Danijel Žlaus

Zaznava in prikaz vidnih ploskev na GPU

Diplomsko delo

Maribor, september 2010

Diplomsko delo univerzitetnega študijskega programa računalništvo in informacijske tehnologije

Zaznava in prikaz vidnih ploskev na GPU

Študent:	Danijel Žlaus
Študijski program:	UN RIT – Računalništvo in informacijske tehnologije
Smer:	Računalništvo in informacijske tehnologije
Mentor:	red. prof. dr. Borut Žalik
Somentor:	doc. dr. Gregor Klajnšek

Maribor, september 2010

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko

Smetanova ulica 17
2000 Maribor

Številka: BRIT-16
Datum in kraj: 31. 08. 2010, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 1/2010)

SKLEP O DIPLOMSKEM DELU

1. **Danijelu Žlausu,** študentu univerzitetnega študijskega programa Računalništvo in informacijske tehnologije, se dovoljuje izdelati diplomsko delo pri predmetu Uvod v računalniško geometrijo.

2. **MENTOR:** **red. prof. dr. Borut Žalik**
   SOMENTOR: **doc. dr. Gregor Klajnšek**

3. **Naslov diplomskega dela:**
   ZAZNAVA IN PRIKAZ VIDNIH PLOSKEV NA GPU

4. **Naslov diplomskega dela v angleškem jeziku:**
   DETECTION AND VIZUALIZATION OF VISUAL SURFACES ON GPU

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (en vezan izvod in dva nevezana izvoda) ter en izvod elektronske verzije do 31. 08. 2011 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO RAČUNALNIŠTVO IN INFORMATIKO 8

Obvestiti:
- kandidata,
- mentorja,
- somentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se mentorju in somentorju za pomoč in vodenje pri opravljanju diplomskega dela. Zahvala tudi staršem, ki so mi omogočili študij.

Naslov diplomskega dela:

Zaznava in prikaz vidnih ploskev na GPU

Ključne besede: računalniška geometrija, zaznavanje vidnosti, GPE, delitev prostora, osmiško drevo

UDK: 004.94(043.2)

Povzetek

V diplomskem delu opišemo algoritem, namenjen zaznavanju vidnih ploskev. Algoritem določi, katere ploskve v sceni so vidne iz izbranega gledišča, pri tem pa učinkovito izrablja grafično procesno enoto (GPE). Za dodatno pospešitev smo uporabili tudi druge, znane, pohitritvene metode, kot sta odstranjevanje objektov, ki ležijo izven vidnega prostora ter hierarhična delitev prostora. S kombiniranjem teh metod smo dobili učinkovito rešitev, ki določi množico vidnih ploskev v realnem času tudi v primeru relativno kompleksne scene.

Title of diploma thesis:

Detection and visualization of visual surfaces on GPU

Key words: computational geometry, visibility detection, GPU, spatial subdivision, octree

UDK: 004.94(043.2)

Abstract

In this diploma thesis we present an algorithm for visible surface detection. The algorithm identifies which surfaces are visible from a viewpoint with the utilization of the graphics processing unit (GPU). In order to maximize the efficiency of the algorithm we use other common speed-up techniques like frustum culling and hierarchical space subdivision. Using a combination of these techniques, an efficient GPU based solution has been obtained. It is able to analyse fairly complex scenes in real time.

VSEBINA

1 UVOD

1.1 Področje raziskav

Področji računalniške grafike in računalniške geometrije sta relativno mladi. Področje računalniške grafike se je sicer začelo razvijati že leta 1940[1], s prihodom prvega digitalnega računalnika. Vendar pa so bile prve raziskave še zelo omejene s sposobnostmi razpoložljive strojne opreme, zato se je intenzivnejši razvoj pričel šele okoli leta 1965. Obe področji vsako leto napredujeta z neverjetno hitrostjo, kar omogoča predvsem hiter razvoj strojne opreme. Kljub temu pa še danes obstajajo problemi, ki jih z uporabo naivnih algoritmov ni mogoče učinkovito reševati. Zato na področju računalniške grafike in računalniške geometrije še naprej potekajo raziskave v smeri optimizacije obstoječih algoritmov kot tudi v smeri razvoja novih algoritmov. Velika večina algoritmov se posveča predvsem optimizaciji hitrosti prikazovanja, eden od ključnih pristopov k temu problemu pa je uporaba algoritmov za hitro oceno vidnih in nevidnih področij.

1.2 Smisel raziskave

Večina obstoječih algoritmov, namenjenih določanju vidnosti objektov, se izvaja na centralni procesni enoti (CPE). Problem namreč nastane, pri prenosu podatkov iz pomnilnika grafične procesne enote (GPE), pogosto imenovane tudi grafični pospeševalnik, do pomnilnika CPE. Čas, ki ga potrebujemo za prenos podatkov, je namreč mnogokrat daljši, kot pa sam čas izvajanja algoritma na CPE. Ustrezno prirejen algoritem za uporabo GPE bi delo opravil znatno hitreje. V naši raziskavi se ne lotevamo samo določanja vidnih in nevidnih objektov, temveč želimo imeti natančen podatek o vidnosti tudi za vsako ploskev, ki sestavlja določen objekt. V tem primeru pa se hitro pokažejo prednosti uporabe GPE. Ta pristop naj bi omogočal enostavnejšo implementacijo, prav tako pa naj bi bil učinkovitejši.

Danes so v razvoju tudi že nove CPE enote, ki bodo imele vgrajeno tudi GPE. Takšne enote se imenujejo pospešene procesne enote (PPE) [2]. S tesno integracijo CPE in GPE se

lahko izognemo zamudnemu prenosu podatkov, zato je tudi iz tega stališča smiselno prirediti obstoječe algoritme za odkrivanje vidnosti, da lahko uporabijo kapacitete GPE.

2 OPREDELITEV PROBLEMA

Problem, ki ga želimo rešiti je, da z uporabo grafične procesne enote zaznamo vse vidne ploskve v neki poljubni sceni, ki lahko vsebuje tudi večje število objektov. Pri tem je lahko gledišče postavljeno v poljubno točko.

2.1 Obseg in vsebina podatkov

Podatki, ki jih obravnavamo, so množice 3D geometrijskih objektov, predstavljenih z ovojnico, ki kot skupek tvorijo sceno.

Obseg podatkov, ki sestavljajo celotno sceno, je lahko poljuben. Objekti so sestavljeni iz ploskev, ki so poljubni n-kotniki. Za najboljše rezultate je priporočena uporaba trikotnikov, katerih velikost ne variira pretirano.

Zaradi specifične rešitve problema lahko naenkrat obdelujemo omejeno število ploskev in objektov. Le-ta je 2^x ploskev in 2^y objektov, kjer morata držati predpostavki, da $x + y \leq 40$ in $x \leq 32$. Ta omejitev je podrobneje obrazložena v poglavjih 3.2.3 in 3.2.4.

2.1.1 Testni podatki

Testni podatki, nad katerimi bomo preverjali pravilnost in učinkovitost algoritma, bodo statične scene, katerih objekti so zgrajeni iz trikotnikov. V samo statično sceno bomo dodali nekaj dinamičnih objektov, da bomo preverili pravilno delovanje metode.

2.2 Krajši pregled treh znanih metod reševanja vidnosti

Dandanes lahko na področju zaznavanja vidnih ploskev že najdemo veliko število raznolikih algoritmov, vendar pa večina rešitev, ki ne zahtevajo realnočasovnega delovanja, kot osnovo uporablja postopek metanja žarkov (angl. ray casting). Ta tehnika je počasna in zato ni primerna za uporabo v realnočasovnih aplikacijah, sploh kadar

obravnavamo večje količine podatkov. V nadaljevanju bomo opisali tri rešitve, ki so sicer namenjene prikazovanju vidnih ploskev, vendar jih s pravim pristopom lahko uporabimo tudi za zaznavanje vidnih ploskev.

2.2.1 Slikarjev algoritem

Ploskve izrisujemo tako kot slikar, od tod tudi ime algoritma, saj rišemo od najbolj oddaljene ploskve do najbližje glede na gledišče. Algoritem deluje v naslednjih korakih [3].

- Potrebno je izračunati središča vseh ploskev, pri čemer lahko statične ploskve v celoti predprocesiramo, dinamične pa posodabljamo, kadar se premaknejo.
- Izračunamo oddaljenosti vsakega središča od gledišča.
- Razvrstimo ploskve po oddaljenosti središč.
- Najprej rasteriziramo vse ploskve, ki se nahajajo na največji oddaljenosti. V primeru, da je na tej oddaljenosti več ploskev, rešimo potencialne konflikte v prekrivanjih tako, da ločeno rasteriziramo vsako ploskev po vrsticah. Na mestih kjer se rasterizirani piksli prekrivajo, upoštevamo tiste, ki se nahajajo najbližje gledišču. Ploskve, ki smo jih rasterizirali, odstranimo iz urejenega seznama.
- Vzamemo naslednjo najbolj oddaljeno ploskev oziroma ploskve in ponovimo postopek, opisan v prejšnji točki. Postopek ponavljamo, dokler ne obdelamo vseh ploskev.

Do informacije o vidnosti ploskev pridemo tako, da za vsako rasterizirano ploskev vodimo evidenco, koliko pikslov smo rasterizirali. Pri tem se lahko zgodi, da rasterizacija nove ploskve prekrije piksle katere od predhodno rasteriziranih ploskev. V takem primeru je potrebno ugotoviti, katera ploskev bo prekrita ter njen števec vidnih pikslov primerno zmanjšati, glede na to, koliko njenih pikslov bo prekritih. Ko števec doseže vrednost nič, lahko ploskev zavržemo, saj so jo druge ploskve, ki se nahajajo pred njo, popolnoma prekrile in ploskev ni več vidna. Ko zaključimo rasterizacijo vseh ploskev, sprejmemo kot vidne tiste ploskve, ki imajo vrednosti števcev različne od nič.

