SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1765

STRUKTURE PODATAKA ZA CAD OBJEKTE

Goran Obradović

Zagreb, listopad 2008.

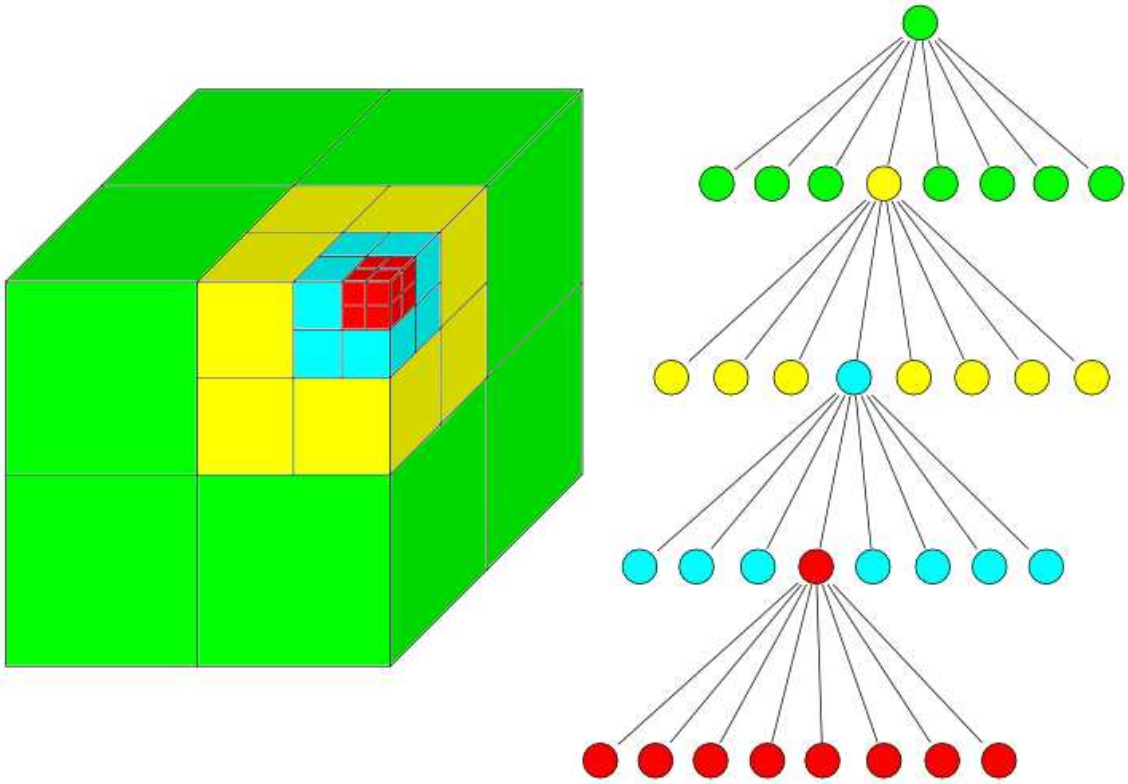
Sadržaj

1. Uvod

Računalom potpomognut dizajn (eng, Computer Aided Design, CAD) je danas vrlo važan u raznim granama industrije. Omogućuje dizajn, odabir materijala, simulacije itd. prije nego što se napravi prototip proizvoda. Time se smanjuju troškovi proizvodnje i omogućuje brži plasman na tržište. Na primjer u automobilskoj industriji se izrađuju računalni modeli, tj. CAD objekti i rade se simulacije na tim modelima poput simulacija rada motora, simulacija aerodinamike karoserije, itd. Modeli na kojima se izvode simulacije su dosta kompleksni i zahtijevaju dosta računalnih resursa te ih je potrebno strukturirati jer to znatno utječe na uštedu vremena procesiranja.

2. Oktalno stablo

Oktalno stablo (eng. Octree) je hijerarhijska struktura podataka kod koje svaki čvor ima do osmero djece. Oktalno stablo se najčešće koristi za podjelu 3D prostora gdje se prostor rekurzivno dijeli na osam oktanata tj. čvorova. Svaki čvor u stablu predstavlja dio prostora, čvorovi bez djece su listovi i oni sadrže podatke o tome što se nalazi u tome dijelu prostora (npr. vrhovi, poligoni, ...).



Slika 1: Prikaz oktalnog stabla u 3D prostoru i pripadajuće strukture

Oktalno stablo stvara se na temelju ulaznog modela, točnije njegovih vrhova, bridova i poligona. Domena koja obuhvaća geometriju modela je sadržana u korijenu oktalnog stabla. Taj korijenski čvor se zatim rekurzivno dijeli na osam djece-oktanata dok se ne dobije željena rezolucija oktalnog stabla. Tijekom dijeljenja tj. rafiniranja pojedinog oktanta on sadrži informacije o vrhovima, bridovima i poligonima koji su unutar njega. Ako neki poligon presijeca više oktanata informacija o tome će biti spremljena u svaki od tih oktanata. Svaki rafinirani oktant još sadrži informaciju o svom roditelju i djeci što omogućuje vertikalno kretanje po stablu u oba smjera. Na slici 1

prikazan je primjer oktalnog stabla. Svaki čvor u oktalnom stablu predstavlja jedan oktant u prostoru.

Korijen oktalnog stabla je poravnat sa koordinatnim osima, a time su i svi oktanti rezultirajućeg oktalnog stabla poravnati sa koordinatim osima. Time se pojednostavljuje određivanje koji element modela je u kojem oktantu i ubrzava se generiranje oktalnog stabla..

Listovi oktalnog stabla su oni oktanti koji se više ne dijele i oni sadrže informacije o elementima unutar njega. Na temelju listova se zatim stvara računska mreža, tako da rezolucija oktalnog stabla određuje konačnu distribuciju veličine ćelija.

Svaki list oktalnog stabla je klasificiran u jednu od ove tri kategorije:

- vanjski list – predstavlja oktant koji se nalazi izvan modela
- unutarnji list – predstavlja oktant koji se nalazi unutar modela
- podatkovni ili granični list – predstavlja oktant koji se nalazi na granici modela i sadrži podatke o elementima unutar tog oktanta

2.1. Potpuno oktalno stablo

Oktalno stablo koje se širi u svim smjerovima se naziva potpuno oktalno stablo, tj. to je stablo kod kojeg se svi oktanti rafiniraju do neke zadane razine. Za potpuno oktalno stablo moguće je izračunati broj čvorova na pojedinoj razini s time da je početna razina nula, tj. korijen stabla je na nultoj razini:

$$n = 8^k \tag{1}$$

tj.

$$n = 2^{3k} \tag{2}$$

gdje su:

n – broj čvorova na pojedinoj razini,

k – razina za koju se računa broj čvorova

Moguće je i izračunati i ukupni broj čvorova u stablu:

$$N = \sum_{i=0}^{d} 2^{3i} \tag{3}$$

gdje su:

N – ukupan broj čvorova u oktalnom stablu,

d – dubina oktalnog stabla

Prednost potpunog oktalnog stabla je to što se adresa svakog čvora može izračunati na temelju njene pozicije u stablu tj. može napraviti bez spremanja pokazivača, ali kod veće dubine stabla taj dobitak postaje nevažan.

Nedostatak potpunog stabla je veliko zauzeće memorije, npr. za 10 razina broj čvorova je 153391689. Drugi nedostatak je taj što će većina tih čvorova, tj. pripadajućih oktanata će biti prazna.

2.2. Oktalno stablo koje se grana prema potrebi (Branch-on-need)

Oktalno stablo kod kojeg se pojedini čvorovi dijele prema potrebi, tj. ako je neki kriterij zadovoljen za razliku od potpunog stabla kod kojeg se dijele svi čvorovi do određene razine, npr. ako je broj elemenata unutar oktanta veći od nekog zadanog broja. Ovo će rezultirati neuniformnom raspodjelom oktanata, tj. neki oktanti će biti veći, a neki manji.

Budući da je većina prostora kojeg model zauzima prazan, tj. ne sadrži nikakve informacije, ovakva struktura je vrlo ekonomična u potrošnji memorije u usporedbi sa potpunim oktalnim stablom.

2.3. Interaktivna selekcija i rafiniranje oktalnog stabla

Katkad rafiniranje oktalnog stabla na temelju nekog kriterija poput maksimalnog broja objekata unutar oktanta ili maksimalne dubine stabla ne daje željene ili dovoljno dobre rezultate, te je potrebno da korisnik sam može odrediti koji će se oktanti dalje

rafinirati. To se najlakše ostvaruje prikazom oktalnog stabla i omogućavanjem interaktivne selekcije željenog oktanta ili više njih.

2.4. Balansiranje oktalnog stabla

Kako je već rečeno kod oktalnog stabla koje se grana prema potrebi oktanti će biti različitih veličina, a time će i razlika među veličinama susjednih oktanata biti velika. Radi toga će potraga za susjednim oktantima biti kompleksnija i vremenski zahtjevnija. Da bi se to izbjeglo na oktalnom stablu se obavlja balansiranje. Balansiranje je dodatno rafiniranje onih susjednih oktanata koji su udaljeni više od jedne razine u oktalnom stablu, tj. oni među kojima je razlika između razina rafiniranosti veća od jedan, to se još zove kriterij razlike od maksimalno jedne razine. Time se osigurava da je susjedni oktant maksimalno dvostruko veći ili dvostruko manji što rezultira sa ove dvije posljedice:

1.) smanjuje se broj potencijalnih susjeda jer onda oktant po svakoj stranici može imati maksimalno četiri susjedna oktanta, a time se ubrzava potraga za susjednim oktanima

2.) postiže se glađi prijelaz u rezoluciji tj. glađu promjenu veličine između susjednih oktanata

Na slici 2 prikazan je primjer balansiranja oktalnog stabla, prikazano je nekoliko koraka rafiniranja. Na prikazu:

a) imamo neko početno stablo koje je već jednom rafinirano

b) odabran je jedan oktant za daljnje rafiniranje, crveni oktanti su oni oktanti koji će se u sljedećem koraku rafinirati, na prikazu

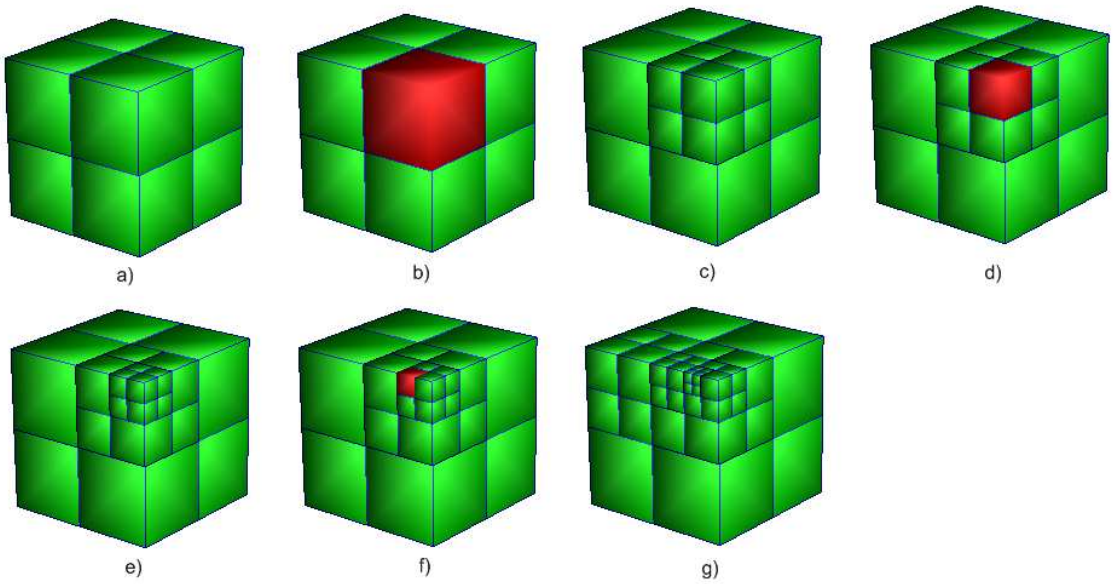
c) rezultat rafiniranja odabranog oktanta

d) isto kao i pod b)

e) isto kao i pod c)

f) odabir oktanta za rafiniranje kojem je jedan susjeda dva puta veći

g) dolazi do balansiranja gdje se dodatno rafiniraju i oni oktanti koji nisu eksplicitno odabrani za daljnje rafiniranje.



