SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2220

VIZUALIZACIJA TRODIMENZIJSKIH FRAKTALNIH OBJEKATA U WEBGL2 OKRUŽENJU

Martin Golub

Zagreb, lipanj 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2220

VIZUALIZACIJA TRODIMENZIJSKIH FRAKTALNIH OBJEKATA U WEBGL2 OKRUŽENJU

Martin Golub

Zagreb, lipanj 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 2. ožujka 2026.

ZAVRŠNI ZADATAK br. 2220

Pristupnik:	**Martin Golub (0036559810)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Vizualizacija trodimenzijskih fraktalnih objekata u WebGL2 okruženju**

Opis zadatka:

U suvremenoj računalnoj grafici za izradu složenih objekata, kakva je većina prirodnih objekata, često se koriste fraktalne strukture. Potrebno je istražiti matematičko definiranje fraktalnih objekata, te mogućnosti WebGL2 tehnologije za njihovu vizualizaciju u stvarnom vremenu. Osmisliti, razviti i optimizirati izradu prikaza fraktala u okviru web preglednika. Implementirati pripadno korisničko sučelje koje omogućava interaktivno definiranje potrebnih parametara. Omogućiti unos i bilo koje funkcije za koju je moguć prikaz funkcije udaljenosti SDF (engl. Signed Distance Function). Provesti testiranje na nizu primjera te analizirati i ocijeniti ostvarene rezultate. Razmotriti upotrebljivost dobivenih rezultata i moguće smjernice proširenja. Izraditi odgovarajući programski proizvod koristeći programski jezik JavaScript i grafičko programsko sučelje WebGL2. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 12. lipnja 2026.

Sadržaj

1. Uvod

Fraktali su zanimljivi oblici koji se često pojavljuju u prirodi, a da nismo toga svjesni. Mogu se opaziti na snježnoj pahuljici ili na listovima biljke. Bilo bi interesantno takve strukture rekonstruirati i istražiti ih u virtualnom okruženju.

Cilj ovog rada je izgraditi virtualnu scenu koja može prikazivati fraktale velikom razinom detalja i omogućiti korisniku kretanje kroz scenu za razgledavanje fraktala. Da bi intuitivno kretanje bilo izvedivo, brzina iscrtavanja mora biti u stvarnom vremenu, što znači da treba postupak optimirati i treba omogućiti upravljanje kvalitetom prikaza. Dodatno bi bilo poželjno da se mogu mijenjati parametri fraktala i čak mijenjati koji se fraktali/tijela prikazuju. Za to je potrebno izgraditi korisničko sučelje za unos vrijednosti.

Dodatni fokusa rada je programsko ostvarenje koristeći *WebGL2* biblioteku. Nakon iskustva korištenja tog alata mogu se usporediti njegove prednosti ili mane u odnosu na druge grafičke biblioteke u *web* okolini.

2. Teorijska pozadina

2.1. Funkcije udaljenosti

Postoji više načina na koje se može zapisati neko tijelo. U računalnoj grafici se najčešće trodimenzionalni modeli opisuju mrežom trokuta. Trokuti su praktični jer imaju jednostavnu strukturu i mogu se efikasno iscrtati. Unatoč tome, postoje nedostatci takvog pristupa. Kako bi se postigla zakrivljena površina, potrebno je izgraditi površinu s velikim brojem trokuta. Time površina nikad neće biti uistinu zakrivljena, nego će se sastojati od puno ravnih dijelova. Neke strukture (kao na primjer spužvaste strukture) zahtijevaju toliko veliki broj trokuta da se treba napraviti kompromis brisanjem detalja strukture radi optimizacije brzine iscrtavanja. Detalji se općenito dodaju modelima koristeći teksture, no u tim slučajevima stvarna geometrija još uvijek ostaje pojednostavljena.

Način prikaza tijela koji će se razmatrati u ovom radu je matematički opis. Specifično, razmatrat će se funkcije udaljenosti (*eng. Signed Distance Function, SDF*). Funkcija ima jedan parametar p, gdje je p točka u prostoru, a rezultat funkcije $f(p)$ je udaljenost te točke od površine tijela koju funkcija opisuje. Dimenzija prostora u kojem se nalazi ulazna točka određuje funkcija (2D ili 3D). Primjer jedne takve funkcije za opis sfere je navedena u jednadžbi (2.1).

$$f(p)_{\text{sfera}} = |centar - p| - radius \tag{2.1}$$

Slika 2.1. prikazuje vrijednosti funkcije udaljenosti pravokutnika. Crte na slici označavaju gdje su vrijednosti funkcije jednake, to jest mogu se zamisliti kao izohipse na topografskoj karti. Bijela crta označava gdje je vrijednost udaljenosti nula, odnosno radi se o površini tijela. Osim vraćanja vrijednost udaljenosti, drugo važno svojstvo funk-

cije je predznak rezultata. Vrijednost je pozitivna kada je točka izvan tijela (slika 2.1., narančasti dio), a negativna kada je točka unutar tijela (slika 2.1., plavi dio). Koristeći navedena svojstva moguće je opisati neko tijelo. Nadalje, to je i dovoljno informacija za izvesti prikaz tog tijela.

Slika 2.1. Ilustracija funkcije udaljenosti [1]

2.2. Fraktali

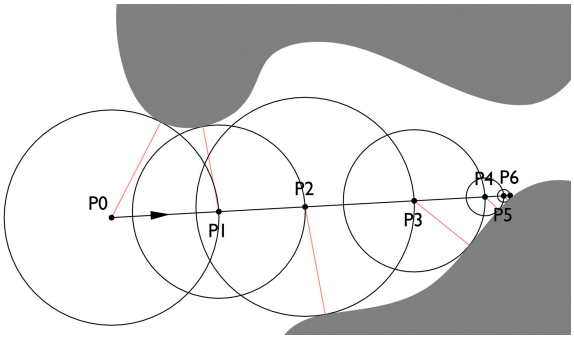
Fraktali su oblici sa svojstvom samoponavljanja. Odnosno, postoji uzorak (*engl. pattern*) ili neko svojstvo koji se ponavlja u tijelu, na raznim mjestima i raznim veličinama. Nema ograničenja koliko se taj uzorak može ponavljati, što znači da fraktal u teoriji ima beskonačno detalja. Takve fraktalne strukture se mogu pronaći u prirodi, kao na primjer stabla, lišće (slika 2.2.) i reljef površine Zemlje.

Slika 2.2. Fraktalna struktura lista paprati [2]

Najčešće se koristi matematički zapis za opis jednostavnih oblika kao što su primjerice ravnine, sfere ili trokuti. Ako želimo matematički opisati neki složeniji oblik kao na primjer zeca, očito je da će to biti teško. Može se pretpostaviti da složenost izraza koji opisuje taj fraktal raste sa složenošću oblika. No, to nije uvijek slučaj. Naime, jednostavna matematički izraz može definirati vrlo složen i detaljan oblik, a dobar primjer su fraktali. Stoga će se u ovom radu koristiti funkcija udaljenosti za opis fraktala.

2.3. Algoritam iscrtavanja

Cilj je grafički prikazati fraktale. Potrebno je osmisliti algoritam koji uzima opis tijela u obliku funkcije udaljenosti i omogućuje iscrtavanje trodimenzionalnog prikaza tog tijela na ekranu. Koristi se metoda praćenja sfere (*engl. Sphere Tracing*), što je podvrsta metode marširanja zrake (*engl. Ray Marching*), što je još podvrsta metode praćenja zrake (*engl. Ray Tracing*). Metoda praćenja zrake određuje boju svakog piksela s pomoću zrake koja počinje u očištu kamere i prolazi kroz taj piksel. Metoda koračanja duž zrake koristi definiranu zraku tako da postepeno korača u smjeru zrake tijekom računanja. Praćenje sfere dalje definira koračanje tako da je duljina svakog koraka određena na temelju funkcije udaljenosti. Prikazivanjem svakog koraka i pripadajuće vrijednosti udaljenosti vizualizira se sfera (ili kružnica ako se gleda u 2D prostoru), što se može vidjeti na slici 2.3.