2.2.2 Warnockov algoritem

Spada v kategorijo algoritmov deli in vladaj. Algoritem se izvaja tako, da vidni prostor rekurzivno delimo na štiri dele tako dolgo, dokler ni izpolnjen eden od naslednjih pogojev [4]

- V podpodročju se ne nahaja nobena ploskev. Celotno podpodročje zapolnimo z barvo ozadja.
- V podpodročju se nahaja samo ena ploskev. Za hitrejše delovanje uporabimo dva kriterija:
    - Če je ploskev delno ali v celoti vsebovana v podpodročju, podpodročje zapolnimo z barvo ozadja ter rasteriziramo del ploskve, ki se nahaja v podpodročju.
    - Če ploskev obdaja celotno podpodročje, zapolnimo celotno podpodročje z barvo ploskve. V tem primeru se izognemo potrebi po rasterizaciji ploskve.

Če bi upoštevali samo ta kriterija, bi se lahko zgodilo, da bi delitev potekala v neskončnost. Zato potrebujemo dodaten pogoj, s katerim omejimo globino delitve. Delitev tako prekinemo tudi v primeru, ko je podpodročje veliko samo en piksel. V takem primeru je potrebno določiti še najbližjo ploskev med tistimi, ki se nahajajo v tem podpodročju. Najbližja ploskev je seveda tudi tista, ki je edina vidna, zato piksel obarvamo z barvo najbližje ploskve.

Do informacije o vidnosti ploskev pridemo enostavno, saj se rekurzivna delitev področja preneha šele takrat, ko določimo nič ali eno vidno ploskev v podpodročju. Če si za vsak podprostor zabeležimo, katera ploskev je bila vidna, dobimo informacijo o vseh vidnih ploskvah v sceni. Sam algoritem zna biti počasen, saj je vsakič potrebno na novo izvesti rekurzivno delitev prostora oziroma podprostora in ponovno izvesti ustrezne vsebnostne teste. Kot vse algoritme, ki temeljijo na strategiji deli in vladaj, je tudi ta algoritem možno paralelizirati, a smo omejeni na število centralno procesnih enot, ki se nahajajo v računalniškem sistemu.

2.2.3 Metanje žarkov (Ray casting)

Ime algoritma izhaja iz koncepta, da iz gledišča pošiljamo ('mečemo') žarke v smeri gledanja [5]. Za vsak piksel pošljemo en žarek in preverimo, katere ploskve seka ter izračunamo koordinate presečišč. Za presečišče, ki je najbližje gledišču vemo, da leži na vidni ploskvi, zato lahko to ploskev dodamo v seznam vidnih ploskev. Na ta način imam ob koncu izvajanja algoritma (ko pošljemo vse žarke) na voljo informacijo o vseh vidnih ploskvah. Sam algoritem je enostaven za implementacijo, a je zelo časovno zahteven. Možno ga je paralelizirati tako, da razdelimo vidno območje na toliko delov, kot imamo CPE in vsaka CPE obdeluje svoj del slike (pikslov).

3 REŠITEV PROBLEMA VIDNOSTI Z OBELAVO PODATKOV Z GPU

3.1 Inicializacija

V fazi inicializacije preberemo vse podatke ter opravimo predprocesiranje celotne scene. V koraku predprocesiranja sceno preberemo, nato pa vsak objekt razdelimo na manjše enote z uporabo hierarhične delitve prostora, pri čemer si pomagamo z osmiškim drevesom.

3.1.1 Branje podatkov

Omejili smo se na lasten datotečni format, kjer datoteka vsebuje naslednje podatke:

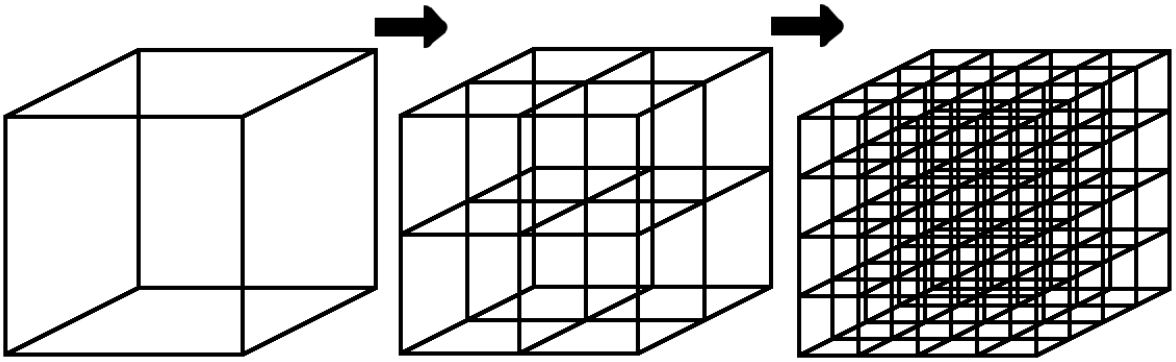
- število vseh točk,
- seznam vseh točk, ki sestavljajo sceno,
- število objektov in
- zapis geometrije objektov, ki so predstavljeni s trikotniško mrežo.

V kolikor dobimo podatke v kakšnem drugem datotečnem formatu, namenjenem zapisu 3D geometrijskih podatkov, ki ga znamo prebrati, preberemo iz originalne datoteke celoten nabor podatkov, nato pa zapišemo samo zgoraj omenjene podatke v novo datoteko, zapisano v opisanem formatu.

3.1.2 Osmiško drevo

Osmiško drevo je struktura, s katero lahko predstavimo hierarhično delitev prostora [6]. Delitev izvedemo tako, da prostor rekurzivno delimo na osem, med seboj enako velikih, podprostorov. Deljenje prenehamo, ko dosežemo neko vnaprej postavljeno omejitev. Tipična omejitev je določitev maksimalnega števila delitev, s čimer omejimo globino drevesa. Z delitvijo prav tako prenehamo, ko se v določenem podprostoru nahaja dovolj majhno število ploskev. Minimalno število ploskev lahko individualno določimo za vsak

objekt, pri čemer upoštevamo tako število ploskev, ki sestavljajo objekt, kot tudi njegovo geometrijo, ali pa vzamemo za celotno sceno neko smiselno postavljeno omejitev. Potrebno je paziti, kako postavimo omejitev, saj lahko zaradi slabo postavljenih omejitev izvedemo premajhno ali pretirano delitev prostora, pri čemer v obeh primerih dobimo slabše rezultate od možnega optimalnega rezultata.



Slika 1 – Prikaz dvakratne osmiške delitve prostora.

Primer osmiške delitve prostora prikazuje slika 1. V tem primeru smo celoten prostor najprej razdelili na osem podprostorov, nato pa smo vsakega od teh podprostorov še enkrat razdelili z osmiško delitvijo. Takšna enakomerna delitev prostora je značilna za osmiško delitev prostora, kadar uporabimo samo omejitev maksimalne delitve prostora, saj bo prostor deljen tudi kadar ne vsebuje geometrijskih podatkov. Če bi upoštevali omejitev za minimalno število ploskev, praznega prostora ne bi delili. V nadaljevanju podrobneje opišemo postopek delitve.

V prvem koraku izračunamo oklepajoč kvader objekta, ki ga s kratico imenujemo tudi AABB (angl. axis aligned bounding box). AABB je najmanjši kvader, ki oklepa celotno geometrijo objekta, katerega robovi so vzporedni z osmi kartezičnega koordinatnega sistema. Zaradi simetrične narave prostora je možno AABB zapisati z dvema vektorjema:

$$P_{center} = (x_{center}, y_{center}, z_{center}),$$

kjer je:

- P_{center} - središče oklepajočega kvadra.

$$P_{razdalje} = (x_{razdalja}, y_{razdalja}, z_{razdalja}),$$

kjer je:

- $P_{razdalje}$ - vektor razdalj od središča oklepajočega kvadra do posameznih stranic prostora.

Dejansko velikost robov oklepajočega kvadra izračunamo tako, da pomnožimo $P_{razdalje}$ s skalarjem 2.

$$P_{velikost} = 2 * P_{razdalje}$$

Osmiško drevo je tipično realizirano kot skupek povezanih vozlišč, izmed katerih lahko ima vsako vozlišče največ osem sinov [7]. Kadar vozlišče nima nobenega sina, ga imenujemo list. V naši rešitvi smo uporabili nekoliko nekonvencionalno implementacijo drevesa, pri kateri dostopamo do sinov vozlišča preko seznama vozlišč, shranjenega v korenskem vozlišču. Predpostavili smo namreč, da bo, zaradi pogoste uporabe, seznam vozlišč večino časa naložen v predpomnilniku procesorja, s čimer se izognemo prenosu podatkov med pomnilnikom in predpomnilnikom.

Korensko vozlišče vsebuje naslednje podatke:

- seznam vseh vozlišč drevesa, preko katerega se vsako vozlišče lahko sklicuje na svoje sinove in
- geometrijo objekta predstavljeno s seznamom trikotnikov.

Vsako vozlišče pa vsebuje naslednje podatke:

- referenco na začetek geometrijskih podatkov, zapisanih v korenskem vozlišču, ki pripadajo temu vozlišču,
- število ploskev v tej geometriji,
- referenco na seznam vozlišč, zapisan v korenskem vozlišču, preko katere lahko dostopamo do sinov vozlišča,
- število sinov, ki jih ima vozlišče,
- P_{center} - sredinsko točko prostora oz podprostora,

- $P_{razdalje}$ - vektor, ki vsebuje razdalje od središča (pod)prostora P_{center} do stranic (pod)prostora in
- r_{sfere} - polmer oklepajoče sfere (pod)prostora.

3.1.3 Postopek hierarhične delitve prostora

Na začetku ustvarimo indeksne reference na našo geometrijo (0…N-1, kjer je N – število ploskev v geometriji). Z referenco 0 označimo prvo ploskev v geometriji z N-1 pa zadnjo. Preden se lahko lotimo delitve, moramo še določiti oklepajoč kvader.

3.1.3.1 Določanje oklepajočega kvadra poljubnega objekta

Da določimo velikost in položaj oklepajočega kvadra (AABB) za poljuben objekt, moramo poiskati dve nasprotno ekstremni točki. To naredimo tako, da preiščemo vse točke, ki sestavljajo objekt, ter shranimo maksimalne in minimalne položaje po vsaki koordinatni osi.

Predpostavimo, da je geometrija objekta sestavljena iz točk:

$$TockeObjekta = (T_0, T_1, T_2, T_3 ... T_{P-1}),$$

kjer je:

- vsaka točka definirana v 3D prostoru: $T_{poljubna} = (x, y, z)$ in
- P - število vseh točk, ki sestavljajo ta objekt.