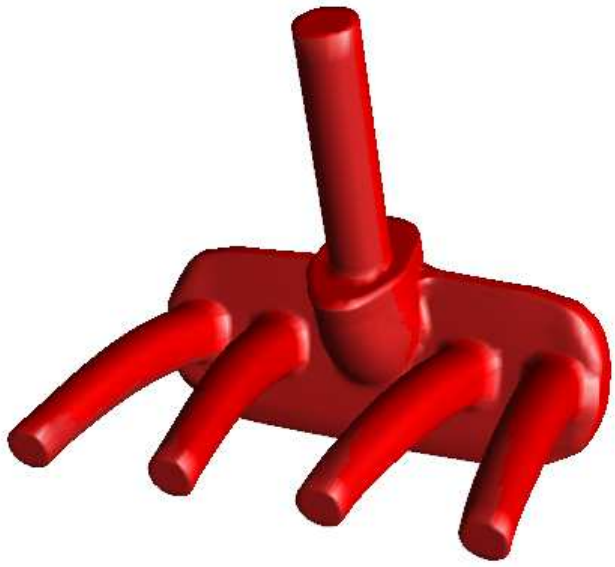
Slika 2: Balansiranje oktalnog stabla

Na prikazu f) se vidi da je došlo do rafiniranja oktanta koji nije bio direktan susjed odabranom oktantu. To znači da kod balansiranja može doći do efekta širenja valova (eng. ripple effect) kod kojeg će se dodatno rafinirati svi oktanti u stablu koji ne zadovoljavaju kriterij razlike od maksimalno jedne razine.

2.5. Generiranje oktalnog stabla

2.5.1. Ulazni model

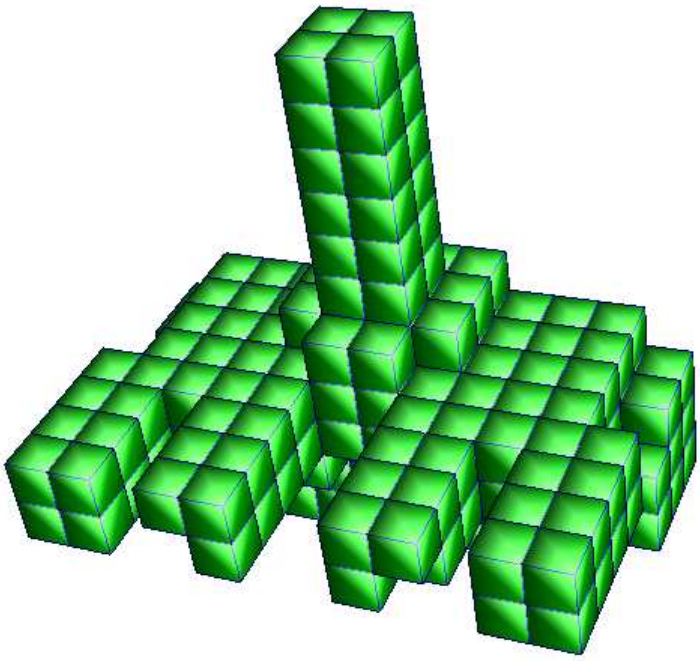
Kao što je već rečeno oktalno stablo se gradi na temelju ulaznog modela ili scene. Sam model se sastoji od niza vrhova, poligona i bridova. Vrhovi predstavljaju pozicije u trodimenzionalnom prostoru, a bridovi i poligoni (trokuti, četverokuti, ...) govore kako su ti vrhovi spojeni. Zajedno čine mrežu koja predstavlja neko tijelo iz stvarnog svijeta u računalu. Sam model je iznutra šupalj, tj. ne sadrži ništa, dok bi u stvarnom svijetu mogao biti i popunjen, npr. model Zemlje. Na slici 3 je prikazan jedan model dijela ispušnog sustava automobila za koji će se generirati oktalno stablo. Taj model se sastoji od 8804 vrhova i 17604 poligona. Prije samog generiranja odabiru se ulazni parametri npr. minimalna i maksimalna dubina stabla, maksimalni broj elementa koji oktant smije sadržavati itd.



Slika 3: Primjer modela za koji se generira oktalno stablo

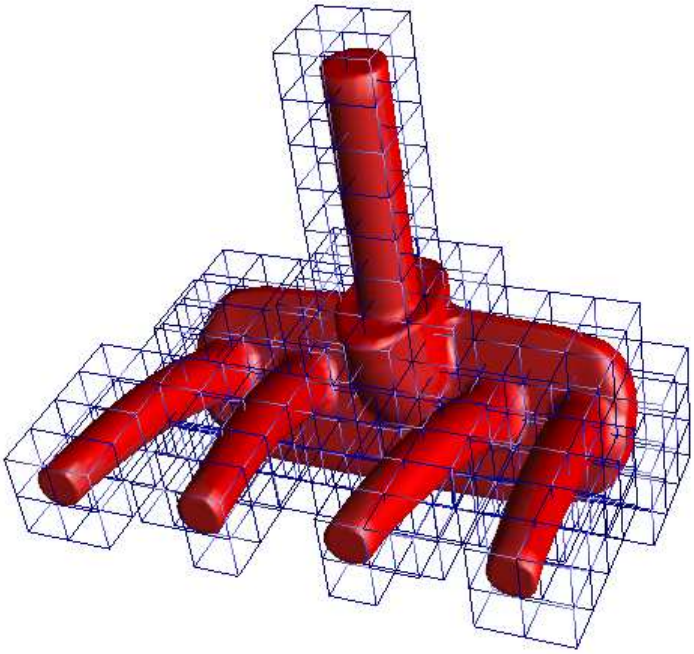
2.5.2. Generirano stablo

Na slici 4 su prikazani podatkovni (granični) listovi generiranog stabla tj. oni koji sadrže barem jedan element modela na temelju kojeg je oktalno stablo izgrađeno. Mogli bi se prikazati i svi oktanti, ali to ne bi bilo pregledno niti bi imalo smisla. Na slici se vidi da podatkovni oktanti prikazuju osnovni oblik početnog modela iako je dubina generiranog stabla mala, u ovom slučaju dubina stabla je 4 razine.



Slika 4: Listovi oktalnog stabla koji sadrže poligone

2.5.3. Usporedba modela i oktalnog stabla



Slika 5: Prikaz mreže bridova oktalnog stabla

Na slici 5 je prikazana mreža bridova generiranog stabla na kojoj se bolje vidi model i generirano oktalno stablo. Kao što se vidi iz slike oktalno stablo aproksimira ulazni model. Ta aproksimacija je sve točnija što je dubina oktalnog stabla veća. U ovom primjeru oktant s najmanjim brojem objekata sadrži 1 poligon i 0 vrhova, oktant s najvećim brojem objekata sadrži 295 poligona i 125 vrhova, prosječan broj poligona po podatkovnom oktantu je 86.97, a vrhova 31.22, ako se uzmu u obzir svi listovi prosječan broj poligona po oktantu je 5.99, a vrhova 2.15.

2.6. Primjene oktalnog stabla

Oktalno stablo u računalnoj grafici ima više primjena. One se uglavnom odnose na podjelu prostora, a time i podjelu modela na manje dijelove. Time se smanjuje broj proračuna i ubrzava algoritam koji se koristi jer se ne treba ispitivati sve na sceni nego samo određeni dijelovi. Osim kod ubrzavanja izvođenja proračuna kod simulacije oktalno stablo primjenjivo je u postupku praćenja zrake, kod problema uklanjanja

dijelova objekata obzirom na volumen pogleda, detekcije kolizije i u drugim područjima.

2.6.1. Postupak praćenja zrake (Ray-tracing)

Postupak praćenja zrake (eng. Ray-tracing) je jedna od tehnika stvaranja dvodimenzionalnog prikaza (slike) trodimenzionalne scene. Slika se stvara tako da se za svaki slikovni element (piksel) generira jedna zraka. Za tu zraku se onda određuje najbliži objekt koji je pogodila. Taj objekt je najčešće poligon. Kada se nađe taj objekt, ovisno o njegovim svojstvima, za njega se mogu stvoriti nove rekurzivne zrake. Te zrake mogu biti zraka odbijanja, zraka refrakcije i zraka sjene. Za zraku odbijanja se opet računa objekt s kojim se sudara i opet sve kreće ispočetka. Obično je određen maksimalni broj odbijanja koji će se računati. Za zraku refrakcije vrijedi isto pravilo. Zraka sjene se stvara za svaki izvor svjetlosti i provjerava se da li može doći do izvora svjetlosti, pritom potencijalno stvarajući zrake odbijanja i zrake refrakcije.

Iz ovoga se vidi da ima dosta generiranih zraka za koje treba računati s kojim objektima se sijeku. Točnije za svaku zraku bi trebalo proći kroz sve poligone da se provjeri sa kojim se siječe, a to zahtjeva mnogo procesorskog vremena. Da bi se broj provjera smanjio trebao bi se prostor scene podijeliti na više dijelova. Oktalno stablo je struktura podataka koja radi upravo to. Dijeli scenu na manje dijelove, npr. dijeli svaki oktant na sceni dok sadrži više od nekog zadanog broja poligona.

Kada je generirano stablo, za svaku zraku se zatim prvo provjerava kroz koje oktante prolazi i zatim provjerava samo one poligone koji se nalaze u tim oktantima. Provjera počinje od korijena i provjerava kroz koje oktante zraka prolazi, ako ne prolazi kroz neki oktant one ne prolazi ni kroz njegovu djecu. Ovime se može dosta uštedjeti na broju provjera, a time i na sveukupnom broju proračuna i znatno ubrzati postupak praćenja zrake.

2.6.2. Odbacivanje po volumenu pogleda (view frustum culling)

Volumen pogleda (eng. view frustum) je geometrijska reprezentacija volumena vidljivog iz očišta. Sve što se nalazi unutar tog volumena je nevidljivo i nije potrebno iscrtavati. Bez strukuturiranja podataka trebalo bi ispitati svaki poligon da li je unutar ili izvan volumena pogleda. Kako oktalno stablo dijeli prostor na manje dijelove potrebno

je samo ispitati koji oktanti su unutar, a koji su izvan volumena pogleda. Naravno i ovdje vrijedi činjenica da ako neki oktant nije unutar volumena pogleda da ni njegova djeca nisu unutar tog volumena. Time se brzo mogu izbaciti veliki komadi geometrije koju nije potrebno iscrtavati.

2.6.3. Detekcija kolizije

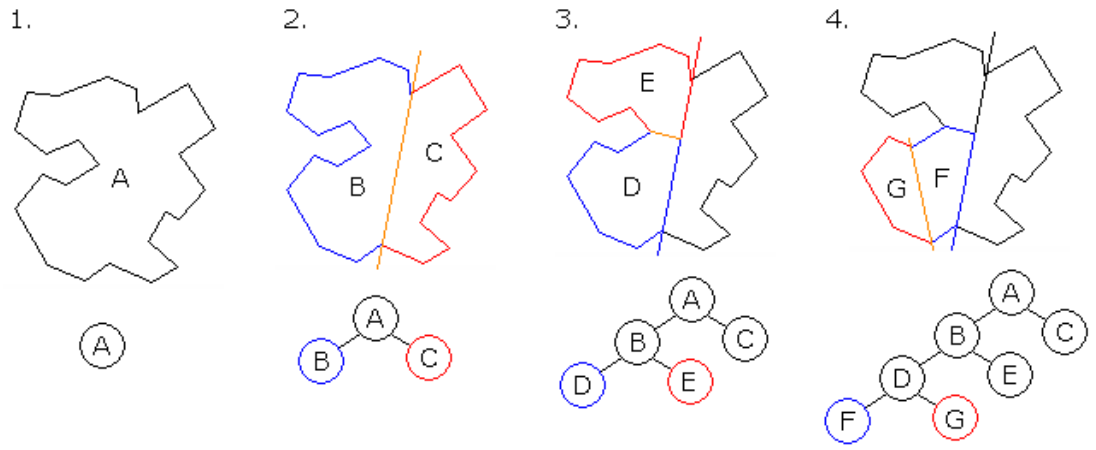
Detekcija kolizije je jedan od osnovnih problema u fizikalnim simulacijama, računalnoj animaciji, igrama, itd. Kod detekcije kolizije je potrebno izračunati da li je došlo do kolizije između dva tijela, točnije potrebno je provjeriti za svaki poligon jednog tijela i za svaki poligon drugog tijela postoji li presjek između njih. Naravno tu ima dosta računanja i taj se proces pokušava ubrzati. Jedan od načina na koji se to može ubrzati je pomoću oktalnog stabla. Kako je već prije rečeno oktalno stablo dijeli prostor na manje dijelove i svaki oktant sadrži podatke o objektima koji su sadržani unutar njegovog volumena. Tako da je prilikom izračuna kolizije potrebno provjeravati samo one poligone unutar određenog oktanta. Time se izbacuje mnogo bespotrebnih provjera i znatno se ubrzava detekcija kolizije.

3. BSP stablo

BSP stablo (eng. Binary Space Partitioning tree) je struktura podataka koja predstavlja rekurzivnu, hijerarhijsku podjelu n-dimenzionalnog prostora u konveksne skupove (dijelove). Konveksni skup je skup poligona takav da je svaki poligon ispred svakog drugog poligona u tom skupu. Poligon je ispred nekog poligona ako su svi njegovi vrhovi sa pozitivne strane tog poligona. Na slici 6 su prikazani konveksni i nekonveksni skupovi.

