirati veći broj *post processing* efekata koji na različite načine utječu na izgled scene. Oni se moraju izvoditi u stvarnom vremenu, a kombiniranjem efekata postiže se realističnija ili stilizirana slika.

Za izradu završnog rada korišten je *Unity*[3] koji renderira i sjenča *3D* scenu, znatno olakšava komunikaciju s grafičkom karticom te omogućuje pisanje *cross-platform shader* programa. *Unity* se programira uz pomoć *C#* programskog jezika, a za definiranje *shader* objekta koristi se deklarativni jezik *ShaderLab*. Za pisanje samog *shader* koda koristi se *HLSL* ili *CG*[4]. Efekti se za demonstraciju primjenjuju na besplatnu *3D* scenu[5] šume preuzetu s interneta.

[3] popularan besplatni *3D game engine*
[4] *High-Level Shader Language* ili *C for Graphics*: programski jezik za pisanje *shadera* koji razvijaju *Microsoft* i *Nvidia* za *DirectX* API
[5] *Fantasy Forest Environment - Free Demo;* preuzeto s *Unity Asset Storea:* https://assetstore.unity.com/packages/3d/environments/fantasy/fantasy-forest-environment-free-demo-35361 (pristupljeno 19. veljače 2024.)

2. RAZRADA

Prije početka programiranja *shader* programa potrebno je poznavati osnovne pojmove vezane uz računalnu grafiku te tijek renderiranja (engl. *rendering pipeline*). Također ih je potrebno implementirati u *Unity* okruženju kako bi se primjenjivali na renderiranu sliku na željeni način.

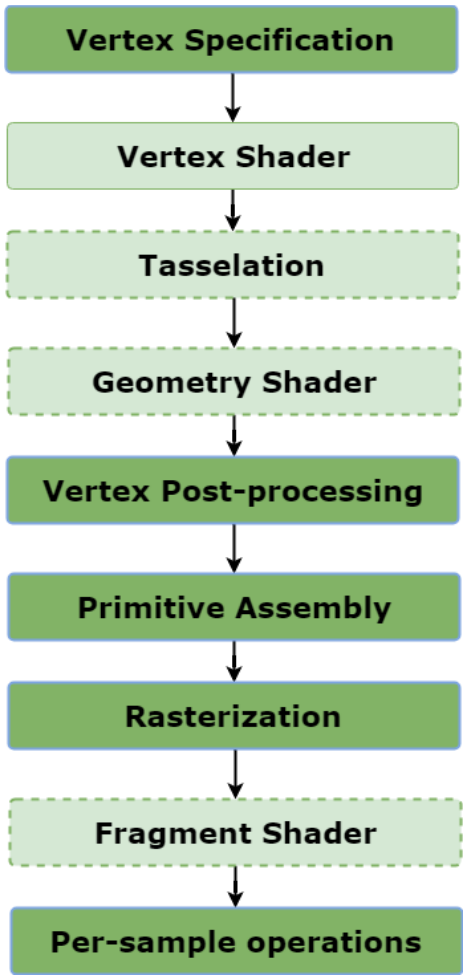
2.1 TIJEK RENDERIRANJA

Tijek renderiranja je niz koraka koje računalo radi kako bi iscrtalo sliku na ekranu. Primjer tijeka prikazuje *Slika 2.1*. Prvi korak je zadati pozicije i ostale atribute za svaki vrh (engl. *vertex*) svakog objekta u sceni. Tada se za svaki od tih vrhova pokreće *vertex shader* koji manipulira dobivenim podatcima kako bi objekti u sceni mogli biti adekvatno prikazani na ekranu. Pozicije vrhova na početku su zapisane kao trodimenzionalni vektori u prostoru objekta[6]. One se množe s matricama transformacija kako bi se pretvorile iz točke u prostoru u koordinate na ekranu. Prvo se transformiraju u prostor svijeta[7] kako bi se prikazala pozicija, rotacija i veličina pojedinog objekta u odnosu na ostale objekte. Vrhovi se nakon toga transformiraju u prostor kamere[8] jer to znatno pojednostavljuje izračune koji slijede. Na kraju se vrhovi transformiraju u prostor odsijecanja[9] koristeći se matricom projekcije. Nakon te transformacije i homogenizacije vektora pozicije dijeljenjem *w* komponentom nastaje perspektiva te izračunate *x* i *y* koordinate vrhova direktno koreliraju s pozicijom točke na ekranu. Promjenom *vertex shadera* mijenja se oblik objekta što se koristi za primijene kao što je generiranje proceduralnih valova, ali se gotovo nikad ne koristi pri izradi *post processing* efekata jer se oni ne primjenjuju na trodimenzionalni objekt, već na sliku.

Nakon *vertex shadera* odsijeku se (engl. *clip*) vrhovi koji nisu vidljivi na kameri te se između preostalih vrhova formiraju trokuti. Tada slijedi rasterizacija: proces koji vrhove i trokute pretvara u rastersku sliku odnosno skup piksela. Posljednji korak je *fragment shader*

[6] engl. *object space*: koordinatni sustav u kojem je definiran model, ishodište se najčešće postavlja u donji lijevi kut ili centar modela; za spremanje relativne pozicije vrha unutar modela
[7] engl. *world space*: koordinatni sustav koji je zajednički svim objektima u sceni; pozicija vrha objekta u svijetu
[8] engl. *camera space*: koordinatni sustav kojem je kamera u ishodištu te orijentirana na način da ona gleda po *z* osi
[9] engl. *clip space*: koordinatni sustav u kojem se prostor vidljivosti kamere zapisuje u prostor od -1 do 1 po svakoj osi

poznat i pod imenom *pixel shader* koji se pokreće jedanput za svaki piksel. On određuje boju pojedinog piksela te će se u nijemu izračunavati sjenčanje objekta. Ovo je najvažniji dio tijeka renderiranja za izradu *post processing* efekata je će se izmjenama *fragment shadera* postići mnogi od njih.



Slika 2.1. primjer tijeka renderiranja[10]

[10] izvor: GeeksForGeeks, OpenGL Rendering Pipeline | An Overview, https://www.geeksforgeeks.org/opengl-rendering-pipeline-overview/ (pristupljeno 19. veljače 2024.)

2.2 OSNOVNI POJMOVI I PRINCIPI

Početak svakog *fragment shadera* su *UV* koordinate. U kontekstu *post processing* efekata, *UV* koordinate označavaju normalizirane koordinate piksela u prostoru zaslona gdje je ishodište donji lijevi kut slike, a točka *(1, 1)* gornji desni kut slike (neovisno o omjeru stranica ekrana). One se koriste za razlikovanje piksela u *fragment shaderu*. Najčešće se vizualiziraju u crvenom i zelenom kanalu kao što prikazuje *Slika 2.2*.

Slika 2.2. vizualizacija UV koordinata

S*hader* program sastoji se od jednog ili više prolaza (engl. *pass*). Kada se *shader* primjenjuje na teksturu pokreću se svi prolazi redoslijedom u kojem su definirani, ako nije zadani određeni prolaz. Svaki od njih ima svoj *vertex* i *fragment shader*. Funkcije i varijable u *shaderu* mogu biti dostupne svim prolazima tog *shadera* ili samo za pojedini prolaz. Kompleksni efekti koji se sastoje od više jednostavnijih efekata, ali ipak profitiraju od nekih zajedničkih parametara i funkcija, koriste više prolaza.

