

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2221

OPTIMIZACIJSKE STRUKTURE PODATAKA BSP I PVS

Stjepan Džidžić

Zagreb, lipanj 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2221

OPTIMIZACIJSKE STRUKTURE PODATAKA BSP I PVS

Stjepan Džidžić

Zagreb, lipanj 2026.

ZAVRŠNI ZADATAK br. 2221

Pristupnik: **Stjepan Džidžić (0036559030)**
Studij: Elektrotehnika i informacijska tehnologija i Računarstvo
Modul: Računarstvo
Mentorica: prof. dr. sc. Željka Mihajlović

Zadatak: **Optimizacijske strukture podataka BSP i PVS**

Opis zadatka:

Prilikom prikaza velikog niza povezanih prostorijskih objekata važno je ostvariti prikaz u stvarnom vremenu. Kako bi takav prikaz bio izvediv u stvarnom vremenu važno je primijeniti optimizacijske strukture podataka pogodne za zatvorene prostore. Proučiti način izgradnje i korištenje optimizacijskih struktura podataka binarne podjele prostora BSP (engl. Binary Space Partitioning) i potencijalno vidljivih skupova PVS prostorijskih objekata (engl. Potentially Visible Sets). Na temelju provedenog istraživanja potrebno je osmisliti, razviti i optimizirati aplikaciju koja implementira navedene optimizacijske strukture. Provesti testiranje na nizu primjera te analizirati i ocijeniti ostvarene rezultate. Usporediti rezultate uz korištenje i bez korištenja optimizacijskih struktura podataka. Razmotriti upotrebljivost dobivenih rezultata i moguće smjernice proširenja. Izraditi odgovarajući programski proizvod koristeći programski jezik C++ i grafičko programsko sučelje OpenGL. 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

Uvod	1
1. Teorijska podloga	2
1.1. Prikaz 3D grafike u stvarnom vremenu	2
1.1.1. Grafički protočni sustav	2
1.1.2. Problem vidljivosti objekata	3
1.2. Binarna podjela prostora (BSP)	4
1.2.1. Definicija BSP stabla	4
1.2.2. Konstrukcija BSP stabla	4
1.2.3. Obilazak BSP stabla	6
1.2.4. Prednosti i nedostaci BSP-a	7
1.3. Potencijalno vidljivi skupovi (PVS)	8
1.3.1. Koncept potencijalno vidljivih skupova	8
1.3.2. Portali i povezanost prostora	8
1.3.3. Generiranje PVS podataka	9
1.3.4. Korištenje PVS-a tijekom izvođenja	11
2. Implementacija aplikacije	12
2.1. Korištene tehnologije	12
2.2. Arhitektura aplikacije	13
2.3. Implementacija BSP strukture	14
2.3.1. Struktura BSP čvora	14
2.3.2. Algoritam izgradnje stabla	15
2.3.3. Obilazak i iscertavanje	17
2.4. Implementacija PVS sustava	18
2.4.1. Struktura portala i njihova usmjerenost	18
2.4.2. Gruba faza vidljivosti	19

2.4.3.	Fina faza vidljivosti	21
2.4.4.	Rekurzivni tok vidljivosti	22
2.4.5.	Korištenje PVS-a pri iscertavanju	24
3.	Testiranje i analiza rezultata	26
3.1.	Testno okruženje.....	26
3.2.	Testne scene.....	26
3.3.	Metodologija mjerenja.....	30
3.4.	Rezultati.....	30
3.5.	Analiza i rasprava	31
	Zaključak	33
	Literatura	34
	Izjava o korištenju umjetne inteligencije.....	35
	Sažetak.....	36
	Summary.....	37
	Skraćenice.....	38

Uvod

Računalna grafika se koristi u velikom broju područja, uključujući videoigre, simulacije, virtualnu i proširenu stvarnost te različite vizualizacije. Razvojem grafičkog hardvera omogućilo je prikazivanje sve složenijih trodimenzionalnih scena koje sadrže velik broj objekata i detalja.

Iako je napredak hardvera značajan, prikaz složenih scena u stvarnom vremenu i dalje predstavlja izazov. Tijekom iscrtavanja scene potrebno je obraditi veliki broj objekata, od kojih značajan dio nije vidljiv promatraču. Obrada takvih objekata nepotrebno utječe na performanse aplikacije čime se može narušiti osjećaj iscrtavanja u stvarnom vremenu.

Kako bi se smanjio broj obrađenih i iscrtanih objekata, koriste se različite optimizacijske metode za određivanje vidljivosti objekata. Posebno su zanimljive zatvorene scene sastavljene od prostorija i hodnika, gdje zidovi ograničavaju vidljivosti velikog dijela scene. U slučaju takvih scena, moguće je postići značajna ubrzanja primjenom odgovarajućih struktura podataka i algoritama.

Cilj ovog rada je istražiti optimizacijske strukture podataka binarne podjele prostora (engl. *Binary Space Partitioning, BSP*) i potencijalno vidljivih skupova (engl. *Potentially Visible Sets, PVS*), analizirati njihove karakteristike te implementirati prikaz zatvorenih scena koristeći navedene metode optimizacije. Implementacija je ostvarena korištenjem programskog jezika C++ i grafičkog sučelja OpenGL. Na kraju rada provedena je usporedba performansi s optimizacijama i bez njih te je analiziran njihov utjecaj na učinkovitost iscrtavanja scene.

1. Teorijska podloga

1.1. Prikaz 3D grafike u stvarnom vremenu

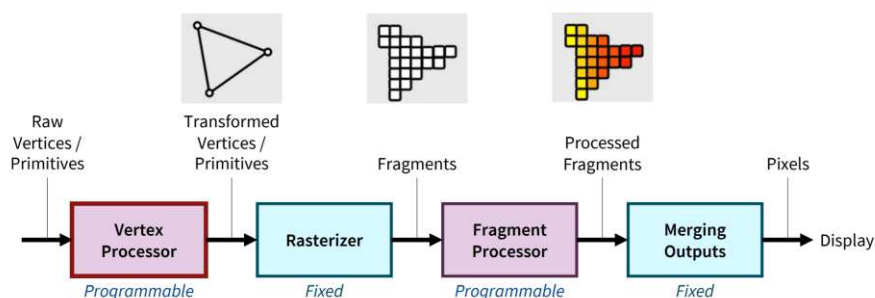
1.1.1. Grafički protočni sustav

Postupak iscrtavanja se sastoji od tri glavna koraka: obrađivanje vrhova, rasterizacija i obrađivanje fragmenata (Sl. 1.1).

Prvi korak je procesiranje vrhova gdje obrađujemo informacije o vrhovima i prosljeđujemo obrađene podatke dalje u sljedeće korake. Glavni podatak koji se obrađuje kod vrhova je sama pozicija vrha, moguće je obrađivati i druge podatke npr. normala vrha i boja vrha. Program koji obrađuje vrhove naziva se sjenčar vrhova. Objekti su obično zapisani tako da je pozicija vrhova u lokalnom prostoru objekta. Posljedica ovog je da će se morati transformirati taj lokalni prostor objekta u prostor koji ćemo na kraju vidjeti. Transformacija u projekcijski prostor se vrši u sljedećem redoslijedu: transformacija lokalnog prostora objekta u globalni prostor svijeta, transformacija iz globalnog prostora svijeta u prostor pogleda i na kraju iz prostora pogleda u prostor ekrana.

U drugom koraku, nakon što smo obradili vrhove, pokreće se postupak rasterizacije koji pojedinačne primitive dobivene od sjenčara vrhova pretvara u fragmente. Te primitive mogu biti crte ako iscrtavamo objekt u žičnoj formi ili više uobičajeno trokuti ako iscrtavamo objekte popunjene punom bojom. Fragment je dio primitive koji je pretvoren u dio zaslona koji je spreman za iscrtavanje i kojem će kasnije biti pridružena boja.

Na kraju, u sjenčaru fragmenata, dobivene fragmente iz postupka rasterizacije je potrebno obraditi tako što izaberemo željenu boju piksela na tom dijelu zaslona.



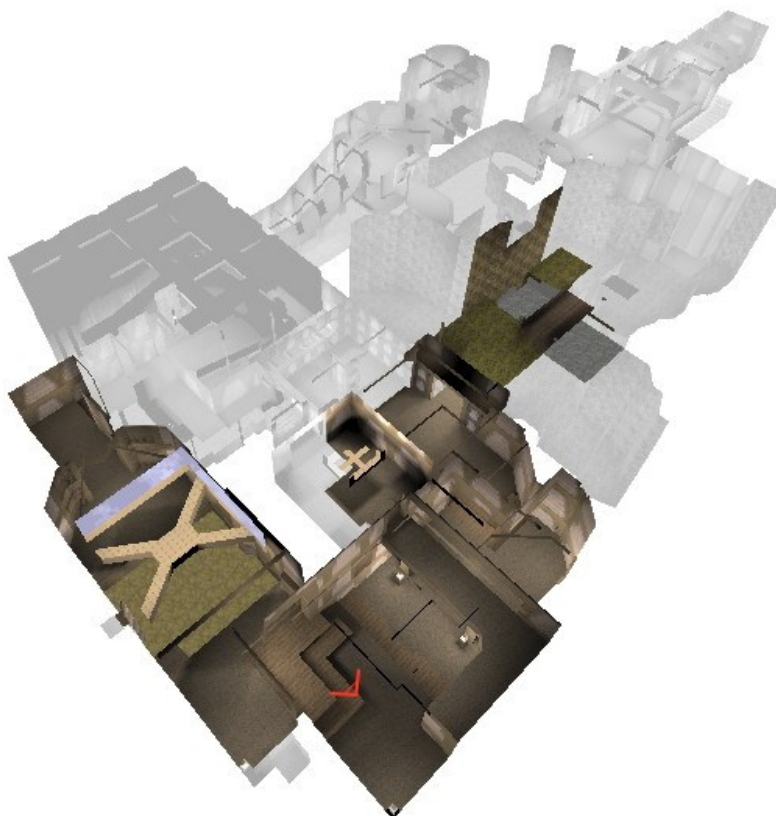
Sl. 1.1 Grafički protočni sustav (engl. Rendering pipeline) [5]

1.1.2. Problem vidljivosti objekata

Za prikaz složenih i realističnih scena potrebno je iscrtati mnogo objekata. Obrada svakog objekta povećava vrijeme iscrtavanja. Cilj je smanjiti vrijeme iscrtavanja scene što je više moguće kako bi se omogućio prikaz u stvarnom vremenu. Zbog toga je nepraktično slati svaki objekt na iscrtavanje. U idealnom svijetu će se poslati samo oni objekti koji su vidljivi, ali algoritmi za odabir takve skupine objekata znaju biti skupi.

Poseban tip scene gdje se može puno vremenskih resursa uštedjeti je tip zatvorenih scena. To je niz međusobno povezanih prostorija i hodnika. U takvim scenama većina objekata nije vidljiva zbog zidova. Zbog toga nije potrebno iscrtati cijelu scenu, već samo prostorije koje mogu biti vidljive iz položaja kamere kao što je prikazano na slici. (Sl. 1.2).