Slika 2.3. Koraci zrake u praćenju sfere

Algoritam na slici 2.4. opisuje postupak metode praćenja sfere. U svakoj iteraciji petlje se računa udaljenost s pomoću funkcije udaljenosti i obavi se korak duljine te udaljenosti u smjeru zrake. Petlja završava kada je zraka dotaknula tijelo ili kada prođe granicu maksimalnog broja koraka. Potrebno je zadati granicu koraka u slučaju kada zraka ne udari tijelo, nego nastavi koračati u beskonačnost. Zbog nepreciznosti brojeva s pomičnim zarezom može doći do slučaja da zraka nikada ne dođe do udaljenosti nula od tijela. Stoga se u uvjet dodira dodaje tolerancija *epsilon*, koja ima vrijednost vrlo malog broja.

```
pracenje_sfere(org, dir, max, epsilon) {
    // org - ishodiste zrake
    // dir - smjer zrake
    // max - maksimalan broj koraka
    // epsilon - vrlo mali broj
    pos = org
    for (count = 0; count < max; count++) {
        dist = sdf(pos) // funkcija udaljenosti
        if (dist < epsilon)
            break;
        pos += dir * dist
    }
    return pos
}
```

Slika 2.4. Algoritam praćenja sfere

3. Program za prikaz fraktala

3.1. Korišteni alati

Primarni alat koji je korišten za ovaj rad je biblioteka *WebGL2*. To je varijanta popularne grafičke biblioteke *OpenGL* koja je prilagođena za rad na internetskom pregledniku i jeziku *JavaScript*. Mogućnost korištenja grafičkih funkcija u internetskom pregledniku ima mnoge prednosti: omogućuje grafičke vizualizacije unutar web stranica, izradu video igara koje su dostupne bez instaliranja izvršne datoteke i generalni pristup procesorske moći grafičke kartice unutar jezika *JavaScript*. Biblioteka *WebGL2* nije sasvim identična *OpenGL*, a jedna bitna razlika je što ne omogućuje izradu računskih sjenčara (*engl. compute shaders*). Takvi sjenčari su pogodni za iscrtavanje postupcima temeljenim na zrakama kao što je praćenje sfera, no moguće je izvesti takvo iscrtavanje koristeći sjenčare fragmenata.

Programski jezik *JavaScript* ima svojih prednosti što se tiče grafičkih aplikacija jer s alatima za prikaz web stranica *HTML* i *CSS* pruža jednostavan i prilagodljiv način izrade korisničkog sučelja za upravljanje programom. Programska implementacija grafičkih sučelja u drugim jezicima kao što je primjerice *C++* bi bila znatno teža i bilo bi potrebno koristiti biblioteke kao na primjer *Dear ImGui* [3], koja ima manju fleksibilnost u dizajnu. Dodatno, web aplikacije su orijentirane na portabilan dizajn tako da neka web stranica može prikazivati na više vrsta uređaja. To uključuje i ovaj projekt: prikaz fraktala se može otvoriti na računalu, ali i na mobitelu, a sučelje se prilagođava na orijentaciju i veličinu ekrana. Naravno, kvaliteta i brzina iscrtavanja će ovisiti o grafičkom procesoru uređaja na kojem se izvodi.

Aplikaciji se može pristupiti na poslužitelju *GitHub Pages*. To je praktično jer se prilikom izrade aplikacije *Github* koristio za kontrolu verzija pa se izvorni tekst programa već

nalazi na toj platformi. Platforma *GitHub Pages* pruža statičko posluživanje, tj. program se izvodi na klijentu, a ne na poslužitelju, što je sasvim dovoljno za ovaj program.

3.2. Komponente programa

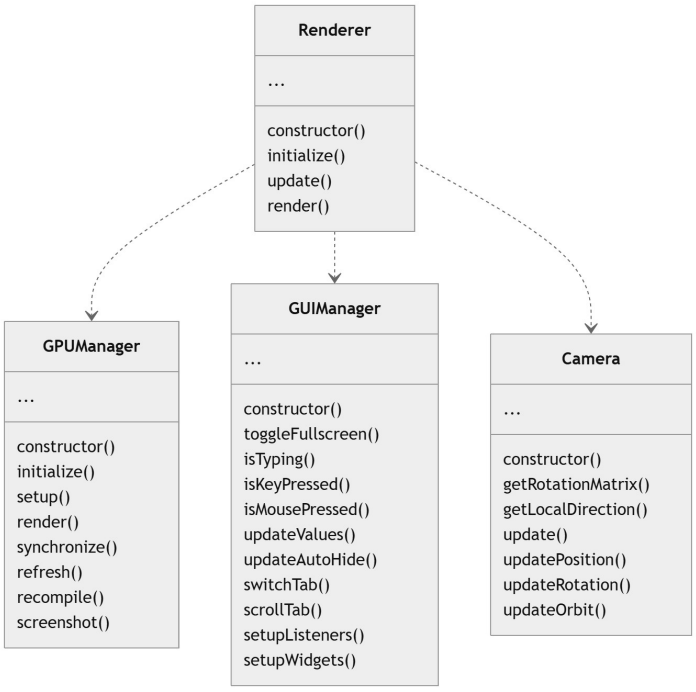
Najvažniji dijelovi programa su četiri klase, a njihov međusobni odnos je prikazan u skraćenom dijagramu razreda na slici 3.1. U dijagramu nisu navedene lokalne varijable, i na njihovo mjesto piše "..." zato što one nisu nužne za opis funkcionalnosti klase.

Klasa *Renderer* upravlja svim ostalim dijelovima programa: poziva inicijalizaciju scene i sučelja, učitava pohranjene postavke i upravlja komunikacijom između objekata.

Slijedi klasa *GPUManager*, koja je najsloženija. Ona stvara *WebGL2* kontekst, poziva *WebGL2* funkcije, upravlja sjenčarima, sinkronizira rezoluciju iscrtavanja s veličinom kanvasa i definira kako će se izvršiti postupak iscrtavanja okvira, što je opisano detaljnije u sljedećem poglavlju.

U okviru klase *GUIManager* ostvarene su funkcionalnosti korisničkog sučelja tako da se pri učitavanju stranice sagradi sučelje i aktivno ažuriraju uniformne varijable sjenčara kada korisnik promijeni vrijednosti.

Klasa *Camera* služi za čuvanje informacija potrebnih za inicijalizaciju zraka i upravlja tim podacima na temelju korisničkih ulaza (tipkovnica i miš).