Spremnik dubine (engl. *depth buffer*) posebna je tekstura koja sadrži udaljenosti svakog piksela od kamere te se koristi pri renderiranju *3D* scene. Ova je tekstura vrlo korisna za izradu velikog broja *post processing* efekata te će se vrlo često koristiti u ovom radu. Udaljenost je spremljena u nelinearnom obliku u rasponu od 0 do 1, pa ju je često potrebno linearizirati kako bi se dobila udaljenost u prostoru svijeta. Izgled spremnika dubine prikazuje *Slika 2.3* gdje se brojevi od 0 do 1 prikazuju kao crno-bijela slika.

Slika 2.3. prikaz spremnika dubine

Prilikom programiranja *shader* programa većina podataka (pozicije, boje i sl.) sprema se u obliku vektora. Tip podataka vektora se u *HLSL* jeziku sastoji od jednostavnog tipa podataka i broja koji označava broj dimenzija vektora (npr. *float3* označava trodimenzionalni vektor koji sadrži decimalne brojeve s pomičnom točkom). Kako bi se pojednostavila manipulacija vektora, *HLSL* i slični jezici uvode *swizlling*. Članovima vektora pristupa se sa sufiksom *.x* za prvi član, *.y* za drugi član, *.z* za treći član itd. Mogu se koristiti slova *x, y, z* i *w* (najčešće pri spremanju pozicije ili opće informacije u vektoru) ili *r, g, b,* i *a* (najčešće pri spremanju boje u vektoru). *Swizlling* omogućuje pristupanje više članova vektora odjednom te njihovo proizvoljno raspoređivanje tako da se koristi sufiks s kombinacijom tih slova (npr. *v1.xy* vraća dvodimenzionalni vektor koji se sastoji od prva dva člana vektora *v1*, a *v2.bgr* vraća trodimenzionalni vektor koji se sastoji od prva tri člana vektora *v2*, ali sa zamijenjenim prvim i trećim članom).

Boje se u grafičkom programiranju spremaju u jednom od dva oblika. To su niski dinamički raspon (*LDR*) ili visoki dinamički raspon (*HDR*). Niski dinamički raspon jednostavniji je jer se u njemu crveni, zeleni i plavi kanal spremaju kao realni brojevi iz intervala $[0, 1]$. Kako svjetlost u stvarnosti nema ograničenja na svoj intenzitet te da bi se postigla veća preciznost, često je bolje koristiti se visokim dinamičkim rasponom. Kada se koristi visoki dinamički raspon boje se spremaju u intervalu $[0, +\infty\rangle$. Tako se omogućuje spremanje vrlo tamnih i vrlo svijetlih boja u istoj teksturi bez gubitka preciznosti. Ipak, pri

korištenju visokog dinamičkog raspona dolazi do problema prilikom prikazivanja slike na zaslonu jer zaslon može prikazati samo boje u niskom dinamičkom rasponu. Zbog toga se slika prije prikazivanja mora prebaciti u *LDR* koristeći se mapiranjem tonova. Također neke grafičke kartice za mobitele i igrače konzole ne podržavaju spremanje tekstura u *HDR* obliku pa je za njih potrebno spremati boje u niskom dinamičkom rasponu.

Prilikom primjenjivanja *post processing* efekata na sliku vrlo je bitan redoslijed u kojem se primjenjuju. Postoje osnovna pravila kojih se treba pridržavati, ali sam redoslijed često ovisi o željenom efektu. Pravilo za određivanje redoslijeda primjene efekata glasi: Prvo se primjenjuju efekti koji se odnose na okoliš, zatim efekti koji repliciraju fenomene koje izaziva leća ili kamera, a na kraju efekti koji mijenjaju svojstva slike odnosno promjene koje bi se radile na fotografiji u programu kao što je *Adobe Photoshop* nakon što je ona fotografirana. *Tablica 2.1* prikazuje sve efekte objašnjene u ovom radu podijeljene u ove tri kategorije. Redoslijed unutar kategorije može varirati.

Tablica 2.1 Podjela efekata u kategorije za redoslijed primjenjivanja

Promjena izgleda scene	**Replikacija svojstva kamere**	**Promjena svojstva boja**
▪ magla	▪ zamućivanje ▪ izoštravanje ▪ *bloom* ▪ dubinska oštrina ▪ kromatska aberacija ▪ zrnatost filma ▪ vinjeta	▪ isticanje obruba ▪ korekcija boja ▪ mapiranje tonova ▪ kvantizacija boja ▪ pikselizacija

2.3 IMPLEMENTACIJA SHADERA

Unity game engine pruža velik broj alata koji olakšavaju rad sa *shaderima*. Implementacija započinje izradom *shader* datoteke koja sadrži *shader* definiran *ShaderLab* i *HLSL* jezikom. Za potrebe izrade *post processing shadera* najbolje je započeti s *Image effect* predloškom te ga prilagoditi potrebama efekta. Ovako izgleda osnovni *post processing shader* program koji ne mijenja izgled slike:

```
Shader "Hidden/TESTSHADER" // naziv shadera
{
    // svojstva koja se mogu mijenjati kroz Unity Inspector
    Properties
    {
        // potrebno kako bi shader radio
        // automatski postaje tekstura na koju se primjenjuje efekt
        _MainTex ("Texture", 2D) = "white" {}
    }

    SubShader
    {
        // konfiguracija postavki renderiranja:
        Cull Off ZWrite Off ZTest Always

        // dio shadera koji je zajednički svakom prolazu
        CGINCLUDE

        //zadaje da se vertex shader funkcija zove vert
        #pragma vertex vert
        //zadaje da se fragment shader funkcija zove frag
        #pragma fragment frag

        // koriste se Unity funkcije i makro naredbe
        #include "UnityCG.cginc"

        // podatci o svakom vrhu koje dobiva vertex shader
        struct appdata
        {
            float4 vertex : POSITION; // pozicija u prostoru objekta
            float2 uv : TEXCOORD0; // UV koordinate
        };
```

```
        // podatci koje vertex shader prosljeđuje fragment shaderu
        struct v2f
        {
            float2 uv : TEXCOORD0; // UV koordinate
            // pozicija u prostoru izrezivanja
            float4 vertex : SV_POSITION;
        };

        // vertex shader
        v2f vert (appdata v)
        {
            v2f o;
            // transformira poziciju iz prostora objekta
            // u prostor odsijecanja koristeći se makro naredbom
            o.vertex = UnityObjectToClipPos(v.vertex);
            o.uv = v.uv; // prosljeđuje UV koordinate
            return o; //prosljeđuje podatke do fragment shadera
        }
        ENDCG

        // prvi prolaz
        Pass
        {
            Name "PrviProlaz" // naziv prolaza

            CGPROGRAM
            // globalna varijabla, moguće ju je zadati u C# kodu
            // naziv započinje donjom crtom kako bi se razlikovale od
            // lokalnih varijabli
            sampler2D _MainTex;

            // fragment shader
            fixed4 frag (v2f IN) : SV_Target
            {
                // uzorkovanje teksture na dobivenim UV koordinatama
                fixed4 col = tex2D(_MainTex, IN.uv);
                return col; // fragment shader vraća očitanu boju
            }
            ENDCG
        }
    }
}
```

Kada se definira željeni *shader* potrebno ga je referencirati (koristeći se nazivom *shadera* ili u inspektoru) u *C#* programu te iz njega napraviti materijal. Na *Unity* kameru se dodaje skripta u kojoj se nalazi metoda *OnRenderImage* koja ima dva parametra: ishodište i odredište. Ishodište je tekstura koju je renderirala kamera, a ono što metoda upisuje u

odredište bit će prikazano na zaslonu. To omogućuje da se uz pomoć metode *Blit* sadržaj ishodišta kopira u odredište te se primjeni željeni *post processing* efekt.