Problem vidljivosti u slučaju zatvorenih scena je moguće učinkovito riješiti korištenjem optimizacijskih struktura podataka. U ovom radu se koriste binarna podjela prostora (engl. Binary Space Partitioning, BSP) i potencijalno vidljivi skupovi (engl. Potentially Visible Sets, PVS).



Sl. 1.2 Primjer zatvorene scene gdje su obojene prostorije koje se mogu vidjeti [4]

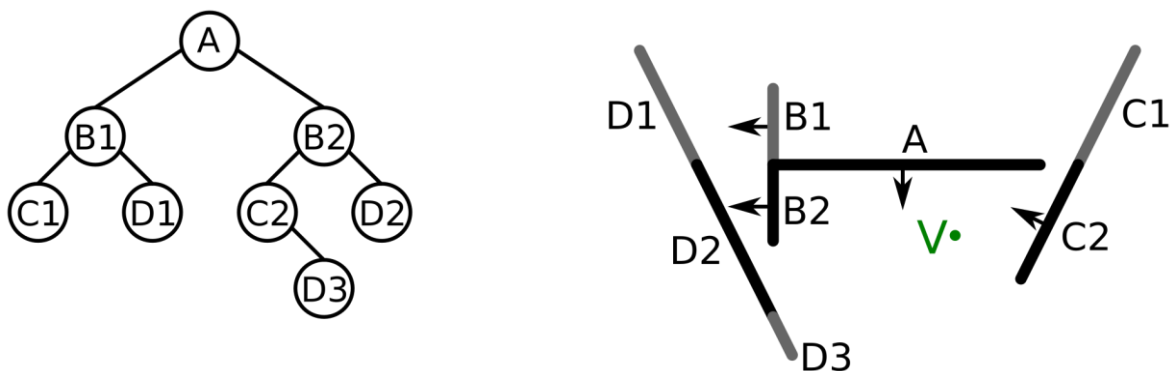
1.2. Binarna podjela prostora (BSP)

1.2.1. Definicija BSP stabla

Binarna podjela prostora predstavlja strukturu podataka za rekurzivnu podjelu prostora. Prostor se rekurzivno dijeli na dva prostora, stražnji prostor i prednji prostor. Prostor se uobičajeno dijeli ravninom u tom prostoru.

Prirodan način zapisivanja binarne podjele prostora je binarno stablo. Stablo je sačinjeno od čvorova i listova. Čvorovi sadrže particijsku ravninu koja dijeli prostor. Listovi predstavljaju konačne dijelove podijeljenog prostora. Rekurzivnom podjelom prostora dobiva se hijerarhijska organizacija scene koja je pogodna za razne optimizacije u računalnoj grafici.

Primjer BSP podjele i pripadnog BSP stabla je prikazano na slici (Sl. 1.3).



Sl. 1.3 Primjer prostora s pripadnim BSP stablom [6]

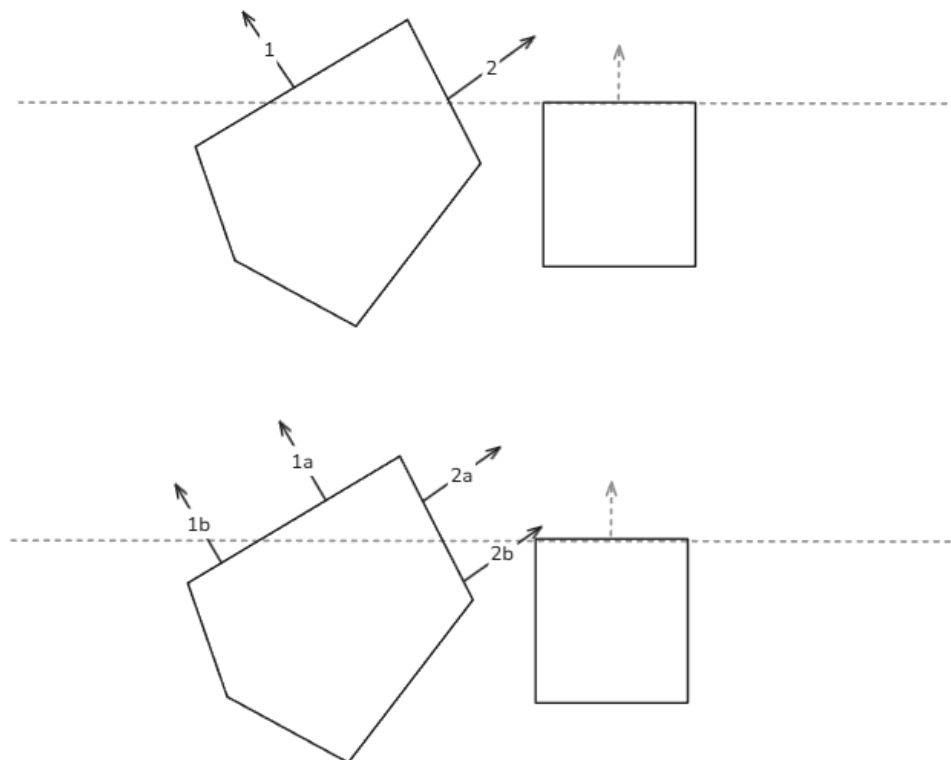
1.2.2. Konstrukcija BSP stabla

Konstrukcija BSP stabla započinje odabirom particijske ravnine koja dijeli prostor na prednji i stražnji poluprostor. Prednji poluprostor se nalazi u smjeru normale ravnine, dok se stražnji poluprostor nalazi u suprotnom smjeru.

Prilikom odabira particijske ravnine važno je smanjiti broj presijecanja poligona scene. Ako particijska ravnina presijeca velik broj poligona, potrebno je stvarati nove poligone, što povećava složenost stabla i memorijske zahtjeve algoritma.

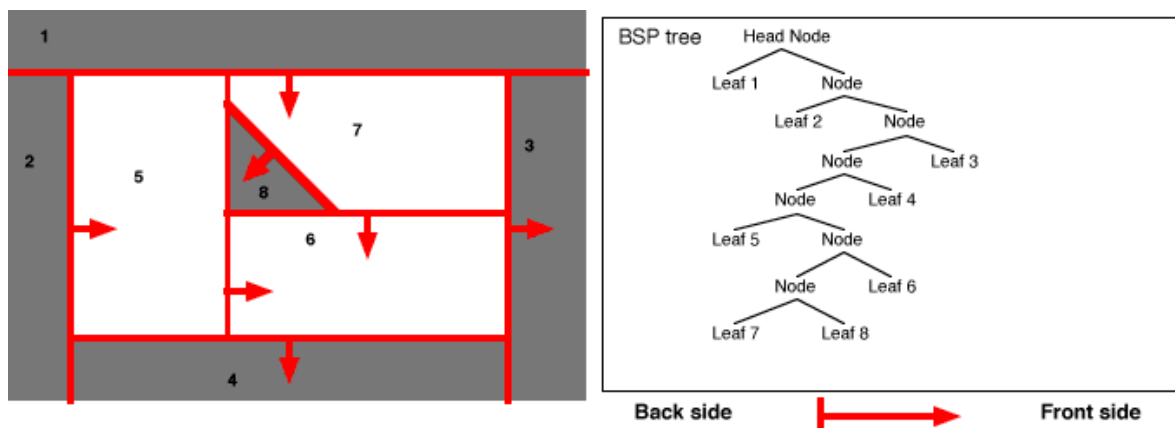
Nakon odabira particijske ravnine svaki poligon klasificira se prema svom položaju u odnosu na ravninu. Poligon može pripadati prednjem ili stražnjem poluprostoru, a može i presijecati particijsku ravninu. U tom slučaju, kada presijeca ravninu potrebno je poligon podijeliti na dva nova poligona.

Podjela poligona provodi se određivanjem sjecišta bridova poligona s particijskom ravinom. Na temelju dobivenih sjecišta grade se novi poligoni koji pripadaju odgovarajućim poluprostorima. Primjer podjele poligona prikazan je na slici (Sl. 1.4).



Sl. 1.4 Prije i poslije podjele jednog poligona ravinom

Nakon što se prostor podijeli na dva dijela, postupak se rekurzivno ponavlja za prednji i stražnji poluprostor. Konstrukcija BSP stabla završava kada više nema poligona koje je potrebno podijeliti ili kada se dostigne unaprijed zadana maksimalna dubina stabla. Primjer rekurzivne podjele prostora se može vidjeti na slici (Sl. 1.5).



Sl. 1.5 Primjer rekurzivne podjele prostora i pripadnog BSP stabla [7]

1.2.3. Obilazak BSP stabla

Nakon što je izgrađeno BSP stablo, ono se može obilaziti na dva načina. Ta dva načina su front-to-back i back-to-front.

Kod back-to-front obilaska prvo se obrađuju dijelovi scene koji su udaljeniji od kamere, a zatim dijelovi koji su bliži. Redoslijed obilaska određuje se prema položaju kamere u odnosu na particijsku ravninu svakog čvora. Takav način obilaska omogućuje ispravno iscrtavanje scene bez korištenje Z-spremnik te se može koristiti za implementaciju slikarovog algoritma (*engl. Painter's algorithm*).

Kod front-to-back obilaska prvo se obilaze dijelovi scene koji su bliže kameri. Ovakav pristup je posebno koristan uz korištenje Z-spremnik jer omogućuje rano odbacivanje fragmenata koji se nalaze iza već iscrtanih objekata. Time se smanjuje broj fragmenata koje je potrebno obraditi te se može povećati učinkovitost iscrtavanja.

Za oba načina obilaska redoslijed obilaska podstabla ovisi o položaju kamere u odnosu na particijsku ravninu trenutnog čvora stabla. Ako je kamera ispred ravnine onda prvo obilazimo prednje podstablo, a zatim stražnje podstablo. Ako je kamera iza ravnine onda idemo obrnutim redoslijedom.

1.2.4. Prednosti i nedostaci BSP-a

Velika prednost kod BSP stabla je što je scena podijeljena i sortirana. Sortiranost nam omogućuje vrlo lagan obilazak stabla i iscrtavanje scene. BSP stablo se može iskoristiti za detekciju sudara. Kod scene bez optimizacije potrebno je ispitati svaki objekt scene, dok BSP stablo pruža učinkovitiji način. Za BSP stablo potrebno je ispitati maksimalno čvorova onoliko kolika je maksimalna dubina stabla. Maksimalna dubina stabla će biti značajno manja od broja objekata. Zbog toga je potrebno izgraditi stablo koje ima najmanju dubinu, a to stablo se naziva balansirano stablo, gdje je broj čvorova jednak između prednjeg i stražnjeg čvora.

Jedan nedostatak je taj što je izgradnja BSP stabla računalno neučinkovita. Nije moguće ostvariti stvaranje BSP stabla u realnom vremenu. Posljedica toga je što moramo unaprijed izgraditi BSP stablo prije korištenja. Konstrukcija BSP stabla prije korištenja sprječava nam korištenje dinamičkih scena koje sadrže objekte koji se pomjeraju u sceni. Za takve objekte potrebno je konstantno računati poziciju u BSP stablu prije svakog iscrtavanja.