Slika 3.1. Dijagram glavnih razreda

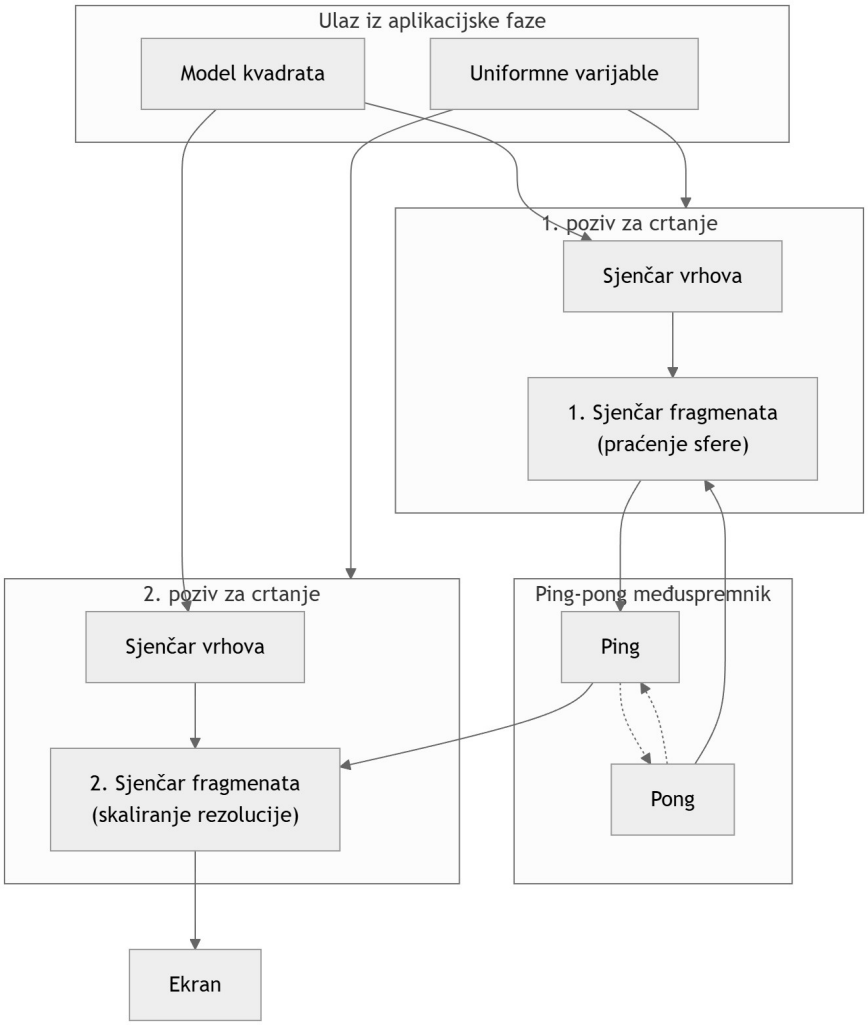
3.3. Postupak iscrtavanja

Iscrtavanje je stvaranje slike virtualne scene koja se prikazuje na ekranu. Za scene s ciljem prikaza u stvarnom vremenu, frekvencija iscrtavanja mora biti dovoljno velika. Radi brzine, veliki dio postupka iscrtavanja je izveden sklopovski i postoji definirani grafički protočni sustav kojeg se treba pridržavati da bi se nacrtao neki 3D model. Prilikom korištenja biblioteke *WebGL2* potrebno je obavezno definirati barem sjenčar vrhova (*engl. vertex shader*) i sjenčar fragmenata (*engl. fragment shader*). U ovome slučaju, modeli koje želimo prikazati se ne sastoje od trokuta i nije potrebna geometrijska faza. Bilo bi praktično koristiti računske sjenčare (*engl. compute shaders*), no kao što je već navedeno, biblioteka *WebGL2* ne omogućuje izradu računskih sjenčara. Stoga je postupak iscrtavanja izveden onime što nam je na raspolaganju i što se može koristiti u okviru biblioteke. U nastavku su objašnjeni postupci iscrtavanja u sastavljenom programu i navedeni su ključni objekti koji se koriste pri iscrtavanju. Slika 3.2. prikazuje redoslijed grafičkih programa i objekte koji se opisuju.

Crtanje jednog okvira se izvodi u dva poziva na crtanje (*engl. draw calls*). U svakom pozivu na crtanje se izvodi standardna grafička protočna struktura: sjenčar vrhova se iterira po točkama modela, a sjenčar fragmenata se iterira po fragmentima. Jedini pravi model koji se koristi je kvadrat (dva trokuta) koji pokriva cijeli ekran. Sjenčar vrhova samo prosljeđuje pozicije točaka modela kvadrata. Model i sjenčar vrhova su identični za oba poziva za crtanje. Gotovo sva logika koja se opisuje je u sjenčarima fragmenata.

Prvo iscrtavanje izvodi algoritam praćenje sfere i on obavi sjenčanje. Za izračun boje piksela se uzima u obzir boja prošlog okvira. Kako nije moguće iz istog spremnika čitati i pisati u isto vrijeme, rezervirani je par spremnika koji zajedno čine ping-pong međuspremnik (engl. *ping-pong buffer*), jer u jednom iscrtavanju, jedan spremnik služi za čitanje, a drugi za pisanje. Nakon svakog iscrtavanja uloga se tih spremnika zamijeni. Taj par spremnika koji omogućuje programu pristup informacijama o prošlom okviru se koristi za uklanjanje *alias artefakta*, izvedbu globalnog osvjetljenja i efekt fokusa kamere.

Drugi poziv uzima sliku u ping-pong međuspremniku, podesi boju slike i prikaže rezultat na *canvas HTML* element stranice. Koriste se dva poziva za crtanje jer u prvom pozivu (koji je više računski intenzivan) se može iscrtati slika manje rezolucije, a zatim se u drugom pozivu može skalirati međurezultat u rezoluciju za prikazivanje.



Slika 3.2. Dijagram postupka iscrtavanja

3.4. Korisničko sučelje

Dizajn korisničkog sučelja je načinjen uz pomoć standardnih *HTML* elemenata i *CSS* pravila, a funkcionalnost sučelja je ostvarena u programskom jeziku *JavaScript*. Izgled sučelja je vidljiv na slici 3.3. Cilj sučelja je omogućiti korisniku pristup što većem broju varijabli. Uz kretanje kroz virtualni prostor, ostvarena je mogućnost podešavanja načina prikaza i mijenjanje funkcija udaljenosti koje se prikazuju. Tekst sučelja je na engleskom jeziku. Upravljanje varijablama je ostvareno raznim kontrolama. Većina elemenata imaju kratak opis koji se pojavi ako se kratko vrijeme ostavi miš iznad njih.

Sučelje je podijeljeno u četiri kartice: generalne postavke, postavke vezane uz praćenje sfere, unos funkcije udaljenosti i informacije o kontrolama. U kartici za informacije o kontrolama su navedene i opisane funkcionalnosti tipki na tipkovnici. Generalne postavke uključuju podešavanje kamere, rezolucije i slikanje ekrana. Postavke za praćenje sfere imaju najveći utjecaj na brzinu iscrtavanja i mogu upravljati oblikom tijela. Konačno, u zadnjoj kartici se može unijeti funkcija udaljenosti. Postoje kratke upute za izradu funkcija i jedan primjer koda.

Slika 3.3. Korisničko sučelje (lijevo: stolno računalo, desno: mobilni uređaj)

3.5. Unos funkcije

Funkcija udaljenosti se unosi u tekstualni okvir u obliku izvornog teksta programa u programskom jeziku *GLSL*. Biblioteka *WebGL2* koristi programski jezik *GLSL* za sjenčare. Nakon unosa programskog koda treba se pokrenuti kompajliranje kako bi se uneseni programski kod primijenio. Prvo se izbriše postojeći program sjenčar kojeg biblioteka *WebGL2* koristi za iscrtavanje. Zatim se uneseni izvorni kod u obliku teksta spoji s izvornim tekstom programa ostatka sjenčara fragmenata. Konačno se novi spojeni izvorni kod pošalje grafičkoj kartici, kompajlira se i stvori se novi program za iscrtavanje. Ako dođe do greške prilikom kompajliranja (kao na primjer sintaksna greška unesenog koda), ispisat će se poruka koja daje informaciju o greški radi jednostavnijeg popravljanja koda.

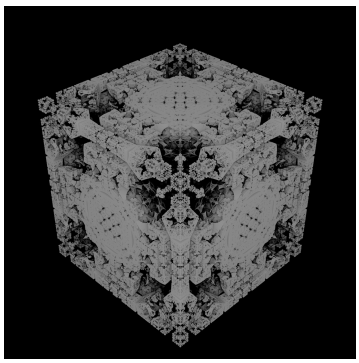
Zadana funkcija udaljenosti koja se koristi kada se pokrene program je navedena na slici 3.4. To je ujedno i implementacija izraza 2.1 iz potpoglavlja 2.1. za oblik sfere. Sfera se nalazi u centru koordinatnog sustava i radius sfere poprima vrijednost varijable *uniforms.custom_b*. Program ima nekoliko uniformnih varijabli sjenčara koje su namijenjene za korištenje u funkcijama udaljenosti (u to je uključena varijabla *uniforms.custom_b*). Te varijable omogućuju podešavanje vrijednosti proizvoljnih brojeva u funkciji kroz sučelje tako da nije potrebno ponovno prevođenje koda nakon svake promjene.

```
float SDF(vec3 p) {
    float radius = uniforms.custom_b;
    return length(p) - radius;
}
```

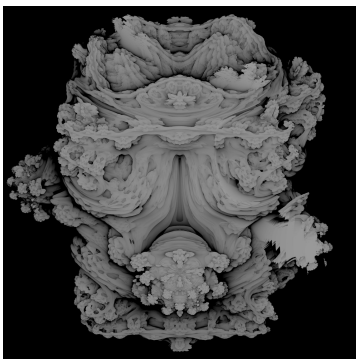
Slika 3.4. Funkcija udaljenosti koja opisuje sferu

3.6. Ugrađeni fraktali

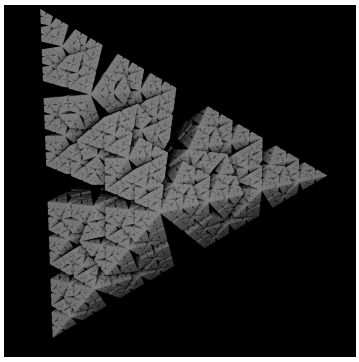
Sučelje nudi odabir funkcije udaljenosti koja se prikazuje. Prvi odabir je *"Custom"*, što znači da prikazuje funkciju koja se nalazi u polju za unos funkcije. A druge opcije su gotove funkcije četiri popularnih fraktala: *mandelbox* [4] (slika 3.5.), *mandelbulb* [5] (slika 3.6.), Kochova krivulja [6] (slika 3.7.) i *juliabulb* [7] (slika 3.8.). Funkcije tih fraktala su pronađene na Internetu, te se njihove poveznice nalaze u literaturi.