Zbog mogućnosti dodavanja većeg broja efekata u ovom radu je primjenjivanje implementirano na način da se efekti primjenjuju na ishodišnu teksturu, a na kraju se sadržaj ishodišta kopira u odredište. Svaki je efekt implementiran u zasebnoj *C#* klasi koja nasljeđuje zajedničku apstraktnu klasu. Apstraktna klasa sadrži varijablu koja pamti je li efekt aktivan, varijablu koja određuje na koji se udio ekrana primjenjuje efekt (kako bi se olakšala usporedba izgleda slike bez efekta te s efektom) te metodu koja primijeni efekt na teksturu. Moguće je napraviti objekte iz klasa koje nasljeđuju iz te apstraktne klase, postaviti njihove parametre te ih primijeniti na ishodišnu teksturu pozivajući metodu *apply*. Osnovni oblik klase efekta izgleda ovako:

```
using System;
using UnityEngine;

[Serializable] // označava da se objekt može prikazati u inspektoru
public class TESTEFFECT : Effect
{
	// metoda za primjenjivanje efekta
	public override void apply(RenderTexture tex)
	{
		// na početku materijal još nije definiran
		// pa ga je potrebno inicijalizirati
		if (mat == null)
		{
			// napravimo materijal koristeći se shaderom
			// koji se referencira koristeći se nazivom shadera
			Shader shader = Shader.Find("Hidden/TESTSHADER");
			mat = new Material(shader);
			mat.hideFlags = HideFlags.HideAndDontSave;
		}
		// zadaju se parametri shadera
		mat.SetFloat("_Swipe", swipe);

		// shader se primjenjuje na teksturu
		Graphics.Blit(tex, tex, mat);
	}
}
```

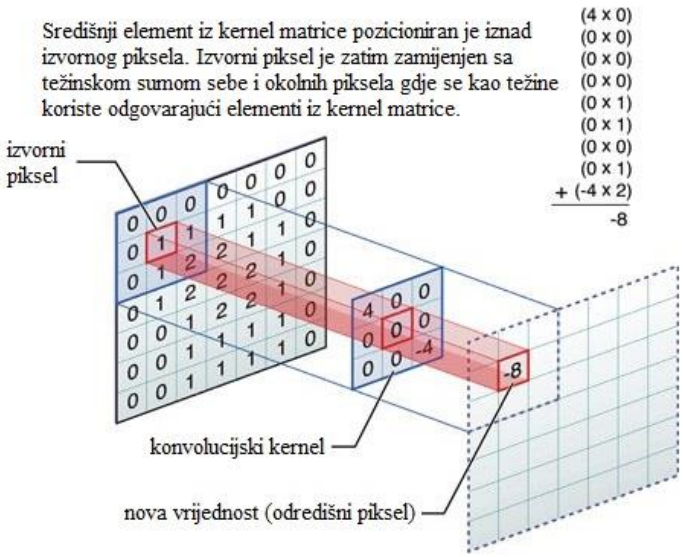
2.4 POST PROCESSING SHADERI

U ovom djelu rada opisani su neki *post processing* efekti te njihova implementacija u *Unityu*.

2.4.1 Zamućivanje

Zamućivanje (engl. *blur*) je jedan od najosnovnijih *post processing* efekata. Kao što ime nalaže on zamuti sliku koja se prikazuje na ekranu. Sam po sebi je rijetko koristan, ali je sastavni dio mnogih kompliciranijih efekata (npr. dubinska oštrina).

Za implementaciju zamućenja kao i mnogih drugih efekata koristi se konvolucija matrica. To je matematička operacija gdje manja matrica koja sadrži težine (*kernel* matrica) kliže po drugoj matrici te se pomnože elementi matrica koji se preklapaju, a umnošci se zbrajaju i zapisuju u novu matricu. Konvoluciju matrica jasnije prikazuje *Slika 2.4*. U praksi konvolucija nekom *kernel* matricom znači da se za svaki piksel izračunava njegova boja tako da se boje svih okolnih te tog piksela pomnože s odgovarajućom težinom iz *kernel* matrice te se zbroj rezultata prikazuje na tom pikselu.



Slika 2.4. konvolucija matrica[11]

[11] izvor: Elster, Convolution, https://mriquestions.com/what-is-convolution.html (19. veljače 2024.)

Najjednostavnija vrsta zamućivanja je zamućivanje okvirom (engl. *box blur*). Za takvo zamućivanje koristi se *kernel* matrica koja sadrži samo jedinice. Za *kernel* veličine *3x3* piksela ona izgleda ovako:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Nakon konvolucije slike matricom, dobiveni rezultat podijeljen je se sa zbrojem težina (u ovom primjeru je to 9). Ove dvije operacije ekvivalentne su uzimanju aritmetičke sredine boja svakog piksela i boja njemu susjednih piksela, pa je na taj način i implementiran za potrebe ovog rada. Iako je najjednostavniji način implementacije zamućivanja, zamućivanje okvirom koristi se rijetko jer rezultati često nisu estetski prihvatljivi.

Kako bi se postigao bolji efekt zamućivanja, potrebno je uzeti u obzir udaljenost susjednog piksela od središnjeg te na taj način odrediti njegovu težinu. Težine nisu raspoređene linearno, već po Gaussovoj krivulji. Zbog toga ova vrsta zamućivanja nosi ime Gaussovo zamućivanje (engl. *Gaussian blur*). To je najčešća vrsta zamućenja koja se koristi za velik broj namjena. Težine je moguće izračunati u stvarnom vremenu koristeći se formulom:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

U formuli parametri *x* i *y* označuju udaljenost susjednog piksela od središnjeg, a σ označava standardnu devijaciju krivulje. Ipak, kako bi se izbjeglo nepotrebno izračunavanje bolje je unaprijed izračunati *kernel* sa željenom standardnom devijacijom te njime činiti konvoluciju slike. Primjer takvog *kernela* sa standardnom devijacijom *0.25* je:

$$K = \begin{bmatrix} 0.0005 & 0.0217 & 0.0005 \\ 0.0217 & 0.9111 & 0.0217 \\ 0.0005 & 0.0217 & 0.0005 \end{bmatrix}$$

Konvolucija *full HD* slike s *kernelom* dimenzija *7x7* sastoji se od više od sto milijuna uzorkovanja teksture. Uzorkovanje je jedna od najsporijih operacija pa je takav *shader* prilično zahtjevan za grafičku karticu te može znatno produljiti vrijeme izvađanja. Kako bi se optimizirao, ovaj efekt implementira se u dva prolaza. Prvi zamuti sliku jednodimenzionalnim *kernelom* po horizontalnoj osi, a drugi po vertikalnoj osi. Rezultat je isti, a zahtijeva znatno manje uzorkovanja. *Slika 2.5* prikazuje usporedbu izgleda scene s Gaussovim zamućenjem i bez Gaussova zamućenja.