Drugi nedostatak je što se pri konstrukciji poligoni dijele. Dijeljenjem poligona uzastopno može nastati vrlo visok broj poligona ako je scena složena. Ovo može dodatno usporiti iscrtavanje scene.

Tablica 1.1 Prednosti i nedostaci BSP stabla

Prednosti	Nedostaci
Učinkovit obilazak	Skupa konstrukcija
Optimizacija vidljivosti	Dijeljenje poligona
Detekcija sudara	Loša za dinamičke scene
Dobro za indoor scene	Manje korisno za outdoor scene

1.3. Potencijalno vidljivi skupovi (PVS)

1.3.1. Koncept potencijalno vidljivih skupova

BSP sam po sebi organizira prostor, ali za učinkovito određivanje vidljivosti zatvorenih prostorija potrebno je dodatno opisati njihovu međusobnu povezanost. Tu ulogu preuzimaju portali, oni bilježe koji su listovi stabla susjedni i time omogućuju da se unaprijed, izvan vremena izvođenja, izračuna skup listova koji mogu biti potencijalno vidljivi iz svakog pojedinog pista. Taj skup se naziva potencijalno vidljivi skup (engl. *Potentially Visible Set, PVS*).

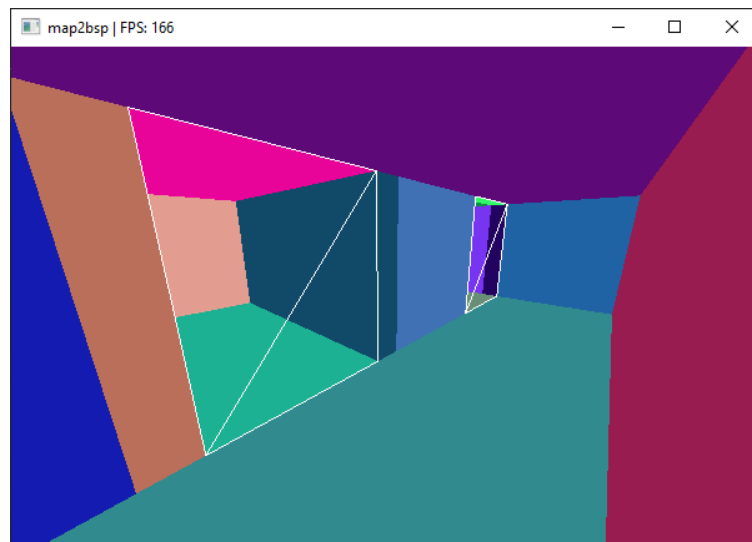
PVS za svaki list određuje skup drugih listova koji mogu potencijalno biti vidljivi iz tog lista. Bitno je naglasiti smjer skupa vidljivosti: ako je list uvršten u PVS neke prostorije, to ne znači da će on stvarno biti vidljiv iz svake točke te prostorije, nego suprotno, da niti jedan list koji nije u PVS-u ne može biti vidljiv. PVS je nadskup svih stvarno vidljivih listova, a njegova je svrha eliminacija listova koji sigurno nisu vidljivi prije iscrtavanja.

1.3.2. Portali i povezanost prostora

Glavna struktura podataka kod izračuna PVS-a su portali. Portal je poligon koji leži na granici između dva susjedna lista BSP stabla i predstavlja prijelaz iz jednog u drugi. Nakon generiranja svih portala pri izgradnji stabla, potrebno je zadržati samo one koji spajaju dva prazna lista. Portali koji prolaze uz krutu tj. sve što nije prazno nisu relevantni jer kroz njih nije moguće gledati. S pomoću takvih portala možemo izračunati vidljivost unutrašnjeg prostora.

Svaki portal uz svoj poligon, pamti još dva podatka: list iz kojeg izlazi i list u koji se ulazi. Kako bi izračun vidljivosti mogao ispravno propagirati kroz lanac više portala, ti se podaci moraju odrediti usmjerenom, tj. za svaki list će se zasebno generirati varijanta portala koje taj list predstavlja izlazni, čime se dobiva jedan smjer gledanja kroz portal.

Primjer portala koji povezuju hodnik i sobe se može vidjeti na slici ispod (Sl. 1.6). Na ovom primjeru hodnik posjeduje dva portala koji izlaze iz hodnika te ulaze u pojedinu sobu. Sobe imaju iste portale u smislu geometrije, ali im je smjer gledanja suprotan, tj. izlaze iz soba te ulaze u hodnik.



Sl. 1.6 Primjer portala koji povezuju prostorije i hodnik

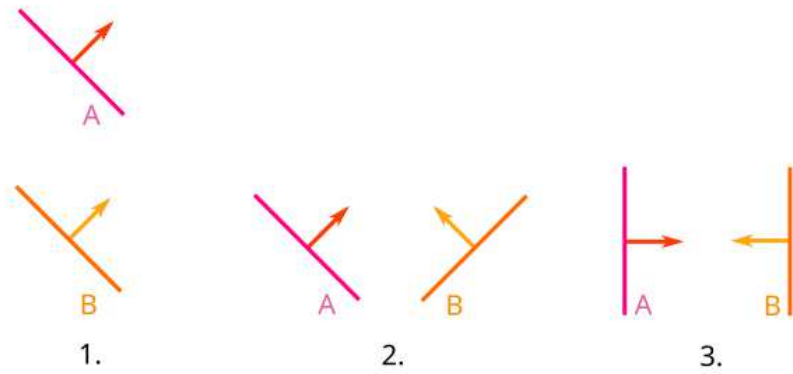
1.3.3. Generiranje PVS podataka

Poput BSP stabla, izračun vidljivosti se odvija unaprijed (offline). Izračun PVS-a se događa nakon izgradnje BSP stabla i nakon izgradnje pripadajućih portala. Izrada PVS-a se odvija u dvije faze: grubu (aproksimativnu) fazu te finu (točniju) fazu vidljivosti.

Gruba faza vidljivosti uzima jednostavne i brze provjere te procjenjuje može li se kroz jedan portal vidjeti drugi portal. Provjera je zadovoljena samo ako vrijede sve tri sljedeće tvrdnje:

1. Poligon drugog portala mora imati barem jednu točku ispred ravnine portala kojeg testiramo.
2. Poligon portala kojeg testiramo mora imati barem jednu točku iza ravnine drugog portala.
3. Ravnine dvaju portala ne smiju biti gotovo suprotno usmjerene, tj. portali ne smiju „gledati“ jedan u drugi.

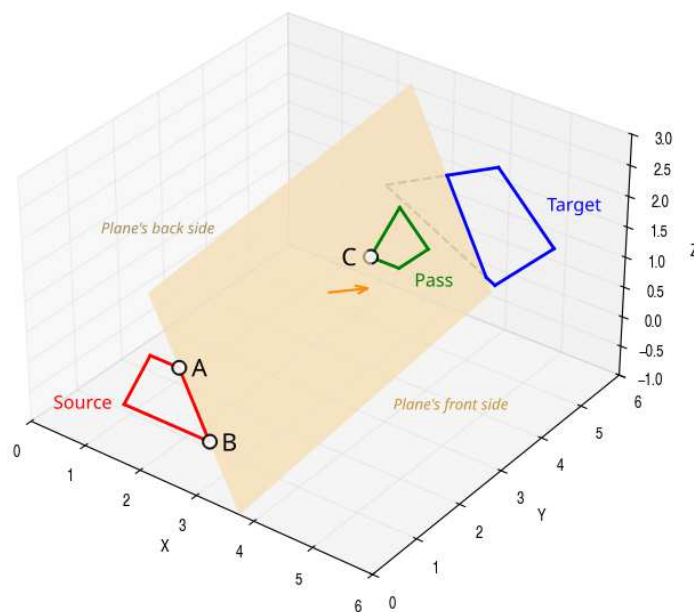
Na slici (Sl. 1.7) su prikazani odgovarajući testovi koji ne prolaze provjeru vidljivosti za svake tvrdnje.



Sl. 1.7 Provjere unutar grube faze vidljivosti [4]

Provjera se rekurzivno propagira dalje kroz susjedne portale sve dok navedeni uvjeti vrijede. Time se pravi za svaki portal gornju granicu tj. nadskup potencijalno vidljivih listova.

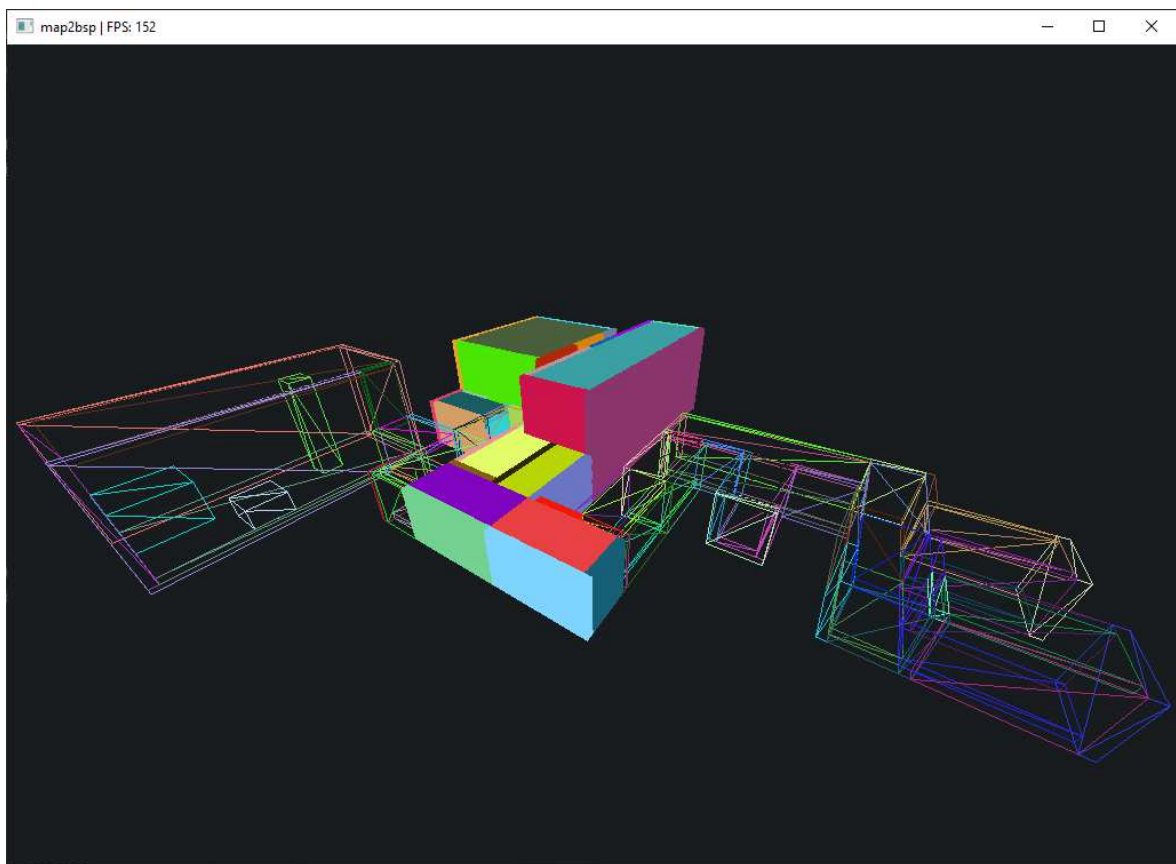
Nakon grube faze, PVS je funkcionalno gotov. Dobiveni skup je korektan nadskup stvarno vidljivih listova, no taj nadskup se može smanjiti bliže stvarnom skupu vidljivosti. Fina faza će zatim, dodatnim i geometrijski točnijim testovima (odrubljivanjem poligona uz separacijske ravnine) ukloniti listove koje je gruba faza pogrešno označila kao potencijalno vidljive (Sl. 1.8). Time se PVS dodatno sužava i približava stvarnom skupu vidljivih listova.