Slika 3.5. *Mandelbox*



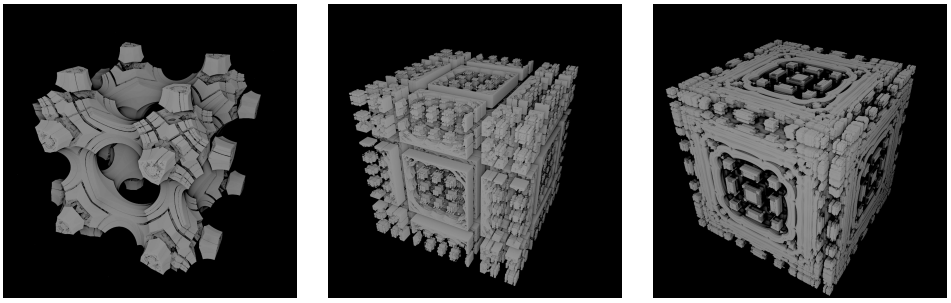
Slika 3.6. *Mandelbulb*



Slika 3.7. Kochova krivulja

Slika 3.8. *Juliabulb*

Za svaki od fraktala je moguće mijenjati nekoliko njegovih parametra. Slika 3.9. je primjer fraktala *mandelbox* s različitim vrijednostima parametra. Interesantno je vidjeti koliko se oblik izmijeni promjenom par brojeva.



Slika 3.9. Fraktal *mandelbox* s različitim parametrima

3.7. Načini prikaza

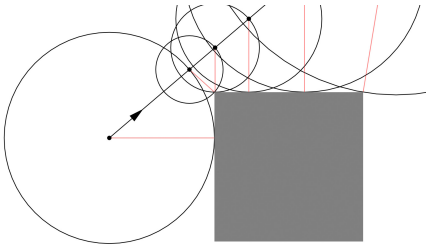
Prikaz tijela se ostvaruje u dva koraka: određivanje točke na površini tijela algoritmom praćenja sfere i pretvaranje informacija o toj točki u boju na ekranu. Algoritam praćenja sfere je ostvaren prema pseudokodu prikazanom na slici 2.4., a programski kod rješenja je naveden u algoritmu na slici 3.10. Za algoritam se koriste dvije strukture: *Ray* i *Data*. Struktura *Ray* opisuje zraku i koristi se kao ulaz u algoritam, a struktura *Data* služi za čuvanje podataka koje algoritam izračuna u jednom paketu. Pretvaranje podataka u boju je izveden na četiri načina i oni su u programu nazvani "načini prikaza".

```
// struktura za opis zrake
struct Ray {
    vec3 origin; // ishodiste zrake
    vec3 direction; // smjer zrake
};

// kontejner informacija za rezultat algoritma
struct Data {
    vec3 position; // tocka presjeka zrake i tijela
    bool collided; // je li zraka udarila tijelo
    vec3 normal; // normala povrsine u tocki presjeka
    float dist; // duljina puta kojem je zraka putovala
    int marches; // broj koraka kojih je zraka otputovala
};

// algoritam pracenja sfere
Data rayMarch(Ray ray) {
    Data data;
    data.position = ray.origin;
    data.collided = false;
    data.marches = 0;

    for (; data.marches < int(uniforms.max_marches); data.marches++) {
        float d = SDF(data.position);
        if (d < pow(2.0, -uniforms.detail)) {
            data.collided = true;
            break;
        }
        data.position += d * ray.direction;
    }

    data.normal = deriveNormal(data.position, uniforms.normals_precision);
    data.dist = length(data.position - ray.origin);
    return data;
}
```

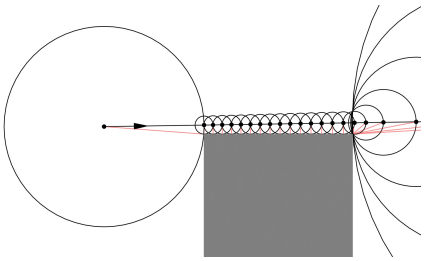
Slika 3.10. Isječak programskog koda za praćenje sfere

3.7.1. Broj koraka

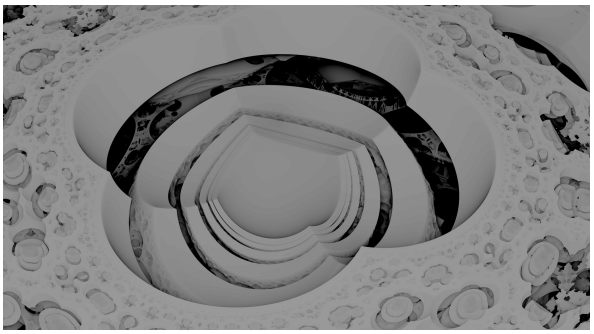
Prvi način prikaza objekta je temeljen na broju koraka kojih zraka obavi prije nego dotakne površinu. Broj koraka se preslikava na sljedeći način: ako je broj koraka nula, boja piksela je bijela, a ako je broj koraka došao do maksimuma, onda je boja crna. Sve između je linearno interpolirano prema algoritmu prikazanom na slici 3.14. Ovaj način prikaza iskorištava svojstvo algoritma da zrake koje prolaze blizu tijela izvedu više koraka. Usporedba zrake koja prolazi blizu tijela u odnosu na zraku koja prolazi dalje od tijela je prikazana na slikama 3.11. i 3.12. Konačna slika ostvarena ovim načinom prikaza (3.13.) istovremeno prikazuje dojam dubine i označava mjesta gdje ima najviše detalja.



Slika 3.11. Zraka koja ne prolazi blizu površine tijela



Slika 3.12. Zraka koja prolazi blizu površine tijela, a ne dotakne ga



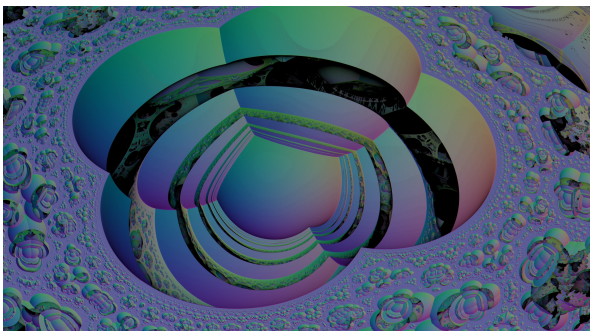
Slika 3.13. Prikaz temeljen na broju koraka zrake

```
void main() {
    ...
    Data data = rayMarch(camera_ray);
    float factor = 1.0 - (float(data.marches) / uniforms.max_marches);
    pixel_color = vec3(mix(0.0, factor, float(data.collided)));
    ...
}
```

Slika 3.14. Isječak programskog koda za način prikaza brojem koraka zrake

3.7.2. Normale

Drugi način prikaza služi za prikaz vrijednosti normale (slika 3.15.). Normala je jedinični vektor koji je usmjeren okomito na površinu. Vrijednost normale se može izračunati derivacijom funkcije udaljenosti za svaku os zasebno.



Slika 3.15. Prikaz temeljen na vektoru normale površine

Za aproksimaciju derivacije funkcije koristi se metoda središnjih razlika (*engl. central difference method*) navedena u izrazu (3.1). Funkcija se mora evaluirati dva puta za svaku os (šest puta za tri osi) i kao parametar se koristi proizvoljan vrlo mali broj dx. Izračuna normale u točki je naveden u algoritmu na slici 3.16.