Z uporabo vseh točk objekta lahko poiščemo ti dve ekstremni vrednosti

$$T_{\min} = (x_{\min}, y_{\min}, z_{\min})$$

$$T_{\max} = (x_{\max}, y_{\max}, z_{\max})$$

Iz ekstremov $T_{\min}$ in $T_{\max}$ izračunamo središče in oddaljenost od središča do stranic oklepajočega kvadra:

$$P_{center} = \frac{T_{\min} + T_{\max}}{2}$$

$$P_{razdalje} = T_{\max} - P_{center} = \frac{T_{\max} - T_{\min}}{2}$$

Iz teh podatkov lahko hitro izračunamo tudi polmer sfere, ki oklepa oklepajoč kvader. Polmer izračunamo tako, da določimo razdaljo od središča P_{center} do enega izmed oglišč:

$$r_{sfere} = \sqrt{(P_{razdalje}x)^2 + (P_{razdalje}y)^2 + (P_{razdalje}z)^2}$$

3.1.3.2 Rekurzivna delitev prostora

Ko poznamo velikost začetnega oklepajočega kvadra, lahko začnemo z rekurzivno delitvijo prostora. Najprej preverimo, ali je že izpolnjena katera od omejitev, ki smo jih opisali v poglavju 3.1.2, saj je možno, da sploh ni potrebno deliti objekta. Podatke o prostoru zapišemo v korensko vozlišče. V kolikor nobena omejitev ni izpolnjena, izvedemo delitev prostora na osem podprostorov. Za to delitev moramo najprej izračunati lastnosti teh podprostorov; torej središča in velikosti podprostorov.

Vektor razdalj, ki ga uporabimo za posreden zapis velikosti oklepajočega kvadra, je za vseh osem podprostorov enak in ga izračunamo tako, da vzamemo vektor razdalj trenutnega prostora in ga prepolovimo.

$$P'_{razdalje} = \frac{P_{razdalje}}{2},$$

kjer je:

- $P'_{razdalje}$ - vektor, ki vsebuje razdalje od središča podprostora do stranic podprostora.

Prav tako lahko na enak način izračunamo tudi velikost omejujočih krogel za vseh osem podprostorov tako, da vzamemo polmer trenutne oklepajoče krogle ter ga prepolovimo:

$$r'_{sfere} = \frac{r_{sfere}}{2},$$

kjer je:

- r'_{sfere} - polmer oklepajoče krogle podprostora.

Preostane nam še izračun središč novih podprostorov. Koordinate središč lahko hitro izračunamo s pomočjo tabele odmikov (tabela 1).

Tabela 1 – Tabela odmikov, ki jo uporabimo za izračun središč podprostorov.

Indeks podprostora	Odmik		
	X	Y	Z
1	-1	1	1
2	1	1	1
3	-1	-1	1
4	1	-1	1
5	-1	1	-1
6	1	1	-1
7	-1	-1	-1
8	1	-1	-1

Središča podprostorov izračunamo tako, da vzamemo izračunan vektor razdalj in ga pomnožimo s pripadajočim odmikom iz tabele 1. Tako dobimo relativni odmik središča podprostora od trenutnega središča. Vse kar še preostane, je da odmik prištejemo k trenutnemu središču:

$$P'_{center\ i} = P_{center} + P'_{razdalje} * Odmik_i,$$

kjer je:

- $P'_{center\ i}$ - središče podprostora i,

- P_{center} - središče podprostora, ki ga delimo,

- $P'_{razdalje}$ - vektor razdalj podprostora,

- $Odmik_i$ - vektor odmika izbran iz tabele 1, glede na indeks podprostora i in

- i indeks podprostora, pri čemer velja $1 \leq i \leq 8$.

Sedaj, ko imamo določene velikosti vseh podprostorov, se lahko lotimo delitve geometrije objekta. Za vsakega od osmih podprostorov izračunamo minimalno in maksimalno točko. Ti točki uporabimo, da preverimo, katere ploskve, v našem primeru trikotniki, so vsebovani v testiranem podprostoru. Testiranje izvedemo z uporabo 3D inačice algoritma Cohen-Sutherland [8]. V primeru, da se trikotnik nahaja v podprostoru, ločimo dve možnosti; trikotnik je lahko popolnoma vsebovan v testiranem podprostoru ali pa se znotraj podprostora nahaja le del trikotnika. Algoritem Cohen-Suterland, ki ga uporabimo za ločevanje takih primerov, je podrobneje opisan v nadaljevanju. Kot smo že omenili, potrebuje algoritem kot vhodna podatka minimalno in maksimalno točko podprostora:

$$T_{\min} i = P'_{center\ i} - P'_{razdalje},$$

$$T_{\max} i = P'_{center\ i} + P'_{razdalje},$$

kjer je:

- $T_{\min} i$ minimalna točka v podprostoru i,

- $T_{\max} i$ maksimalna točka v podprostoru i.

Algoritem Cohen-Sutherland inicializiramo z našo minimalno in maksimalno točko podprostora ter za vsakega od trikotnikov iz prejšnjega prostora preverimo, ali je vsebovan v tem podprostoru.

Ker je Cohen-Sutherland namenjen preverjanju vsebnosti daljic, je potrebno trikotnik:

$$Trikotnik = (T_1, T_2, T_3),$$

razbiti na tri daljice:

$$Daljica_1 = (T_1, T_2),$$

$$Daljica_2 = (T_2, T_3),$$

$$Daljica_3 = (T_3, T_1).$$

Kadar so vse tri daljice v celoti vsebovane v podprostoru, potem je tudi trikotnik v celoti vsebovan. V kolikor pa katera od daljic sega izven podprostora, potem vemo, da tudi trikotnik sega izven tega podprostora.

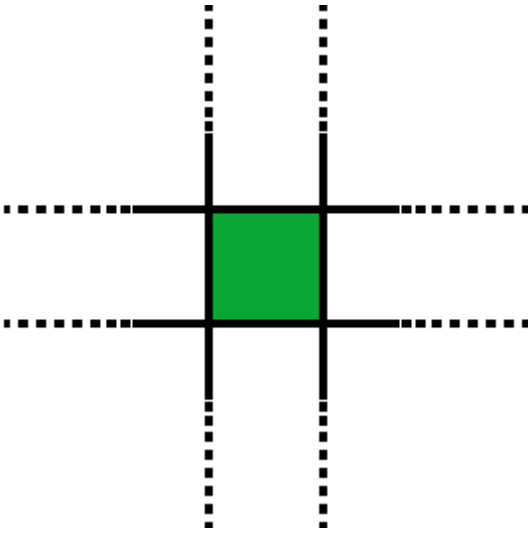
Po izvedbi algoritma Cohen-Sutherland imamo za vsakega od osmih novih podprostorov določeno, kateri trikotniki se nahajajo v njem. Opisan postopek nato rekurzivno ponovimo za vsak podprostor.

3.1.4 Algoritem Cohen-Sutherland

Osnovni algoritem je namenjen delu v 2D prostoru, a ga je preprosto razširiti v poljubno višjo dimenzijo. Mi smo ga razširili v 3D. Za lažje razumevanje algoritma, bomo najprej opisali algoritem v 2D, nato pa opišemo še spremembe, ki so potrebne za razširitev v 3D.

3.1.4.1 Cohen-Sutherland 2D

Algoritem se uporablja za obrezovanje daljic ali za preverjevanje vsebnosti daljic v omejenem 2D prostoru [9]. Ta omejen prostor, s katerim preverjamo, ali je daljica vsebovana, obkrožuje 8 neskončno velikih prostorov, kot je prikazano na sliki 2.

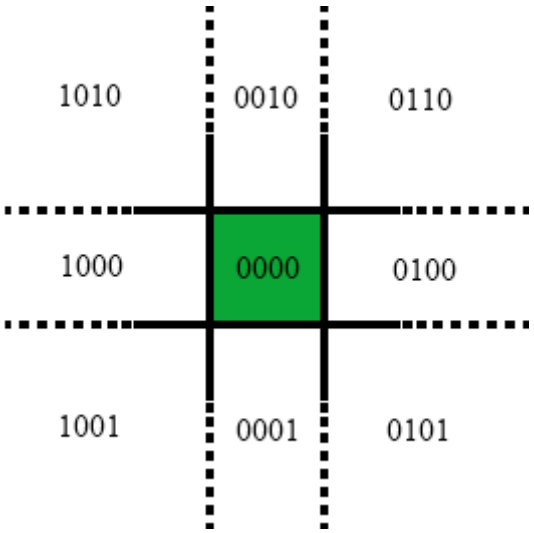


Slika 2 – Zeleni omejen 2D prostor obdaja osem neskončno velikih 2D prostorov.

Vsakemu od teh prostorov določimo bitno kodo glede na njegov položaj, relativno na srednji prostor. Način določitve bitnih kod prikazujeta slika 3 in tabela 2.

Tabela 2 – Bitne kode položajev

Bitna koda	Položaj
0000	center
0001	spodaj
0010	zgoraj
0100	desno
1000	levo



Slika 3 – Grafična predstavitev določitev bitnih kod.

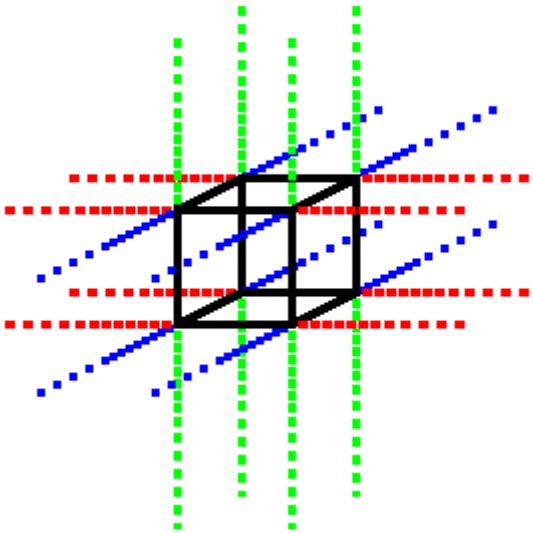
V inicializaciji algoritma določimo dve točki, ki tvorita aktivni prostor 0000. Vsaka od daljic, ki jih preverjamo, je sestavljena iz dveh točk (P_1, P_2). Točkama določimo dve bitni kodi B_1 in B_2 glede na njun položaj relativno na aktivni prostor. S kodama B_1 in B_2 preverjamo vsebnost daljice v aktivnem prostoru po naslednjih kriterijih:

1. Logični ***ali*** med B_1 in B_2 je enak 0000, kar implicira, da sta obe točki znotraj aktivnega prostora, saj morata B_1 in B_2 imeti bitno kodo 0000. S tem vemo, da je znotraj aktivnega prostora celotna daljica.
2. Logični ***ali*** med B_1 in B_2 ni enak 0000, je pa ena izmed teh dveh bitnih kod enaka 0000. To implicira, da je samo ena točka vsebovana v aktivnem prostoru, druga pa je v enem izmed preostalih osmih prostorov. Če je samo ena točka daljice vsebovana v aktivnem prostoru, je daljica delno vsebovana.
3. Logičen ***in*** med B_1 in B_2 je različen od 0000. V tem primeru vemo, da daljica ne seka aktivnega prostora.