Sl. 1.8 Odrubljivanje poligona portala s pomoću separacijske ravnine [4]

1.3.4. Korištenje PVS-a tijekom izvođenja

Provjera vidljivosti tijekom izvođenja uopće više ne koristi portale, nego svaki list BSP stabla unaprijed posjeduje gotov popis vidljivih listova. Taj popis je izračunat u prethodnoj fazi. Prvi korak je odrediti u kojem se listu stabla trenutačno kamera nalazi. Ovaj izračun je trivijalan zbog same strukture BSP stabla. Nakon toga dovoljno je iscrtati geometriju tog lista te geometriju svakog lista iz njegova popisa vidljivih listova. Primjer vidljivosti jedne scene se može vidjeti na slici ispod (Sl. 1.9).



Sl. 1.9 Primjer scene gdje je vidljivi dio scene popunjen, dok dio koji nije vidljiv je u žičnoj formi

2. Implementacija aplikacije

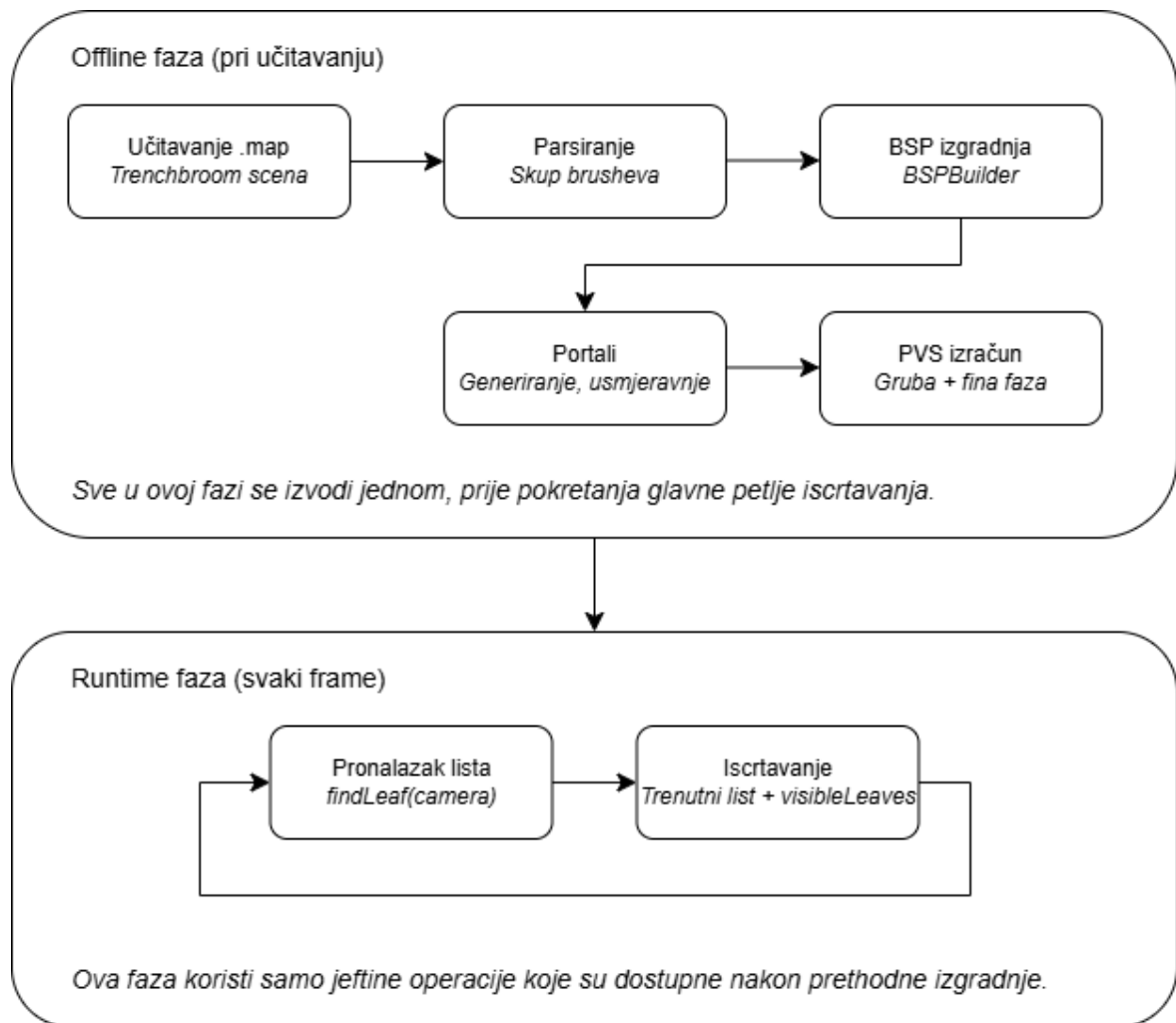
2.1. Korištene tehnologije

Aplikacija je izrađena u programskom jeziku C++.

Grafičko sučelje koje se koristi je OpenGL. Također uz OpenGL koristi se biblioteka GLFW koja je biblioteka za upravljanje prozorima te korisničkim ulazima u aplikaciju. Biblioteka za razne matematičke operacije koja se koristi je GLM (OpenGL Mathematics). Build sustav je CMake koji koristi MSVC kao kompajler.

Izrada scene se događa u alatu TrenchBroom[1] koju aplikacija učitava, parsira i optimizira. Scene su u formatu *.map* koje koriste razne igre (Quake/Half-Life).

2.2. Arhitektura aplikacije



Slika 2.1 Dijagram arhitekture aplikacije

Aplikacija učitava *.map* datoteku te je parsira. Parser iz datoteke uzima podatke koje su potrebne za izgradnju scene. Nakon toga aplikacija izgrađuje BSP stablo iz geometrije scene te izgradnja portala koji povezuju geometriju scene. Kada se izgradi BSP stablo i portali, onda se vrši PVS izračun. Ovaj proces se izvrši samo jednom prije iscrtavanja, dok se iscrtavanje događa mnogo puta sve dok korisnik ne izađe iz aplikacije. Iscrtavanje je omogućeno posebnim rendererom koji obilazi stablo za pronalazak kamere u sceni, te iscrtavanje vidljivih objekata.

2.3. Implementacija BSP strukture

2.3.1. Struktura BSP čvora

Struktura podataka koja se koristi za binarnu podjelu prostora je binarno stablo. Stablo se sastoji od čvorova, pri čemu neki čvorovi predstavljaju listove stabla.

U implementaciji su čvorovi i listovi predstavljeni istom strukturom `BSPNode`. Status lista je određen po vrijednosti varijable `isLeaf`, koja označava je li pojedini čvor list. Varijable `isLeaf` i `isSolid` koriste se prvenstveno u implementaciji PVS sustava te nisu nužne za samu konstrukciju i obilazak BSP stabla.

Svaki čvor BSP stabla sadrže sljedeće podatke:

- Particijsku ravninu zapisanu u implicitnom obliku
- Kolekciju poligona (*brushvi*)
- Pokazivač na prednje dijete
- Pokazivač na stražnje dijete

Particijska ravnina spremljena je pomoću tipa `glm::vec4`, pri čemu komponente predstavljaju koeficijente implicitne jednadžbe ravnine oblika prema izrazu:

$$Ax + By + Cz + D = 0$$

Za upravljanje ravninom korišten je `std::shared_ptr`, čime je omogućeno automatsko oslobađanje memorije bez potrebe za ručnim upravljanjem pokazivačima.

```
struct BSPNode {
    Plane plane;
    std::vector<Brush> brushes;
    bool isLeaf = false;
    bool isSolid = false;
    Contents contents = Contents::SOLID;

    std::shared_ptr<BSPNode> parent;
    std::shared_ptr<BSPNode> backChild;
    std::shared_ptr<BSPNode> frontChild;

    std::vector<Portal *> portals; // EMPTY - EMPTY
    std::vector<BSPNode *> solidNeighbors; // EMPTY -> SOLID
```

```

int leafIndex = -1;
std::vector<BSPNode *> visibleLeaves;

glm::vec3 color{ 1.0 };
};

```

Kôd 2.1 Struktura BSP čvora

2.3.2. Algoritam izgradnje stabla

Izgradnja BSP stabla provodi se rekurzivnim algoritmom podjele prostora. U svakom koraku odabire se particijska ravnina koja dijeli prostor na prednji i stražnji poluprostor. Odabir particijske ravnine značajno utječe na kvalitetu i balansiranost BSP stabla pa je cilj pronaći ravninu koja omogućuje što ravnomjerniju raspodjelu poligona uz minimalan broj podjela poligona.

Algoritam konstrukcije sastoji se od dva dijela:

- Odabira particijske ravnine
- Klasifikacije i podjele poligona (*brusheva*)

Za odabir particijske ravnine koristi se heuristika koja procjenjuje kvalitetu kandidata za particijsku ravninu. Heuristika uzima u obzir razliku između broja poligona u prednjem i stražnjem poluprostoru te broj poligona koje bi trebalo podijeliti.

```

int Splitter::selectSplitPlane(const std::vector<Brush>
&brushes, std::vector<Plane> &planes) {
    ...
    for (int planenum : candidatePlanes) {
        int front = 0, back = 0, splits = 0;
        for (const auto &brush : brushes) {
            for (const auto &face : brush.sides) {
                PlaneSide side =
classifyPolygon(face.winding, planes[planenum]);
                if (side == PlaneSide::Back)
                    back++;
                if (side == PlaneSide::Front)
                    front++;
                if (side == PlaneSide::Straddling)
                    splits++;
            }
        }
    }
}

```

```

    }
    if (back == 0 && front == 0)
        continue;
    int currentScore = 5 * (front + back) - 5 * splits -
std::abs(back - front);
    if (currentScore > bestScore) {
        bestScore = currentScore;
        bestPlane = planenum;
    }
}
return bestPlane;
}

```

Kôd 2.2 Isječak funkcije heuristike za odabir particijske ravnine

Izračunom heuristike za pojedinu ravninu dobije se broj koji označava koliko je dobra ravnina za podjelu prostora. U ovom slučaju heuristika je napravljena da veći broj označava bolji odabir ravnine. To je postignuto tako da se izračuna apsolutna razlika poligona koji su ispred i onih koji su iza ravnine. Razlikom se dobije broj koji označava balansiranost stabla, gdje se dodatno kažnjavaju ravnine koje dijele dodatne poligone. Kazna za podjelu poligona je veća od nebalansiranosti stabla zato što se svakom podjelom poligona dodatno komplicira scena i može znatno utjecat na dubinu stabla. Množenje broja podjela poligona s četiri se pokazalo kao dobar kompromis između balansa i fragmentacije stabla.

Nakon odabira particijske ravnine, potrebno je prostor podijeliti odabranom ravninom. Podjelom prostora će se dobiti poligoni koji su u prednjem prostoru, poligoni koji su u stražnjem prostoru i poligoni koji prolaze kroz tu ravninu. Poligone koji prolaze kroz particijsku ravninu potrebno je podijeliti i razvrstati u prednji prostor ili stražnji prostor.

```

std::vector<Brush> back, front;

for (auto &brush : brushes) {
    auto [backBrush, frontBrush] = Splitter::split(brush,
splitPlaneNum, planes);

    if (!backBrush.sides.empty())
        back.push_back(backBrush);
}

```

```

        if (!frontBrush.sides.empty())
            front.push_back(frontBrush);
    }

```

Kôd 2.3 Podjela poligona particijskom ravninom

Razvrstavanjem poligona koji pripadaju prednjem prostoru i stražnjem prostoru jednostavno se rekurzivno prosljeđuju ti poligoni prednjem djetetu i stražnjem djetetu. Algoritam prestaje kad čvor nema poligona za odabir particijske ravnine i tad se taj čvor zapisuje kao list stabla.

2.3.3. Obilazak i iscrtavanje

Obilazak BSP stabla provodi se ovisno o položaju kamere u odnosu na particijsku ravninu trenutnog čvora. Na temelju položaja kamere određuje se redoslijed obilaska podstabala.

Kod front-to-back obilaska prvo se obrađuje podstablo bliže kameri, dok se kod back-to-front obilaska prvo obrađuje udaljenije stablo.

U implementaciji se koristi front-to-back obilazak kako bi se omogućilo učinkovitije korištenje Z-spremnik i rano odbacivanje fragmenata koji nisu vidljivi.

```

void BSPRenderable::renderNode(BSPNode* node) {
    if (!node) return;

    vec4 cameraPos = vec4(camera->getPosition(), 1.0f);
    float side = dot(node->plane, cameraPos);

    if (side > 0) {
        renderNode(node->frontChild.get());
        drawNode(node);
        renderNode(node->backChild.get());
    } else {
        renderNode(node->backChild.get());
        drawNode(node);
        renderNode(node->frontChild.get());
    }
}

```

Kôd 2.4 Rekurzivni obilazak BSP stabla u ovisnosti o položaju kamere na particijsku ravninu

2.4. Implementacija PVS sustava

2.4.1. Struktura portala i njihova usmjerenost

Za korištenje portala potrebno je imati sljedeće podatke: poligon portala te pokazivači na susjedne listove. Ostali podaci koji su u strukturi `Portal` se koriste kako bi se olakšala implementacija vidljivosti. Geometrija portala, tj. poligon se naziva *winding*. Korisno je imati spremljeno u samu strukturu portala ravninu na kojoj je taj portal kako se ne bi svaki put trebao raditi izračun ravnine iz poligona.

Kod izračuna grube faze potrebno je imati skup pokazivača na vidljive listove. Za izračun fine faze korisno je imati poseban skup potencijalno vidljivih listova.

```
struct Portal {
    std::vector<glm::vec3> winding;
    Plane plane;
    BSPNode *front;
    BSPNode *back;
    std::vector<BSPNode*> visibleLeaves; // gruba faza
    std::vector<bool> mightSee;
    std::vector<bool> vis; // rekurzivni tok vidljivosti
    bool done = false;
};
```

Kôd 2.5 Struktura podataka portala

Kako bi se popunila struktura podataka portala (Kôd 2.5) moraju se pretvoriti već generirani dvosmjerni portali u par jednosmjernih portala. Usmjerenost portala omogućava lakši izračun propagacije portala u fazama izračuna vidljivosti. Proces pretvaranja u usmjerene portale je jednostavan te se radi na sljedeći način: iteracijom kroz svaki portal se napravi dodatni portal koji je u suprotnom smjeru i taj se portal pridruži listu koji povezuje prvobitni portal (Kôd 2.6).

```

std::vector<Portal> newPortals;
for (Portal& portal : tree.portals) {
    if (!portal.front || !portal.back)
        continue;

    Portal reverse = portal;
    reverse.plane.n = -portal.plane.n;
    reverse.plane.d = -portal.plane.d;
    std::reverse(reverse.winding.begin(),
reverse.winding.end());
    std::swap(reverse.front, reverse.back);
    newPortals.push_back(std::move(reverse));
}

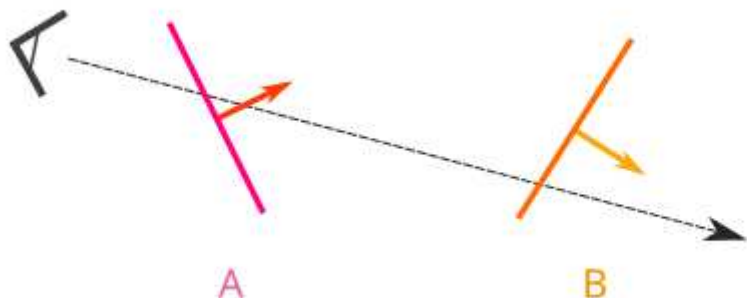
```

Kôd 2.6 Generiranje usmjerenih portala

2.4.2. Gruba faza vidljivosti

Sada kada je konstruiran usmjereni graf listova povezanim portalima, može se započeti izračun vidljivosti. Prvo je potrebno izračunati koje sve listove svaki portal može vidjeti i tek onda na kraju napraviti vidljivi skup za svaki list. Grubom fazom se izračunaju listovi koji su teoretski vidljivi kroz bilo koji portal.

Portal A može „vidjeti“ list ako postoji linija vida kroz portal B koja vodi u taj list (Sl. 2.1).



Sl. 2.1 Linija vida koja prolazi kroz portal A i kroz portal B [4]

Portal A može imati liniju vida kroz portal B ako su zadovoljeni sljedeći uvjeti:

- Poligon portala B ima barem jednu točku ispred ravnine portala A,
- Poligon portala A ima barem jednu točku iza ravnine portala B,
- Portal A i portal B ne gledaju jedan na drugi, tj. njihova ravnina nije obrnutog smjera.

Ove tri provjere su trivijalne te se mogu lagano i efikasno testirati koristeći ravnine portala i točke poligona portala.

Nakon što su definirane ove tri provjere može se početi s rekurzivnom propagacijom grube faze vidljivosti. Koristi se algoritam traženja u dubinu kroz listove svakog portala. Započinje se destinacijskim listom prvog portala i prolazeći kroz druge portale, ali samo ako su oni vidljivi kroz početni portal. Rezultat je skup listova za svaki portal koji mogu biti teoretski vidljivi (Kôd 2.7).

```
vector<BSPNode*> basePortalFlow(const vector<Portal>&
portals, const Portal& portal) {
    vector<BSPNode*> potentiallyVisible;
    simpleFlood(potentiallyVisible, portal, portal.front);
    return potentiallyVisible;
}

void simpleFlood(vector<BSPNode*>& potentiallyVisible, const
Portal& portal, BSPNode* leaf) {
    if (find(potentiallyVisible.begin(),
potentiallyVisible.end(), leaf) != potentiallyVisible.end())
        return;

    // portal sees leaf
    potentiallyVisible.push_back(leaf);
    // tests for visibility
    for (const auto& otherPortal : leaf->portals) {
        if (portalMightSeePortal(portal, *otherPortal)) {
            simpleFlood(potentiallyVisible, portal,
otherPortal->front);
        }
    }
};
```

Kôd 2.7 Tok izračuna vidljivosti kod grube faze vidljivosti

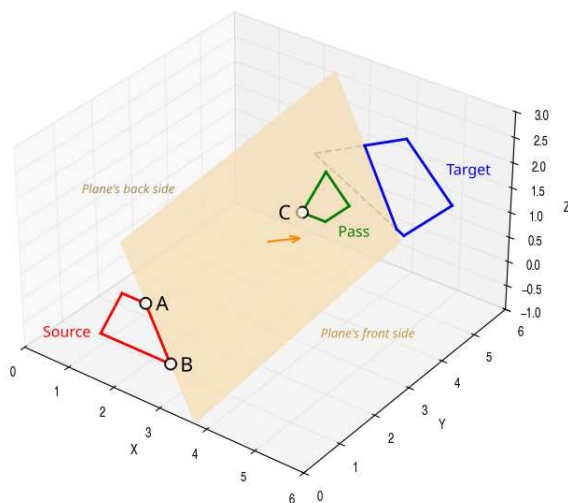
2.4.3. Fina faza vidljivosti

Gruba faza vidljivosti je izračunala listove koji se možda mogu vidjeti iz trenutnog portala. Izračun točne vidljivosti je vrlo težak problem, ali se može uzeti konzervativnija aproksimacija koja je relativno blizu točne vidljivosti portala. U ovoj finoj fazi se nikada neće sakriti list koji treba biti vidljiv.

Fundamentalna ideja je konstruirati volumen koji sadrži sve linije vida koje prolaze kroz slijed portala. Ako je neki portal van konstruiranog volumena onda se on ne može vidjeti kroz portale u trenutnom slijedu.

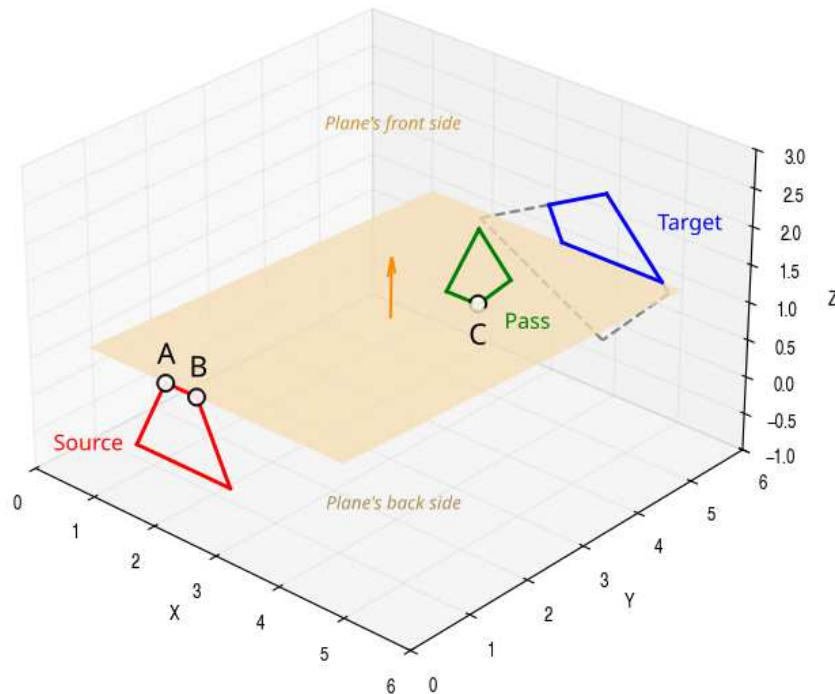
Način koji je implementiran je pomoću odsijecanja poligona ciljnog portala. Odsijecanje se događa ako je portal djelomično vidljiv. Zbog jednostavnosti uzima se slijed od tri povezana portala: izvorišni portal, prijelazni portal te ciljni portal kojeg se pokušava odrediti je li nevidljiv. Činjenica je da se uvijek može vidjeti prolazni portal zato što su listovi geometrijski konveksni, pa će uvijek biti linija vida koja prolazi kroz izvorišni portal u prolazni portal.