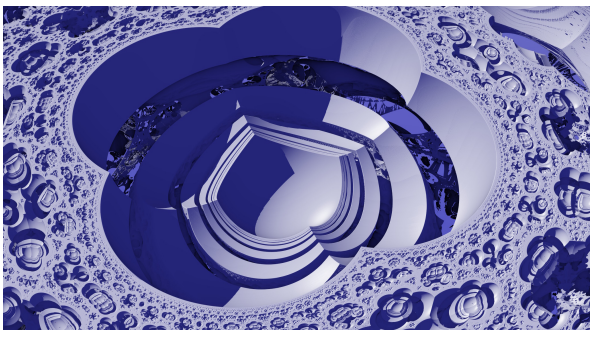
$$f'(x) \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} \tag{3.1}$$

```
// varijabla "epsilon" je ekvivalentna "dx" iz gornje formule
vec3 deriveNormal(vec3 position, float epsilon) {
    vec3 normal;
    normal.x = SDF(position + vec3(epsilon, 0.0, 0.0))
        - SDF(position - vec3(epsilon, 0.0, 0.0));
    normal.y = SDF(position + vec3(0.0, epsilon, 0.0))
        - SDF(position - vec3(0.0, epsilon, 0.0));
    normal.z = SDF(position + vec3(0.0, 0.0, epsilon))
        - SDF(position - vec3(0.0, 0.0, epsilon));
    return normalize(normal  / (2.0 * epsilon));
}
// pretvaranje vektora normale u boju
void main() {
    ...
    Data data = rayMarch(camera_ray);
    float factor = 1.0 - (float(data.marches) / uniforms.max_marches);
    if (data.collided)
        pixel_color = vec3((data.normal * 0.5 + 0.5) * factor);
    else
        pixel_color = vec3(0.0);
    ...
}
```

Slika 3.16. Isječak programskog koda za način prikaza normala

3.7.3. Lokalno osvjetljenje

Treći način prikaza je lokalno osvjetljenje ostvareno s Phongovim modelom (slika 3.17.). Postoji samo jedan izvor svjetlosti: sunce. Uz lokalni model se prikazuje oštra sjena. Sjena je ostvarena slanjem zrake od točke na površini prema izvoru svjetlosti i provjerom je li zraka putem udarila nešto. Programsko ostvarenje je prikazano u algoritmu na slici 3.18.



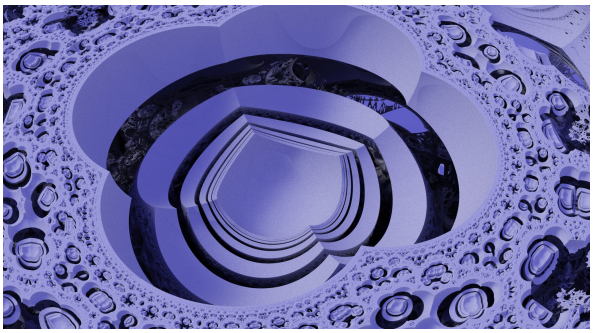
Slika 3.17. Prikaz s lokalnim osvjetljenjem i sjenama

```
void main() {
    ...
    Data camera_data = rayMarch(camera_ray);

    // izracun komponenta osvjetljenja
    float diffuse = clamp(dot(camera_data.normal, uniforms.sun_direction),
        0.0, 1.0);
    float specular = clamp(pow(dot(camera_data.normal
        , uniforms.sun_direction), 50.0), 0.0, 1.0);
    vec3 ambient = skyValue(vec3(0.0, 0.0, -1.0)) * 0.5;
    float ao = 1.0 - (float(camera_data.marches) / uniforms.max_marches);
    vec3 intensity = skyValue(uniforms.sun_direction)
        / uniforms.sun_intensity * 2.0;

    // slanje zrake i izracun sijene
    Ray shadow_ray;
    shadow_ray.direction = uniforms.sun_direction;
    shadow_ray.origin = camera_data.position + camera_data.normal
        * uniforms.epsilon;
    Data shadow_data = rayMarch(shadow_ray);
    float shadow = float(shadow_data.dist > 100.0);

    // spajanje komponenata u boju
    if (camera_data.collided)
        pixel_color = vec3((1.0 + specular) * diffuse * shadow
            * intensity + ambient) * ao;
    else
        pixel_color = vec3(0.0);
    ...
}
```

Slika 3.18. Isječak koda za implementaciju lokalng osvjetljenja i sjena

3.7.4. Globalno osvjetljenje

Zadnji, četvrti način prikaza je globalno osvjetljenje (slika 3.19.). Koristi se pojednostavljeni algoritam praćenja putanje zrake (*engl. path tracing*) s ping-pong spremnikom. Kada zraka dođe do površine, odbije se u nasumičnom smjeru i nastavi se odbijati dok ne prijeđe maksimalan broj odbijanja (rezultat je crna boja) ili otiđe u nebo (rezultat je boja neba u tom smjeru) (algoritam na slici 3.20.). Rezultat je prosjek svih boja koje su dobivene za dani piksel. Kako je ovaj postupak računski intenzivan, radi responzivnosti i optimizacije računa se prosjek između prošlog i trenutačnog okvira koji se iscrtava. Time treba manje vremena za izračun pojedinog okvira, a slika se postepeno razvija u bolji rezultat što se duže program iscrtava.



Slika 3.19. Prikaz s globalnim osvjetljenjem

```
vec3 pathTrace(Ray camera_ray, inout uint seed) {
    Ray ray = camera_ray;
    for (int bounce_counter = 0; bounce_counter < int(uniforms.max_bounces)
        ; bounce_counter++) {
        Data data = rayMarch(ray);

        if (!data.collided && data.dist > 10.0)
            return skyValue(ray.direction);

        ray.direction = normalize(randomDirection(seed) + data.normal);
        ray.origin = data.position + ray.direction * uniforms.epsilon;
    }
    return vec3(0.0);
}

void main() {
    ...
    pixel_color = pathTrace(camera_ray, seed);
    // izracun prosjeka trenutne boje i boje proslog okvira
    output_color = vec4(previous_color.xyz
        * ((uniforms.temporal_counter - 1.0) / uniforms.temporal_counter)
        + pixel_color * (1.0 / uniforms.temporal_counter), 1.0);
    ...
}
```

Slika 3.20. Isječak koda za implementaciju globalnog osvjetljenja

3.8. Web stranica i izvorni tekst programa

Izvorni tekst kod programa rada se nalazi na *GitHub* repozitoriju na sljedećoj poveznici:
`https://github.com/martzin23/webgl2-fractal-raymarcher`

Aplikacija se izvodi na web pregledniku i kompatibilna je s mobilnim uređajima. Bitno je naglasiti da program neće ispravno raditi ako uređaj ili web preglednik ne podržavaju sve korištene značajke biblioteke *WebGL2*. Aplikacija je dostupna na sljedećoj poveznici:
`https://martzin23.github.io/webgl2-fractal-raymarcher/`

4. Rezultati i rasprava

4.1. Mjerenje brzine izvođenja programa

Ispitivanje je provedeno na tri uređaja. Tablica 4.1. prikazuje specifikacije uređaja. Postoje slučajevi kada brzina iscrtavanja prestaje rasti i ostane konstantna. To je zato što internetski preglednik automatski ograniči brzinu iscrtavanja na brzinu osvježavanja ekrana. U svakom testu koriste se sve zadane postavke i vrijednosti varijabli, osim varijable koja se u testu mijenja i odabrana funkcija udaljenosti (u svakom testu je odabran fraktal *mandelbox*). Za ispitivanje su odabrane varijable za koje se očekuje da imaju značajan utjecaj na performanse, a to su djelitelj rezolucije, maksimalan broj koraka zrake i način prikaza. S obzirom na to da je složenost programa najveća unutar sjenčara, očekivano je da će performanse najviše ovisiti o grafičkom procesoru uređaja.

Tablica 4.1. Specifikacije uređaja za testiranje

Uređaj	Naziv	Procesor (CPU)	Grafički procesor (GPU/iGPU)	Maksimalna brzina iscrtavanja (FPS)
1	Desktop PC	i5-14400F @ 2.50GHz	RTX 4060Ti	144 Hz
2	HP ProBook 650 G5	i5-8265U @ 1.60GHz	UHD Graphics 620	60 Hz
3	Redmi Note 9	Octa-core Max @ 2.00GHz	Mali-G52 MC2	60 Hz

Prvo mjerenje uključuje podešavanje veličine rezolucije iscrtavanja (tablica 4.2.). U ovom slučaju rezolucija ne znači veličinu ekrana, nego se odnosi specifično na vrijednosti prozora za prikaz (*engl. viewport*) *WebGL2* programa za prvi poziv na iscrtavanje. Iz rezultata se vidi da rezolucija ima veliki utjecaj na brzinu iscrtavanja. To je zato što se algoritam nalazi u sjenčaru fragmenata koji se izvodi za svaki piksel slike.

Tablica 4.2. Brzina iscrtavanja u ovisnosti o rezoluciji

Uređaj	Djeljitelj rezolucije	Rezolucija (širina x visina)	Brzina scrtavanja (FPS)
1	1	2048 x 1131	144 Hz
1	2	1024 x 565	144 Hz
1	3	682 x 377	144 Hz
2	1	1408 x 779	10 Hz
2	2	704 x 389	34 Hz
2	3	469 x 259	60 Hz
3	1	365 x 292	23 Hz
3	2	182 x 149	46 Hz
3	3	121 x 99	60 Hz

Drugo mjerenje odnosi se na parametar u algoritmu praćenja sfere (tablica 4.3.). Mijenja se maksimalan broj koraka kojih zraka može obaviti prije nego algoritam zaključi da zraka nije naišla na prepreku. Povećavanjem vrijednosti varijable se smanjuje brzina iscrtavanja do neke razine, a zatim varijabla prestaje negativno utjecati na brzinu. To je zato što sve zrake imaju dovoljno koraka na raspolaganju da stignu do površine tijela i ostatak koraka ne koriste. To se događa zato što je kamera bila okrenuta prema tijelu. Ako bi kamera gledala od tijela, zrake bi išle u beskonačnost, odnosno uvijek bi iskoristile sve dostupne korake i varijabla bi u višim vrijednostima nastavila negativno utjecati na brzinu.

Tablica 4.3. Brzina iscrtavanja u ovisnosti o maksimalnom broju koraka zrake

Uređaj	Maksimalan broj koraka	Brzina iscrtavanja (FPS)
1	50	144 Hz
1	100	144 Hz
1	200	144 Hz
1	400	144 Hz
2	50	31 Hz
2	100	26 Hz
2	200	24 Hz
2	400	23 Hz
3	50	48 Hz
3	100	42 Hz
3	200	41 Hz
3	400	41 Hz

Treće mjerenje se odnosi na ispitivanje načina prikaza. Rezultati su bili predvidivi, tj. složeniji načini su naravno sporiji. Jedino se ističe mala je razlika u brzini između

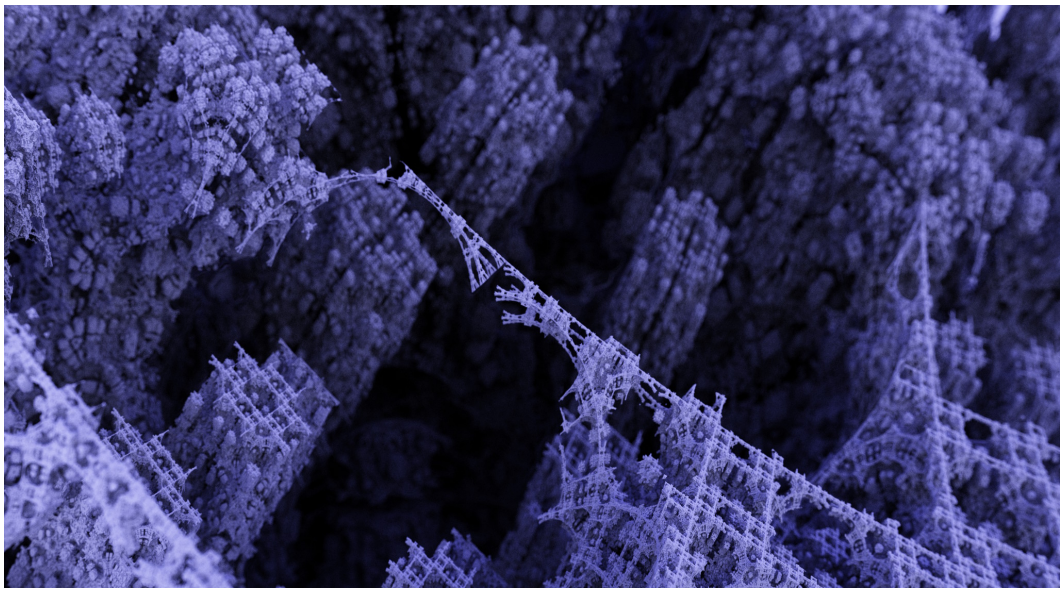
lokalnog i globalnog osvjetljenja. No, to je zato što način globalnog osvjetljenja koristi prosjek do tada iscrtanih okvira pa se jedan okvir ne može klasificirati kao gotova slika.

Tablica 4.4. Brzina iscrtavanja u ovisnosti o načinu iscrtavanja

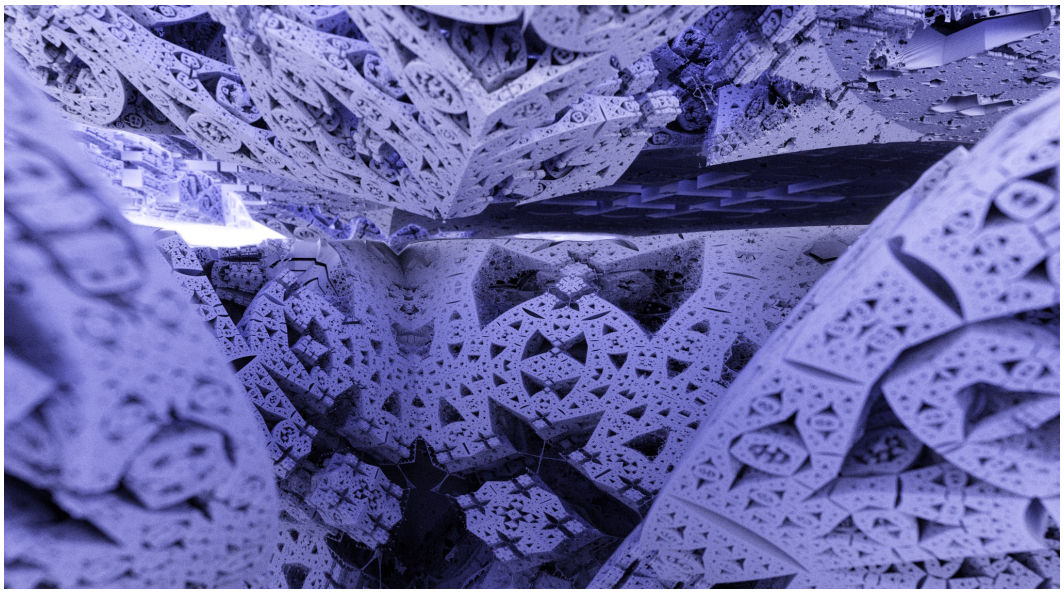
Uređaj	Način prikaza	Brzina scrtavanja (FPS)
1	Broj koraka	144 Hz
1	Normale	144 fps
1	Lokalno osvjetljenje	103 Hz
1	Globalno osvjetljenje	64 Hz
2	Broj koraka	31 Hz
2	Normale	26 Hz
2	Lokalno osvjetljenje	14 Hz
2	Globalno osvjetljenje	10 Hz
3	Broj koraka	48 Hz
3	Normale	45 Hz
3	Lokalno osvjetljenje	31 Hz
3	Globalno osvjetljenje	22 Hz

4.2. Slike rezultata

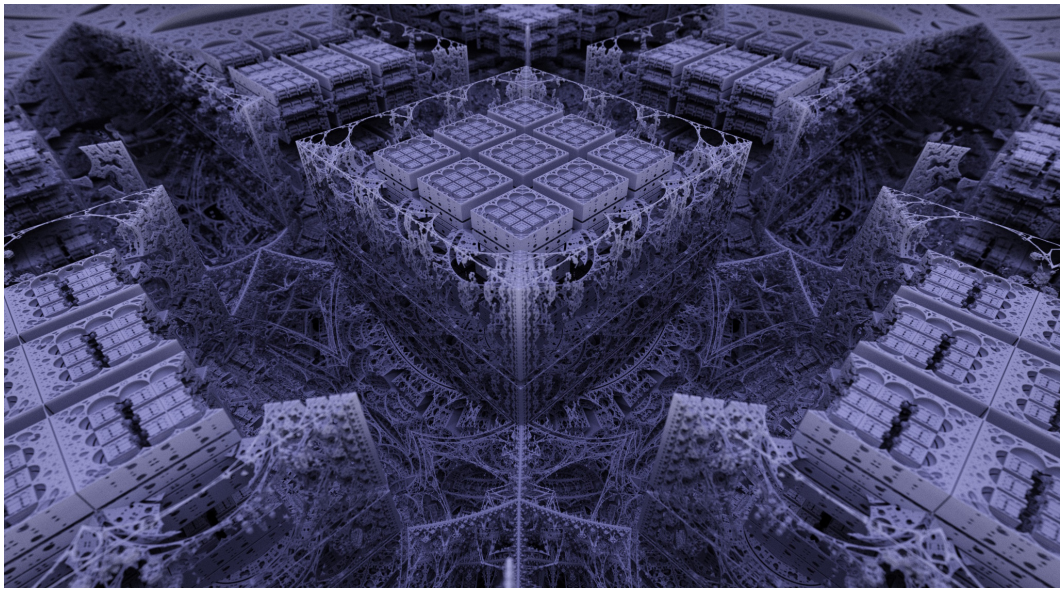
Na slikama koje slijede vizualizirano je nekoliko fraktala s globalnim osvjetljenjem (slike 4.1. do 4.7.). Svaka slika se iscrtavala oko dvije minute. Na slikama je naveden fraktal koji je odabran s vrijednostima korištenih parametara.



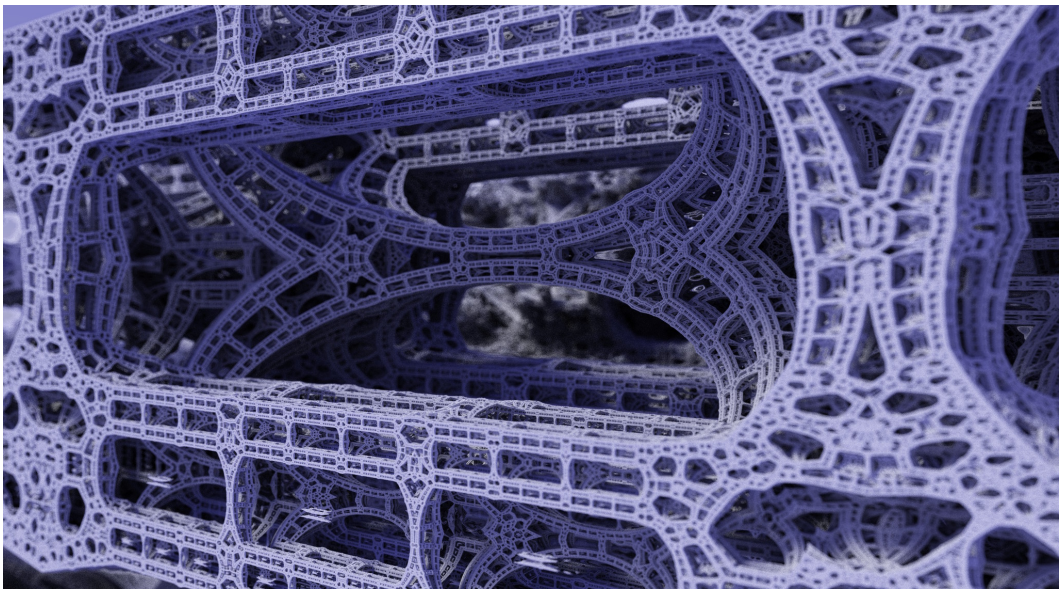
Slika 4.1. Fraktal *mandelbox* (parametri: *Scale*: -2, *Folding limit*: 1.09, *Min radius*: 0.27, *Fixed radius*: 1, *Folding value*: 2)



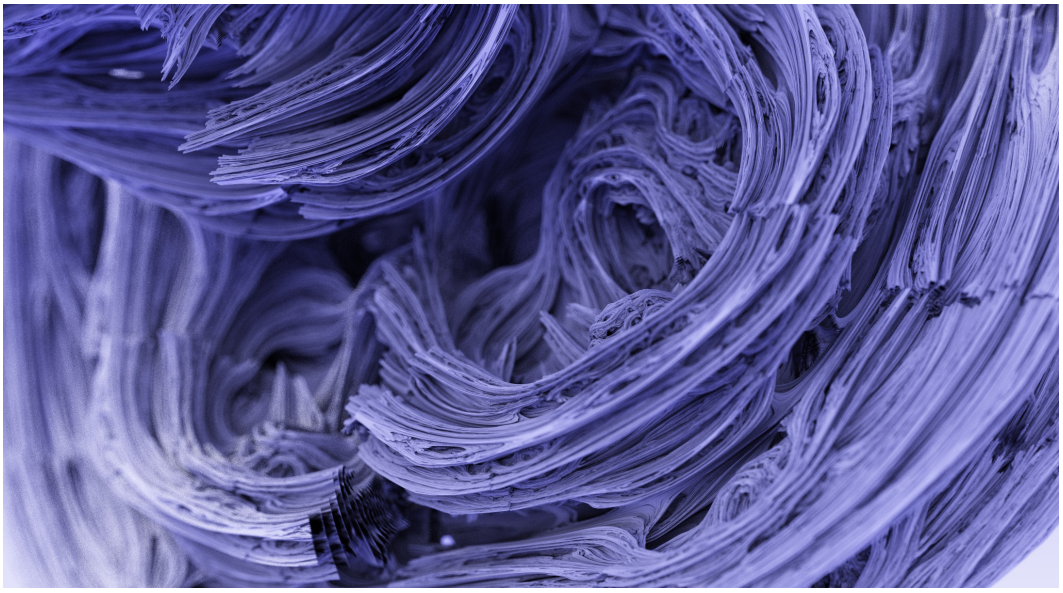
Slika 4.2. Fraktal *mandelbox* (parametri: *Scale*: -1.73, *Folding limit*: 0.77, *Min radius*: 0.68, *Fixed radius*: 2.78, *Folding value*: 2.01)



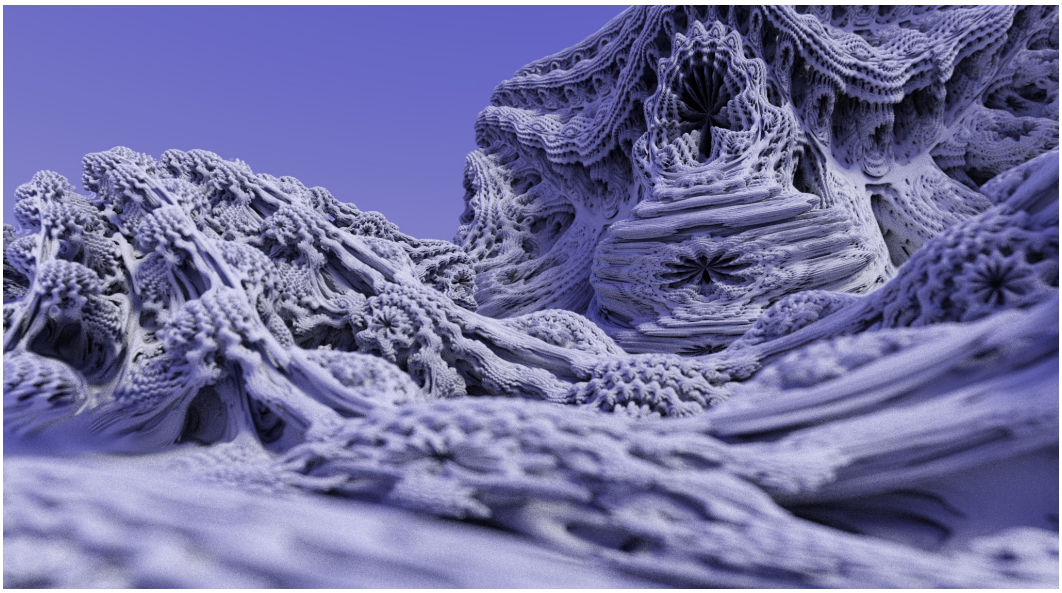
Slika 4.3. Fraktal *mandelbox* (parametri: *Scale*: -1.86, *Folding limit*: 0.65, *Min radius*: 0.73, *Fixed radius*: 2.09, *Folding value*: 2)