4. Logičen ***in*** med B_1 in B_2 je enak 0000. V tem primeru obstaja možnost, da daljica seka aktivni prostor. Daljico je potrebno krčiti tako, da izberemo eno izmed dveh točk (P_1, P_2), ter jo premikamo po daljici proti drugi točki, dokler premikajoča točka ne pade v aktivni prostor, s čimer je izpolnjen drugi kriterij oziroma, dokler premikajoča se točka ne pade v tisti prostor, v katerem se nahaja druga točka, s čimer je izpolnjen tretji kriterij.

3.1.4.2 Cohen-Sutherland 3D

Da razširimo algoritem v 3D, potrebujemo dodatno dimenzijo (globino) oziroma os (z os), kar prikazuje slika 4.



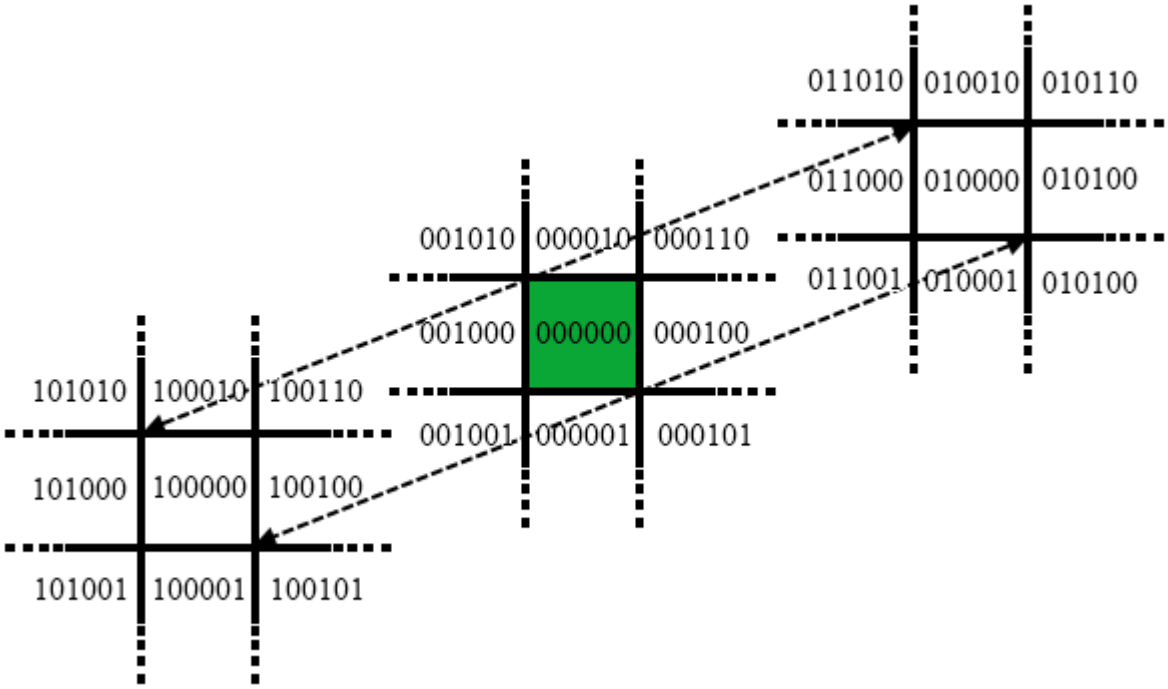
Slika 4 - Črno kocko obdaja šestindvajset neskončno velikih 3D prostorov.

Prav tako moramo tudi v bitno kodo dodati dva nova položaja. Razširitev tabele bitnih kod za uporabo v 3D prostoru prikazuje tabela 3.

Tabela 3 – Bitne kode položajev v 3D prostoru

Bitna koda	Položaj
000000	center
000001	spodaj
000010	zgoraj
000100	desno
001000	levo
010000	zadaj
100000	spredaj

Bitne kode ponovno določimo vsakemu 3D prostoru glede na njegov relativen položaj na srednjo celico, kot prikazuje slika 5.



Slika 5 – Grafična predstavitev določitve bitnih kod za algoritem Cohen-Sutheland v 3D prostoru.

Tudi sedaj je potrebno inicializirati algoritem, le da tu uporabimo dve 3D točki, ki bosta tvorili naš aktivni prostor. Kot pri 2D algoritmu vsako daljico obravnavamo kot dve točki, katerima določimo dve bitni kodi, glede na to, v katerem od sedemindvajsetih prostorov se nahajata. Da preverimo ali je daljica vsebovana v aktivnem prostoru, uporabimo iste štiri kriterije kot pri 2D algoritmu, le da nadomestimo primerjave z bitno kodo 000000.

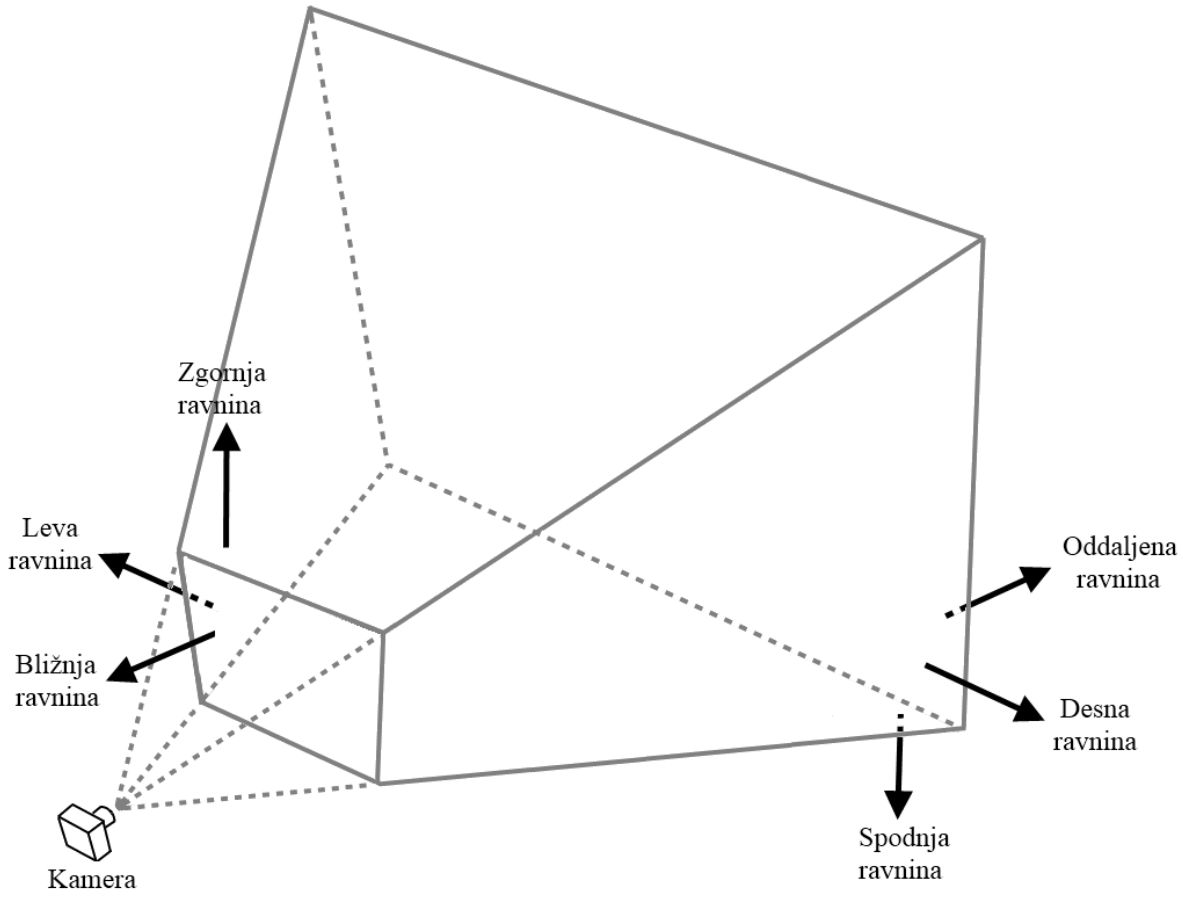
3.2 Postopek zaznavanja vidnih ploskev

Šele po opravljenem predprocesiranju se lahko lotimo analize vidnosti posameznih ploskev v sceni. Najprej izločimo vse objekte, ki se nahajajo izven vidnega področja, saj lahko s tem znatno pospešimo sam postopek. Nato je potrebno vsaki ploskvi določiti unikatno barvo, da lahko kasneje, v fazi analize vidnosti, že samo iz barve določenega piksla razberemo, katero ploskev predstavlja. Te, potencialno vidne, ploskve pošljemo na grafično enoto, ki jih rasterizira. Po opravljeni rasterizaciji prenesemo dobljeno 2D sliko iz pomnilnika GPE nazaj v delovni pomnilnik. Iz te slike lahko sedaj pridobimo podatke o vidnosti ploskev. V nadaljevanju vsakega od teh korakov opišemo podrobneje.