Odsijecanje ciljnog portala odvija se pomoću separacijskih ravnina. Separacijske ravnine se konstruiraju od kombinacije točaka izvorišnog te prijelaznog portala. Dvije točke su iz susjednih vrhova izvorišnog portala, a zadnja je jedna točka prijelaznog portala. Separacijska ravnina se uzima samo u slučaju ako su izvorišni portal i prijelazni portal na suprotnoj strani prijelazne ravnine, te ako je barem jedan vrh prijelaznog portala ispred separacijske ravnine. Primjer ispravno konstruirane separacijske ravnine, te odsijecanja ciljnog portala se može vidjeti na slici ispod (Sl. 2.2).



Sl. 2.2 Primjer odsijecanja portala pomoću separacijske ravnine [4]

Uzima se svaka ispravna separacijska ravnina za odsijecanje ciljnog portala, što znači da je potrebno provjeriti svaku kombinaciju ruba izvorišnog portala i svaki vrh prijelaznog portala. Slika ispod pokazuje drugo odsijecanje za iste portale (Sl. 2.3).



Sl. 2.3 Primjer odsijecanja portala pomoću separacijske ravnine [4]

2.4.4. Rekurzivni tok vidljivosti

Nakon što je gruba faza vidljivosti odredila za svaki portal konzervativan skup potencijalno vidljivih listova `mightSee`, potrebno je taj skup smanjiti geometrijskim točnijim postupkom fine faze vidljivosti. Taj postupak implementiran je kroz funkcije `portalFlow` i `leafFlow`.

Funkcija `portalFlow` predstavlja izračun vidljivosti jednog portala. Ona pokreće rekurzivni obilazak koji počinje od lista na koji portal gleda (`portal.front`), s početnim skupom mogućih vidljivih listova (`portal.mightSee`). Ovaj se postupak izvršava za svaki portal u sceni (Kôd 2.8).

```

void BSPBuilder::portalFlow(Portal& portal, int testLevel) {
    leafFlow(portal.front, portal.mightSee, portal.winding,
portal.plane, nullptr, portal, testLevel);
    portal.done = true;
}

```

Kôd 2.8 Tok portala

Nakon izračuna, portal je označen kao obrađen, što omogućuje drugim portalima koji će računati vidljivost kroz taj portal da koriste novu, precizniju vidljivost umjesto grube procjene.

Rekurzija se odvija u funkciji `leafFlow`, koja za trenutni list prvo označava kao vidljiv iz izvornog portala. Zatim se prolazi kroz sve portale koje trenutni list gleda prema susjednim listovima. Tim portalima se prvo provjerava trivijalno odbacivanje, tj. je li ciljni list uopće u skupu mogućih vidljivih listova (`mightSee`). Zatim se skup mogućih vidljivih listova smanjuje presjekom s odgovarajućim skupom ciljnog portala, `vis` skupom ako je obrađen ili ako nije grubom procjenom `mightSee`. Ako taj presjek ne donosi niti jedan novi list koji izvorni portal još ne vidi, onda se grananje prekida jer nema potrebe za daljnjim provjerama.

Ako provjera prođe, pokrenut će se odsijecanje. Poligon ciljnog portala se prvo odsiječe prema ravnini izvornog portala, čime se zadržava dio kojeg je moguće vidjeti iz izvorne pozicije (Kôd 2.9).

```

auto targetPoly = Splitter::clipFront(pt->winding, source.plane, true);

```

Kôd 2.9 Odsijecanje ciljnog portala prema ravnini izvornog portala

Za neposredno susjedne listove (slučaj kada je prvi korak rekurzije) ovime se izračun završava i rekurzija nastavlja odmah u susjedni list. Daljnji koraci rekurzije se obrađuju dodatnim odsijecanjem. Odsijecanje se vrši prijelaznom ravninom iz prethodnog koraka, a izvorni poligon se reže sa stražnjom stranom ravnine trenutnog portala, čime se osigurava da se uzme dio prostora koji je stvarno vidljiv kroz cijeli lanac portala.

Konačna i najpreciznija provjera se odvija kroz četiri stupnja odsijecanja separacijskim ravninama. Svaki stupanj promatra odnos izvora, prijelaznog i ciljnog poligona iz različitih kutova. Svaki stupanj može suziti poligone koji se prenose dalje u rekurziju (Tablica 2.1). Viša razina testiranja daje precizniji skup vidljivih listova, ali uz veću računsku cijenu, pa broj maksimalnih stupnjeva predstavlja kompromis između točnosti i brzine izgradnje PVS podataka.

Tablica 2.1 Razine odsijecanja separacijskim ravninama [4]

Stupanj	Prvi poligon	Drugi poligon	Odsijecani poligon
1.	Izvorišni	Prijelazni	Ciljni
2.	Prijelazni	Izvorišni	Ciljni
3.	Ciljni	Prijelazni	Izvorišni
4.	Prijelazni	Ciljni	Izvorišni

Ako nijedno odsijecanje nije u potpunosti uklonilo ciljni ili izvorni poligon, rekurzija će se nastaviti u susjedni list, prenoseći manji skup mogućih vidljivih listova. Izrezani poligoni će biti novi izvorni i prijelazni par za sljedeći korak.

2.4.5. Korištenje PVS-a pri iscrtavanju

Iz PVS-a se dobiju samo prazni vidljivi listovi, jer se samo kroz njih mogu prostirati linije vida. Za prikaz solidne geometrije potrebno je u te prazne listove dodati informacije o geometriji. U ovoj implementaciji uzeta je jednostavnija varijanta koja obavlja tu ulogu. Struktura pojedinog lista će se proširiti sa skupom njegovih susjednih solidnih listova. Ovo je učinjeno prije samog PVS procesa zato što u izračunu vidljivosti ne trebaju portali koji povezuju prazne i solidne listove. Iteracijom kroz sve portale, te ako postoji portal koji povezuje prazan i solidan list, u skup punih listova tog praznog lista će se dodati puni list (Kôd 2.10).

```

for (Portal& portal : tree.portals) {
    BSPNode* a = portal.front;
    BSPNode* b = portal.back;
    if (!a || !b) continue;
    if (a->contents == Contents::EMPTY &&
        b->contents == Contents::SOLID) {
        a->solidNeighbors.push_back(b);
    }
    else if (a->contents == Contents::SOLID &&
        b->contents == Contents::EMPTY) {
        b->solidNeighbors.push_back(a);
    }
}

```

Kôd 2.10 Umetanje listovi koji su puni u skup pojedinog praznog lista

Iscrtavanje scene s pomoću PVS-a se odvija nešto drugačije od običnog BSP iscrtavanja. Nije potrebno obilaziti cijelo stablo nego je potrebno samo pronaći u kojem listu se nalazi pozicija kamere. Zbog izračuna PVS-a listovi sadrže skup potencijalno vidljivih listova, a iscrtavanje je trivijalno prolaženje kroz taj skup (Kôd 2.11).

```

void BSPRenderable::draw() {
    glBindVertexArray(VAO);
    currentFrame++;

    BSPNode* cameraLeaf = findLeaf(tree.rootNode.get(),
    camera->getPosition());
    if (!cameraLeaf) return;
    renderNode(cameraLeaf);
    for (const auto visible : cameraLeaf->visibleLeaves) {
        for (const auto solid : visible->solidNeighbors) {
            drawNode(solid);
        }
    }
    glBindVertexArray(0);
}

```

Kôd 2.11 Iscrtavanje scene s pomoću strukture PVS-a

3. Testiranje i analiza rezultata

3.1. Testno okruženje

Sva mjerenja se odvijaju na istoj hardverskoj i softverskoj podršci (Tablica 3.1).

Tablica 3.1 Specifikacije hardvera i softvera mjernog računala

CPU	Intel(R) Core(TM) i5-3570 @ 3.40GHz
GPU	NVIDIA GeForce GTX 1050 Ti
RAM	16GB DDR3
OS	Windows 10 Pro 64-bit (10.0, Build 19045)
Compiler	MSVC 14.51.36231
OpenGL	4.6.0

3.2. Testne scene

Mjerenje se odvija na različitim tipovima scena. Korišteno je 5 scena za testiranje:

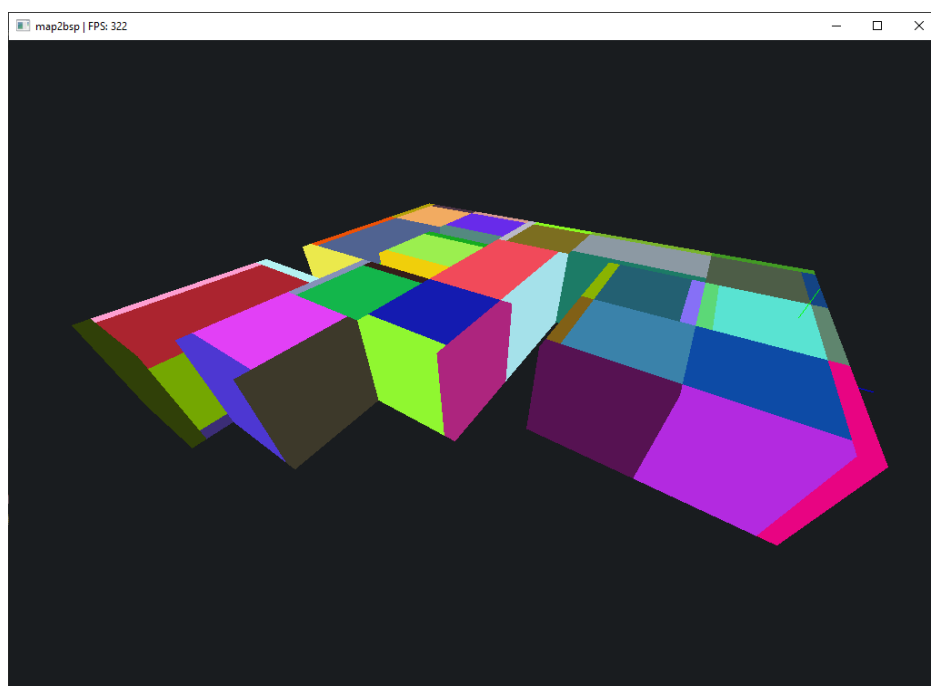
1. Kontrolna scena, ovo je mala scena koja se sastoji od jedne prostorije i u njoj je jedna kocka.
2. Manja scena samo s hodnicima. Ovakav tip scena je vrlo dobar izbor za PVS sustav.
3. Scena srednje veličine s hodnicima i s prostorijama. Prostorije omogućuju maksimalan učinak PVS-a jer je vidljivost iz njih mala.
4. Veća scena, nalik na labirint. Labirint je zapravo skup međusobno povezanih uskih i malih hodnika. PVS će moći dobro iskoristiti ovakvu strukturu scene.