Slika 6: Lijevo je prikazan konveksni skup, a desno nekonveksni skup

BSP stablo je binarno stablo koje se koristi za sortiranje i pretragu n-dimenzionalnog prostora. Ako BSP stablo gledamo kao cjelinu ono predstavlja cijeli prostor (scenu, domenu), a svaki čvor u stablu sadrži jednu "hiper-ravninu" koja dijeli prostor koji taj čvor predstavlja na dva dijela. Kod n-dimenzionalnog prostora "hiper-ravnina" je n – 1 dimenzionalni objekt koji se može koristiti za podjelu tog prostora na dva polu-prostora. U jednodimenzionalnom prostoru "hiper-ravnina" je točka i dijeli pravac na dva polu-pravca, u dvodimenzionalnom prostoru "hiper-ravnina" je pravac i dijeli ravninu na dvije polu-ravnine, a u trodimenzionalnom prostoru "hiper-ravnina" je ravnina i dijeli prostor na dva polu-prostora. Svaki čvor još dodatno sadrži i dva čvora-djece, prednji koji predstavlja prostor ispred "hiper-ravnine" i stražnji koji predstavlja prostor iza "hiper-ravnine". Dalje se ti dijelovi mogu rekurzivno dijeliti. Na slici 7 je prikazan primjer stvaranja BSP stabla u dvodimenzionalnom prostoru.



Slika 7: Primjer stvaranja BSP stabla u dvodimenzionalnom prostoru

3.1. Generiranje BSP stabla

Algoritam za generiranje BSP stabla se sastoji od 3 koraka:

1. odaberi ravninu za podjelu prostora
2. podijeli skup poligona odabranom ravninom
3. rekurzivno prođi kroz algoritam za svaki od dva nova skupa

Odabir ravnine za podjelu prostora ovisi o tome kako će se BSP stablo upotrebljavati. Moguće je odabrati ravninu iz ulaznog skupa poligona ili odabrati ravninu koja je poravnata sa koordinatnim osima. Poželjno je da stablo bude balansirano, tj. da svaki list stabla sadrži približno jednak broj poligona. Ako poligon siječe ravninu podjele podijelit će se na dva ili više dijela. Loš odabir ravnine podjele rezultirat će povećanjem ukupnog broja poligona. Potrebno je naći ravnotežu između balansiranog stabla i velikog broja podjela.

Podjela skupa poligona ravninom radi se tako da se klasificira svaki poligon iz skupa u odnosu na ravninu podjele. Ako je poligon potpuno s jedne strane ravnine onda se on ne modificira i dodaje se u skup strane na kojoj se nalazi. Ako poligon siječe ravninu dijeli se na dva ili više dijela i svaki dio se dodaje u skup strane na kojoj se nalazi. Ako je poligon koincidentan s ravninom, tj. leži na ravnini on se dodaje u jedan od skupova. Poligon se nalazi potpuno na jednoj strani ravnine ako se svi njegovi vrhovi

nalaze na jednoj strani ravnine. Ako se neki vrhovi nalaze na jednoj strani ravnine, a drugi na drugoj strani te ravnine, taj poligon siječe ravninu i dijeli se na dva ili više dijela. Ako se svi vrhovi nalaze na ravnini onda se i taj poligon nalazi na ravnini. To je ispred/iza provjera i provodi se za svaki vrh poligona uvrštavanjem koordinata (x, y i z) tog vrha u jednadžbu ravnine:

$$Ax + By + Cz + D = 0 \tag{4}$$

gdje su:

A, B i C – vektor normale te ravnine

D – pomak ravnine po normali od ishodišta koordinatnog sustava, tj. udaljenost najbliže točke na ravnini od ishodišta koordinatnog sustava

tj. potrebno je izračunati:

$$s = Ax + By + Cz + D \tag{5}$$

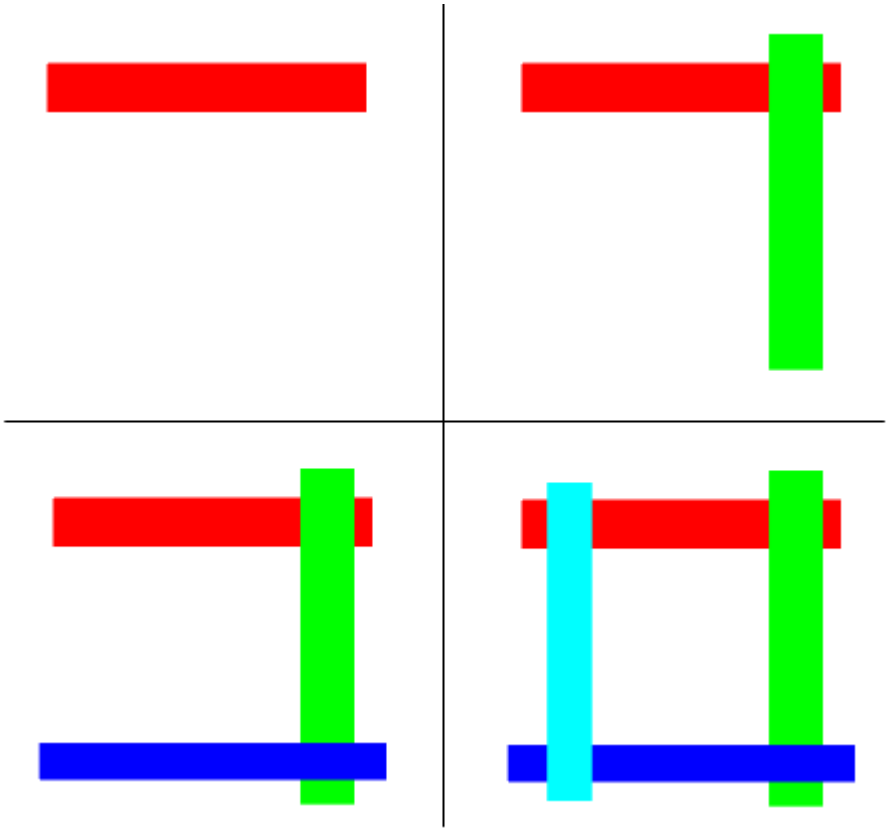
ako je s > 0 vrh se nalazi s pozitivne strane (ispred) ravnine, a ako je s < 0 vrh se nalazi s negativne strane (iza) ravnine, a ako je s = 0 vrh se nalazi na ravnini, tj. koincidentan je s ravninom.

Proces se može zaustaviti kada je broj poligona u pojedinom listu manji od nekog zadanog broja, druga mogućnost je zaustavljanje kada je dosegnuta maksimalna dubina stabla.

3.2. Primjene BSP stabla

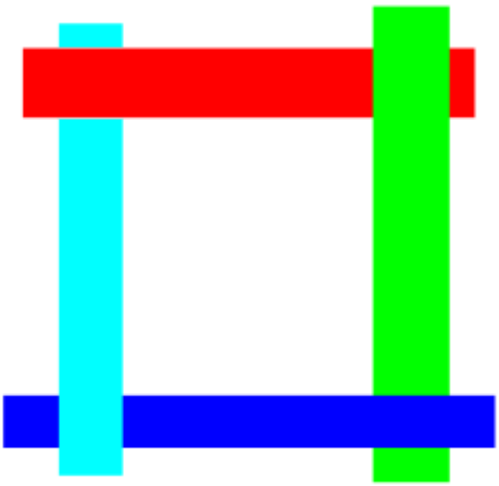
Glavna prednost BSP stabla je to što može sortirati i pretraživati poligone po njihovoj udaljenosti od neke točke. Moguće je pretraživati poligone od nazad prema naprijed (eng. back to front) i od naprijed prema nazad (eng. front to back).

Slikarev algoritam (eng. Painter's algorithm) je način iscrtavanja scene kod kojeg se iscrtavaju objekti počevši od najudaljenijeg od očišta do najbližeg. Skriveni dijelovi objekata (poligoni) će biti precrtani sa onim objektima bliže očištu. Jedini uvjet je da postoje ravnine koje međusobno odvajaju te objekte, tako da je svaki objekt sam u svom dijelu prostora. Na slici 8 su prikazani koraci slikarevog algoritma.



Slika 8: Slikarev algoritam

Ako objekte nije moguće objekte međusobno odvojiti ravninama tako da je svaki objekt u svom dijelu prostora slikarev algoritam neće dati točne rezultate. Na slici 9 je prikazan slučaj cikličkog preklapanja za koji slikarev algoritam neće točno raditi.



Slika 9: Cikličko preklapanje

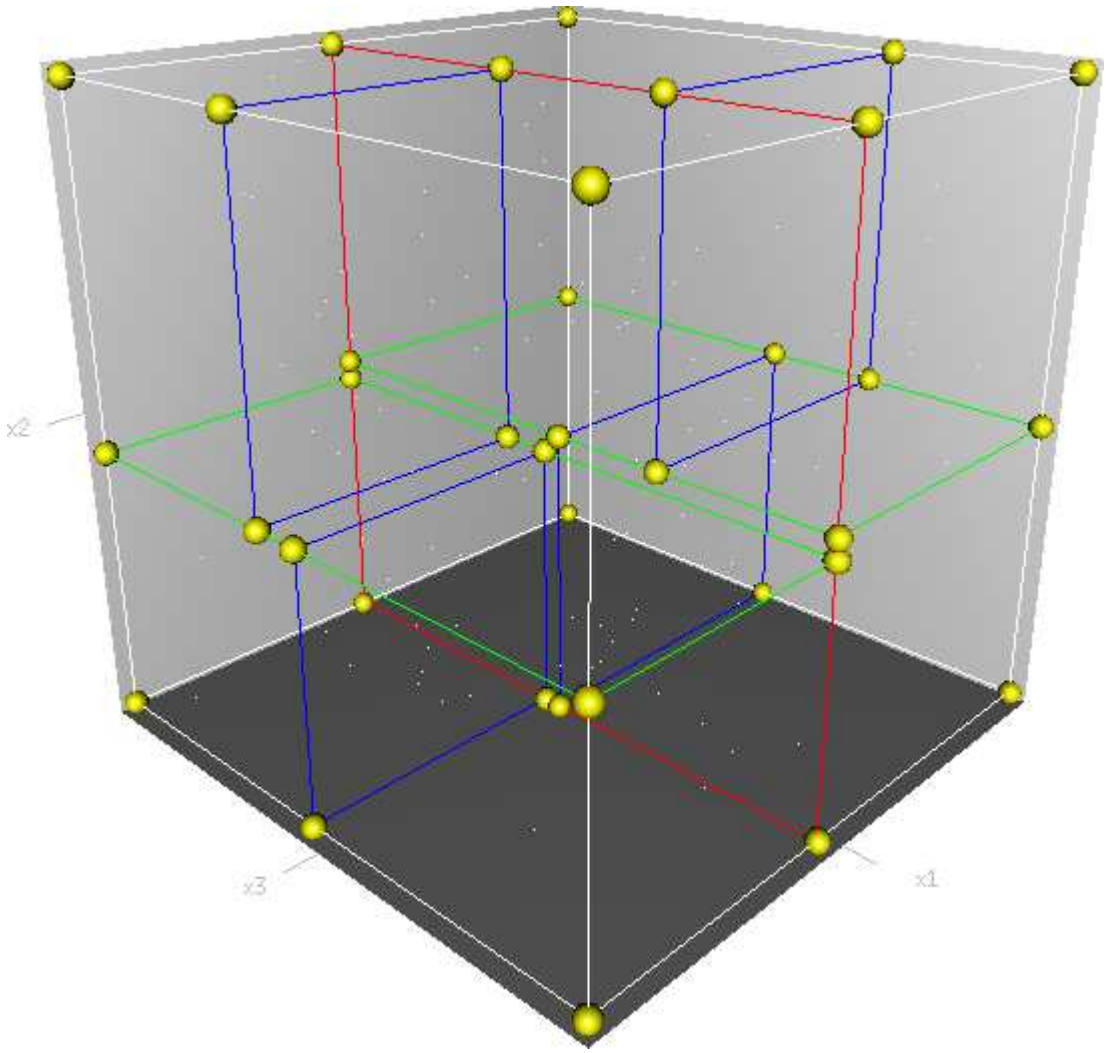
Da bi slikarev algoritam točno radio potrebno je podijeliti objekte na takav način da ih je moguće iscrtati od najudaljenijeg do najbližeg prema očištu. BSP stablo upravo omogućuje rekurzivan prolaz kroz stablo u smjeru od nazad prema naprijed. Da bi se iscrtao sadržaj BSP stabla slikarevim algoritmom potrebno je proći kroz stablo u smjeru od nazad prema naprijed. Počevši od korijena stabla odrediti gdje je očište u odnosu na ravninu podjele. Zatim iscrtati udaljeniji čvor od očišta, zatim iscrtati poligone u trenutnom čvoru, zatim iscrtati bliži čvor. Sljedeći pseudokod pokazuje prolaz kroz stablo od nazad prema naprijed (back to front).

```
void Draw_BSP_Tree (BSP_tree *tree, point eye)
{
   real result = tree->partition.Classify_Point (eye);
   if (result > 0)
   {
      Draw_BSP_Tree (tree->back, eye);
      tree->polygons.Draw_Polygon_List ();
      Draw_BSP_Tree (tree->front, eye);
   }
   else if (result < 0)
   {
      Draw_BSP_Tree (tree->front, eye);
      tree->polygons.Draw_Polygon_List ();
      Draw_BSP_Tree (tree->back, eye);
   }
   else // result is 0
   {
      // the eye point is on the partition plane...
      Draw_BSP_Tree (tree->front, eye);
      Draw_BSP_Tree (tree->back, eye);
   }
}
```

Kod praćenja zrake BSP stablo se prolazi od očištu najbližeg do najdaljeg čvora u stablu. To se postiže tako da se u gornjem pseudokodu zamjeni redoslijed rekurzivnog pozivanja pretrage čvora. Tako se osigurava da se prvo provjeri da li je zraka pogodila bliže poligone, a zatim one udaljenije. Kod praćenja zrake nije potrebno dijeliti poligone da bi se dobili točni rezultati, što omogućuje da se ravnine podjele tj. "hiper-ravnine" biraju samo radi balansiranja stabla i smanjuje broj novostvorenih poligona. To omogućuje brže stvaranje BSP stabla i njegovu pretragu.