3.2.1 Izločanje objektov, ki se nahajajo izven vidnega področja

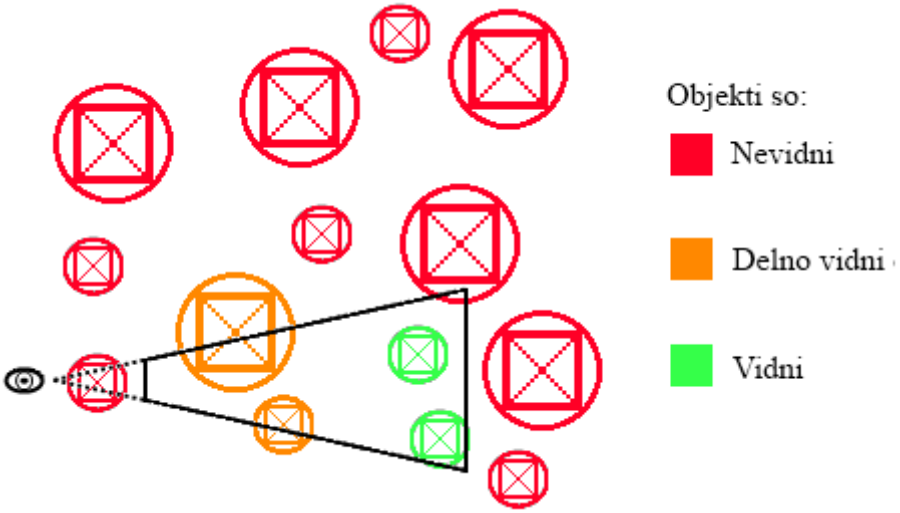
Da lahko opravimo čim hitrejšo analizo vidnosti, je smiselno, da poiščemo objekte, ki se ne nahajajo v vidnem področju kamere. Tako lahko že vnaprej hitro izločimo veliko število objektov. V našem primeru uporabljamo perspektivno projekcijo, zato vidno področje predstavlja prisekana štiristrana piramida, ki jo prikazuje slika 6. To piramido določa šest ravnin:

- bližnja ravnina (angl. near plane),
- leva ravnina (angl. left plane),
- desna ravnina (angl. right plane),
- zgornja ravnina (angl. up plane),
- spodnja ravnina (angl. bottom plane),
- oddaljena ravnina (angl. far plane).



Slika 6 – Prisekana štiristrana piramida, ki predstavlja vidno področje pri perspektivni projekciji.

Takšno vidno polje ni omejeno na isti način kot človeški vid, saj obstajata dve dodatni ravnini, ki na nenaraven način omejujeta pogled (bližnja in oddaljena ravnina). Vse kar se nahaja med bližnjo ravnino in gledišče, obravnavamo kot nevidno, čeprav vsekakor je v vidni poti našega pogleda v vidni prostor. Da to težavo vsaj delno odstranimo, izberemo čim manjšo razdaljo do bližnje ravnine. Oddaljena ravnina pa omejuje naš pogled v daljavo. Kot posledica te omejitve nam lahko objekti v sceni izginjajo, ko se dovolj oddaljijo od gledišča, če razdalja do te ravnine ni dovolj velika. Na sliki 7 vidimo shematski prikaz rezultatov postopka zaznavanja objektov, ki se nahajajo v vidnem polju. Na sliki lahko hitro opazimo tudi tiste objekte, ki jih je postopek napačno določil kot nevidne zaradi neustrezno nastavljene oddaljenosti bližnje in oddaljene ravnine.



Slika 7 - Določitev vidnosti objektov glede na vidno polje.

Da preverimo ali so objekti v vidnem polju, moramo zapisati šest ravnin vidnega polja v vektorski obliki.

Zapis enačbe ploskve je v splošni obliki:

$$Ax + By + Cz = D,$$

kjer so:

- A, B, in C komponente normalnega vektorja ravnine glede na os x, y in z in
- D pravokotna razdalja od ploskve do gledišča.

Enačbo lahko zapišemo tudi z vektorji, tako da uporabimo:

$n = (A, B, C)$ ter ločeno vrednost D,

kjer je:

- n normalni vektor ravnine.

Ko poznamo vseh šest normalnih vektorjev ter oddaljenosti ploskev, lahko začnemo preverjati vidnost objektov. Najprej izvedemo vsebnostni test z uporabo omejujočih krogel. S tem testom lahko odkrijemo večino objektov, ki so v celoti izven vidnega prostora ter vse tiste, ki so v celoti znotraj vidnega prostora. Ta vsebnostni test izvedemo najprej, ker je izjemno hiter, hkrati pa je v večini primerov izven vidnega polja mnogo večje število

objektov kot v njem. V posameznih primerih se nam lahko zgodi, da na osnovi prvega testa ne moremo vedeti ali se objekt nahaja znotraj ali zunaj vidnega prostora. V takem primeru izvedemo drugi test, pri katerem preverjamo vsebnost oglišč oklepajočega kvadra v vidnem prostoru.

V nadaljevanju oba testa opišemo podrobneje.

1. Preverjanje z uporabo omejujočih krogel

Najprej za vsako ravnino vidnega prostora izračunamo skalarni produkt med normalo ravnine in vektorjem, ki predstavlja položaj krogle, ter rezultatu prištejemo odmik D.

$$s = (n_{ravnine} \cdot P_{center}) + D_{ravnine}$$

Glede na dobljen rezultat ločimo tri možnosti:

a) $s \leq -r_{sfere}$

Kadar je rezultat skalarnega produkta manjši ali enak negativnemu polmeru krogle, se krogla nahaja izven vidnega področja. Tak rezultat namreč pomeni, da se samo središče nahaja na napačni strani ravnine, ki jo trenutno preverjamo, in je preveč oddaljeno od te ravnine, da bi jo krogla lahko sekala. V tem primeru lahko preskočimo vse nadaljnje teste.

b) $-r_{sfere} < s < r_{sfere}$

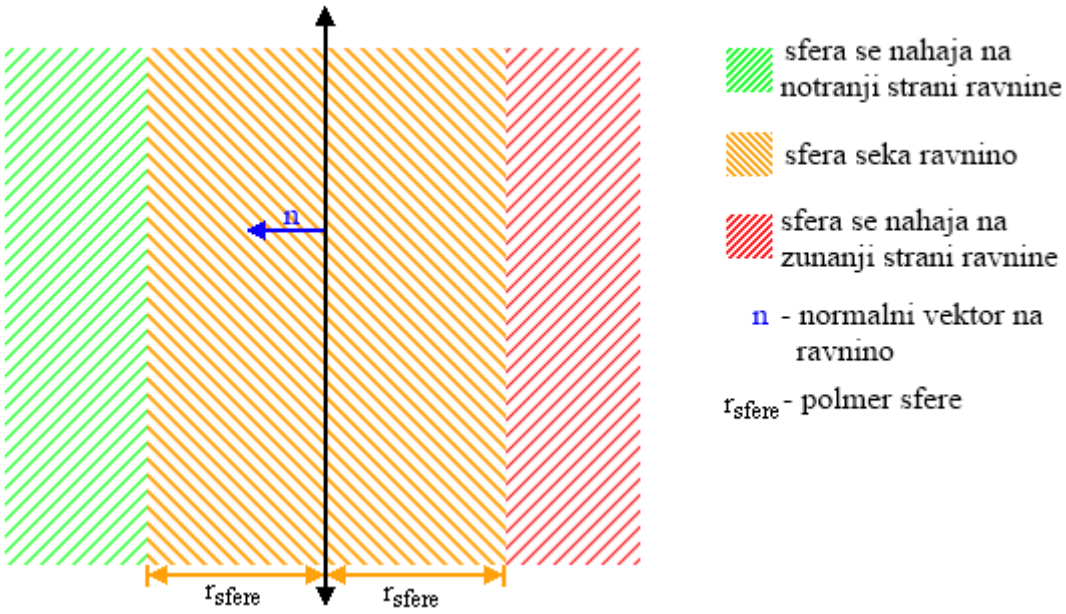
V primeru, da je izračunana vrednost med negativnim in pozitivnim polmerom krogle vemo, da krogla seka ravnino, torej se del krogle nahaja znotraj vidnega prostora.

c) $r_{sfere} \leq s$

Kadar je izračunana vrednost večja ali enaka polmeru, pa vemo le, da se središče krogle nahaja na pravilni strani ravnine, ne vemo pa, ali je v samem vidnem prostoru.

Tri območja, ki predstavljajo možnosti a, b in c, prikazuje slika 8. Če s predhodnimi testiranji nismo mogli določiti, da se krogla v celoti nahaja izven vidnega polja, nam preostaneta dve možnosti. V kolikor je vseh šest testov vrnilo rezultat c, se krogla v

celoti nahaja znotraj vidnega polja, s tem pa vemo, da se znotraj vidnega polja nahaja tudi celoten objekt. V nasprotnem primeru pa se znotraj vidnega polja nahaja samo del krogle, zato moramo za natančno določitev vidnosti izvesti še nadaljnja preverjanja.



Slika 8 - Območja v okolici ravnine

2. Preverjanje z uporabo oklepajočih kvadrov

Če smo pri testiranju oklepajoče krogle ugotovili, da se le-ta v celoti nahaja znotraj vidnega področja, potem vemo, da je znotraj vidnega območja tudi oklepajoč kvader, zato lahko ta test izpustimo. V primeru, da je bil rezultat katerega od preverjanj vsebnosti oklepajoče krogle, da krogla seka ravnino vidnega področja, pa je možno, da je oklepajoč kvader v celoti znotraj, delno znotraj ali pa v celoti izven vidnega prostora. Zato moramo izvesti vsebnosti test za vsako od osmih oglišč oklepajočega kvadra. Za vsako od teh oglišč moramo preveriti, na kateri strani vsake od šestih ravnin, ki predstavljajo vidni prostor, leži. Za izračun koordinat vseh osmih oglišč oklepajočega kvadra si lahko ponovno pomagamo s tabelo odmikov (tabela 4).

Tabela 4 – Tabela odmikov, ki jo uporabimo za izračun koordinat oglišč oklepajočega kvadra.

Indeks oglišča	Odmik		
	X	Y	Z
1	-1	1	1
2	1	1	1
3	-1	-1	1
4	1	-1	1
5	-1	1	-1
6	1	1	-1
7	-1	-1	-1
8	1	-1	-1

Oglišče O_i izračunamo tako, da vektor razdalj pomnožimo z *i*-tim vektorjem odmika v tabeli 4 ter rezultat prištejemo k vektorju, ki predstavlja središče prostora.

$$O_i = P_{center} + P_{razdalje} * Odmik_i ,$$

kjer je :

- $Odmik_i$ - *i*-ti vektor odmika iz tabele 4.