5. Veća otvorena scena koja nema baš prostoriya. Velike i otvorene scene nisu najbolji izbor za PVS optimizaciju zato što nema puno zaklanjanja i očekuje se minimalna promjena performanse s PVS-om.

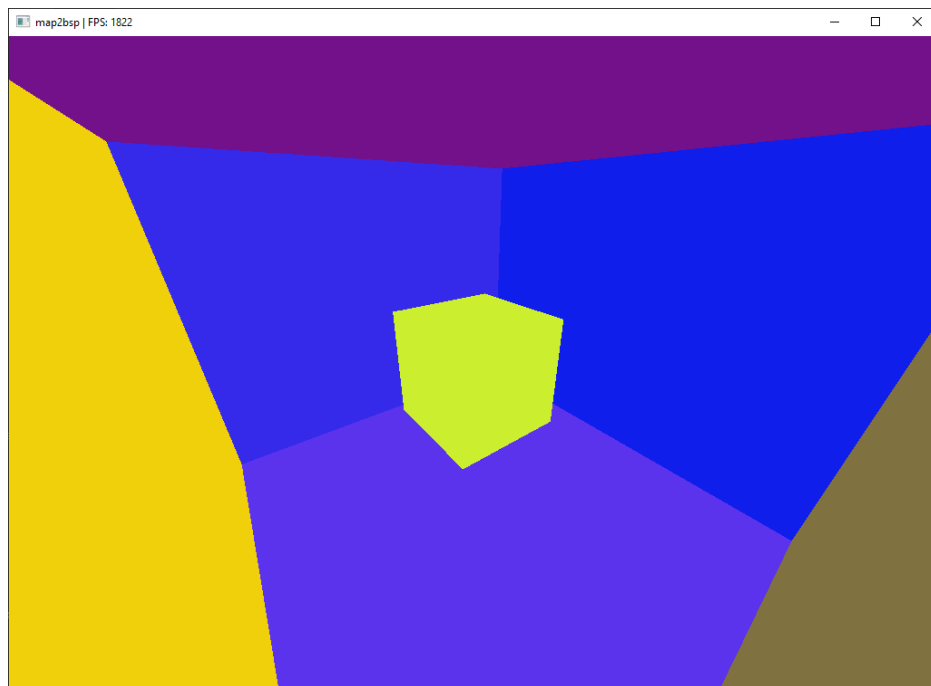
Rezultati mjerenja ovih scena su u poglavlju 3.4 Rezultati dok su ispod u tablici navedeni osnovni podaci testnih scena (Tablica 3.2). Broj listova i broj portala su uzeti nakon BSP i PVS generacije.

Tablica 3.2 Osnovni podaci testnih scena

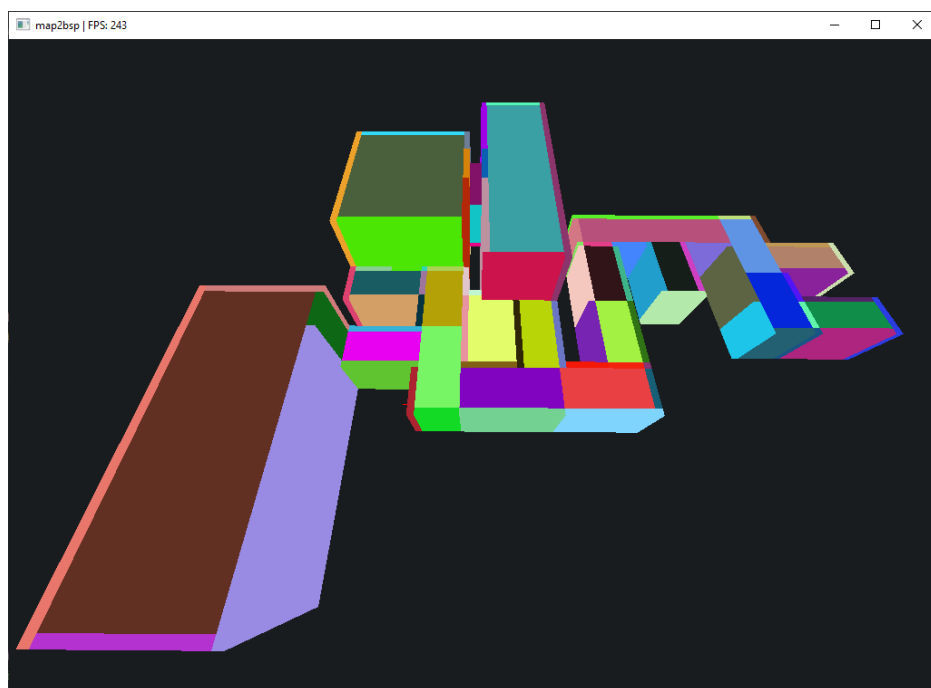
RB	Naziv scene	Broj objekata	Broj listova	Broj portala
1.	<i>control.map</i>	7	41	24
2.	<i>hallways.map</i>	78	373	58
3.	<i>rooms.map</i>	116	561	90
4.	<i>maze.map</i>	20	82	52
5.	<i>outside.map</i>	10	62	126



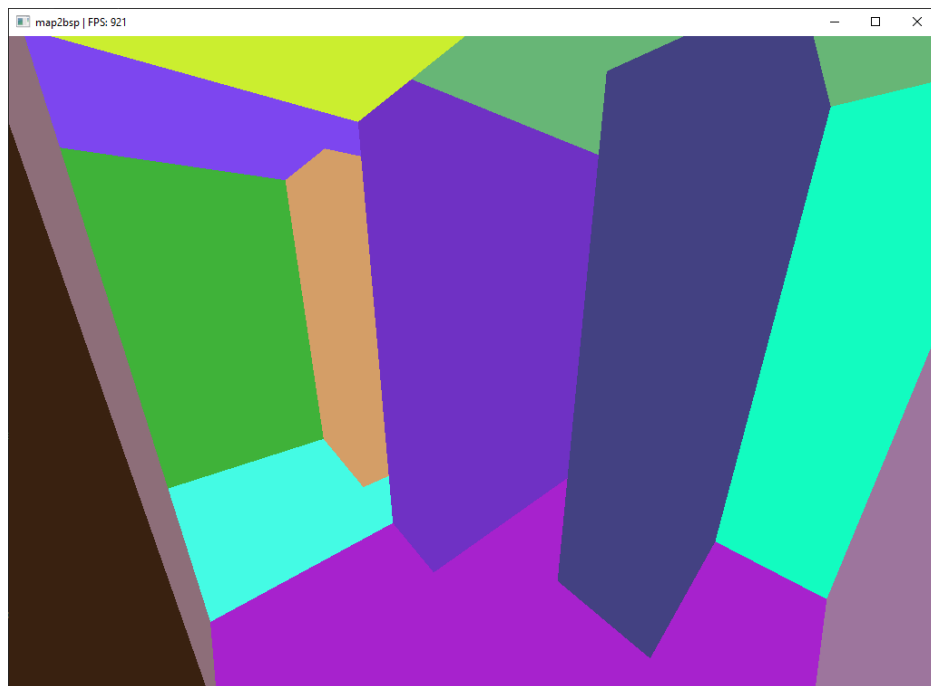
Slika 3.1 Prikaz scene hallways.map unutar aplikacije



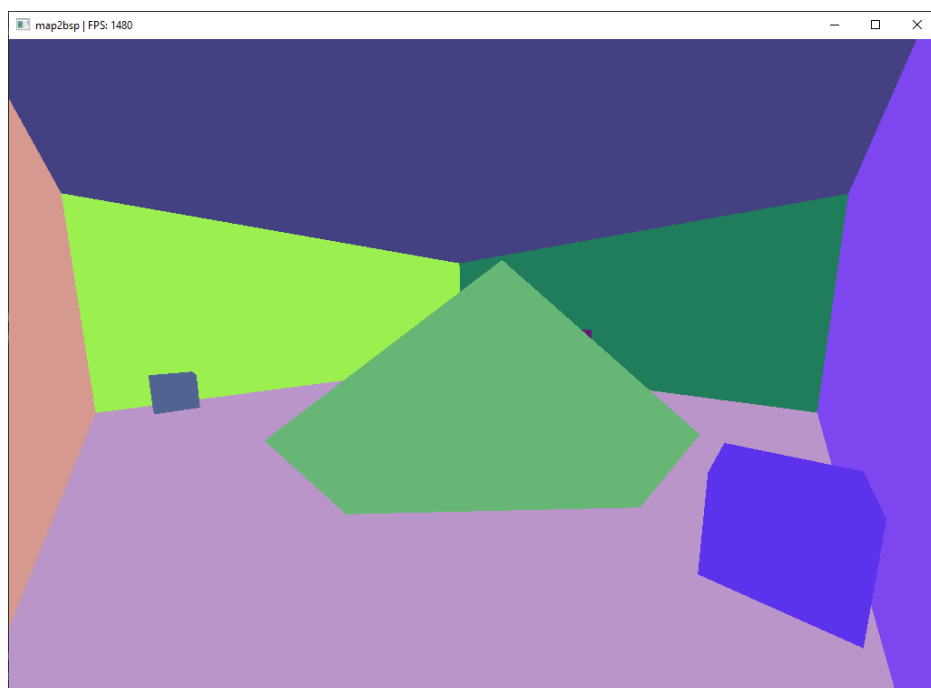
Slika 3.2 Prikaz scene control.map unutar aplikacije



Slika 3.3 Prikaz scene rooms.map unutar aplikacije



Slika 3.4 Prikaz scene maze.map unutar aplikacije



Slika 3.5 Prikaz scene outside.map unutar aplikacije

3.3. Metodologija mjerenja

Mjerenje se provodi bilježeći razne ispitne parametre unutar pojedinih scena. Prva i osnovna je prosječan broj slika po sekundi (FPS). Dodatno renderer koristi jedan poziv za iscrtavanje (*draw call*) za svaki objekt, tj. za svaki brush, pa će se zbog toga mjeriti i broj poziva za iscrtavanje. Korisno je znati i podatke o listovima. Podaci o listovima su minimalan i maksimalan broj vidljivih listova u nekom listu te prosječan vidljivih listova. Svi podaci se bilježe u centru svakog lista scene na tri različite metode iscrtavanja. Metode iscrtavanja su sljedeće: bez optimizacija, iscrtavanje s pomoću BSP obilaska, te iscrtavanje s pomoću BSP+PVS sustava. Svi podaci se bilježe statično, kamera se redom postavlja u centar svakog lista, a mjeri se prosjek kroz kraći period na toj poziciji, za svaku od tri metode iscrtavanja.