4. Kd-stablo

Kd-stablo (eng. kd-tree) je struktura podataka za podjelu prostora i organiziranje objekata u k-dimenzionalnom prostoru. Kd-stablo je poseban slučaj BSP stabla kod kojeg su ravnine podjele poravnate sa koordinatnim osima. Oktalno stablo je poseban slučaj kd-stabla kod kojeg osi podjele prolaze kroz središte oktanta koji se dijeli. Na slici 10 je prikazano 3-dimenzionalno kd-stablo.



Slika 10: kd-stablo za 3 dimenzije

4.1. Generiranje kd-stabla

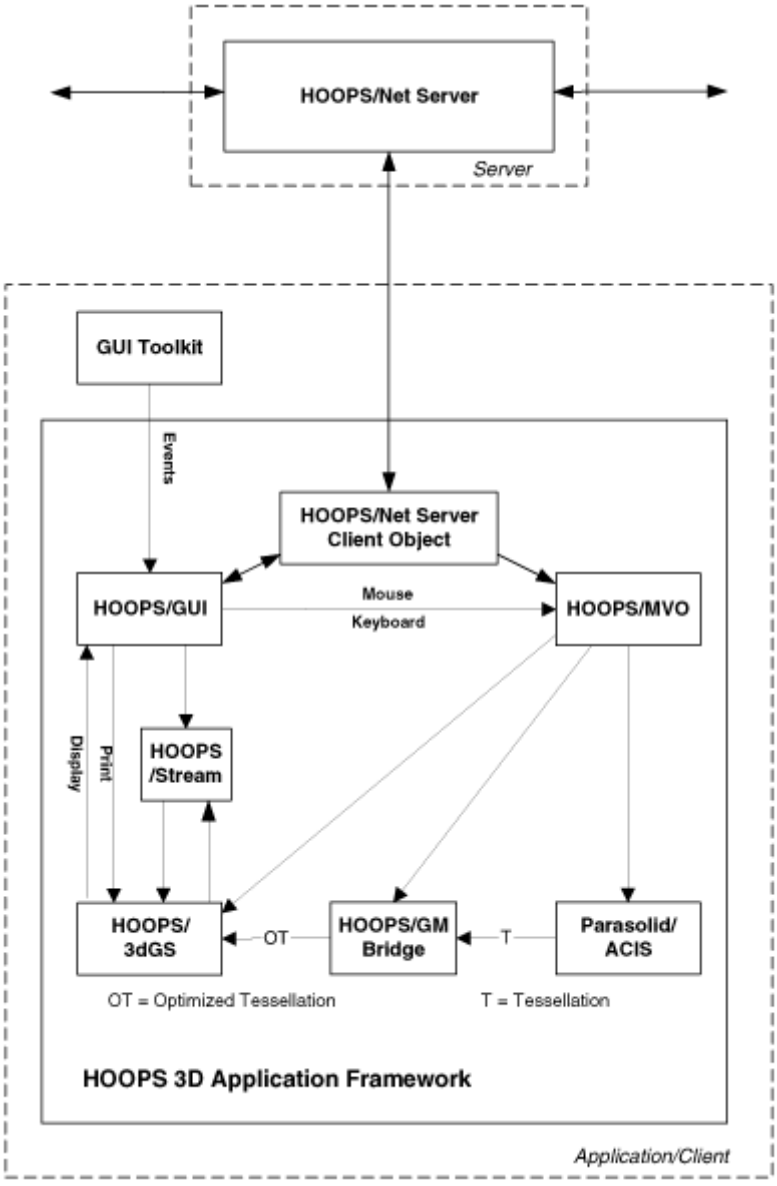
Kd-stablo se generira tako da se prvo odabere jedna ravnina za podjelu. Ta ravnina se bira ciklički prolazeći kroz osi (npr. korijen će biti podijeljen s ravninom koja je poravnata s x-osi, njegova djeca bi onda bila podijeljena s ravninom koja je poravnata s y-osi, a njihova djeca bi bila podijeljena s ravninom koja je poravnata sa z-osi, ravnina podijele na sljedećoj razini bi bila poravnata s x-osi). Ravnina podjele se bira kao medijan objekata u trenutnom čvoru s obzirom na koordinate na osi koja se koristi za podjelu. Ovo će rezultirati balansiranim kd-stablom kod kojeg je svaki list jednako udaljen od korijena stabla.

4.2. Primjene kd-stabla

Budući da je kd-stablo specijalni slučaj BSP stabla, a oktalno stablo specijalni slučaj oktalnog stabla kd-stablo se može koristiti za iste primjene. Te primjene se odnose na podjelu prostora i modela na manje dijelove, a time i smanjenje broja potrebnih proračuna i ubrzavanje algoritama. To znači da je kd-stablo primjenjivo u postupku praćenja zrake, kod uklanjanja geometrije s obzirom na volumen pogleda, detekciju kolizije i u drugim područjima.

5. HOOPS 3D Aplikacijsko Okruženje (HOOPS/3dAF)

HOOPS 3D je komercijalno aplikacijsko okruženje koje se sastoji od više integriranih komponenti koje omogućuju brzi razvoj vizualizacijskih, inženjerskih i drugih aplikacija visokih performansi. HOOPS 3D je višeplatformsko rješenje koje radi na Windows, UNIX, LINUX i Mac OS X operacijskim sustavima.



Slika 11: HOOPS 3D Aplikacijsko okruženje

HOOPS 3D Aplikacijsko Okruženje se sastoji od sljedećih komponenti:

- HOOPS 3D Grafički Sustav
- HOOPS 3D Set alata za tokove
- HOOPS 3D Model/Pogled/Operator biblioteka
- HOOPS 3D GUI Moduli za Sučelja
- HOOPS 3D Mostovi za Geometrijski Modeler
- HOOPS Aplikacijski Klijent/Server Set alata

HOOPS/3dGS

HOOPS 3D Grafički Sustav (eng. HOOPS 3D Graphics System) je objektno orijentirani API grafa scene visokih performansi koji enkapsulira grafičku bazu podataka i optimizirane algoritme za spremanje, stvaranje, editiranje, manipulaciju, ispitivanje, iscrtavanje i ispisivanje 2D i 3D grafičkih informacija. Ti algoritmi dramatično pojednostavljuju izradu 2D i 3D interaktivnih aplikacija raznih namjena poput CAD/CAM/CAE aplikacija, aplikacija za znanstvenu vizualizaciju, GIS aplikacija i drugih.

HOOPS/Stream

HOOPS 3D Set alata za tokove (eng. HOOPS Stream Toolkit) je skup alata za rad sa datotekama u HSF formatu koji omogućuje uvoz/izvoz korisničkih HSF datoteka u/iz HOOPS 3D Grafički Sustav ili grafički sustav nekog drugog proizvođača.

HOOPS/MVO

HOOPS 3D Model Pogled Operator biblioteka (eng. HOOPS Model View Operator library) je skup platformski neovisnih C++ klasa koje implementiraju osnovnu funkcionalnost poput stvaranja, pogleda, manipulacije, selekcije objekata.

HOOPS/GUI

HOOPS 3D GUI Moduli za Sučelja (eng. HOOPS GUI Interface Modules) je skup modula koji omogućuju spajanje HOOPS Grafičkog Sučelja s nekoliko drugih alata za izradu korisničkih sučelja (GUI alata), HOOPS/GUI moduli postoje za: MFC, .Net Windows Forme, ATL, QT, Java/Swing i Win32 API.

HOOPS/GMB

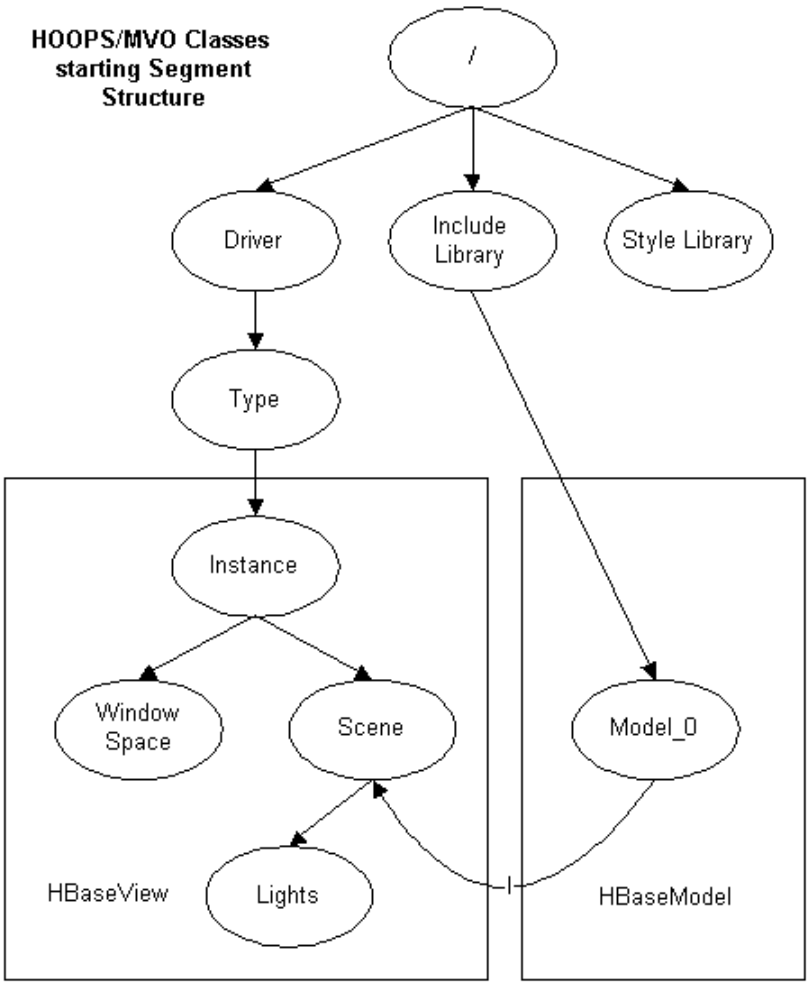
HOOPS 3D Mostovi za Geometrijski Modeler (eng. HOOPS Geometric Modeler Bridges) omogućuju spajanje raznih jezgri za modeliranje drugih proizvođača što omogućuje njihovo direktno spajanje sa HOOPS 3D Grafičkim Sustavom bez potrebe da korisnik sam dizajnira i implementira sučelje između njih.

HOOPS/Net

HOOPS Aplikacijski Klijent/Server Set alata (eng. HOOPS Application Client/Server Toolkit) je skup alata koji omogućuje izradu serverskih i klijentskih aplikacija za međusobno slanje poruka preko mreže.

5.1. Vizualizacija objekata pomoću HOOPS 3D Grafičkog Sustava

HOOPS koristi bazu podataka za spremanje objekata s kojima radi. Podaci se spremaju kao hijerarhijska struktura segmenata. Korijenski segment je "/" i sadrži segmente "Driver", "Include Library" i "Style Library". Na slici 12 je prikazana početna struktura segmenata koja se automatski stvara prilikom inicijalizacije.