Slika 2.5. razlika slike sa i bez Gaussova zamućivanja

2.4.2 Vinjeta

Vinjeta (engl. *vignette*) je efekt koji zatamnjuje rubove slike. Vinjeta ponekad nastaje na stvarnim fotografijama, a dodavanjem vinjete moguće je postići dramatičnu scenu te privući pozornost na sredinu slike.

Prvo je potrebno izračunati udaljenost piksela od centra slike. To se postiže na način da se najprije od *UV* koordinata piksela oduzme *0.5*. Rezultat je vektor koji pokazuje od središta slike do pozicije piksela. Magnituda tog vektora je udaljenost piksela od središta slike. Prema udaljenosti od središta moguće je izračunati jačinu vinjete u svakom pikselu. Za više kontrole nad izgledom koristi se formula:

$$V = 1 - ((d + I - 0.5) * F + 0.5)$$

U ovoj formuli *d* označava udaljenost piksela od središta slike, *I* je intenzitet vinjete, a *F* kontrolira koliko je nagli prijelaz vinjete. Ako se boja pojedinog piksela pomnoži s dobivenom vrijednošću *V*, postići će se željeni rezultat. *Slika 2.6* prikazuje izgled scene s primijenjenim efektom vinjete.

Slika 2.6. slika s efektom vinjete

2.4.3 Kromatska aberacija

Kromatska aberacija (engl. *chromatic aberration*) je pojava vidljiva na nekim fotografijama gdje se crveni, zeleni i plavi kanal ne poklapaju u potpunosti pri rubu fotografije. U videoigrama se često koristi kako bi se prikazalo da nešto nije u redu, npr. kad je glavni lik ozlijeđen.

Kao *post processing* efekt se implementira na način da se zamaknu *UV* koordinate na kojima se uzorkuje tekstura za različite kanale. Ovisno o udaljenosti od središta slike taj pomak je sve veći. *Slika 2.7* prikazuje izgled efekta kada se primijeni na sliku.

Slika 2.7. slika s kromatskom aberacijom

2.4.4 Zrnatost filma

Zrnatost filma (engl. film grain) je efekt koji replicira šum koji je vidljiv na starom filmu. Implementira se na način da se na sliku doda bijeli šum (engl. *white noise*) odnosno tako da se svjetlina svakog piksela poveća ili smanji za nasumičnu količinu. Efekt je sam po sebi vrlo jednostavan, ali ga ograničenja grafičke kartice čine kompliciranijim za implementaciju. Grafičke kartice ne mogu izračunavati nasumične brojeve kao što to radi procesor, već je potrebno izračunati pseudonasumičan broj[12]. Generatori pseudonasumičnih brojeva su u principu *hash* funkcije. Funkcija se koristi nizom operacija množenja, ekskluzivne disjunkcije na razini bitova te pomaka bitova kako bi se ulazna vrijednost pretvorila u pseudonasumičan rezultat. Kao ulazna vrijednost koriste se koordinate piksela kako bi se za svaki piksel izračunala drugačija vrijednost. Sljedeći isječak koda prikazuje implementaciju generatora pseudonasumičnih brojeva koji je korišten u ovom radu:

```
// uzima 'seed' vrijednost tipa uint i vraća pseudonasumičan broj tipa
uint
// za dobivanje broja iz intervala [0, 1] vraćena vrijednost dijeli se
// maksimalnim mogućim brojem tipa uint (0xffffffff)
uint hashi(uint x)
{
    x ^= x >> 17;
    x *= uint(0xed5ad4bb);
    x ^= x >> 11;
    x *= uint(0xac4c1b51);
    x ^= x >> 15;
    x *= uint(0x31848bab);
    x ^= x >> 14;
    return x;
}
```

[12]brojevi izračunati na način da ne postoji nikakva korelacija između ulazne vrijednosti i generiranog pseudonasumičnog broja, ali za određenu ulaznu vrijednost generirani broj će uvijek biti isti

Dobivena nasumična vrijednost iz intervala $[0, 1]$ tada se uvrsti u formulu koja glasi:

$$X = 10000 * \sin(n + t)$$

$$N = X - \lfloor X \rfloor - 0.5$$

Broj *n* pseudonasumičan je broj koji je dobiven ovisno o koordinatama piksela, *10000* je proizvoljan veliki broj koji kontrolira brzinu promijene intenziteta šuma (ako je premalen bit će vidljiva periodičnost sinusa), *t* vrijeme od početka izvođenja programa, a *N* rezultantni šum koji se dodaje na sliku. Rezultat je pseudonasumičan broj iz intervala $[-0.5, 0.5]$ koji se dodaje na boju piksela što ga zatamni ili posvijetli. Funkcija sinus korištena kako promjena intenziteta šuma nije u potpunosti nasumična već piksel postepeno mijenja boju. *Slika 2.8* prikazuje scenu na koju je dodan intenzivan šum.

Slika 2.8. slika s intenzivnim šumom

2.4.5 Magla

Magla je vrlo česti *post processing* efekt jer je korisna za određivanje atmosfere scene, ali i kako bi prikrila modele u udaljenim dijelovima scene koji su niske rezolucije zbog očuvanja performansi. Implementira se na način da se ovisno o udaljenosti piksela od kamere (koja se iščitava iz spremnika dubine) boja piksela interpolira između izvorne boje i boje magle. Udaljenost spremljenu u spremniku dubine prvo je potrebno linearizirati koristeći se formulom:

$$d_L = \frac{f}{\left(1 - \frac{f}{n}\right) \cdot (1 - d) + \frac{f}{n}}$$

Gdje je d_L linearna dubina u prostoru svijeta, d očitana vrijednost iz spremnika dubine, n udaljenost bliske ravnine odsijecanja, a f udaljenost daleke ravnine odsijecanja[13].