Da preverimo, na kateri strani ravnine leži, uporabimo naslednjo enačbo:

$$s = (n_{ravnine} \cdot O_i) + D_{ravnine}$$

Na kateri strani ravnine se nahaja oglišče, preverimo tako, da preverimo predznak vrednosti s. Če je vrednost pozitivna, se oglišče nahaja na notranji strani ravnine, če je vrednost negativna, pa na zunanji strani. Samo kadar se vseh osem oglišč nahaja na zunanji strani ravnine lahko zagotovo rečemo, da celoten oklepajoč kvader leži izven vidnega prostora. Za popolno preverjanje moramo izvesti oseminštirideset testov. Samo če vsak od teh oseminštiridesetih testov vrne podatek, da se oglišče nahaja znotraj vidnega prostora, lahko sklepamo, da se celoten oklepajoč kvader nahaja znotraj

vidnega prostora. Kadar pa ugotovimo, da se del oglišč nahaja zunaj vidnega področja del pa znotraj, pa oklepajoč kvader seka vidni prostor.

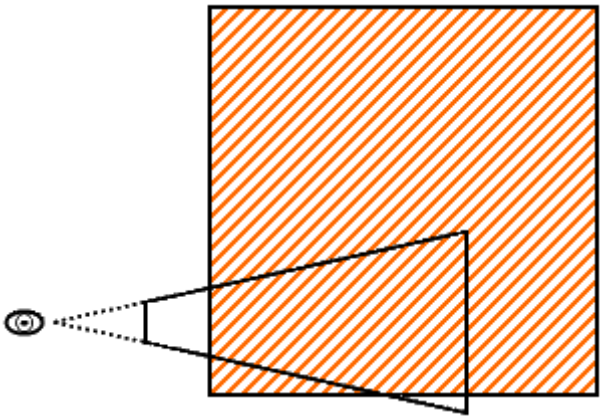
Po opravljenem vsebnostnem testu z oklepajočimi kvadri vemo, kateri objekti so znotraj vidnega prostora, kateri zunaj in kateri ga sekajo.

3.2.2 Postopek odstranjevanja ploskev, ki se nahajajo izven vidnega prostora.

Če smo za neki objekt ugotovili, da njegov oklepajoč kvader seka vidni prostor, lahko sklepamo, da je del tega objekta znotraj vidnega prostora del pa zunaj. Lahko se sicer zgodi, da se celoten objekt nahaja zunaj ali znotraj vidnega prostora, a nam takšnih primerov ni potrebno obravnavati ločeno. Kadar torej ugotovimo, da oklepajoč kvader seka vidni prostor, lahko uberemo dva postopka:

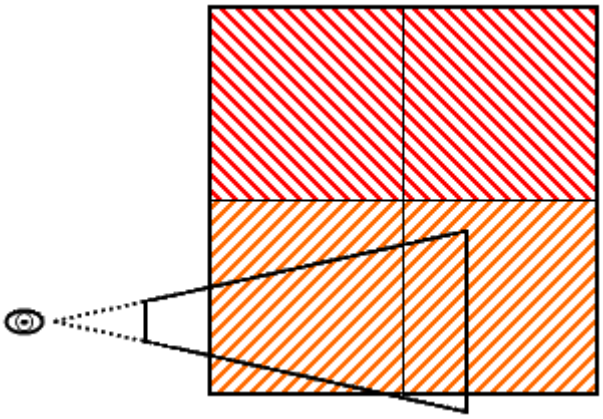
- Objekt privzamemo kot viden, kar pomeni, da bomo v nadaljnji fazi izrisali celotno geometrijo objekta, tudi če se del te geometrije nahaja izven vidnega prostora.
- Če gre za objekt z velikim številom ploskev, je smiselno, da poskusimo pred nadaljnjo fazo odstraniti tudi čim večje število ploskev, ki se v celoti nahajajo izven vidnega področja. Za tak korak uporabimo osmiško drevo, ki smo jih generirali v fazi predprocesiranja. Korensko vozlišče osmiškega drevesa predstavlja celoten objekt, zato ga v tej fazi ne obravnavamo ponovno. Zato pa se sprehodimo čez vse njegove sinove in za vsakega sina izvedemo enake vsebnostne teste, kot smo jih izvajali nad objekti. Najprej torej izvedemo preverjanje z oklepajočo kroglo (gre za oklepajočo kroglo podprostora, ki ga predstavlja vozliče), šele nato pa se, po potrebi, lotimo preverjanja z uporabo oklepajočega kvadra. Vsa ta preverjanja se sedaj izvedejo s primerno manjšimi oklepajočimi kvadri in sferami, ki smo jih izračunali v fazi predprocesiranja. Preverjanje izvajamo rekurzivno, dokler ne preverimo celotnega osmiškega drevesa. V primeru, da se nahajamo v listu in ugotovimo, da oklepajoč kvader seka vidni prostor, privzamemo, da je podprostor, ki ga predstavlja vozlišče, viden. S tem pa seveda privzamemo tudi, da so vidne vse ploskve, ki se nahajajo v tem podprostoru in jih zato pošljemo v nadaljnje procesiranje.

Na slikah 9, 10, 11 in 12 je prikazan zgled opisanega preverjanja vidnosti posameznih delov objekta. Za lažjo predstavo smo zgled naredili v 2D in namesto osmiškega drevesa uporabili štiriško drevo. Pri tem maksimalno globino drevesa omejimo na 4 nivoje. Prav tako smo v primeru prikazali samo preverjanje vsebnosti z uporabo oklepajočih kvadrov.



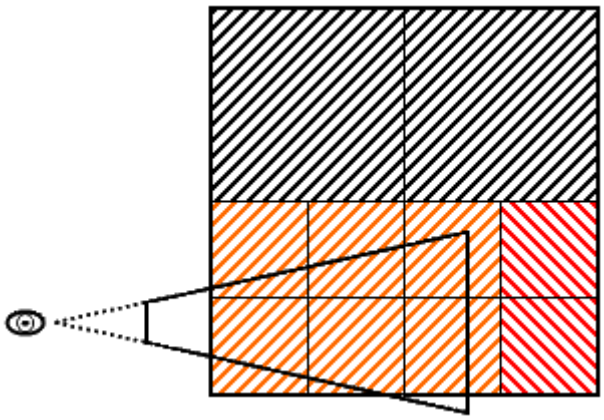
Črtast pravokotnik predstavlja oklepajoč kvader nekega objekta. Del kvadra je znotraj vidnega prostora, del pa zunaj, zato moramo nadaljevati s preverjanjem na naslednjem nivoju

Slika 9 – Preverjanje vidnosti objekta s preverjanjem oklepajočega kvadra na prvem nivoju.



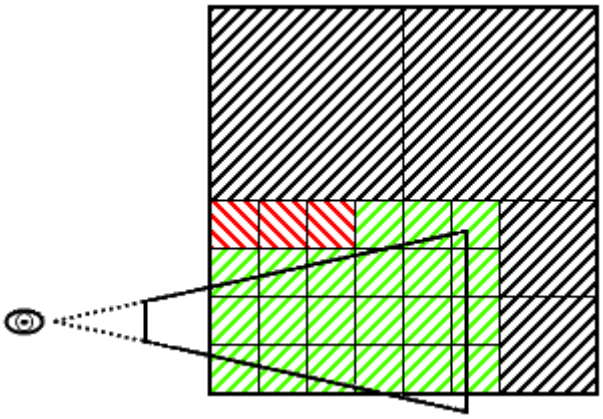
Na tem nivoju najdemo dva podprostora, ki se nahajata izven vidnega prostora in dva ki ga sekata. Za ta podprostora moramo postopek ponoviti na naslednjem nivoju.

Slika 10 – Preverjanje vidnosti objekta s preverjanjem oklepajočih kvadrov na drugem nivoju.



Polovico podprostorov smo izločili že v prejšnjem preverjanju in jih sedaj ne preverjamo več. V tem koraku izločimo še dva dodatna prostora ter se spustimo na končen, četrti, nivo.

Slika 11 - Preverjanje vidnosti objekta s preverjanjem oklepajočih kvadrov na tretjem nivoju.



Sedaj lahko izločimo še tri podprostore, ki so obarvani z rdečo. Ker smo dosegli zadnji nivo, označimo vse preostale podprostore kot v celoti vidne, čeprav nekateri sekajo vidni prostor.

Slika 12 - Preverjanje vidnosti objekta s preverjanjem oklepajočih kvadrov na četrtem nivoju.

3.2.3 Analiza vidnosti

V predhodnih korakih smo izločili večino tistih ploskev, ki se v celoti nahajajo izven vidnega področja. Sedaj bomo izvedli postopek analize vidnosti še nad tistimi ploskvami, ki se nahajajo znotraj vidnega področja. Za to analizo smo razvili lasten postopek, ki učinkovito izrablja zmožnosti modernih grafičnih procesnih enot in je zaradi tega tudi izredno hiter.

Osnovna ideja celotnega postopka je enostavna in jo sestavljajo naslednji koraki:

- rasterizacija vsake ploskve z unikatno barvo,
- branje rasterizirane 2D slike iz pomnilnika GPE in
- sprehod skozi piksle v dobljeni sliki.

Ker vsako ploskev obarvamo z unikatno barvo, lahko iz barve piksla razberemo, katero ploskev predstavlja. S pomočjo tega podatka za vsako ploskev ugotovimo, ali je vidna ali ne.

Metoda, ki smo jo razvili, temelji na izkoriščanju barvnega (angl. color buffer) in šablonskega medpomnilnika (angl. stencil buffer). Skupna uporaba teh medpomnilnikov nam omogoča zapis informacij kot 40 bitnih vrednosti. 32 bitov nam je na voljo v barvnem medpomnilniku, 8 bitov pa v šablonskem medpomnilniku. Metoda prav tako uporablja tudi z-medpomnilnik, ki samodejno opravi globinsko urejanje geometrije v fazi rasterizacije objektov. Vsi ti medpomnilniki so na voljo na vseh modernih GPE.