Slika 12: Prikaz početne strukture segmenata

Svaki segment može sadržavati druge segmente i geometriju koja se prikazuje. Svaki segment ima jedinstveni ključ pomoću kojega ga je moguće otvoriti. Taj ključ je definiran kao HC_KEY. Segment se otvara pomoću funkcije:

```
HC_Open_Segment("segment1");
```

Ako traženi segment ne postoji on se stvara. Kada je segment otvoren s njim se može raditi npr. dodavati drugi segmenti, dodavati geometrija za prikaz, manipulirati sa opcijama segmenta itd. Kada promijenimo neku opciju u segmentu tu promjenu nasljeđuju svi podsegmenti i njihove geometrije. Kada prestanemo s radom na segmentu taj segment je potrebno zatvoriti funkcijom:

```
HC_close_segment();
```

Pomoćna funkcija koja vraća ključ osnovnog segmenta modela (HBaseModel) koji je automatski stvoren prilikom inicijalizacije i omogućuje brzo ubacivanje segmenata i geometrije je:

```
HC_KEY HBaseView::GetModelKey();
```

Funkcije za rad s geometrijom su:

```
HC_Open_Geometry(key);
HC_Close_Geometry();
```

Kada je neki segment ili geometrija otvoren onda možemo mijenjati njegove opcije poput vidljivosti, boje, način prikaza i razne druge. Neke od tih funkcija su:

```
HC_Set_Visibility("geometry = off");
HC_Set_Selectability("geometry = on");
HC_Set_Color("edges=black");
HC_Set_Rendering_Options("lighting interpolation=gouraud");
```

Moguće je postaviti, mijenjati i dohvatiti korisničke opcije pomoću sljedećih funkcija:

```
HC_Set_User_Options("boxId=1234");
HC_Show_One_User_Option("boxId", boxId);
```

5.1.1. Prikaz objekata pomoću HOOPS-a

Da bi nešto prikazali na ekranu to prvo trebamo ubaciti u bazu podataka. Sljedeći dio koda pokazuje kako ubaciti kocku u bazu, taj dio koda se nalazi u petlji pomoću koje se ubacuju sve kocke u bazu:

```
HC_Open_Segment_By_Key(getHoopsCanvas()->getHoopsBaseView()
      ->GetModelKey());
HC_Open_Segment("Kocka");
HC_Open_Geometry(HC_KInsert_Shell(8, points, 30, faces));
boxName = "boxId=";
boxName += number;
HC_Set_User_Options(boxName.c_str());
HC_Close_Geometry();
HC_Close_Segment();
HC_Close_Segment();
```

Primjer funkcije za promjenu prozirnosti nekog objekta:

```
void HoopsTest::setTransparency(float transparency, const
      char* basename, const char* segmentName)
{
      HC_Open_Segment_By_Key(getHoopsCanvas()
            ->getHoopsBaseView()->GetModelKey());
      HC_Open_Segment(basename);
      HC_Open_Segment(segmentName);
      char buf[256];
      sprintf(buf, "faces=(transmission=(r=%f g=%f b=%f))",
            transparency, transparency, transparency);
      HC_Set_Color(buf);
      HC_Close_Segment();
      HC_Close_Segment();
      HC_Close_Segment();
      getHoopsCanvas()->getHoopsBaseView()->Update();
}
```

5.1.2. Selekcija objekata pomoću HOOPS-a

Selekcija objekata omogućuje interaktivan rad s prikazanim modelom. Kada se želi odabrati nešto na ekranu, npr. jedan oktant oktalnog stabla, onda je potrebno kliknuti na prozor. Time program dobije poziciju pokazivača miša. Ta pozicija se dalje predaje HOOPS-u koji određuje koji dio geometrije je odabran. Tada HOOPS vraća ključ odabranog dijela selekcije, u ovom slučaju oktanta. Nakon toga se otvara taj dio geometrije i dohvaća se korisnička opcija u koju je zapisan identifikacijski broj oktanta,

to se zatim sprema u jednu listu i mijenja se boja selektiranog oktanta. Primjer tog koda je:

```
char* boxId = new char[32];
if(HC_Compute_Selection(/*"?Picture"*/"/driver/opengl/window0+0"
      , "", "v", event.GetMouseWindowPos().x,
      event.GetMouseWindowPos().y) > 0)
{
      HC_KEY key;
      int d1;
      int d2;
      int d3;
      HC_Show_Selection_Element(&key, &d1, &d2, &d3);
      HC_Open_Geometry(key);
      HC_Show_One_User_Option("boxId", boxId);
      XS_Data::getInstanceFast()
            ->selection.push_back(atoi(boxId));
      HC_Set_Color("red");
      HC_Close_Geometry();
}
delete boxId;
GetView()->Update();
```

6. Fame Foundation biblioteke

Fame Foundation je skup statičkih biblioteka koje sadrže klase za rad s trodimenzionalnim modelima. Neke od tih klasa su: klase za rad s mrežama poligona, klase za rad sa oktalnim stablima, klase za točke, vektore i matrice, klase za vrhove, bridove i poligone, itd.

6.1. Rad sa Fame Foundation bibliotekama

Da bi se moglo raditi s nekim modelom prvo ga je potrebno učitati sa diska. To se radi pomoću funkcije:

```
FF_MeshTNGTools::ReadMeshFromFile(fileNameString, *meshSP);
```

Kada je učitan model potrebno je prvo translatirati i promijeniti veličinu modela da stane u prostor (0, 0, 0) – (1, 1, 1), zatim stvoriti instancu oktalnog stabla sa željenim parametrima maksimalne i minimalne dubine stabla i maksimalnog broja objekata, postaviti model za koji se generira oktalno stablo, stvoriti instancu upravitelja za rafiniranje i zatim pokrenuti rafiniranje, nakon što je rafiniranje gotovo potrebno je vratit model na početnu veličinu i početnu poziciju. Kad je to gotovo potrebno je stvoriti vizualnu reprezentaciju oktalnog stabla u HOOPS 3D Grafičkom Sustavu. Primjer koda kojim je to moguće napraviti je:

```
XS_MeshTNGTransformationTools::TransformMeshToUnity(ov->myMesh,
      ov->scalingFactor, ov->shift);
ov->octree = new FF_RecursiveOctree(maxR, minR, maxN);
ov->octree->setData(ov->myMesh);
FF_RecursiveOctreeRefinement another_refinement_manager(*ov
      ->octree, 0, FF_True);
another_refinement_manager.refine();
XS_MeshTNGTransformationTools::InvTransformMesh(ov->myMesh, ov
      ->scalingFactor, ov->shift);
ov->hoopsInterface->createOctree(ov->octree, ov->basename, ov
      ->scalingFactor, ov->shift);
```

Kada je prikazano oktalno stablo moguće je odabrati jedan ili više oktanata za daljnje rafiniranje. Nakon što su odabrani svi željeni oktanti, lista tih oktanata se šalje funkciji za rafiniranje:

```
XS_MeshTNGTransformationTools::TransformMeshToUnity(ov->myMesh,
      ov->scalingFactor, ov->shift);
FF_RecursiveOctreeRefinement ror(*ov->octree, 0, FF_True);
ror.refineBoxes(XS_Data::getInstanceFast()->selection);
XS_MeshTNGTransformationTools::InvTransformMesh(ov->myMesh, ov
      ->scalingFactor, ov->shift);
ov->hoopsInterface->deleteOctree(ov->basename);
ov->hoopsInterface->createOctree(ov->octree, ov->basename, ov
      ->scalingFactor, ov->shift);
XS_Data::getInstanceFast()->selection.clear();
```

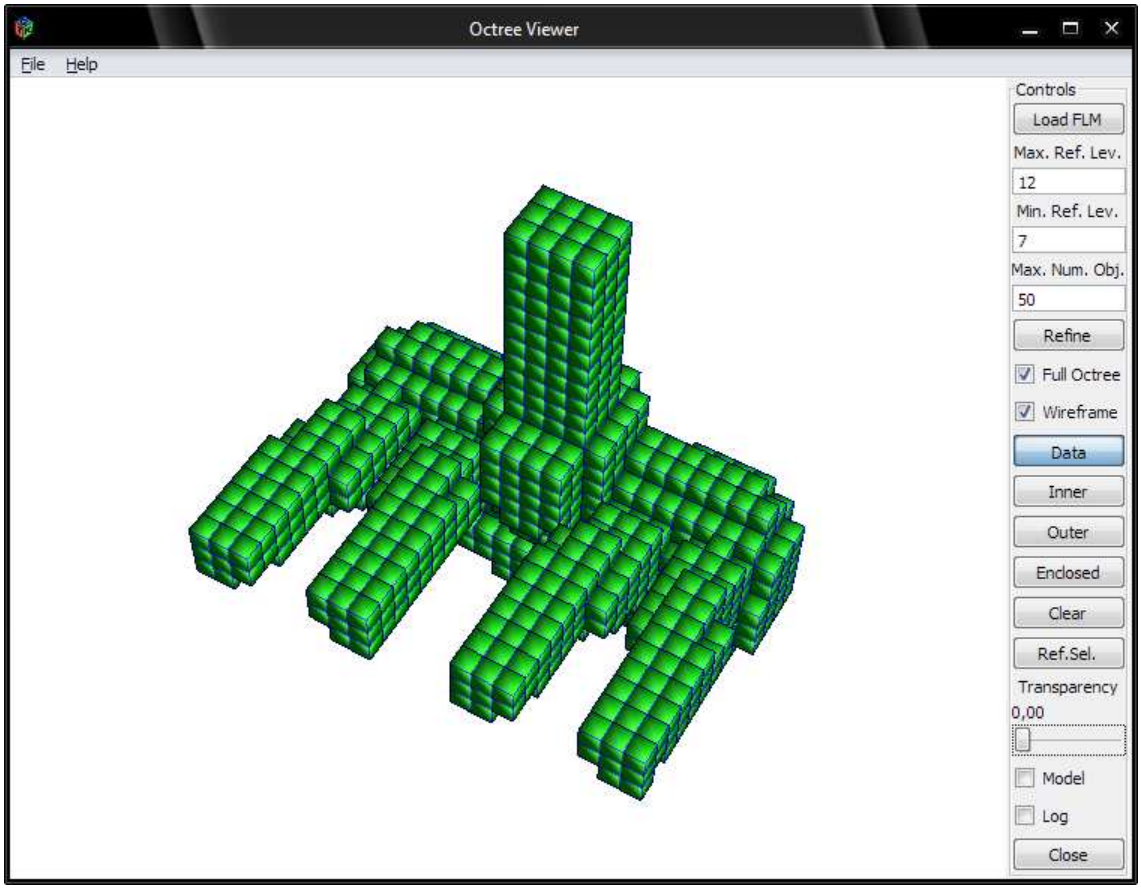
Kod funkcije za rafiniranje liste oktanata je:

```
void FF_RecursiveOctreeRefinement::refineBoxes(vector<int>
      selection)
{
      FF_ALT();
      FF_Int num_refinements = 0;
      FF_LogInfo0(APP_PART_OCTREE, LOG_PRIORITY_NORMAL, "Box
            selection based refinement being applied." );
      octree_.setUpInnerOuterInfo();
      FF_Int i,j,k;
      FF_Array<FF_Uint8> ref_marker_helper;
      ref_marker_helper.setNumElements(octree_.getNumBoxes());
      ref_marker_helper.init(0);
      FF_Array<FF_RecursiveOctreeBox*> leaves;
      leaves.setNumElements(0);
      octree_.collectLeaves(leaves);
      FF_Uint num_leaves = leaves.getNumElements();
      FF_RecursiveOctreeBox* leaf = 0;
      FF_Uint num_refined_boxes = 0;
      FF_Uint total_num_refined_boxes = 0;
      for (j = 0; j < num_leaves; j++)
      {
            leaf = leaves.getElement(j);
            for (i = 0; i < selection.size(); i++)
            {
                  if(selection[i] == leaf->getBoxId())
                  {
                        leaf->setRefinementMarker(1);
                        break;
                  }
            }
      }
      FF_LogInfo0(APP_PART_OCTREE, LOG_PRIORITY_NORMAL, "Box
            selection based refinement - Done." );
      num_refined_boxes = 0;
      octree_.traverseDownRefineMarkedLeaves(num_refined_boxes);
      cout << "Refined " << num_refined_boxes << " boxes." <<
            endl;
      octree_.clearInnerOuterInfo();
```

```
            octree_.setUpInnerOuterInfo();
            octree_.closeLevelDifference();
            cout << "Finished refining octree." << endl;
            cout << "Setting up inner outer info." << endl;
            octree_.clearInnerOuterInfo();
            octree_.setUpInnerOuterInfo();
            cout << "Setting up inner outer info - done." << endl;
        }
```

7. Programska podrška

Program za vizualizaciju i interaktivno rafiniranje oktalnog stabla je napravljen u suradnji sa AVL-om [11], pritom koristeći njihove Fame Foundation biblioteke i osnovni prototip programa. Taj prototip je sadržavao GTK+ HOOPS 3D Viewer kontrolu i primjer njenog korištenja. Program je trebao omogućiti vizualizaciju oktalnog stabla, promjenu vidljivosti oktalnog stabla (npr. prozirnost prikazanih oktanata da se može vidjeti model) i mogućnost odabira oktanata za daljnje rafiniranje. Program je trebao biti napisan u programskom jeziku C++, pritom koristeći HOOPS 3D Aplikacijsko Okruženje, Fame Foundation biblioteke i GTK+ alate za izradu korisničkog sučelja.



Slika 13: Prikazan je program za vizualizaciju oktalnog stabla.

Programsko sučelje se sastoji od dva dijela, to su: prozor u kojem je prikaz modela i oktalnog stabla i dijela za kontrolu rafiniranja i izgleda oktalnog stabla. Nakon

pokretanja programa potrebno je učitati željeni model, upisati parametre za rafiniranje i pokrenuti rafiniranje. Nakon toge je moguće odabrati koji dio oktalnog stabla će biti prikazan. Kada je željeni dio stabla prikazan moguće je sa Alt+desnim klikom odabrati željene oktante i pokrenuti daljnje rafiniranje. Moguće je promijeniti prozirnost oktalnog stabla da se vidi i model na temelju kojeg je stablo izgrađeno, još je moguće promijeniti i prozirnost modela.