Zatim se izračunava faktor magle prema formuli:

$$F = 2^{-(d_L \cdot D)^2}$$

Gdje F označava faktor magle, a D parametar koji određuje gustoću magle. Na kraju *shader* vraća linearnu interpolaciju između boje piksela i odabrane boje magle koristeći se izračunatim faktorom magle. Izgled ovako postignute magle prikazuje *Slika 2.9.*

Slika 2.9. slika s maglom

[13] bliska i daleka ravnina odsijecanja (engl. *near and far clipping plane*) su svojstva kamere koje određuju najbližu i najdalju točku koja se renderira

2.4.6 Izoštravanje

Izoštravanje (engl. *sharpen*) je efekt koji povećava kontrast na rubovima objekata kako bi se postigla oštrija slika. Ovaj efekt može znatno doprinijeti izgledu scene bez znatnog utjecaja na performanse jer može prividno povećati količinu detalja na teksturama niske rezolucije. Moguće ga je implementirati na velik broj načina, a najosnovniji je izoštravanje okvirom (engl. *box sharpen*). Izoštravanje okvirom konvolucija je matricom koja smanjuje utjecaj susjednih piksela na trenutni te rezultira suprotnim efektom zamućivanju okvirom. Koristi se sljedeća kernel matrica:

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Ona se također može prilagoditi kako bi uključivala parametar koji određuje intenzitet efekta (ovdje označen sa *S*):

$$K = \begin{bmatrix} 0 & -S & 0 \\ -S & 4S+1 & -S \\ 0 & -S & 0 \end{bmatrix}$$

Efekt je vrlo jednostavan ali daje rezultat koji u većini slučajeva nije koristan. Uvodi veliku količinu neželjenog šuma te ne izgleda prirodno. Izgled slike izoštrene koristeći se izoštravanjem okvirom prikazuje *Slika 2.10*. Korišten je relativno visok intenzitet kako bi se na slici istaknuli artefakti koji nastaju kao nuspojava ovog efekta. Pri nižem intenzitetu su oni manje vidljivi, ali je i oštrina slike manja.

Slika 2.10. izoštravanje okvirom s izuzetno visokim intenzitetom

Za prirodniji izgled izoštrene slike koristi se izoštravanje koje se prilagođava kontrastu (engl. *contrast adaptive sharpness*). Ono se razlikuje od izoštravanja okvirom na način da izoštrava jedino piksele u dijelovima slike visokog kontrasta te na taj način izbjegava nastajanje artefakata. Prvo se uzorkuje trenutni piksel te susjedni pikseli. Izračuna se minimalna i maksimalna vrijednost za svaki kanal (crveni, zeleni i plavi) među tim pikselima. Koristeći te minimalne i maksimalne vrijednosti izračunava se amplituda za svaki kanal koja određuje potrebnu količinu izoštravanja prema formuli:

$$A = \sqrt{\frac{\min(m, 2 - M)}{M}}, m = m_o + m_d, M = M_o + M_d$$

U formuli A označava amplitudu. Oznake m_o i M_o označavaju minimalnu i maksimalnu vrijednost među trenutnim pikselom i ortogonalno susjednim pikselima, dok oznake m_d i M_d označavaju minimalnu i maksimalnu vrijednost među dijagonalno susjednim pikselima. Zatim se izračunava težina (označena *w*) koristeći se parametrom oštrine (označen *S*) koji je negativan realan broj iz intervala $[-1, 0\rangle$ gdje manji broj rezultira u većem intenzitetu oštrine. Ta težina se uvrštava u formulu za izračun specifične kernel matrice (označena *K*) za pojedini kanal boje tog piksela.

$$w = A \cdot S$$

$$K = \frac{1}{1+4w} \cdot \begin{bmatrix} 0 & w & 0 \\ w & 1 & w \\ 0 & w & 0 \end{bmatrix}$$

Kada se slika izoštri na ovaj način nastaje puno manje artefakata. *Slika 2.11* prikazuje scenu gdje je lijeva polovica slike izoštrena ovom metodom, a desna polovica nije izoštrena. Razlika nije jasno vidljiva, ali takva razlika značajno doprinosi isticanju detalja u sceni bez da slika izgleda neprirodno oštro.

Slika 2.11. usporedba slike izoštrene metodom adaptivnog izoštravanja i izvorne slike

2.4.7 Isticanje obruba

Isticanje obruba (engl. *thicken outlines*) je efekt koji je vrlo popularan među *indie*[14] programerima videoigara jer omogućuje upečatljiv stil nalik na crtani film ili strip bez prevelikog truda, ali njime se koriste i neke visokobudžetne videoigre kao što su *Borderlands* i *Taletale* serijal. Jedan od načina implementacije ovakvog *shadera* je kao *post processing* efekt koji se koristi spremnikom dubine za određivanje dijelova slike gdje postoje velike

[14] jedna osoba ili mali nezavisni tim

promjene u dubini, odnosno rubova objekata. Prvo se uzorkuje dubina na trenutnom pikselu i pikselima koji ga okružuju te izračunava aritmetička sredina tih dubina. Ako se uzorkuje veći broj piksela, obrubi će biti deblji. Razlika između aritmetičke sredine dubina i dubine trenutnog piksela uvrštava se u formulu kako bi se dobila rezultanta boja.

$$C_r = C_i * (1 - (d \cdot F)^S)$$

U formuli C_r označava rezultantnu boju, a C_i izvornu boju. Razlika dubine i aritmetičke sredine uzorkovanih dubina označena je s *d*. Oznaka *F* predstavlja faktor koji određuje osjetljivost odnosno minimalnu potrebnu razliku udaljenosti da se pojavi obrub, a *S* je parametar koji određuje oštrinu obruba. Izgled scene s istaknutim rubovima prikazuje *Slika 2.12*.

Slika 2.12. scena s istaknutim obrubima

2.4.8 Korekcija boja

Korekcija boja (engl. *color correction*) vrlo je važan *post processing* efekt koji je gotovo obavezan dio svake videoigre. On se sastoji od više parametara koji kontroliraju svojstva boja u slici. To su ekspozicija, balans bijele boje, kontrast, svjetlina, zasićenost, filter boja i gama. Većinu tih efekata moguće je implementirati za cjelovitu boju ili za svaki kanal boje posebno kako bi se pružala veća kontrola nad izgledom rezultante slike.

Boje u teksturi su gama korigirane pa ih je prvo potrebno prebaciti u linearni prostor boja. To se postiže na način da se vrijednost boje potencira s eksponentom $\frac{1}{2.2}$. Zatim je moguće primjenjivati efekte.

Ekspozicija (engl. exposure) u fotografiji predstavlja količinu svjetlosti koja ulazi u senzor kamere. Taj efekt se replicira tako da se boja svakog piksela pomnoži sa skalarnom vrijednošću.

Balans bijele boje (engl. *white balance*) određuje temperaturu boja (tople ili hladne boje odnosno žuto ili plavo) te nijansu boja (zeleno ili ružičasto). Izračunava se nizom matematičkih operacija te množenjem matricama koje proizlaze iz teorije boja.

Kontrast (engl. *contrast*) je razlika između svijetlih i tamnih dijelova slike. On se zajedno sa svjetlinom (engl. *brightness*) implementira koristeći se formulom:

$$C_r = c * (C_i - 0.5) + 0.5 + b$$

Gdje su C_i i C_r izvorna i rezultanta boja, c označava kontrast, a b svjetlinu.

Zasićenost (engl. saturation) određuje intenzitet boja. Veća zasićenost dovodi do žarkih boja, a niža zasićenost do zagasitih boja. Dobiva se linearnom interpolacijom između izvorne boje i luminancije (engl. *luminance*) te boje. Luminancija je prividna svjetlina boje za ljudsko oko. Ona se izračunava kao skalarni umnožak boje i konstantnog vektora:

$$L = \begin{bmatrix} r \\ g \\ b \end{bmatrix} \cdot \begin{bmatrix} 0.299 \\ 0.587 \\ 0.144 \end{bmatrix} = 0.299r + 0.587g + 0.144b$$

To proizlazi iz načina na koji ljudsko oko percipira boje. Zelena boja je prividno svjetlija od crvene i plave pa ima najveći koeficijent pri računanju luminancije.