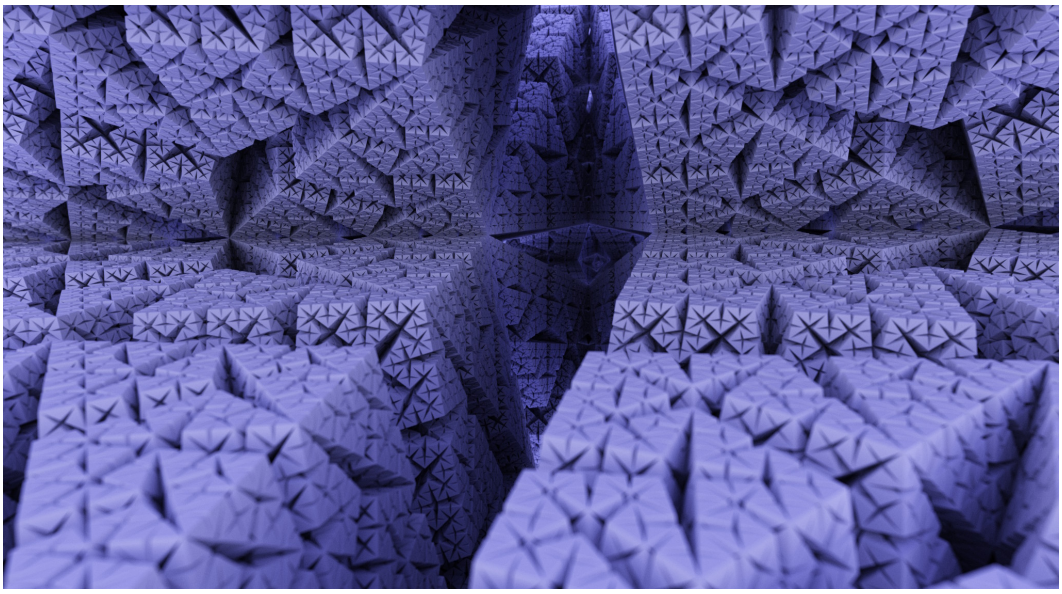
Slika 4.4. Fraktal *mandelbox* (parametri: *Scale*: -1.56, *Folding limit*: 0.96, *Min radius*: 0.87, *Fixed radius*: 2, *Folding value*: 2)



Slika 4.5. Fraktal *juliabulb* (parametri: *Exponent*: 3.13, *Phi multiplier*: 0.85, *Theta multiplier*: 1.72)



Slika 4.6. Fraktal *mandelbulb* (parametri: *Power*: 9.51)



Slika 4.7. Fraktal Kochove krivulje (parametri: *Scale*: 1, *Spacing*: 1, *Inversion*: 1)

5. Zaključak

U ovom radu su vizualizirani fraktali. Prikaz je dovoljno računski jednostavan pa se je brzina iscrtavanja odvija u stvarnom vremenu. Istovremeno je dostupna mogućnost povećanja kvalitete i korištenje globalnog osvjetljenja za dobivanje kvalitetnih slika.

Može se zaključiti da je korištena grafička biblioteka *WebGL2* dovoljno sposobna za ovakve zadatke iako ne nudi mogućnost korištenja računskih sjenčara. Veliki broj uređaja i internetskih preglednika podržava biblioteku *WebGL2* pa nije bilo problema prilikom ispitivanja aplikacije na drugim uređajima, što je prednost u odnosu na moderniju alternativu biblioteke *WebGPU*, koja je naprednija, ali još ne podržava veliki broj uređaja.

Najteže je bilo ostvariti korisničko sučelje. Nije korištena biblioteka za *widgete*, stoga su svi ulazni elementi ostvareni u okviru ovog rada. Trebalo je ostvariti sinkronizaciju vrijednosti varijabli u sučelju s pripadajućim vrijednostima uniformnih varijabli u sjenčarima. Kontrole na mobilnim uređajima su bile neintuitivne za izvesti zato što su na raspolaganju dostupne samo informacije točaka prstiju koji diraju ekran.

Postoji prostor za proširenja i program se može unaprijediti na više načina. U programu je moguće lako mijenjati veličinu rezolucije koja se iscrtava, no bilo bi praktično da se rezolucija automatski prilagođava. Na primjer, kada se kamera kreće, mogla bi se smanjiti rezolucija tako da se brže iscrtava scena te bi kretanje bilo glatko. A kada kamera stoji mirno, rezolucija bi se mogla povećati kako bi slika bila oštrija.

Literatura

[1] I. Quilez, “2d distance functions”, https://iquilezles.org/articles/distfunctions2d/, [mrežno; stranica posjećena: svibanj 2026.].

[2] Arunodhai, “A detailed close-up shot of a green fern leaf against a white background”, https://www.pexels.com/photo/green-leaf-photography-691043/, [mrežno; stranica posjećena: svibanj 2026.].

[3] Omar, “Dear imgui repository”, https://github.com/ocornut/imgui, [mrežno; stranica posjećena: svibanj 2026.].

[4] M. H. Christensen, “Distance estimated 3d fractals (vi): The mandelbox”, *Syntopia*, 2011., [stranica posjećena: svibanj 2026.]. [Mrežno]. Adresa: http://blog.hvidtfeldts.net/index.php/2011/11/distance-estimated-3d-fractals-vi-the-mandelbox/

[5] S. Lague, “Coding adventure: Ray marching”, https://youtu.be/Cp5WWtMoeKg, 2019., [mrežno; stranica posjećena: svibanj 2026.].

[6] Angramme, “fractal_viewer: A simple ray-marched fractal renderer in processing and glsl”, https://github.com/Angramme/fractal_viewer, 2024., [mrežno; stranica posjećena: svibanj 2026.].

[7] A. Sołtysik, “Fractalview: 3d fractal renderer in glsl.” https://github.com/adamsol/FractalView, 2022., [mrežno; stranica posjećena: svibanj 2026.].

Sažetak

Vizualizacija trodimenzijskih fraktalnih objekata u WebGL2 okruženju

Martin Golub

Ovaj rad se bavi vizualizacijom fraktala. Navedena je osnovna teorija o fraktalima, funkcijama udaljenosti koje se koriste za njihov opis te algoritam praćenja sfere za iscrtavanje tih funkcija. Program je ostvaren u programskom jeziku *JavaScript* koristeći biblioteku *WebGL2* unutar internetskog preglednika. Opisana je struktura programskog koda, postupak iscrtavanja i korisničko sučelje koje je ostvareno koristeći alate za prikaz web stranica *HTML* i *CSS*. Istraženi su načini pretvorbe podataka iz algoritma praćenja sfere u sliku za prikaz na ekranu i provedena su mjerenja performansi programa.

Ključne riječi: fraktali; WebGL2; marširanje zrakom; praćenje sfera; funkcija udaljenosti; vizualizacija

Abstract

Visualization of three-dimensional fractal objects in WebGL2

Martin Golub

This paper deals with the visualization of fractals. The basic theory of fractals is presented, along with the distance functions used to describe them and the sphere tracing algorithm for rendering those functions. The program was implemented in the *JavaScript* programming language using the *WebGL2* library within a web browser. The structure of the program code, the rendering process, and the user interface implemented using the web page display tools *HTML* and *CSS* are described. Methods for converting data from the sphere tracing algorithm into an image for display on screen are explored, and performance measurements of the program are conducted.

Keywords: fractals; WebGL2; ray marching; sphere tracing; signed distance function; visualization