Modeli korišteni u ovom radu su zapisani u flma datotekama. Te datoteke sadrže opis modela, tj. popis vrhova i poligona. Izgled datoteke je dan primjerom:

```
8           //broj vrhova, nakon čega slijede x, y i z koordinate
-0.188485 0.199697 0.094768 0.188486 0.199697 0.094768 -0.188485
-0.177274 0.094768 0.188486 -0.177274 0.094768 -0.188485
0.199697 -0.282203 0.188486 0.199697 -0.282203 -0.188485 -
0.177274 -0.282203 0.188486 -0.177274 -0.282203
12          //broj poligona
3 0 2 3     //prvo je dan broj vrhova u poligonu
3 3 1 0     //a onda slijede indeksi vrhova
3 5 7 6
3 6 4 5
3 4 6 2
3 2 0 4
3 1 3 7
3 7 5 1
3 4 0 1
3 1 5 4
3 2 6 7
3 7 3 2
12          //ovo sljedeće je misterij, ali broj dvojki je jednak
2 2 2 2 2 2 2 2 2 2 2 2      //broju poligona
0           //slijedi niz od šest nula – isto nepoznanica
0
0
0
0
0
```

Zbog nedostatka modela u flma zapisu modificiran je program, napravljen za seminar iz predmenta Računalna animacija na Fakultetu Elektrotehnike i Računarstva u Zagrebu, tako da modele učitane u obj formatu može spremiti u flma format.

Postoje razlike između opisane strukture oktalnog stabla i implementacije u Fame Foundationu a to su: brojanje razina počinje od maksimalnog zadanog broja razina i sa svakom razinom taj broj se smanjuje i postoji još jedna vrsta oktanta a to je okruženi oktant (eng. enclosed) koji se nalazi izvan modela ali je okružen njime.

8. GTK+ biblioteke za izradu grafičkog korisničkog sučelja

GTK+ [12] je skup višeplatformskih biblioteka koje služe za izradu grafičkih sučelja programa. Sadrži skup standardnih kontrola npr. gumbi, izbornika, okvira za izbor, kontola za unos teksta itd. U nastavku je dan dio koda za kreiranje gumba, tekstualne oznake i kontrole za unos teksta:

```
GtkWidget* buttonLoadFLM;
buttonLoadFLM = gtk_button_new_with_label("Load FLM");
g_signal_connect(G_OBJECT(buttonLoadFLM), "clicked",
G_CALLBACK(cbOpenFLM), NULL);
gtk_box_pack_start(GTK_BOX(vbox), buttonLoadFLM, false, true,
2);
gtk_widget_show(buttonLoadFLM);

GtkWidget* label;
label = gtk_label_new/*_with_mnemonic*/("Max. Ref. Lev.");
gtk_box_pack_start(GTK_BOX(vbox), label, false, true, 2);
gtk_widget_show(label);
entryMaxRefLev = gtk_entry_new();
gtk_widget_set_size_request(entryMaxRefLev, 30, 18);
gtk_box_pack_start(GTK_BOX(vbox), entryMaxRefLev, false, true,
2);
gtk_widget_show(entryMaxRefLev);
```

Funkcija za stvaranje novog gumba sa tekstom je:

```
gtk_button_new_with_label("Load FLM");
```

nakon toga se može na taj gumb povezati funkcija za događaj (eng. event) klika na taj gumb:

```
g_signal_connect(G_OBJECT(buttonLoadFLM), "clicked",
G_CALLBACK(cbOpenFLM), NULL);
```

onda se taj gumb dodaje u spremnik koji sadrži komponente:

```
gtk_box_pack_start(GTK_BOX(vbox), buttonLoadFLM, false, true,
2);
```

i nakon toga se taj gumb prikazuje na ekranu:

```
gtk_widget_show(buttonLoadFLM);
```

Slična procedura se provodi i za sve ostale kontrole koje GTK+ sadrži.

9. Rezultati

Obavljena su mjerenja kod uniformnog i neuniformnog rafiniranja oktalnog stabla na dva različita modela. Prilikom tih mjerenja dobiveni su rezultati o vremenu trajanja rafiniranja, broju podatkovnih oktanata i ukupnom broju poligona.

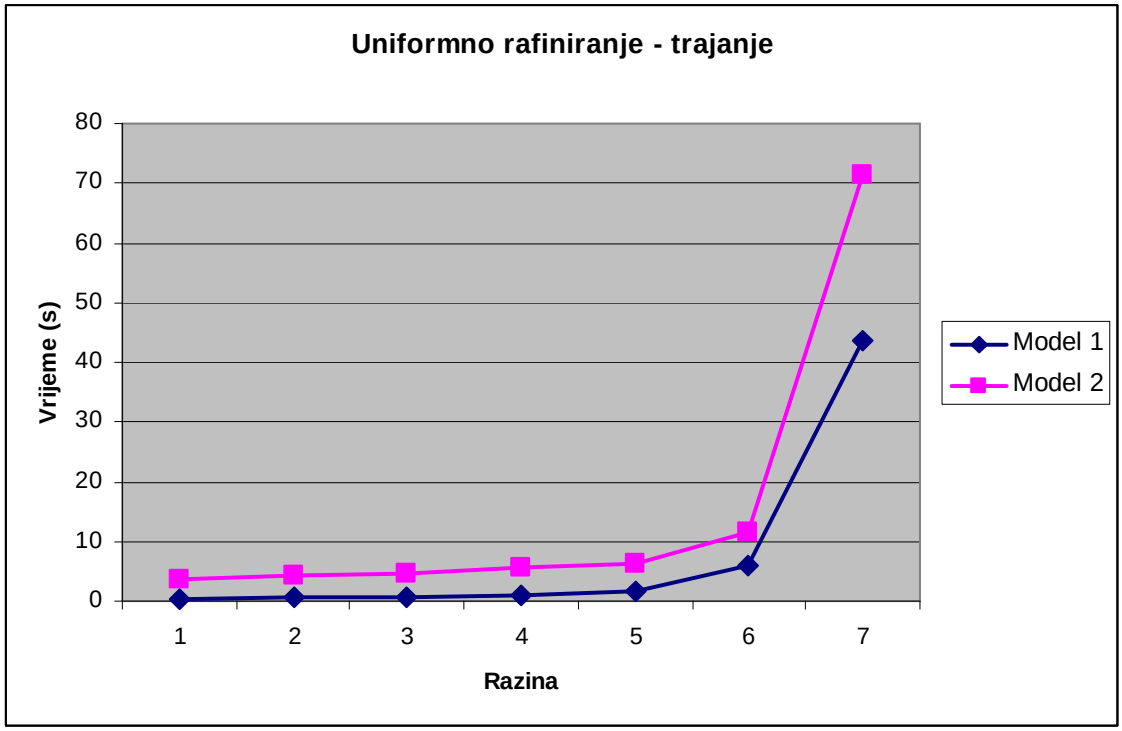
Mjerenja su provedena na računalu: AMD AthlonXP 2000+, 768 MB RAM

Korišteni modeli:

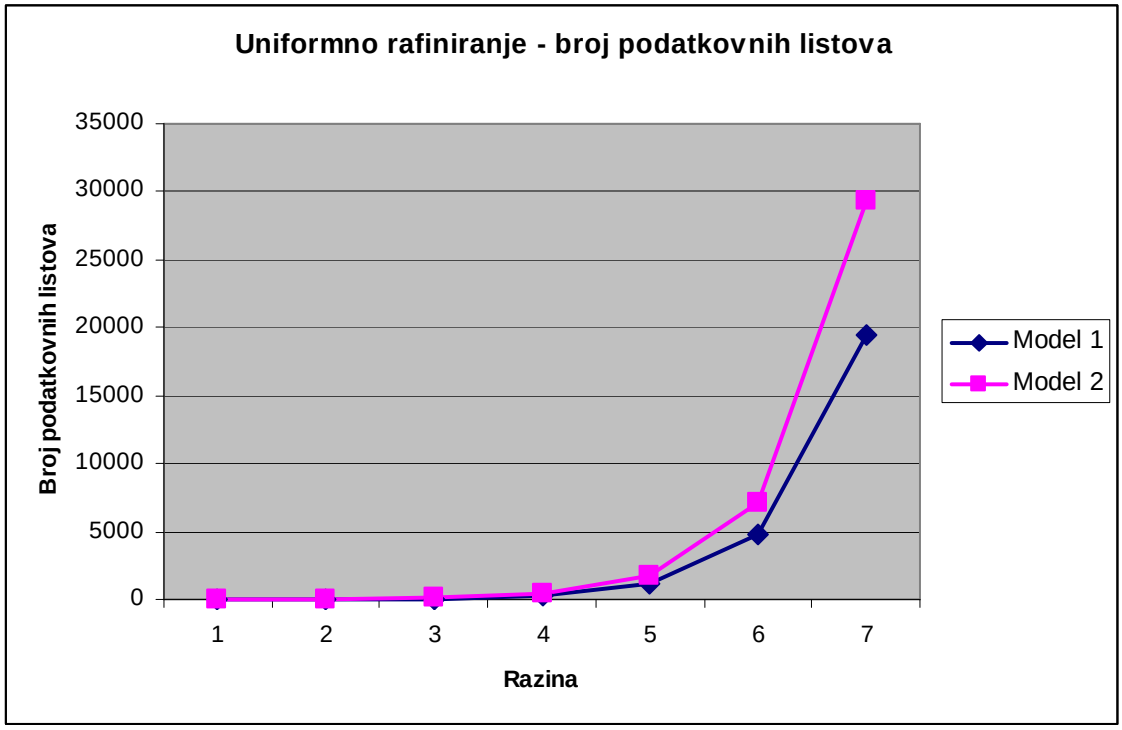
- model 1: 8804 vrhova, 17604 poligona
- model 2: 26596 vrhova, 51411 poligona

9.1. Uniformno rafiniranje

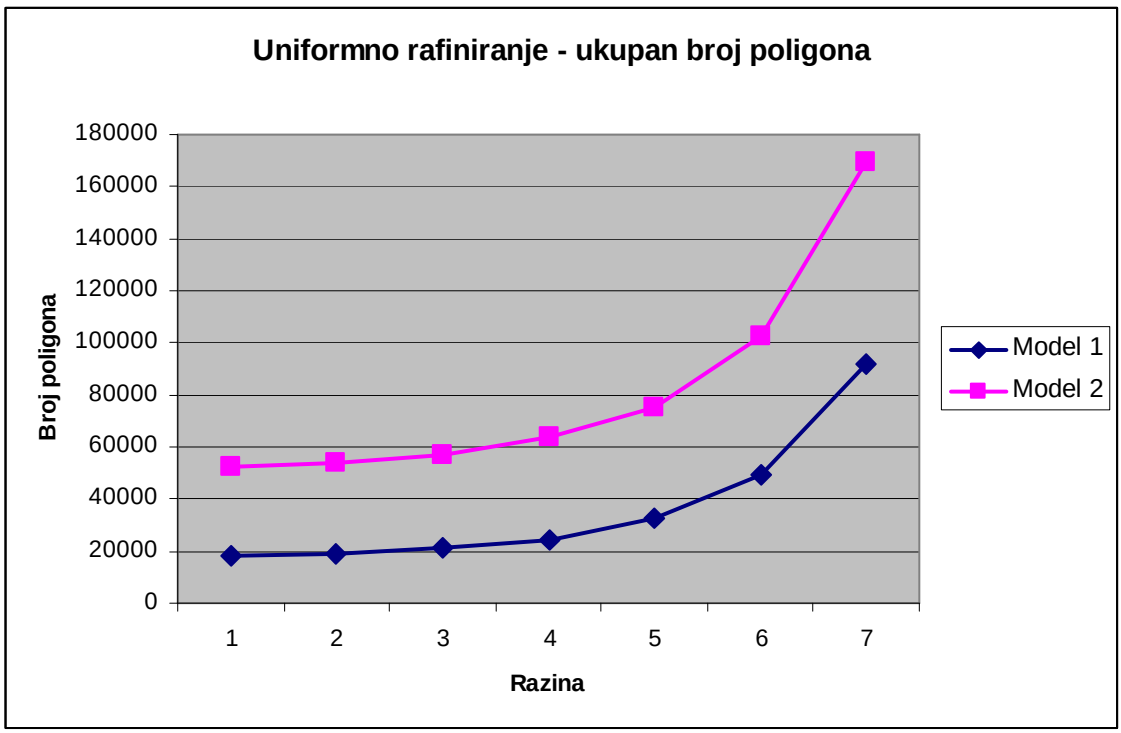
Kod uniformnog rafiniranja prikazana je usporedba između modela 1 i modela 2.



Slika 14: Graf zavisnosti vremena potrebnog za rafiniranje o broju razina u oktalnom stablu



Slika 15: Graf zavisnosti broja podatkovnih listova o broju razina u oktalnom stablu

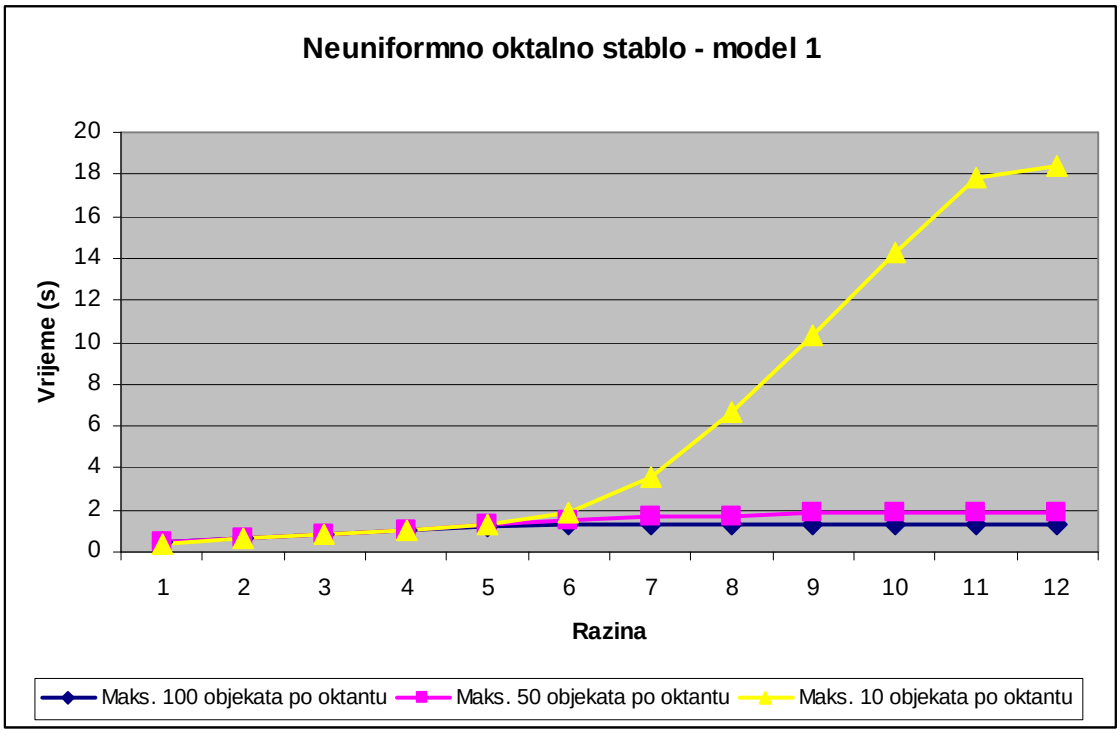


Slika 16: Graf zavisnosti ukupnog broja poligona o broju razina u oktalnom stablu

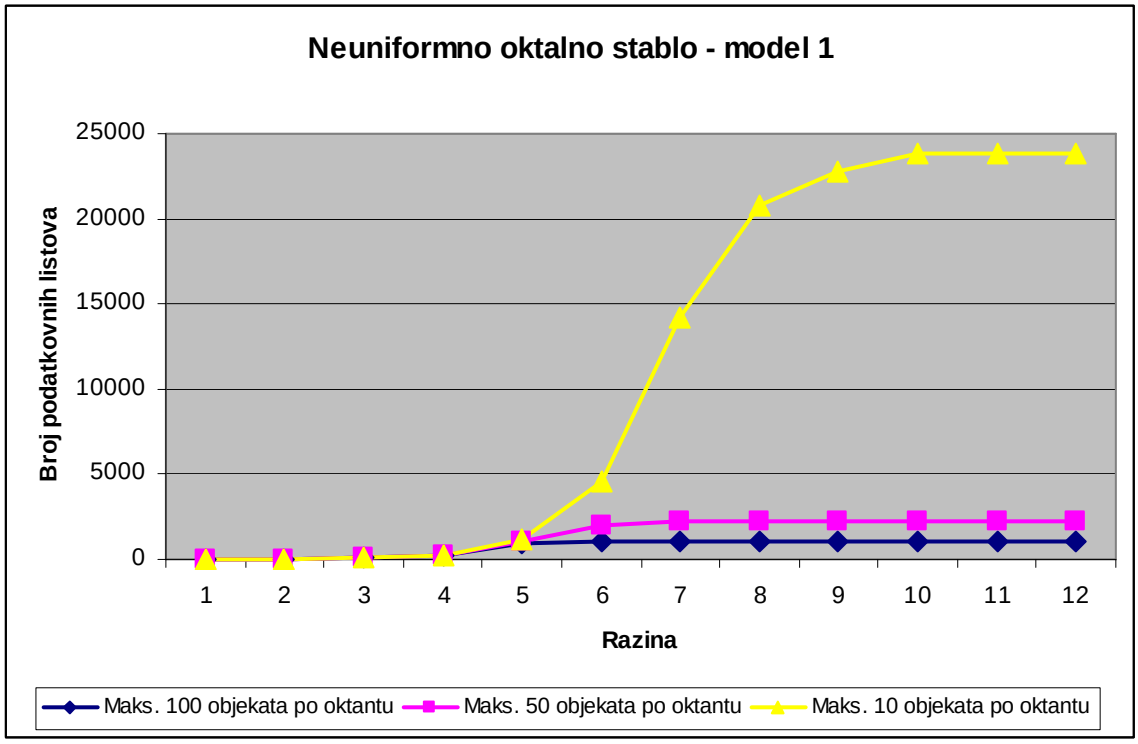
Iz grafa na slici 14 se vidi eksponencijalni rast vremena potrebnog za rafiniranje porastom broja razina u oktalnom stablu, što je logično jer se kod uniformnog tj. potpunog oktalnog stabla broj oktanata eksponencijalno povećava s brojem razina po formuli (2). Na slici 15 se vidi da i broj podatkovnih oktanata raste eksponencijalno. Na slici 16 se vidi eksponencijalni porast ukupnog broja poligona u oktalnom stablu što znači da broj redundantnih poligona brzo raste s brojem razina. Redundantni poligoni su oni poligoni za koje je informacija spremljena u više oktanata. Ako se neki poligon nalazi unutar više oktanata (tj. ako siječe više oktanata) informacija o tome će biti spremljena u svakom od njih.

9.2. Neuniformno rafiniranje

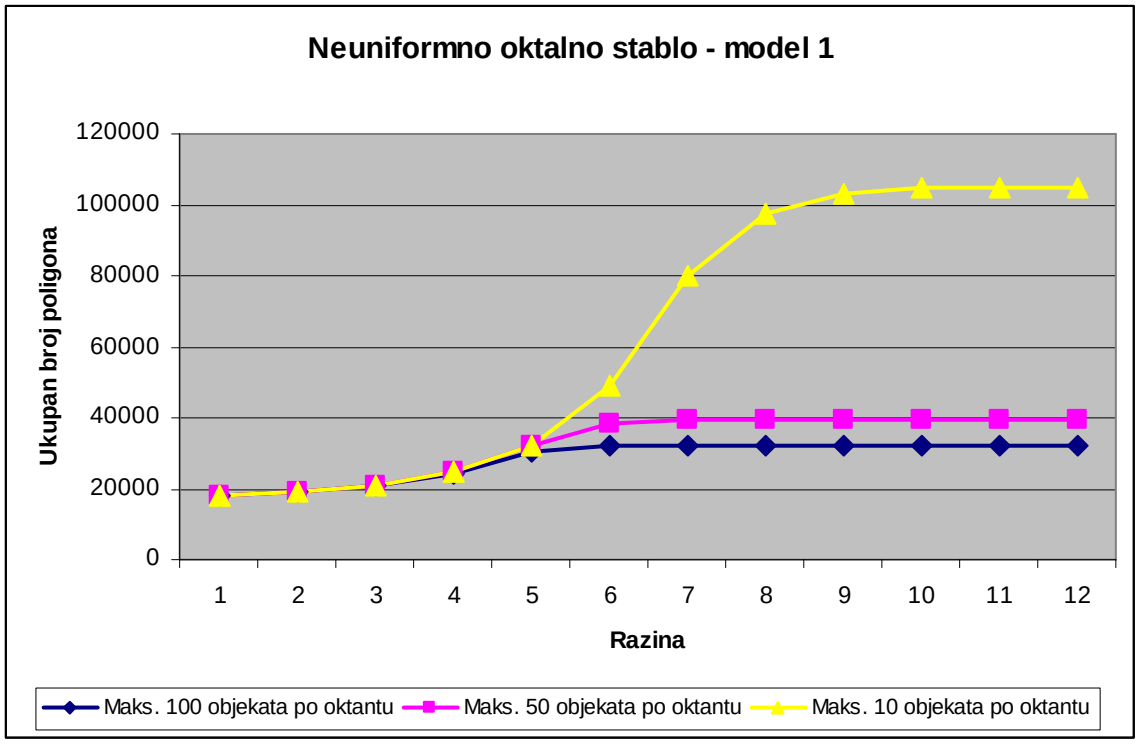
Kod neuniformnog rafiniranja napravljena je usporedba kod modela 1 i 2 za različit maksimalan broj objekata po oktantu.



Slika 17: Graf zavisnosti vremena potrebnog za rafiniranje o broju razina u neuniformnom oktalnom stablu za model 1



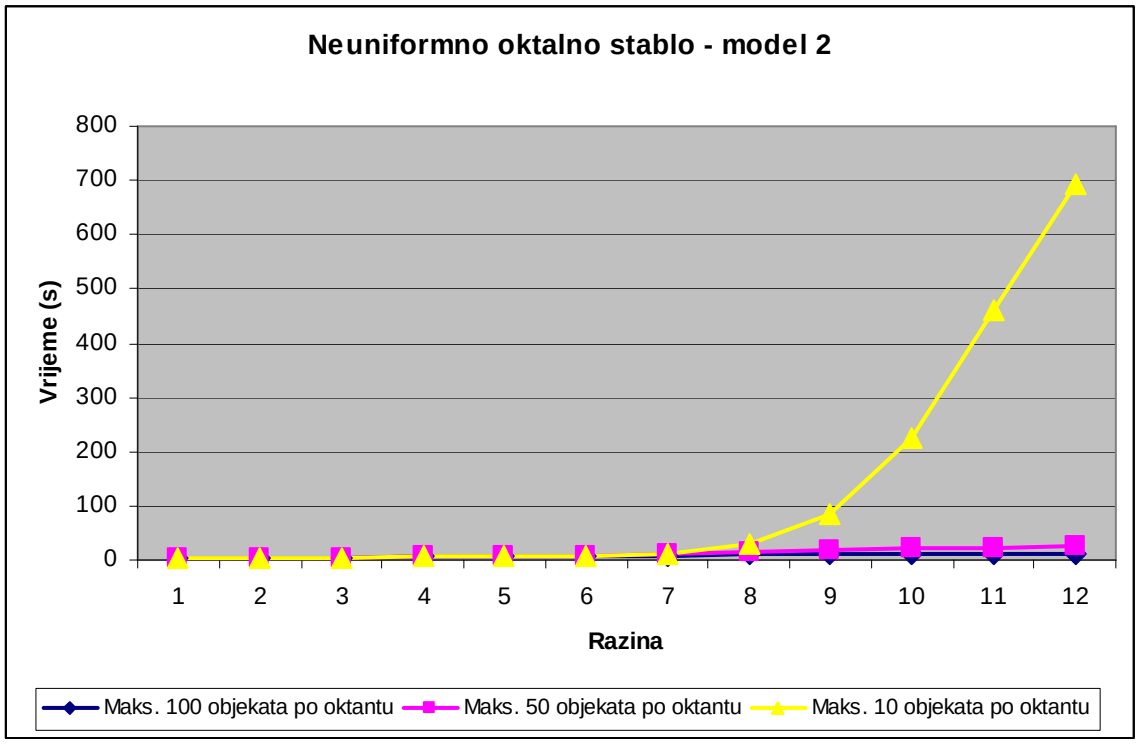
Slika 18: Graf zavisnosti broja podatkovnih listova o broju razina u neuniformnom oktalnom stablu za model 1



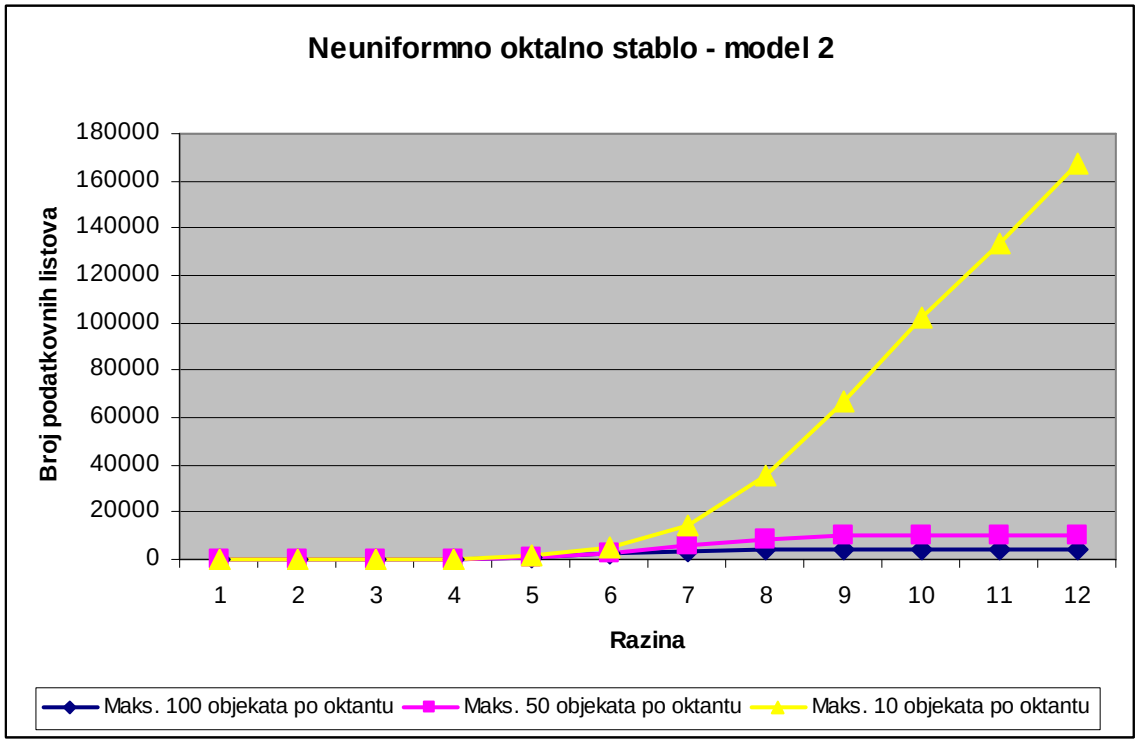
Slika 19: Graf zavisnosti ukupnog broja poligona o broju razina u neuniformnom oktalnom stablu za model 1

Na slici 17 se vidi sve brži rast vremena trajanja rafiniranja u ovisnosti o broju razina stabla. Nakon određenog broja razina to vrijeme sve sporije raste jer sve više oktanata zadovoljava uvjet maksimalnog broja objekata po oktantu. Tako da nakon određenog broja razina dolazi do "zasićenja" kada se stablo više dalje ne rafinira. Iz grafa je vidljivo da nizak zahtjev za maksimalnim brojem objekata po oktantu ima znatan utjecaj na vrijeme rafiniranja stabla, dok između stabala s maksimalnim brojem objekata po oktantu od 100 i 50 nema velike razlike.

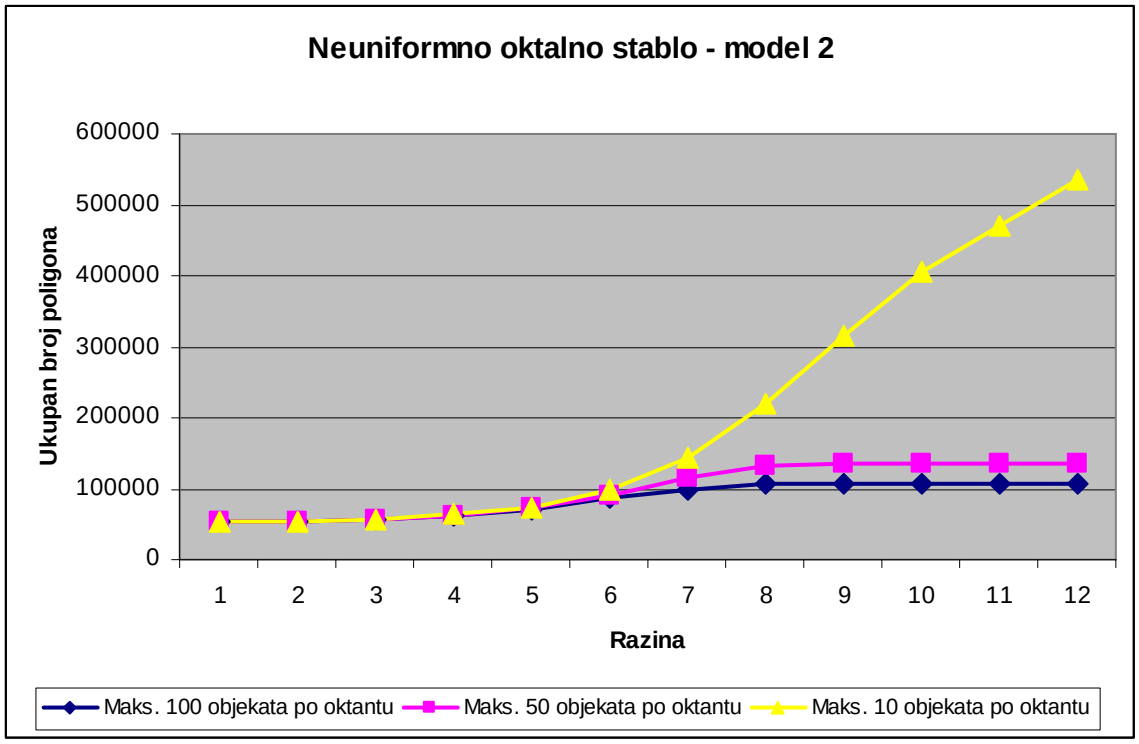
Na slikama 18 i 19 se vidi slično ponašanje kao i na slici 17 iz istog razloga. Utjecaj na memorijske zahtjeve i performanse može imati velik broj redundantnih poligona koji nastaju pogotovo kod preniskog zahtjeva za maksimalnim brojem objekata po oktantu.



Slika 20: Graf zavisnosti vremena potrebnog za rafiniranje o broju razina u neuniformnom oktalnom stablu za model 2

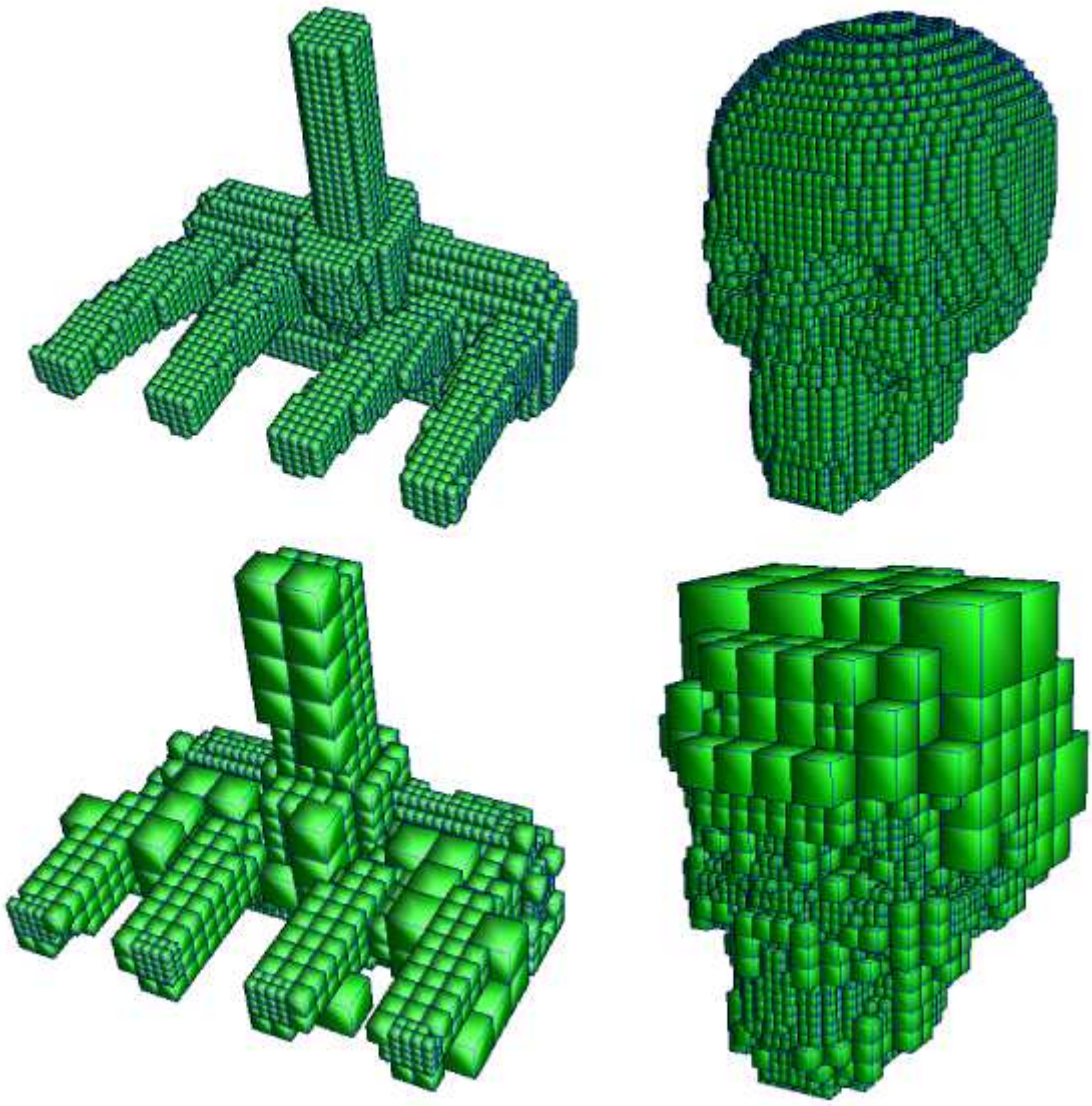


Slika 21: Graf zavisnosti broja podatkovnih listova o broju razina u neuniformnom oktalnom stablu za model 2



Slika 22: Graf zavisnosti ukupnog broja poligona o broju razina u neuniformnom oktalnom stablu za model 2

Ponašanje za model 2 je slično kao i za model 1, što je vidljivo kod oktalnih stabala sa zahtjevom od maksimalno 100 i 50 objekata po oktantu. Na primjeru sa 10 maksimalnih objekata po oktantu se vidi da je vrijeme potrebno za rafiniranje znatno veće od onog za 50 ili 100, zatim da je ukupan broj poligona znatno veći nego kod stabala sa 50 ili 100 maksimalno objekata po oktantu. Kod modela sa većim brojem poligona utjecaj maksimalnog broja objekata po oktantu je dosta značajan.



Slika 23: Usporedba uniformne i neuniformne podjele oktalnog stabla za model 1 i 2

10. Zaključak

U ovom radu predstavljene su strukture podataka pogodne za pohranu, analizu, simulaciju i vizualizaciju CAD objekata. Posebna pažnja obraćena je na oktalno stablo kao strukture podataka pomoću koje je moguće dobiti znatna ubrzanja prilikom raznih proračuna, pogotovo ako ju je moguće iskoristiti za više namjena istovremeno, npr. izbacivanje dijela geometrije koja se ne vidi i detekciju kolizije. Ona se koristi u raznim segmentima računalne industrije, od izrade profesionalnih alata za računalom potpomognut dizajn (Computer Aided Design, CAD) do računalnih igara.

Iz obavljenih mjerenja vidljivo je da oktalno stablo može zauzeti velike količine memorije zbog velikog broja oktanata i redundantnih poligona. Tako da je vrlo važno obratiti pažnju na parametre stvaranja oktalnog stabla.

Dodatno je napravljena programska implementacija koja omogućuje vizualizaciju i interaktivno upravljanje neuniformnom izradom oktalnog stabla. Prilikom izrade programske implementacije najveći problem je bio nedostatak dokumentacije za AVL-ove Fame Foundation biblioteke.

11. Popis slika

12. Literatura

1. Octree - Wikipedia: http://en.wikipedia.org/wiki/Octree
2. DmWiki: http://www.devmaster.net/wiki/Octree
3. Knoll, Aaron: A Short Survey of Octree Volume Rendering Techniques GI Lecture Notes in Informatics, (Proceedings of 1st IRTG Workshop, June 14-16 2006, Dagstuhl, Germany)
4. Kamburelis, Michalis: VRML processing and rendering engine, http://stoma.name/michalis/vrml_engine_doc/output/xsl/html-nochunks/vrml_engine.html#chapter.octree, 2006
5. Ray Tracing - Auxiliary Data Structures, http://undergraduate.csse.uwa.edu.au/units/CITS4241/Handouts/Lecture14.html
6. Binary space partitioning – Wikipedia: http://en.wikipedia.org/wiki/Binary_space_partitioning
7. DevMaster.net - BSP Trees: http://www.devmaster.net/articles/bsp-trees
8. BSP Tree FAQ: ftp://ftp.sgi.com/other/bspfaq/faq/bspfaq.html
9. kd-tree – Wikipedia: http://en.wikipedia.org/wiki/Kd-tree
10. Tech Soft 3D: HOOPS 3D, http://hoops3d.com
11. AVL: Fame Foundation, http://www.avl.com
12. The GTK+ Project: http://www.gtk.org

13. Sažetak

U ovom radu su proučene strukture podataka pogodne za pohranu, simulaciju i vizualizaciju CAD objekata. Posebno je obraćena pažnja na strukturu oktalnog stabla. Proučene su moguće primjene navedene strukture za različite namjene. Proučen je i opisan programski alat za izradu sučelja GTK. Načinjena je programska implementacija koja omogućuje interaktivno upravljanje neuniformne izrade oktalnog stabla. Za izradu je korišten programski jezik C++, grafičko programsko sučelje HOOPS i GTK.

Ključne riječi: oktalno stablo, BSP stablo, kd-stablo, računalom potpomognut dizajn, CAD

14. Abstract

This paper describes data structures suitable for storage, simulation and visualization of CAD objects. Special attention has been focused on octree data structure. Different uses for that data structure have been studied. A programming tool for designing user interfaces has been studied. A program implementation for interactive control of nonuniform creation of the octree has been created. The C++ language, HOOPS graphical framework and GTK have been used for the program implementation.

Keywords: octree, BSP tree, kd-tree, computer aided design, CAD