Filter boja (engl. *color filter*) je jednostavni efekt koji boje svih piksela pomnoži s odabranom bojom. Najupečatljivija posljedica ovog efekta jest da svi bijeli dijelovi slike postanu odabrana boja.

Na kraju korekcije boje potrebno je opet korigirati gamu slike. Taj korak je iznimno bitan kako bi slika izgledala prirodno. Ljudske oči ne percipiraju svjetlost na isti način kao kamera. Kamera sprema boje u linearnom prostoru što znači da spremljena vrijednost direktno korelira sa stvarnom luminancijom. Ali takve slike ljudima izgledaju blijedo i nerealno jer ljudske oči ne vide svjetlost linearno: svjetlost duplo većeg intenziteta za ljudsko

oko je samo lagano svjetlija. Kako bi se to uračunalo potrebno je transformirati boje u gama prostor. To se radi na način da se boja potencira na neki eksponent. Standardna vrijednost game je *2.2*, ali promjenom ove vrijednosti mogu se naglasiti tamni ili svijetli dijelovi slike.

Kombinacijom ovih efekata moguće je promijeniti boje kako bi slika bila realnija ili upečatljivija. Takve postavke korištene su za korekciju boja koju prikazuje *Slika 2.13* (lijeva strana).

Slika 2.13. usporedba slike sa i bez korekcije boja

Kako je korekcija boja vrlo moćan alat može se koristiti i za stilizirani izgled kao što prikazuje *Slika 2.14*.

Slika 2.14. korištenje korekcije boja za stilizirani izgled

2.4.9 Pikselizacija

Pikselizacija je efekt koji prividno smanjuje rezoluciju prikazane slike kako bi se istaknuli pikseli. Popularan je u *indie* igrama jer omogućuje proceduralni *pixel art* stil bez potrebe za *2D* animacijom ili modelima s puno detalja. Za pikselizaciju slike koristi se tehnika progresivnog smanjivanja te povećavanja rezolucije teksture. To je vrlo jednostavan proces koji ne zahtjeva pisanje *shadera* te je iznimno brz jer je mijenjanje rezolucije teksture implementirano hardverski u grafičkoj kartici. Početna tekstura premjesti se u privremenu teksturu duplo manje rezolucije. To se ponavlja nekoliko puta (ovisno o željenoj veličini piksela), pazeći da ni visina ni širina teksture ne bude manja od jedan. Nakon toga tekstura se isti broj puta premješta u duplo veću teksturu kako bi se vratila do početne rezolucije. Koristeći se ovom tehnikom slika neće postati pikselizirana, već zamućena kao što prikazuje *Slika 2.15*. To se može koristiti kao vrlo efikasan efekt zamućivanja, ali nije poželjno za pikselizaciju. Izvor ovog problema je način filtriranja pri uzorkovanju teksture. Ako se način filtriranja ne navede, koristi se bilinearno filtriranje (engl. *bilinear filtering*). To znači da uzorkovanje *UV* koordinata koje se nalaze između dva piksela, daje boju koja je kombinacija boja tih dvaju piksela. Za potrebe ovog efekta potrebno je koristiti se točkastim filtriranjem (engl. *point filtering*). Tada će uzorkovana boja biti jednaka boji najbližeg piksela, što stvara oštre granice između boja. Ovako pikseliziranu sliku prikazuje *Slika 2.16*. Ova tehnika često se koristi u kombinaciji s kvantizacijom boja kako bi se postigao autentičniji *pixel art* stil.

Slika 2.15. slika zamućena progresivnim smanjivanjem i povećavanjem

Slika 2.16. pikselizirana slika

2.4.10 Kvantizacija boja

Za postizanje stiliziranog izgleda igre, a pogotovo za *pixel art* stil, često je važno ograničiti paletu boja. Jedan od načina na koji se to može učiniti jest kvantizacija boja (engl. *color quantization*). To je proces koji smanjuje količinu različitih boja na slici tako da slika što bolje održi svoj izgled, što se često koristi i prilikom kompresije slika. To se postiže uvrštavanjem boje piksela u formulu:

$$C_r = \frac{\lfloor C_i * (n-1) + 0.5 \rfloor}{n-1}$$

Gdje su C_i i C_r izvorna i rezultantna boja, a n cijeli broj iz intervala $[2, 256]$ koji označava željeni broj boja po kanalu. Ovakva kvantizacija je funkcionalna ali dovodi do vrlo vidljivih pojasa boja (engl. *color banding*) te se gube oblik predmeta i percepcija dubine. To prikazuje *Slika 2.17* (na sliku je primijenjena i pikselizacija jer efekt kvantizacije boja izgleda prirodnije na pikseliziranim slikama).

Slika 2.17. pojasi boja vidljivi pri jednostavnoj kvantizaciji boja

Kako bi se poboljšao efekt kvantizacije boja uvodi se *dithering*: namjerno dodavanje šuma na sliku kako bi se stvorila iluzija veće količine boja s ograničenom paletom boja. *Dithering* je također metoda koja se u *pixel artu* često koristi za sjenčanje te gradijente, pa pridonosi stilu slike. *Dithering* se na sliku dodaje koristeći se matricom za uređeni *dithering* (engl. *ordered dithering*). Za potrebe ovog efekta korištena je sljedeća matrica:

$$D = S \cdot \left(\frac{1}{16} \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 6 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} - 0.5 \right)$$

Parametar *S* određuje intenzitet efekta, odnosno koliko široka su područja između dvije boje na kojima je vidljiv *dithering*. Na svaki blok od *4x4* piksela odgovarajućem pikselu dodaje se odgovarajuća vrijednost iz matrice *D*. Pozicija u matrici se matematički izračunava na sljedeći način:

$$p_x = \lfloor UV_x \cdot w \rfloor$$
$$p_y = \lfloor UV_y \cdot h \rfloor$$
$$i = p_x \bmod 4$$
$$j = p_y \bmod 4$$

Koordinate piksela u teksturi označene su s p_x i p_y. UV_x i UV_y označavaju *UV* koordinate tog piksela, *w* i *h* dimenzije teksture u pikselima, a *i* i *j* poziciju odgovarajućeg elementa u matrici *D*. Kvantizacija boja s uređenim *ditheringom* prikazuje *Slika 2.18* koja sadrži osam boja po kanalu kao i *Slika 2.17*, ali prikazuje puno veći broj detalja.

Slika 2.18. kvantizacija boja s uređenim ditheringom

Kvantizacija boja se može iskoristiti i kako bi se zadala željena paleta boja kojom će se koristiti slika. *Shader* je potrebno prilagoditi na način da se umjesto boje u formulu uvrsti luminancija, što efektivno izvodi kvantizaciju boja na crno-bijeloj slici. Zatim se dobivena vrijednost koristi kao *x* koordinata kojom se uzorkuje tekstura palete boja (npr. *Slika 2.19*) te se ta boja prikazuje na ekranu. Rezultat ovog efekta je upečatljiv stil nalik na *pixel art* kao što prikazuje Slika 2.20.

Slika 2.19. primjer teksture s paletom boja

Slika 2.20. kvantizacija boja s zadanom paletom boja; pixel art stil

2.4.11 Bloom

Bloom je efekt koji replicira svojstvo kamera da se vrlo svjetli dijelovi slike „razlijevaju“ u okolne piksele. Često se koristi za isticanje jake svjetlosti objekata jer oni često ne izgledaju dovoljno svjetlo, osobito na *LDR* zaslonu. Implementira se na način da se izoliraju najsvjetliji dijelovi slike, oni se zamute te se aditivno miješaju (engl. *aditive blending*) na originalnu sliku. Za to je prvo potrebno odrediti dijelove slike na koje će se *bloom* primijeniti, a za to se koristi formula:

$$s = (b + t - k)^2 \cdot \frac{0.25}{k}$$

$$C_r = C_i \cdot \frac{\max(s, b - t)}{b}$$

C_i i C_r oznake su za izvornu i rezultantnu boju, b je svjetlina piksela koja je definirana kao najveća od vrijednosti tri kanala boje, a t željeni prag koji određuje koliko svjetli mora biti piksel da bi se na njega primijenio *bloom*. Rezultat ove operacije je tekstura gdje su izolirani najsvjetliji dijelovi slike, a ostatak teksture je crn. Zatim se tekstura zamuti koristeći se tehnikom progresivnog smanjivanja te povećavanja rezolucije. Pri svakom prolazu na sliku se također primjenjuje zamućivanje okvirom s *3x3 kernel* matricom što omogućuje vrlo intenzivno te estetski prihvatljivo zamućenje s minimalnim utjecajem na performanse.

Prolaz koji se koristi pri postepenom povećavanju rezolucije sadrži oznaku *Blend One One* što znači da se rezultanta boja zbraja na izvornu umjesto da ju zamijeni (aditivno miješanje). U posljednjem prolazu dobivena se slika aditivnim miješanjem dodaje na izvornu. Rezultat prikazuje *Slika 2.21*.

Slika 2.21. bloom na kugli s emisionim materijalom

Zbog prirode aditivnog miješanja, nakon primjenjivanja *bloom* efekta na sliku, boje su često pogurnute u *HDR* raspon. Također je povoljno prag svjetline za *bloom* postaviti izvan *LDR* raspona jer uostalom dolazi do prevelike količine *blooma* koji je nepoželjan. Zbog ta dva razloga *bloom* se koristi gotovo isključivo u visokom dinamičkom rasponu, a rezultantu sliku potrebno je dovesti u *LDR* koristeći se mapiranjem tonova.

2.4.12 Mapiranje tonova

Mapiranje tonova (engl. *tone mapping*) je proces prevođenja *HDR* boja u *LDR* kako bi se mogle pravilno prikazati na zaslonu. Postoji vrlo velik broj funkcija za mapiranje tonova od kojih svaka ima prednosti i mane. Jedna od takvih funkcija je *Lottes* funkcija za mapiranje boja. Ona glasi:

$$C_r = \frac{C_i}{1 + L}$$

C_i i C_r oznake su za izvornu i rezultantnu boju, a L označava luminanciju trenutnog piksela. Ova funkcija je vrlo jednostavna, ali i ograničena time što ne može biti podešena. Zbog toga se mogu koristiti funkcije poput Reinhardove proširene funkcije za mapiranje tonova koja omogućuje zadavanje točke bjeline (engl. *white point*) koja određuje koliko svjetao je najsvjetliji piksel slike u *HDR* teksturi. Ta boja se mapira u bijelu boju u *LDR* teksturi. Reinhardova proširena funkcija za mapiranje tonova glasi:

$$C_r = C_i \cdot \frac{1 + \frac{L}{(w_p)^2}}{1 + L}$$

C_i i C_r oznake su za izvornu i rezultantnu boju, L označava luminanciju trenutnog piksela, a parametar w_p određuje točku bjeline.

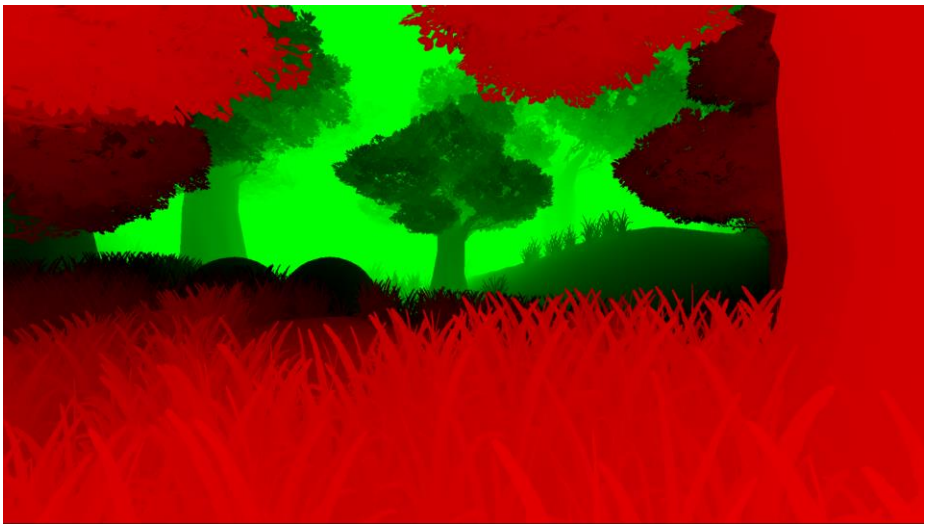
2.4.13 Dubinska oštrina

Dubinska oštrina (engl. *depth of field*) je efekt koji replicira zamućenje dijelova slike koji su izvan fokusa. Može biti vrlo zahtjevan za performanse, ali znatno doprinosi fotorealizmu slike koja postaje nalik pravoj fotografiji, zbog čega je vrlo često korišten za ugrađeni „*photo mode*“ u videoigrama.

Prvi prolaz generira teksturu u kojoj se ovisno o udaljenosti piksela od kamere on označava kao blizak ili dalek koristeći se funkcijom:

$$C_{RGB}(d) = \begin{cases} [1,0,0], & d < F - R \\ \left[\frac{(F - d_L)}{R}, 0, 0\right], & F - R \leq d < F \\ \left[0, 1 - \frac{(F + R - d_L)}{R}, 0\right], & F \leq d < F + R \\ [0,1,0], & F + R \leq d \end{cases}$$

F je udaljenost fokusa od kamere, R je oznaka širine dijela slike koji je oštar (analogno *f-stopu* fizičke kamere), a d_L linearizirana udaljenost trenutnog piksela od kamere. U crveni kanal teksture sprema se bliski dio slike, a u zeleni kanal udaljeni dio slike. Tu teksturu prikazuje *Slika 2.22*. Ona se naziva i tekstura kruga zamagljenja (engl. *circle of confusion*). Spremanje više informacija u različite kanale iste teksture je čest način optimizacije *shadera.*



Slika 2.22. prikaz bliskog i dalekog dijela slike u crvenom i zelenom kanalu teksture

Slijede drugi i treći prolaz koji na crveni kanal (bliski dio) primjenjuju redom *max filter* i zamućivanje okvirom. *Max filter* je operacija koja u svaki piksel upiše maksimalnu vrijednost iz tog i okolnih piksela što efektivno povećava granice bliskog dijela kako bi se on preklapao s dalekim dijelom. To omogućava da se objekt koji je u prednjem planu i izvan fokusa može „prelijevati“ i preko udaljenih objekata. Zamućivanje okvirom služi kako granica između bliskog i dalekog dijela slike ne bi bila toliko izražena.