3.2.4 Rasterizacija

Ko je scena pripravljena za izris, se celotna geometrija transformira iz 3D prostora v 2D rastrsko sliko, ki se nahaja v grafičnem pomnilniku. Zaradi omejene količine prostora za shranjevanje vrednosti 'pikslov', 40 bitov, se nam prejšni postopek (postopek izločanja objektov in ploskev izven vidnega področja) izkaže v korist, saj se ta omejitev nato nanaša na geometrijo v vidnem prostoru in ne na celotno sceno. Ker ponavadi vidimo samo del celotne scene, je možno uporabiti ta postopek tudi na zelo velikih scenah.

Vrednost, ki jo hočemo shraniti v vsak piksel, je sestavljena iz dveh vrednosti:

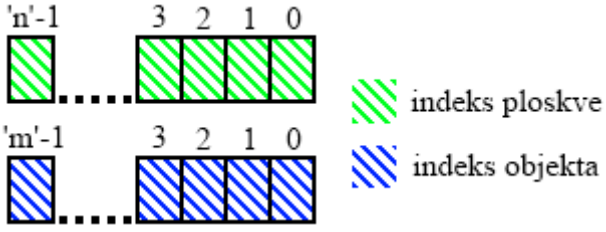
- indeks ploskve (0...N-1, kjer je N število ploskev za posamezen objekt) in

- indeks objekta, ki mu pripada ta ploskev (0...M-1, kjer je M število objektov v sceni).

Zaradi omejene količine prostora je potrebno izbrati tudi učinkovit sistem shrambe teh dveh različnih vrednosti. Če vnaprej poznamo maksimalno število objektov in ploskev, ki se bodo pojavile v scenah, lahko uporabimo fiksno delitev zapisa, kjer imamo X bitov rezerviranih za zapis indeksa ploskve ter Y bitov namenjenih za zapis indeksa objekta, kjer pa mora držati omejitev, da je seštevek X in Y manjši ali enak 40. To situacijo je nemogoče zagotoviti v nekontroliranem okolju, kjer dovolimo nalaganje poljubnih scen. V takem primeru je bolj smiselno poskusiti uporabiti dinamično delitev zapisa. V dinamičnih scenah lahko namreč pričakujemo, da bo število vidnih objektov ter vidnih ploskev variiralo zaradi uporabe vidnostnih testov.

Za dinamično delitev zapisa moramo pred fazo izrisa vedeti, koliko objektov bomo poslali v obdelavo GPE ter kateri od teh objektov vsebuje največ ploskev. Z uporabo teh dveh informacij je možno ustvariti dinamično delitev za vsako posamezno rasterizacijo. Sam postopek izvedemo na naslednji način:

1. Določimo število M', ki predstavlja koliko objektov se bo rasteriziralo.
2. Določimo, kateri od teh M' objektov vsebuje največ ploskev – N' ploskev.
3. Izračunamo, koliko bitov je potrebnih, da shranimo števili N' (n bitov) ter M' (m bitov).
4. S pomočjo vrednosti 'n' in 'm' določimo, kako naj bo zapis razdeljen.



Slika 13 – Posamezni velikosti vrednosti

Ko imamo natančno določen format zapisa, lahko indeks objekta in indeks ploskve (slika 13) združimo v eno samo 40-bitno vrednost. Vse kar potrebujemo, je začasna 64 bitna vrednost, v kateri združimo ta indeksa tako, da indeks ploskve pustimo takšen kot je, indeks objekta pa bitno 'n'-krat zamaknemo (angl. bit shift) v levo, kar ga prestavi v levo

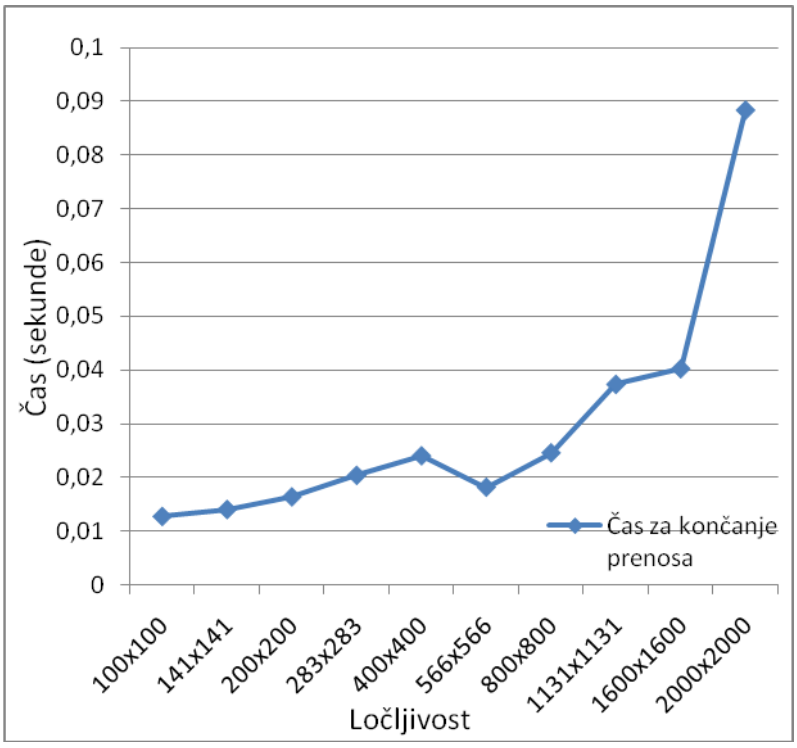
stran ravno dovolj, da ju lahko združimo z logičnim ***ali*** operatorjem, brez da bi se vrednosti prekrivali.

$$ZdruženaVrednost = IndeksPloskve \,|\, (IndeksObjekta << n)$$

Sedaj, ko lahko vsaki ploskvi določimo unikatno vrednost, znamo iz slike, ki jo bo izrisala GPE, razbrati, katere od ploskev, ki smo jih poslali v postopek upodabljanja, so dejansko vidne in katere ne.

3.2.5 Prenos podatkov

Po končani rasterizaciji je potrebno prenesti vrednosti, shranjene v barvnem in šablonskem medpomnilniku, v delovni pomnilnik, kjer jih lahko obdelujemo kot 2D sliko. Sam prenos podatkov poteka preko sistemskega vodila in je zaradi tega tudi najpočasnejša operacija in lahko znatno vpliva na učinkovitost metode. Pri večjih ločljivostih rasterizacije je potrebno prenesti več podatkov, zato je čas trajanja prenosa podatkov še znatno daljši, saj povečanje ločljivosti za X -krat, poveča količino podatkov za X^2 -krat.



Slika 14 – Čas prenosa ob večanju ločljivosti.

Slika 14 prikazuje meritve časa prenosa podatkov iz pomnilnika GPE v delovni pomnilnik pri različnih ločljivostih. V grafu se ločljivost povečuje za faktor $\sqrt{2}$ -krat glede na

prejšnjo ločljivost, število pikslov pa se posledično poveča za faktor 2-krat ($X = \sqrt{2}; X^2 = 2$). Graf se ujema z našo napovedjo kvadratne rasti, saj začne čas hitro rasti ob vedno večjih ločljivostih.

3.2.6 Analiza podatkov

Po prenosu slike moramo celotno sliko pregledati ter za vsak piksel opraviti inverzno transformacijo vrednosti v indeks ploskve in indeks objekta. Da iz 40 bitne vrednosti izluščimo ta indeksa, potrebujemo dve maski.

Slika 15 - Združeni vrednosti je potrebno ločiti.

Na sliki 15 lahko vidimo, da je možno prebrati indeks ploskve tako, da sestavimo masko, katera izreže prvih 'n' bitov oziroma zavrže vse bite po 'n'-tem, za odmik objekta pa potrebujemo ravno obratno masko.

Da ustvarimo maski, ustvarimo vrednost, kjer je prvih 40 bitov postavljenih na 1, na katero se bomo sklicevali kot BaznaMaska.

- V binarni obliki: 0000 0000 0000 0001 1111 1111 1111 1111 1111 1111 1111 1111

- V heksadecimalni obliki: 00 01 FF FF FF FF

S pomočjo bazne maske ustvarimo masko za indeks ploskve tako, da odstranimo 'm' oziroma $40 - 'n'$ bitnih enic s pomočjo bitnih zamikov v desno

$$PloskevM = BaznaMaska >> (40 - 'n'),$$

kjer je:

- PloskevM - bitna maska uporabljena za branje indeksa ploskve in
- operacija >> - bitni zamik v desno.

Za branje indeksa objekta uporabimo inverz zgornje maske. Iz tako dobljene maske moramo potem dodatno odstraniti še zadnjih 25 enic, ki se pojavijo kot posledica invertiranja.

$$ObjektM = (\sim PloskevM) \,\&\, BaznaMaska,$$

kjer je:

- ObjektM bitna maska uporabljena za branje indeksa objekta,
- ~ PloskevM – inverz bitne maske PloskevM in
- & BaznaMaska – odstrani zadnjih 25 enic s pomočjo logičnega ***in*** ter bazne maske.