3.4. Rezultati

Tablica 3.3 Mjerenja scene control.map

Metoda	Renderirani listovi (min., prosjek, max.)	FPS
Bez optimizacije	7, 7, 7	1709
BSP	7, 7, 7	1573
BSP+PVS	7, 7, 7	1530.5

Tablica 3.4 Mjerenja scene hallways.map

Metoda	Renderirani listovi (min., prosjek, max.)	FPS
Bez optimizacije	78, 78, 78	330.5
BSP	78, 78, 78	314.2
BSP+PVS	15, 34.6, 50	715.4

Tablica 3.5 Mjerenja scene rooms.map

Metoda	Renderirani listovi (min., prosjek, max.)	FPS
Bez optimizacije	116, 116, 116	223.4
BSP	116, 116, 116	219.3
BSP+PVS	15, 34.5, 65	689.5

Tablica 3.6 Mjerenja scene maze.map

Metoda	Renderirani listovi (min., prosjek, max.)	FPS
Bez optimizacije	24, 24, 24	818.2
BSP	24, 24, 24	825.2
BSP+PVS	9, 12.1, 19	1374.6

Tablica 3.7 Mjerenja scene outside.map

Metoda	Renderirani listovi (min., prosjek, max.)	FPS
Bez optimizacije	10, 10, 10	1119.3
BSP	10, 10, 10	1134.6
BSP+PVS	10, 10, 10	1248.8

3.5. Analiza i rasprava

Rezultati pokazuju da je učinak BSP+PVS optimizacije izrazito ovisan o tipu scene, dok sam BSP obilazak (bez PVS filtriranja) donosi zanemariv učinak na performanse u odnosu na iscrtavanje bez ikakve optimizacije.

Kod scena s izraženom unutrašnjom strukturom, *hallways.map* i *rooms.map*, BSP+PVS donosi najveći dobitak, s FPS-om koji rase s otprilike 330 na 715 (*hallways.map*, Tablica 3.4) i s 223 na 690 (*rooms.map*, Tablica 3.5), više nego dvostruko u oba slučaja. Ovo prati broj iscertanih listova. Prosječan broj vidljivih listova pada s ukupnih 78, odnosno 116, na 34-35. Ovo potvrđuje da PVS uspješno eliminira velik dio geometrije koja je izvan vidnog polja kroz zidove hodnika. Scena labirinta (*maze.map*, Tablica 3.6) pokazuje sličan učinak, 818 na 1375 FPS, prosječno s 24 na 12.1 vidljivih listova. Dobitak je nešto manji jer je scena manja i broj objekata mal, pa je i bazni FPS visok.

Nasuprot tome, *control.map* i *outside.map* pokazuju da PVS ne donosi korist kad ne postoji zaklanjanje koje bi se moglo iskoristiti. Kod *control.map* broj vidljivih listova ostaje konstantan u sva tri scenarija, što znači da je cijela scena uvijek vidljiva iz svake pozicije, pa i FPS blago pada s BSP+PVS-om s 1709 na 1530 u odnosu bez optimizacije (Tablica 3.3). Ovo je očekivano jer PVS uvodi dodatan trošak, obilazak stabla za pronalazak lista u kojem je pozicija kamere. *outside.map*, iako ima 126 portala (najviše od svih testnih scena), pokazuje minimalan dobitak (s 1119 na 1249 FPS) jer je struktura scene otvorena zbog čega je malo stvarnog zaklanjanja.

Zanimljivo je da sam BSP (bez PVS-a) u četiri od pet scena rezultira blago nižim FPS-om nego iscertavanje bez ikakve optimizacije. Ovo je očekivano zato što je potrebno obići cijelo BSP stablo za iscertavanje. Samo u jednoj sceni (*outside.map*) gdje BSP ima malo veću performansu nego bez optimizacije, no razlika je premala da bi bila značajna s obzirom na broj objekata u sceni.

Sveukupno, rezultati potvrđuju očekivanje iz teorijskog dijela: PVS je najkorisniji u zatvorenim scenama s puno unutrašnjih prijelaza (hodnici, sobe, labirinti), gdje aktivno smanjuje broj listova koje je potrebno iscertati, dok u otvorenim ili vrlo malim scenama njegov trošak može minimalno nadmašiti korist. Valja napomenuti i ograničenje metodologije: sva mjerenja su bilježena u centru svakog lista, a ne duž kontinuirane putanje kamere kroz scenu, odnosno kako bi stvarno korisničko iskustvo bilo. Mogla bi biti razlika zato što sobe i uski prolazi koji posjeduju puno listova mogu povećati prosjek iz razloga zato što se uzima mjerenje iz svakog lista.

Zaključak

Ovaj rad je istražio dvije optimizacijske strukture podataka za određivanje vidljivosti u zatvorenim trodimenzionalnim scenama. Istražene su strukture binarne podjele prostora (BSP) i potencijalno vidljivih skupova (PVS). Navedene strukture su implementirane u sklopu aplikacije razvijene u C++-u i OpenGL-u.

Izrađen je kompajler koji iz *.map* scene izrađene u alatu *TrenchBroom* scene gradi BSP stablo, generira usmjerene portale te nad njima izvodi PVS izračun u dvije faze (grubu i finu fazu). Rezultat izračuna je popis potencijalno vidljivih listova po svakom listu BSP stabla koji se koristi tijekom iscrtavanja kako bi se izbjeglo obrađivanje geometrije koja sigurno nije vidljiva iz trenutne pozicije kamere.

Provedena mjerenja su potvrdila očekivanja iz teorijskog dijela rada: kombinacija BSP stabla i PVS izračuna donosi značajan dobitak u zatvorenim scenama s puno soba i hodnika. Razlog je zato što PVS eliminira velik dio geometrije zaklonjenim zidovima. Pokazao se isto i nedostatak djelovanja PVS sustava u otvorenim ili malim scenama gdje nema puno zaklanjanja geometrije. U ovim slučajevima PVS sustav daje minimalna poboljšanja performansi.

Trenutačna implementacija ima određena ograničenja. Kompajler ne podržava izgradnju iznimno kompleksnih *.map* scena, a ne provodi ni provjeru curenja (*leak*) portala. Kod kompleksnih scena ova ograničenja dovode do pogrešnog izračuna scene i taj izračun nije upotrebljiv. Mjerenja performansi provedena su statično u centru svakog lista. Rezultati bi se mogli dodatno potvrditi mjerenjem kroz kontinuiranu putanju kroz scenu umjesto statičnog načina mjerenja.

Moguća proširenja budućeg rada uključuje podršku za izgradnju kompleksnih *.map* scena, uvođenje tekstura u scene te optimizaciju izgradnje trenutnog BSP+PVS sustava.

Literatura

- [1] Duske K., *TrenchBroom Level Editor*, Poveznica: <https://trenchbroom.github.io/>
- [2] Eric W., ericw-tools, Poveznica: <https://ericwa.github.io/ericw-tools/>
- [3] Abrash M., Black Book, Poveznica: <https://www.jagregory.com/abrash-black-book/#chapter-59-the-idea-of-bsp-trees>
- [4] Väänänen P., Portals and Quake, Poveznica: <https://30fps.net/pages/pvs-portals-and-quake/>
- [5] Lee Y., Rendering Pipeline, Poveznica: <https://leeyngdo.github.io/blog/computer-graphics/2024-02-29-graphics-pipeline/>
- [6] Chrisjohnson, English Wikipedia, CC BY-SA 3.0, Poveznica: <https://commons.wikimedia.org/w/index.php?curid=20655693>
- [7] Sejievan, Binary space partitioning, Poveznica: https://developer.valvesoftware.com/wiki/Binary_space_partitioning

Izjava o korištenju umjetne inteligencije

Osvrt na korištenje umjetne inteligencije: Umjetna inteligencija je korištena za provjeru pravopisnih i stilskih pogrešaka u pisanju rada. Generativni modeli se nisu dobro pokazali za programski dio rada. Prilikom pokušaja korištenja takvih uočeno je više halucinacija i pogrešaka u shvaćanju povezanosti različitih komponenata rada.

Izjava o korištenju umjetne inteligencije: Potvrđujem da sam tijekom izrade ovog završnog rada koristio umjetnu inteligenciju u skladu s Politikom primjerenog korištenja umjetne inteligencije na Fakultetu elektrotehnike i računarstva, te da umjetna inteligencija nije zamijenila moje kritičko razmišljanje niti moj doprinos.

Sažetak

Optimizacijske strukture podataka BSP i PVS

Cilj ovog rada je istražiti i implementirati optimizacijske strukture podataka binarne podjele prostora (engl. *Binary Space Partitioning*, BSP) i potencijalno vidljivih skupova (engl. *Potentially Visible Sets*, PVS) te njihova primjena u prikazu zatvorenih scena u stvarnom vremenu. Opisana je teorija iza struktura podataka BSP i PVS te njihova uloga u rješavanju problema vidljivosti objekata. Aplikacija je razvijena u programskom jeziku C++ koristeći grafičko sučelje OpenGL koja implementira navedene optimizacijske metode. Provedena su mjerenja performansi i učinkovitosti na više testnih scena te je napravljena usporedba prikaza scene s optimizacijama i bez njih. Rezultati pokazuju da korištenjem BSP i PVS struktura u iscrtavanju zatvorenih scena može značajno ubrzati iscrtavanje i smanjiti broj objekata za prikaz.

Ključne riječi: binarna podjela prostora, BSP, potencijalno vidljivi skupovi, PVS, računalna grafika, optimizacija prikaza, vidljivost objekata, OpenGL, zatvorene scene, iscrtavanje u stvarnom vremenu

Summary

Optimization Data Structures BSP and PVS

Goal of the thesis is to investigate and implement optimization data structures Binary Space Partitioning (BSP) and Potentially Visible Sets (PVS) and their application to real-time rendering of indoor scenes. Theory behind BSP and PVS techniques and their roles in solving visibility of objects is presented. An application implementing these optimization methods was developed in C++ using OpenGL graphics API. Performance measurements were conducted on several test scenes and rendering with and without optimizations was compared. The results demonstrate that using BSP and PVS optimizations can significantly reduce the number of processed objects and improve the efficiency of rendering indoor scenes.

Keywords: Binary Space Partitioning, BSP, Potentially Visible Sets, PVS, Computer Graphics, Rendering Optimization, Object Visibility, OpenGL, Indoor Scenes, Real-Time Rendering

Skraćenice

BSP *Binary Space Partitioning*

binarna podjela prostora

PVS *Potentially Visible Sets*

potencijalno vidljivi skupovi