Prije daljnje obrade slike postoji mogućnost primjenjivanja inverznog mapiranja tonova kako bi se boje pomaknule u *HDR* te time poboljšao izgled dijelova slike koji su izvan fokusa. Ovaj korak moguće je preskočiti u slučaju da se boje već nalaze u visokom dinamičkom rasponu ili ako korišteno sklopovlje ne podržava *HDR*. Inverzni oblik Reinhardove proširene funkcije za mapiranje tonova glasi:

$$C_r = \frac{C_i}{w_p \cdot (1 - \frac{C_i}{w_p})}$$

U sljedećem koraku se bliski i daleki dio slike odvajaju u posebne privremene teksture. Za bliski dio se uzima kopija izvorne teksture, a za daleki dio izvorna tekstura pomnožena zelenim kanalom iz teksture kruga zamagljenja. Zatim se obje teksture zamute koristeći se posebnom vrstom zamućivanja zvanom *bokeh* zamućivanje. *Bokeh* u doslovnom prijevodu s japanskog jezika znači zamućenje, ali se koristi i kao naziv za pojavu gdje svjetle točke izvan fokusa poprimaju oblik otvora objektiva kojim je fotografija fotografirana. Bokeh zamućivanje je tip zamućivanja u kojem svi uzorci imaju istu težinu, a *kernel* je oblika nekog geometrijskog lika (npr. kvadrat, krug, trokut, šesterokut). Jedna od vrsta *bokeh* zamućivanja je i zamućivanje okvirom, ali za dubinsku oštrinu se najčešće koriste okrugli ili šesterokutni *kerneli* jer je takav otvor na većini objektiva.

Kako bi se dodatno istaknule najsvjetlije točke slike može se koristiti tehnika koja se zove inverzni Karisov prosjek. Karisov prosjek koristi se za prigušivanje „krijesnica“ (jedan iznimno svijetli piksel koji nastaje kao nuspojava nekih efekata), ali ako se pri zamućivanju koristi inverzni Karisov prosjek on pojačava svjetle točke što doprinosi izgledu slike izvan fokusa na kojoj se gube tamniji detalji. To se postiže prilagodbom težina uzoraka ovisno o *luminanciji* piksela. Očitana boja piksela se množi s odgovarajućom težinom, te se zbroj svih tako dobivenih boja dijeli sa zbrojem težina. S ovom tehnikom je potrebno biti pažljiv jer pojačava neželjene artefakte.

Na kraju se sve tri teksture povezuju u jednu linearnom interpolacijom između izvorišne boje te boja iz zamućenih tekstura, koristeći se teksturom kruga zamagljenja. Ako je primijenjeno inverzno mapiranje tonova, tada se primjenjuje mapiranje tonova s istom točkom bjeline kako bi se boje vratile u *LDR*. Izgled efekta dubinske oštrine prikazuje *Slika 2.23*.

Slika 2.23. dubinska oštrina na sceni

3. ZAKLJUČAK

Post processing efekti koriste se principima matematike i računalstva kako bi sliku učinili upečatljivijom. Mogućnost izvađanja u realnom vremenu čini ih iznimno korisnim pri izradi interaktivnih sadržaja. Zbog malog broja stručnjaka, potražnja za grafičkim programerima vrlo je visoka, a izvori za učenje su rijetki. Cilj ovog rada bio je istraživanje o vrstama *post processing* efekata te o detaljima njihove implementacije kao uvod u širok svijet grafičkog programiranja. Principi korišteni u ovom radu primjenjuju se i prilikom izrade mnogih naprednijih efekata te pri programiranju nekih paralelnih algoritama. Za tu svrhu odabrani su *post processing* efekti jer su ograničeni u dvije dimenzije te jer je kombiniranjem nekolicine relativno jednostavnih *shadera* moguće postići značajni doprinos vizualnom identitetu videoigre. Osim u razvoju videoigara, *shaderi* omogućuju brzo izvađanje raznovrsnih paralelnih algoritama što ih čini nezamjenjivima u obradi slike, izradi grafičkih sučelja, ali i računalnom vidu te umjetnoj inteligenciji.

4. LITERATURA

1. Garrett Gunnell, Acerolla, Youtube, https://www.youtube.com/@Acerola_t (pristupljeno 29. siječnja 2024.)

2. Garrett Gunnell, Acerolla, GitHub, https://github.com/GarrettGunnell/ (pristupljeno 29. siječnja 2024.)

3. Jasper Flick, C# and Shader Tutorials for the Unity Engine, https://catlikecoding.com/unity/tutorials/ (pristupljeno 29. siječnja 2024.)

4. Matt Taylor, Delta, Tone Mapping, https://64.github.io/tonemapping/ (pristupljeno 29. siječnja 2024.)

5. LearnOpenGL, Bloom, https://learnopengl.com/Advanced-Lighting/Bloom (pristupljeno 29. siječnja 2024.)

6. GeeksForGeeks, OpenGL Rendering Pipeline | An Overview, https://www.geeksforgeeks.org/opengl-rendering-pipeline-overview/ (pristupljeno 29. siječnja 2024.)

7. LearnOpenGL, Coordinate Systems, https://learnopengl.com/Getting-started/Coordinate-Systems (pristupljeno 29. siječnja 2024.)

8. Unity Documentation, White Balance Node, https://docs.unity3d.com/Packages/com.unity.shadergraph@6.9/manual/White-Balance-Node.html (pristupljeno 29. siječnja 2024.)

9. Hrvatska internetska enciklopedija, 3D računalne grafike, https://enciklopedija.cc/index.php?title=3D_računalne_grafike (pristupljeno 29. siječnja 2024.)

10. Unity Documentation, Graphics, https://docs.unity3d.com/Manual/Graphics.html (pristupljeno 29. siječnja 2024.)

11. Elster, Convolution, https://mriquestions.com/what-is-convolution.html (pristupljeno 19. veljače 2024.)

12. Papers with Code, Convolution Explained, https://paperswithcode.com/method/convolution (pristupljeno 29. siječnja 2024.)

5. POPIS SLIKA

KONZULTACIJSKI LIST IZRADE ZAVRŠNOG RADA

Ime i prezime učenika: **Jakov Biškup**

Razred: **4. RT**

Program – zanimanje: **Tehničar za računalstvo**

Mentor: **Krešimir Kočiš**

R. br.	DATUM KONZULTACIJE	SADRŽAJ RADA	POTPIS MENTORA
1.			
2.			
3.			
4.			

Završni rad ocijenjen pozitivno

DA / NE

_____

(potpis mentora)