Da izluščimo indeks ploskve z logičnim ***in***, združimo skupno vrednost ter masko za indeks ploskve:

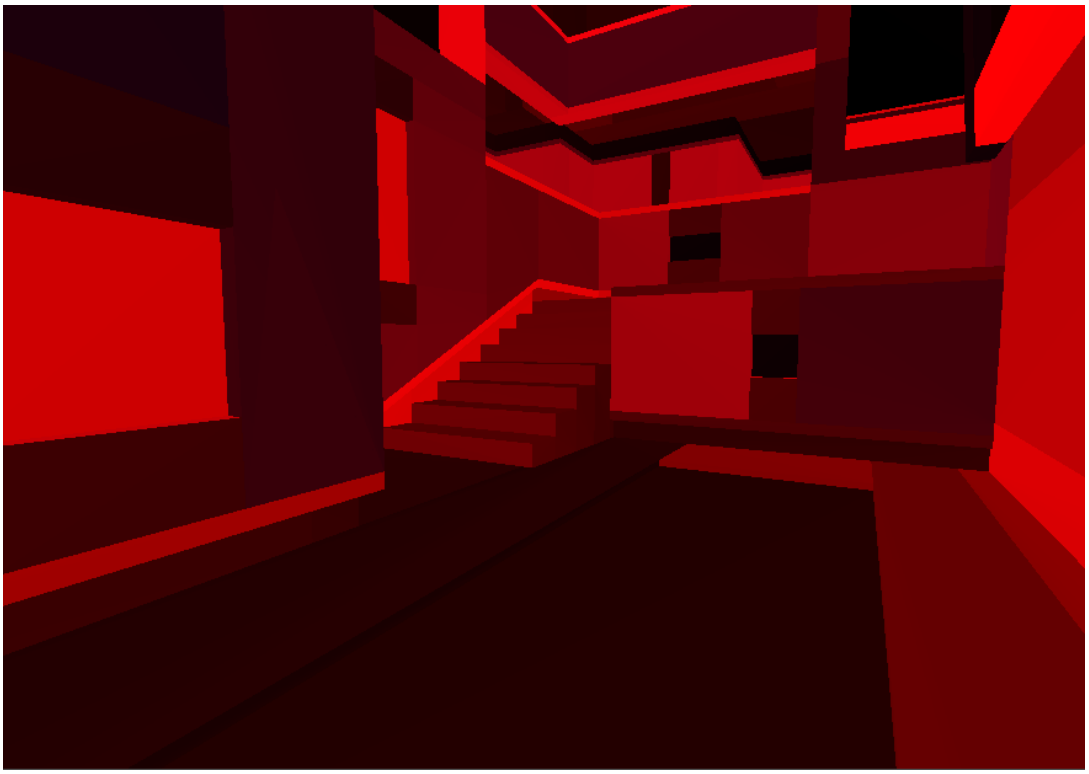
$$IndeksPloskve = SkupnaVrednost \,\&\, PloskevM$$

Da pa izluščimo indeks objekta, moramo z logičnim ***in*** združiti skupno vrednost in masko za indeks objekta, nato pa dobljeno vrednost bitno zamakniti v desno za 'n' mest:

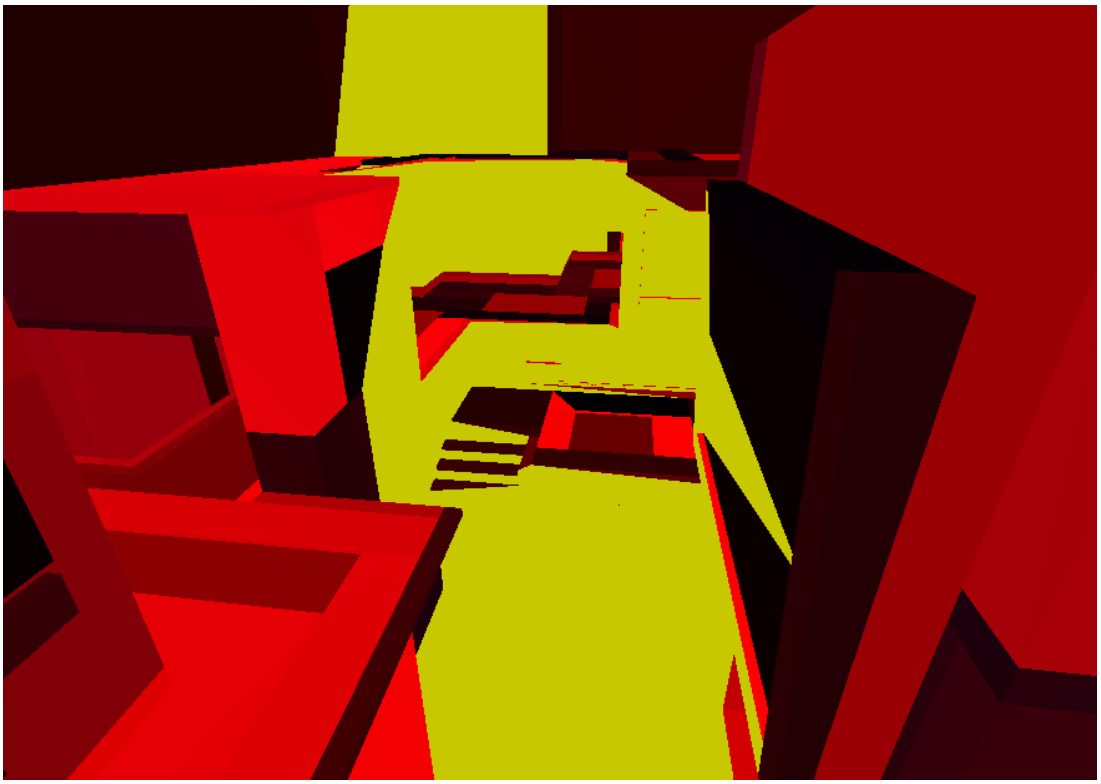
$$IndeksObjekta = (SkupnaVrednost \,\&\, ObjektM) >> 'n'$$

Iz dobljenih indeksov lahko določimo, katera ploskev je bila rasterizirana v ta piksel ter kateremu objektu ta ploskev pripada. Enak postopek izvedemo za vse piksle v sliki. Po končanem postopku dobimo kot rezultat množico vidnih ploskev.

Rezultat opisanega postopka prikazujeta sliki 16 in 17. Na sliki 16 je prikazana scena, na kateri bomo izvedli analizo vidnosti. Na sliki 17 je prikazana ista scena po opravljeni analizi vidnosti z nekoliko drugače postavljenim gledišče m. Ploskve, ki so obarvane rumeno, so tiste ploskve, ki jih je postopek zaznavanja vidnih ploskev zaznal kot vidne.



Slika 16 - Poljubna lokacija znotraj scene, kjer želimo analizirati vidnost.



Slika 17 - Pogled na sceno iz ptičje perspektive po analizi vidnosti.

4 OCENITEV SISTEMA

Implementiran sistem zmore obdelati veliko količino geometrijskih podatkov in deluje na vseh scenah, ki jih je možno rasterizirati (statične in dinamične). Hierarhična delitev prostora deluje pravilno, a ker je bila večina testne geometrije zelo lokalna in večinoma v vidnem polju, je bil sistem pohitren šele, kadar je bila večina objektov izven vidnega polja.

Dosežena je bila realnočasovna interakcija pri sinhronem prenosu podatkov iz grafičnega v sistemski pomnilnik, saj je sistem zmogel izvesti več kot 40 detekcij vidnosti ploskev v celotni sceni na sekundo pri rasterizacijski ločljivosti 800x800 pikslov. Seveda je to odvisno tudi od kapacitet računalnika, na katerem se postopek izvaja, ter ločljivosti rasterizacije, saj sinhroni prenosi podatkov med sistemskim pomnilnikom in pomnilnikom grafičnega pospeševalnika in nazaj krepko upočasnijo celotno metodo.

5 SKLEP

Implementirali smo dokaj robustno in hitro metodo, ki poišče vse ploskve, vidne iz poljubnega gledišča. Doseženi so bili zadovoljivi realnočasovni rezultati, čeprav je to zelo odvisno tudi od same zmogljivosti računalniškega sistema ter od ločljivosti okna aplikacije. Da bi povečali odzivnost, bi bilo potrebno vzpostaviti asinhrone prenose podatkov med grafičnim in sistemskim pomnilnikom. V takšne raziskave bo usmerjeno tudi naše nadaljnje delo.

6 VIRI

[1] M. Friendly, D. J. Denis, Milestones in the history of thematic cartography, statistical graphics, and data visualization, 2001,
http://www.math.yorku.ca/SCS/Gallery/milestone/ (21.8.2010)

[2] AMD Fusion Family of APUs, bela knjiga,
http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf (18.7.2010)

[3] J. Foley, A. Dam, S. Feiner, J.Hughes, R.Phillips, Introduction to computer graphics, Addison-Wesley, Reading, 1994, strani 465-467

[4] J. Foley, A. Dam, S. Feiner, J.Hughes, R.Phillips, Introduction to computer graphics, Addison-Wesley, Reading, 1994, strani 469-471

[5] J. Foley, A. Dam, S. Feiner, J.Hughes, R.Phillips, Introduction to computer graphics, Addison-Wesley, Reading, 1994, strani 459-460

[6] H. Samet, Application of Spatical Data Structures, Computer Graphics. Image processing and GIS, Addison-Wesley, Wokingham, England, 1990, stran VII

[7] H. Samet, Application of Spatical Data Structures, Computer Graphics. Image processing and GIS, Addison-Wesley, Wokingham, England, 1990, strani 26-27

[8] A. LaMothe, Tricks of the 3D Game Programming Gurus-Advanced 3D Graphics and Rasterization, Sams Publishing, Indianapolis, 2003, strani 1019-1020

[9] N. Krishnamurthy, Introduction to computer graphics, Tata McGraw-Hill, New Delhi, 2002, strani 179-180

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko

Smetanova ulica 17
2000 Maribor

IZJAVA O USTREZNOSTI DIPLOMSKEGA DELA

Podpisani mentor Borut Žalik (ime in priimek mentorja) izjavljam, da je

študent Danijel Žlaus (ime in priimek študenta-tke) izdelal diplomsko

delo z naslovom: Zaznava in prikaz vidnih ploskev na GPU

(naslov diplomskega dela)

v skladu z odobreno temo diplomskega dela, Navodili o pripravi diplomskega dela in mojimi navodili.

Datum in kraj:

24.9.2010, MB

Podpis mentorja:

B. Žalik

Fakulteta za elektrotehniko, računalništvo in informatiko

Smetanova ulica 17
2000 Maribor

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE DIPLOMSKEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek diplomanta-tke: Danijel Žlaus

Vpisna številka: E1004456

Študijski program: Računalništvo in informacijske tehnologije

Naslov diplomskega dela: Zajemanje in prikaz vidnih ploskev na GPU

Mentor: red. prof. dr. Borut Žalik

Somentor: doc. dr. Gregor Klajnšek

Podpisani-a Danijel Žlaus izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru.
Diplomsko delo sem izdelal-a sam-a ob pomoci mentorja. V skladu s 1. odstavkom 21. clena Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 16/2007) dovoljujem, da se zgoraj navedeno zakljucno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija diplomskega dela je istovetna elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov vezanih na zakljucek študija (ime, priimek, leto in kraj rojstva, datum diplomiranja, naslov diplomskega dela) na spletnih straneh in v publikacijah UM.

Datum in kraj: 24. 9. 2010, Maribor

Podpis diplomanta-tke: