Strokovno-znanstveni prispevek ■

Vizualizacija prostorskih medicinskih podatkov na osebnem računalniku

David Podgorelec, Marko Vinter, Borut Žalik

Izvleček. V članku je predstavljen algoritem za senčenje prostorskih podatkov, ki je zaradi svoje enostavnosti in hitrosti primeren tudi za uporabo na osebnem računalniku. Množice podatkov, ki jih je zmožen vizualizirati, so dovolj velike za učinkovito rabo v medicinskih aplikacijah. Algoritem temelji na sekundarnem geometrijskem modelu, ki smo ga poimenovali seznam vidnih vokslov. Vizualizacijo poskuša pospešiti s predhodnim izločanjem vokslov, nezanimivih za uporabnika. Ti tako imenovani beli voksli predstavljajo zrak, notranjost objektov in dele površja objektov, ki niso vidni iz trenutnega položaja opazovalca. Delovanje in učinkovitost algoritma smo predstavili na nekaj praktičnih primerih uporabe pri vizualizaciji medicinskih podatkov.

Visualisation of Volume Medical Data on a Personal Computer

Abstract. We present an algorithm for volume rendering that is simple and sufficiently fast to be run on a low-cost personal computer. The algorithm supports the visualization of large data sets, which are typically examined in medical applications. The algorithm is based on a secondary model called the list of visible voxels. Visualisation is accelerated by eliminating voxels not interesting to a user. These so-called white voxels present air, objects' interior and parts of objects' boundaries hidden from the user's viewpoint. In the last part of the paper, some practical examples from the medical data sets are considered.

■ **Infor Med Slov** 2002; 7(1):31-45

Institucije avtorjev: Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru (DP, BŽ), Oddelek za radiologijo, Splošna bolnišnica Maribor (MV)

Kontaktna oseba: David Podgorelec, Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru, Smetanova ul. 17, 2000 Maribor. email: david.podgorelec@uni-mb.si.

Uvod

Znanstvena vizualizacija (angl. scientific visualisation) je obetajoče in hitro razvijajoče se področje z možnostmi uporabe v najrazličnejših inženirskih panogah. Med največ uporabljanimi metodami znanstvene vizualizacije je nedvomno realistično senčenje prostorskih podatkov (angl. volume rendering), ki ga bomo v tem sestavku imenovali prostorsko upodabljanje ali tudi vizualizacija prostorskih podatkov, med glavnimi uporabniki pa so biomedicinske znanosti s široko rabo v diagnosticiranju in izobraževanju.[1] Prostorsko upodabljanje medicinskih podatkov je prikaz slikovnih podatkov o predelu telesa, ki je pregledan z diagnostičnimi metodami, običajno radiološkimi. Podatki so v digitalni obliki kot zaporedje dvorazsežnih (2D) slik oziroma rezin (angl. slides ali tudi slices). S predstavitvijo podatkov v trirazsežnem prostoru (3D) odpremo poglavje navidezne resničnosti v medicini ali še bolje resnično navideznost v medicini, saj prikažemo resnične podatke v navideznem svetu, seveda čim bolj resnično. Zato so kmalu po klinični uporabi aparatur računalniške tomografije (angl. Computerised Tomography, CT) začeli izdelovati namensko opremo za tako imenovane 3D rekonstrukcije – izdelavo 3D modelov, dobljenih iz rezin CT. Pred 15 leti je to pomenilo počasne računalnike z dokaj grobimi podatki. Z razvojem računalniške tehnologije so se stvari hitro izboljšale, z uvedbo magnetne resonance (angl. Magnetic Resonance Imaging, MRI), kasneje pa še 3D ultrazvoka in rotacijske DSA (angl. Digital subtraction angiography) pa je prišlo do dokončne uporabe 3D modelov v diagnostične namene. Kar se tiče strojne opreme, so bile najprej uporabljene delovne postaje tipa SUN in SGI z operacijskim sistemom Unix, v zadnjih dveh letih pa se pojavlja težnja prenesti programsko opremo tudi na platformo PC.

Razlogi za uporabo prostorskega upodabljanja oziroma širše 3D rekonstrukcij v medicini so abstraktne slike, število slik in interpretacija podatkov. Vse naštete naprave oziroma tehnike nudijo na zaslonu ali na tiskalniškem izpisu 2D sliko ne glede na način zajemanja slike. Ta slika oziroma serija slik (od 20 do 200) je abstraktna, običajno črno bela, neresnična.

Problem interpretacije je najbolje predstaviti na primeru. Bolnik ima tumor v predelu hrbtenice. Naredimo CT. Dobimo po 15 slik na 4 – 6 folijah velikosti 45 × 43 cm, torej skupno 60 – 90 sličic. Iz teh slik mora radiolog razbrati dolžino patologije ter premer tumorja na posameznih višinah. Na podlagi njegovega izvida se mora kirurg odločiti za diagnostični poseg. Torej morata tako kirurg kot radiolog v glavi pravilno pretvoriti črno bele rezine v barvno, krvavo sliko, ki jo bo operater v resnici zagledal med operacijo.

Kljub različnim zahtevam in pričakovanjem strokovnjakov z različnih področij imajo najpogosteje uporabljane metode vizualizacije prostorskih podatkov mnogo skupnega. Prav vse se spopadajo z osnovnim problemom prostorskega upodabljanja: z velikimi količinami geometrijskih podatkov. Tako je na primer v medicinski vizualizaciji standardni izhod CT ali MRI rezina velikosti 512 × 512 ali celó 1024 × 1024 slikovnih elementov (pikslov), kvantificiranih z 2^{12} diskretnimi vrednostmi. Z drugimi besedami to pomeni, da je za predstavitev vsakega piksla potrebnih po 12 bitov. Prostorski model dobimo iz zaporedja rezin tako, da pikslom dodamo debelino, ki je določena z razdaljo med dvema sosednjima (vzporednima) rezinama. Največkrat se uporablja nekaj sto rezin, kar pomeni, da sestoji prostorski model iz nekaj deset ali tudi nekaj sto milijonov vokslov. Razumljivo je, da so poleg velikih prostorskih zahtev tudi časi izvajanja algoritmov temu primerno dolgi. Zaradi tega je bila (in je pretežno še danes) vizualizacija prostorskih podatkov domena delovnih postaj, opremljenih z grafičnimi pospeševalniki, ali celó domena računalnikov za posebne namene, zasnovanih na paralelnih računalniških arhitekturah.[2-4]. Takšna draga strojna oprema je pogosto predstavljala resno oviro raziskovalcem in uporabnikom. Na srečo so s pocenitvijo spomina in procesorske moči v zadnjem času postali mogoči tudi projekti vizualizacije

prostorskih podatkov na osebnih računalnikih (PC).

Do danes je bilo predlaganih kar nekaj tehnik vizualizacije prostorskih podatkov. V grobem jih lahko razdelimo v dve skupini:[5]

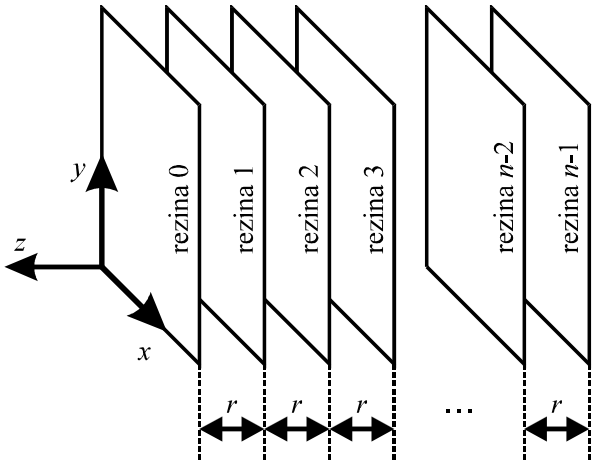
Algoritmi za neposredno senčenje prostora (angl. direct volume rendering algorithms). V tem primeru so prostorski podatki organizirani v preprosto 3D matriko vokslov. Geometrijski model največkrat vizualiziramo z uporabo metanja žarka (angl. ray casting)[6,7] ali vmesnega pomnilnika za koordinato z (angl. z-buffer).[8] V splošnem so te metode računsko precej obsežne. Nepotrebnim izračunom v praznem ali prosojnem prostoru se velikokrat izognemo z uporabo hierarhičnih podatkovnih struktur (npr. osmiških dreves, dreves K-D).[9]

Pridobivanje sekundarnih modelov (angl. secondary model extraction). Objekt predstavimo z alternativnimi modeli, kot so konturne črte ali ploskve, ki jih potem prikazujemo z uporabo klasičnih metod računalniške grafike.[8] Eden izmed najpopularnejših pristopov uporablja t.i. 'korakajoče kocke' (angl. marching cubes) za pridobivanje modela z ovojnico.[10]

V članku obravnavamo naš pristop k vizualizaciji prostorskih podatkov. Metoda temelji na ekstrakciji sekundarnega modela – vidnih stranic vokslov. Te stranice potem senčimo oziroma prikazujemo v zadnjem koraku algoritma. Predstavljena rešitev teče na zmerno zmogljivem osebnem računalniku s 512 MB delovnega pomnilnika in s 600 MHz procesorjem. Za testiranje smo uporabili medicinske podatke s po 512×512 piksli na rezino, ki so shranjeni v grobem (ACR NEMA) formatu. Prek 200 rezin je možno procesirati in senčiti v sprejemljivem času. Ta trditev je podprta z nekaj eksperimentalnimi rezultati, prikazanimi v zadnjem delu članka.

Definicija problema

Vhod v naš algoritem je zaporedje rezin, kjer vsaka rezina predstavlja 2D sliko dela objekta, ki ga želimo prikazati. Predpostavljamo, da so bile vse rezine zajete oziroma posnete v isti geometrijski smeri (v praksi običajno tudi je tako). Bralec si lahko predstavlja, da se je kamera premikala vzdolž premice in v enakomerno porazdeljenih točkah zajemala slike. Vse rezine so enake velikosti in vse so pravokotne na pot kamere. Izhodišče desnosučnega realnega koordinatnega sistema je v levem spodnjem kotu prve rezine, kamera pa se premika v smeri negativne osi z (slika 1). Število rezin bomo označevali z n, razdaljo med dvema sosednjima rezinama oz. debelino voksla pa z r. V članku bomo r izražali kar v pikslih, čeprav je pričakovati, da bo uporabnik določal to vrednost v milimetrih. Pretvorba je seveda trivialna. Naš računalniški program lahko sprejme rezine praktično poljubne velikosti. Edino omejitev določa t.i. maksimalna razsežnost oziroma število bitov, uporabljenih za zapis indeksov (po enega v vsaki koordinatni smeri) posameznega voksla, ki je v našem primeru 32. Velja torej $w + h + d = 32$, kjer je w število bitov, uporabljenih za širino rezine (smer x na sliki 1, angl. width = širina), h število bitov, uporabljenih za višino rezine (smer y na sliki 1, angl. height = višina), d pa je število bitov za predstavitev števila rezin n (angl. depth = globina). Privzete vrednosti so $w = h = 11$ in $d = 10$, ki omogočajo največjo ločljivost slike 2048×2048 pikslov in največ 1024 rezin.



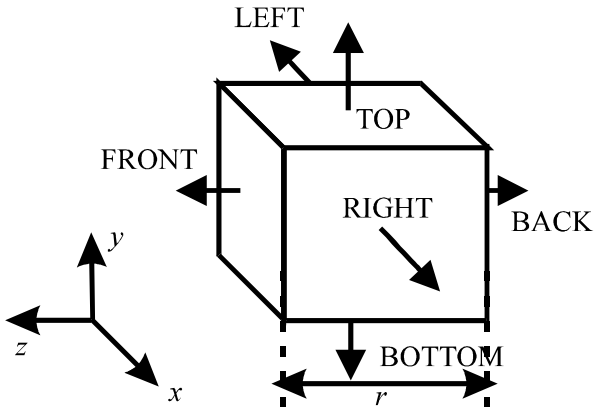
Slika 1 Položaj rezin v realnem koordinatnem sistemu

Rezine preberemo iz datotek v grobem (ACR NEMA) formatu. Za ta format smo razvili tudi preprosto, od podatkov odvisno kodirno tehniko, ki zagotavlja hitro (v realnem času) kodiranje in dekodiranje brez izgub kvalitete ter v povprečju prihrani blizu 80 odstotkov pomnilniškega prostora. Ukvarjamo se tudi z modulom, ki bo zmožen neposredno brati datoteke v standardu DICOM 2. Za sedaj takšne datoteke predhodno pretvarjamo v zaporedje datotek ACR NEMA s posebnim programom.

Vsak piksel vsake rezine je predstavljen z vrednostjo iz obsega [–2048, 2047], ki predstavlja indeks barve piksla v uporabljeni barvni paleti. Paleta vsebuje 4096 polj, v vsakem pa je zapisana 32-bitna vrednost. Prvih 24 bitov je uporabljenih za opis barve v formatu RGB, preostalih 8 bitov pa predstavlja številko skupine. Skupine omogočajo uporabniku, da opremi barve, ki verjetno predstavljajo isti objekt ali isto značilnost objekta, z istimi atributi vidnosti. Z osmimi biti lahko definiramo 256 skupin, ki jih indeksiramo od 0 do 255. Vsaka skupina je opremljena z zastavico, ki določa vidnost barv te skupine. Skupina z indeksom 0 predstavlja zrak, ki je vedno neviden. Privzeta barva te skupine ima indeks –2048 in uporabnik ne more spreminjati te nastavitve. Podobno so barve skupine z indeksom 255 vedno vidne. Zastavice vidnosti ostalih skupin je možno spreminjati. V prihodnji verziji bomo zastavice nadomestili z 8-bitno vrednostjo, ki bo podajala odstotek prosojnosti do pribl. 0,5 odstotka natančnosti.

Tretji del vhoda v program predstavljajo geometrijski podatki, ki opisujejo operacijo gledanja. To so normalni vektor n projekcijske ravnine ter točki p_1 in p_2, ki ležita v tej ravnini. Z normalnim vektorjem in prvo točko opišemo enačbo ravnine, druga točka pa je namenjena le za orientacijo oziroma določitev t.i. vektorja 'up', ki ga označimo z u. Ko preslikamo 3D realne koordinate točk v projekcijski ravnini v 2D koordinate računalniške izhodne naprave (običajno zaslona), je namreč treba vedeti, katera smer v realnem koordinatnem sistemu ustreza vektorju, ki je na zaslonu usmerjen navzgor[11].

Rezine uporabimo za izgradnjo prostorskega modela, predstavljenega s 3D matriko prostorskih elementov ali krajše *vokslov*. Element matrike $v_{i,j,k}$ ustreza vokslu oziroma majhnemu kvadru, ki se razteza v prostoru med k-to in $(k + 1)$-to rezino. Prednja in zadnja stranica voksla se ujemata s piksloma v i-tem stolpcu in j-ti vrstici obeh omenjenih rezin. Vse mejne ploskve voksla dobijo barvo prednje stranice. Voksel z angleškimi imeni posameznih stranic je prikazan na sliki 2. Govorimo o prednji (FRONT), zadnji (BACK), levi (LEFT), desni (RIGHT), spodnji (BOTTOM) in zgornji (TOP) stranici. Prednja in zadnja stranica sta velikosti 1×1.



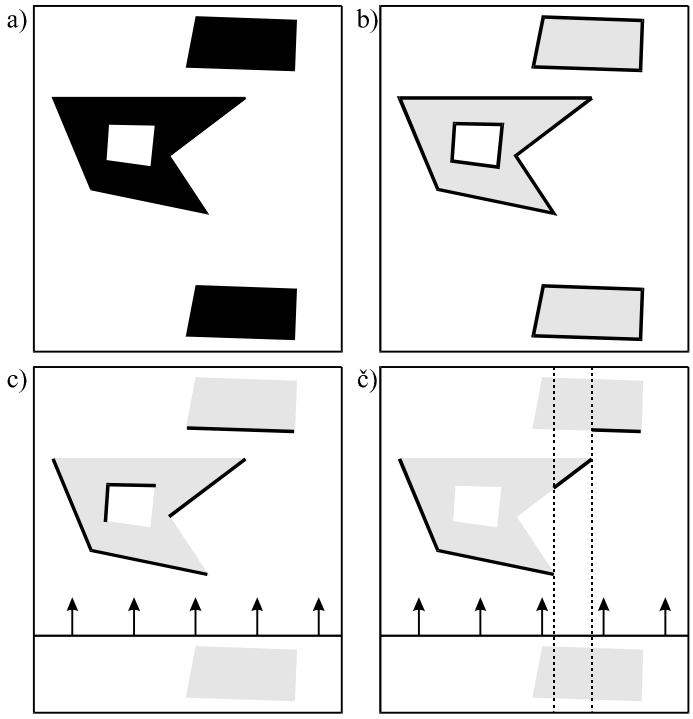
Slika 2 Voksel in njegovih šest stranic

Cilj predstavljenega algoritma je projicirati prostorski model v projekcijsko ravnino in potem prikazati sliko, dobljeno v tej ravnini, na računalniškem zaslonu. To je treba doseči kar najhitreje, še posebej, ker program omogoča uporabniku, da interaktivno spreminja atribute operacije gledanja, pri čemer uporabnik pričakuje hiter odziv po izvedbi zahtevanih geometrijskih transformacij. Zato algoritem ne senči vseh vokslov, ampak le tiste, ki so vidni glede na trenutno operacijo gledanja. Ti voksli predstavljajo izhod iz algoritma in so shranjeni v *seznamu vidnih vokslov* (angl. the list of visible voxels – VVL).

Algoritem

Algoritem se izvaja v naslednjem zaporedju korakov:

1. Branje rezin iz datotek in izgradnja 3D matrike vokslov V.
2. Izločitev vokslov, ki vsebujejo zrak, in tistih, ki so v notranjosti objekta. Preostale voksle prepišemo v seznam vidnih vokslov VVL.
3. Voksle, ki ne morejo biti vidni glede na trenutno operacijo gledanja, izločimo iz VVL.
4. V VVL dodamo še morebitne voksle notranjosti objektov, ki ležijo tik pred projekcijsko ravnino. Ta korak obravnava primere, ko projekcijska ravnina prereže posamezne objekte.
5. Prikaz elementov VVL na izhodni napravi.

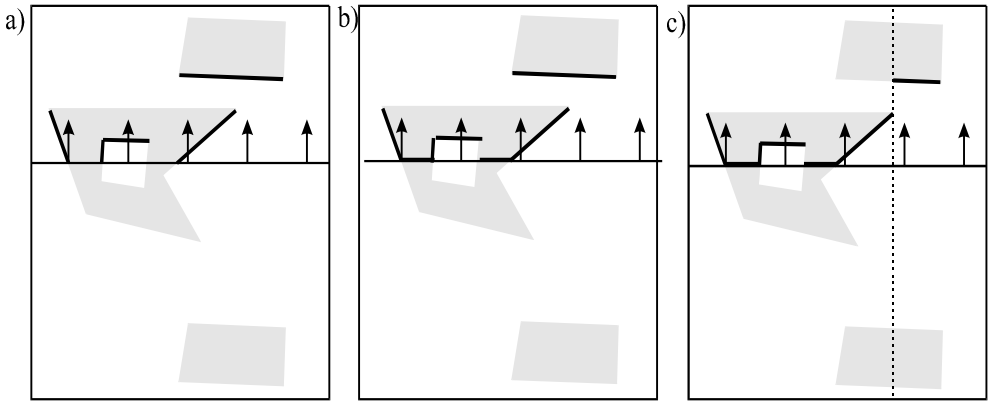


Slika 3 Določanje vidnih vokslov v primeru, ko projekcijska ravnina ne seka objektov

Slika 3 prikazuje prerez skozi vidni prostor, ki vsebuje tri objekte. Na sliki 3a predstavlja bela barva zrak, črna pa objekte. Tako črni kot beli voksli se v 1. koraku algoritma prepišejo v 3D matriko vokslov V. Korak 2 izloči voksle, ki predstavljajo zrak, ter tiste v notranjosti objektov.

Samo voksli, ki predstavljajo meje objektov, so kandidati za prikaz in jih vstavimo v seznam VVL. Ti voksli so predstavljeni s črno barvo na sliki 3b. Slika 3c prikazuje stanje po nadaljnji redukciji VVL, ki jo opravi korak 3. Odstranjeni so voksli za opazovalčevim hrbtom in skrite ploskve objektov pred projekcijsko ravnino. Slika 3č pa v črni barvi prikazuje le tiste voksle, katerih projekcija tvori končno 2D sliko (korak 5).

V zgornjem primeru projekcijska ravnina ne prereže nobenega izmed objektov, torej korak 4 ni spreminjal VVL. Situacija na sliki 4 pa je drugačna. Po koraku 3 je stanje takšno, kot ga prikazuje slika 4a. Projekcijska ravnina očitno seka enega izmed objektov in tudi luknjo v njem. Korak 4 nato doda voksle v notranjosti objekta tik pred projekcijsko ravnino. Na sliki 4b so ti voksli ponazorjeni z dvema vodoravnima daljicama tik nad daljšo in tanjšo črto, ki ponazarja projekcijsko ravnino. Zadnji korak potem podobno kot v prejšnjem primeru le še prikaže vidne voksle, pri čemer odstrani tiste, ki jih zakrivajo drugi objekti ali deli objektov. Situacijo prikazuje slika 4c.

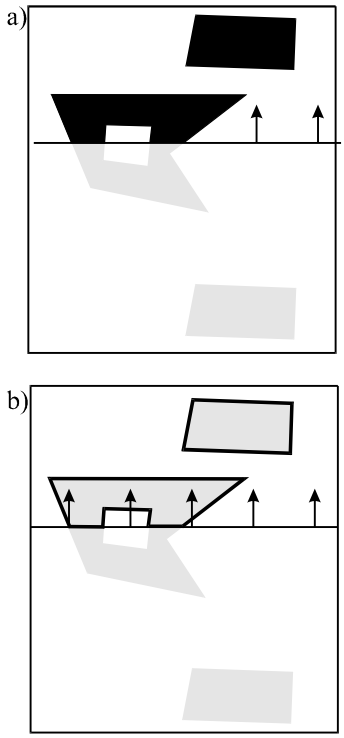


Slika 4 Primer, ko projekcijska ravnina prereže enega izmed objektov

Pripomnimo še, da bi lahko bil vrstni red naštetih korakov tudi drugačen. Korak 4, ki obravnava prereze, bi lahko izpustili, če bi korake organizirali v naslednjem zaporedju:

1. Branje rezin iz datotek in izgradnja 3D matrike vokslov V.
2. Izločitev vokslov za hrbtom opazovalca. Preostale voksle prepišemo v seznam vidnih vokslov VVL.
3. Voksle, ki predstavljajo zrak ali notranjost objektov, izločimo iz VVL.
4. Izločitev vokslov površja objektov pred opazovalcem, ki ne morejo biti vidni glede na operacijo gledanja.
5. Prikaz elementov VVL na izhodni napravi.

Delovanje tako modificiranega algoritma prikazuje slika 5. Scena je enaka kot v predhodno opisanih primerih (slika 3a), projekcijska ravnina pa tako kot v primeru s slike 4 seka enega izmed objektov. Opazimo naslednje spremembe. Modificiran algoritem že v 2. koraku izloči voksle, ki so za hrbtom opazovalca (slika 5a). S tem smo vsem vokslom, ki so tik pred projekcijsko ravnino, na eni strani odvzeli sosede. Zaradi tega jih naslednji (3.) korak, ki poleg zraka izloča voksle v notranjosti objektov, prepozna kot voksle meje oziroma površja objektov. Takšni voksli torej ostanejo v VVL in ne potrebujemo dodatnega koraka, ki bi jih kasneje dodajal v seznam. Stanje po koraku izločanja notranjosti prikazuje slika 5b. Sledi izločanje preostalih vokslov, ki ne morejo biti vidni glede na operacijo gledanja (korak 4), in dobimo stanje, ki se ujema s stanjem po 4. koraku originalnega algoritma (slika 4b). Vizualizacija se potem izvede na enak način kot v prejšnjem primeru (slika 4c).



Slika 5 Koraka 2 in 3 modificiranega algoritma

Bistvena prednost spremenjenega algoritma je lažja implementacija. Žal pa se ta algoritem pokaže za precej manj učinkovitega kot originalni. Test, ali je voksel za hrbtom opazovalca ali pred projekcijsko ravnino, je treba opraviti za vse črne voksle s slike 3a, torej tudi za notranjost objektov. Največkrat je število notranjih vokslov zelo veliko v primerjavi s številom vokslov na površju objektov, sam test pa računsko ni povsem nezahteven. Test, ali je voksel znotraj objekta, je precej enostavnejši, saj temelji le na preverjanju sosedov, medtem ko si je pri prvem testu treba pomagati s skalarnim produktom. Ta slabost je še posebej moteča, kadar uporabnik spreminja operacijo gledanja. Ta sprememba je običajno veliko pogostejša kot pa spreminjanje atributa vidnosti posamezne barve oziroma skupine, saj si želi uporabnik običajno ogledati objekt v izbranih barvah kar z največ strani ter zato pričakuje enostavno in hitro navigacijo. Originalni algoritem lahko v primeru spremembe operacije gledanja uporabi kot vhod močno okleščen VVL brez vokslov notranjosti, modificirani algoritem pa uporablja operacijo gledanja že pri izločanju vokslov za hrbtom opazovalca ter mora zato ob vsaki geometrijski transformaciji obravnavati prav vse črne voksle. V nadaljevanju bomo obravnavali posamezne korake prve (originalne) verzije algoritma. Opisani bodo tudi vsi testi (izločanje vokslov za kamero, izločanje notranjosti in izločanje skritih ploskev objektov pred kamero).

Izgradnja 3D matrike vokslov

V tem koraku se barvni indeksi pikslov vseh rezin kopirajo v 3D matriko vokslov V. Relacija med posameznim pikslom in ustreznim vokslom je opisana v 2. poglavju. Vsak voksel v matriki V predstavimo z dvema zlogoma. Matrika je običajno zelo velika. Pri ločljivosti 512 × 512 pikslov in 220 rezinah na primer dobimo matriko, ki zaseda več kot 100 MB (natančno 115.343.360 zlogov). Ker program uporablja še dodatne podatke (na primer VVL) skupne velikosti približno 20 MB, teče zadovoljivo hitro le na računalnikih z vsaj 256 MB pomnilniškega prostora. Uporabnik lahko znatno prihrani pomnilniški prostor, če izključi tvorbo 3D matrike V. V tem primeru se korak 2 izvaja neposredno v času branja rezin. Žal pa takšen pristop zahteva ponovno branje matrike, kadar uporabnik spremeni zastavice vidnosti posameznih barv ali atribute operacije gledanja.

Izločanje zraka in notranjosti

V naslednjem koraku (korak 2) uporabimo matriko V (ali zaporedje rezin v implementaciji z neposredno tvorbo VVL iz rezin), izhod iz koraka pa predstavlja seznam morebiti vidnih vokslov VVL. Ta korak običajno izloči veliko število vokslov iz množice kandidatov za senčenje. Izločanje temelji na dveh preprostih dejstvih, ki ne potrebujeta dokazovanja:

1. Uporabnik vidi skozi zrak.
2. Uporabnik ne vidi skozi voksle, ki predstavljajo vidne objekte (obravnavana verzija ne vključuje delno prosojnih vokslov).

Izraz »zrak« uporabljamo za voksle, ki dejansko predstavljajo zrak (tiste z barvnim indeksom – 2048), pa tudi za voksle, obarvane z barvami iz skupin, ki imajo zastavico vidnosti postavljeno na »nevidno«. Ker tudi barva z indeksom –2048 zagotovo pripada takšni skupini, je test, ali voksel predstavlja zrak ali ne, zares trivialen. Treba je le preveriti zastavico vidnosti ustrezne skupine.

Drugo dejstvo, omenjeno zgoraj, ima pomembno posledico. Če so vsi neposredni sosedje opazovanega voksla vidni, potem takšen voksel zagotovo ne more biti viden. Zaradi tega lahko tudi voksle v notranjosti objektov odstranimo na povsem enostaven način. Treba je preveriti zastavice vidnosti sosednjih vokslov in če so vse postavljene na »vidno«, potem opazovani voksel ne more biti viden. Sosedje so definirani le s sosednostno relacijo lice – lice. Če naj bo vidno oglišče ali rob voksla, mora biti vidna tudi ena od priležnih mejnih ploskev. Torej je treba obravnavati le šest sosedov opazovanega voksla $v_{i,j,k}$: voksel $v_{i,j,k-1}$ pred njim, $v_{i,j,k+1}$ za njim, $v_{i-1,j,k}$ levo in $v_{i+1,j,k}$ desno od njega, $v_{i,j-1,k}$ pod njim in $v_{i,j+1,k}$ nad njim. Poleg tega voksli na robovih opazovanega prostora nimajo šest sosedov, ampak le tri, štiri ali pet in so avtomatsko vidni.

Dobljeni seznam VVL vsebuje le voksle, ki predstavljajo meje objektov v vidnem prostoru. Te meje se lahko razlikujejo od realnih meja objektov, saj lahko posamezne voksle, ki v realnosti predstavljajo objekte, naredimo za zrak s postavitvijo ustreznih zastavic vidnosti na »nevidno«.

Vsak voksel je v VVL predstavljen s tremi indeksi i, j, k, ki predstavljajo stolpec, vrstico in številko rezine. Ti indeksi so kodirani v zapisu dolžine 32 bitov. Prav zaradi tega je maksimalna razsežnost omejena z obrazcem $w + h + d = 32$. Poleg tega vključuje vsak element seznama še dodaten zlog, imenovan status vidnosti lic. Ta zlog vsebuje zastavice vidnosti vseh šestih mejnih ploskev voksla. Dva bita nista uporabljena. Zastavica vidnosti posamezne mejne ploskve je postavljena na 1, kadar je sosednji voksel, ki se stika z opazovanim v tej mejni ploskvi, zrak.

Izločanje vokslov, ki niso vidni glede na izbrano operacijo gledanja

Prejšnji korak temelji na snovnih lastnostih predstavljene scene. S poznavanjem teh lastnosti običajno znatno zmanjšamo število vokslov, ki jih bo treba senčiti. Nadaljnje zmanjšanje števila kandidatov za senčenje lahko dosežemo z uporabo geometrijskih lastnosti. V tem poglavju opišemo korak algoritma (korak 3), ki iz VVL izloči voksle, ki ne morejo biti vidni iz projekcijske ravnine. Očitno uporabljamo ortografsko paralelno projekcijo, kar je razvidno že iz slik 3, 4 in 5.

Projekcijska ravnina je definirana z normalnim vektorjem n in s točko p_1, ki leži v tej ravnini. Voksle aproksimiramo z njihovimi središčnimi točkami. Če se indeksi v vseh treh koordinatnih smereh začno z 0, potem je središčna točka voksla $v_{i,j,k}$ določena z obrazcem:

$$c_{i,j,k} = (i + 0{,}5, j + 0{,}5, -r \cdot (k + 0{,}5)),$$

kjer r predstavlja debelino posameznega voksla. Opazimo, da je koordinata z negativna (za razlago glej sliko 1).

Algoritem izloči voksle, ki so skriti za drugimi voksli istega objekta, pa tudi tiste, ki se nahajajo na napačni strani projekcijske ravnine (za hrbtom opazovalca). Testa, ki ju uporabimo za ugotavljanje teh dveh lastnosti, sta si precej podobna.

Izločanje skritih vokslov temelji na dejstvu, da je kot med smerjo pogleda (vektorjem n, če uporabljamo ortografsko paralelno projekcijo) in med normalnim vektorjem vidne ploskve objekta večji od 90°, medtem ko je kot med smerjo pogleda in normalnim vektorjem skrite ploskve manjši ali enak 90°. Pri tem mora biti normalni vektor obravnavane ploskve usmerjen iz objekta (v našem primeru voksla).

Za določitev kota med dvema vektorjema uporabimo njun skalarni produkt. Ker so vsi voksli orientirani na enak način, te operacije ni treba

izvesti na licih vseh vokslov v VVL, ampak le na šestih licih enega samega vzorčnega voksla.

Rezultate vseh šestih testov shranimo v masko vidnosti lic, ki je organizirana na enak način kot status vidnosti posameznega voksla. Spomnimo se, da je posamezna zastavica v statusu vidnosti postavljena, kadar je sosednji voksel, ki se stika z opazovanim v ustrezni ploskvi, zrak. Algoritem sedaj primerja status vidnosti vsakega voksla z masko vidnosti lic z uporabo bitnega operatorja IN. Kadar je rezultat različen od 0, je voksel še naprej kandidat za prikaz in ostane v VVL. V nasprotnem primeru voksel odstranimo iz VVL.

Za voksel, ki ga je zgornji test potrdil za vidnega, je treba še ugotoviti, ali leži pred projekcijsko ravnino ali za njo. Voksel je pred projekcijsko ravnino, kadar je kot med normalnim vektorjem ravnine n ter vektorjem, usmerjenim iz točke p_1 v središčno točko voksla, manjši kot 90°. Tudi voksli v projekcijski ravnini, za katere je ta kot natanko 90°, so sprejemljivi.

Prerezi skozi posamezne objekte

Vidni prostor običajno ni tako prazen kot na sliki 3. Kadar je predstavljenih več objektov in predvsem kadar so ti objekti večji, projekcijska ravnina pogosto prereže katerega izmed njih. Velikokrat je to tudi glavni namen uporabnika, ki izbere takšne parametre operacije gledanja, da dobi želeni prerez objekta. V takšnem primeru je treba vse voksle notranjosti objektov (te smo izločili v koraku 2), ki ležijo v projekcijski ravnini, ponovno dodati v VVL.

Grob pristop bi za vsak voksel, ki smo ga predhodno izločili, oziroma za njegovo središčno točko $c_{i,j,k}$ testiral predznačeno razdaljo od projekcijske ravnine. Razdalja je določena z obrazcem:

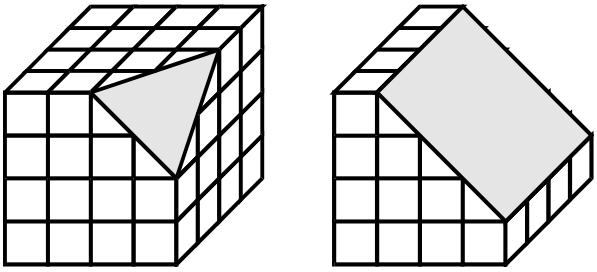
$$\delta(c_{i,j,k}, \Pi) = n \cdot c_{i,j,k} - n \cdot p_1$$

Če je ta razdalja znotraj obsega $[-1/2, r/2]$, potem voksel $v_{i,j,k}$ dodamo v VVL. Ker je voksel aproksimiran s središčno točko, rezultati velikokrat niso stoodstotno natančni. Pogosto se zgodi, da vstavljena plast vokslov ni povsem tanka (slika 4). Prag $r/2$ je namreč precej velik, še posebej, kadar je projekcijska ravnina vzporedna ali skoraj vzporedna z osjo z. Izračunane razdalje so za voksle, ki jih seka projekcijska ravnina, običajno manjše od ½. Pogosto se zgodi, da dodamo še vsaj eno plast sosednjih vokslov pred projekcijsko ravnino. Dodajanju prevelikega števila vokslov za projekcijsko ravnino se izognemo z uporabo nesimetričnega praga. Za negativne razdalje so dovoljena manjša odstopanja.

Ker torej pogosto dodamo preveč vokslov, jih bo treba tudi več senčiti. Vendar pa ima to dejstvo tudi dobro stran. Rezultati so namreč lepši. Ploskev prereza je namreč lahko preveč porozna, če dodamo eno samo plast vokslov.

Slabost opisanega načina je, da je treba testirati vse voksle, ki smo jih predhodno izločili. Teh je običajno zelo veliko, sam postopek za izračun razdalje pa je časovno precej zahteven (6 produktov in 7 seštevanj realnih števil). Zato rajši uporabimo direktno metodo, ki vokslov prereza ne išče v množici vokslov, ampak jih izračuna. Metoda se izvede v naslednjem zaporedju korakov:

1. Izračun prereza prostora vokslov in projekcijske ravnine. Prerez je lahko trikotnik ali štirikotnik (slika 6).
2. Geometrijska transformacija prereza (mnogokotnika) v ravnino xy.
3. Rasterizacija transformiranega mnogokotnika.
4. Inverzna geometrijska transformacija pikslov, dobljenih s transformacijo, nazaj v projekcijsko ravnino.
5. Za vsak transformirani piksel ugotovimo, kateremu vokslu pripada, in voksel dodamo v v VVL.



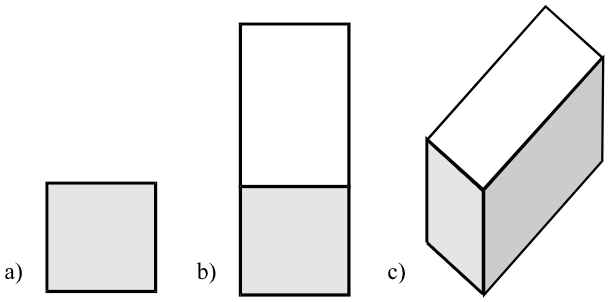
Slika 6 Presek prostora vokslov in projekcijske ravnine

Še enkrat se spomnimo, da smo v predhodnih razdelkih predlagali tudi pristop, v katerem lahko dodajanje prerezov preskočimo. Spremenjeni algoritem najprej spremeni vse voksle za hrbtom opazovalca v zrak, s čimer postanejo morebitni notranji voksli tik pred projekcijsko ravnino voksli površja. Zato jih naslednji korak, ki izloča notranje voksle, ne izloči iz VVL in jih tudi ni treba kasneje ponovno dodajati.

Senčenje

Oblika projekcije voksla je odvisna od orientiranosti projekcijske ravnine, ki vpliva na število lic voksla, vidnih iz projekcijske ravnine. Če je normalni vektor n pravokoten na dve koordinatni osi, je lahko vidno po eno samo lice posameznega voksla (slika 7a). Če je normalni vektor n pravokoten na natanko eno koordinatno os, sta lahko vidni dve sosednji lici posameznega voksla (slika 7b). In končno, če n ni pravokoten na nobeno koordinatno os, je treba senčiti po tri lica vsakega vidnega voksla (slika 7c). V prvih dveh primerih je projekcija voksla pravokotnik, v tretjem primeru pa šestkotnik. Vendar ne bomo razlikovali med temi tremi primeri in bomo za projekcijo vedno vzeli šestkotnik. Na slikah 7a in 7b lahko za manjkajoča lica smatramo daljice, ki sovpadajo z ustreznimi robovi. Na ta način prevedemo oba primera v obliko, enako tretjemu primeru.

Nadalje lahko vsakega izmed paralelogramov, ki predstavljajo vidna lica voksla, senčimo ločeno, ali pa senčimo vsa tri lica kot en sam šestkotnik, ki predstavlja obris voksla. Odločili smo se za drugo možnost, ki je približno 2,7-krat hitrejša.



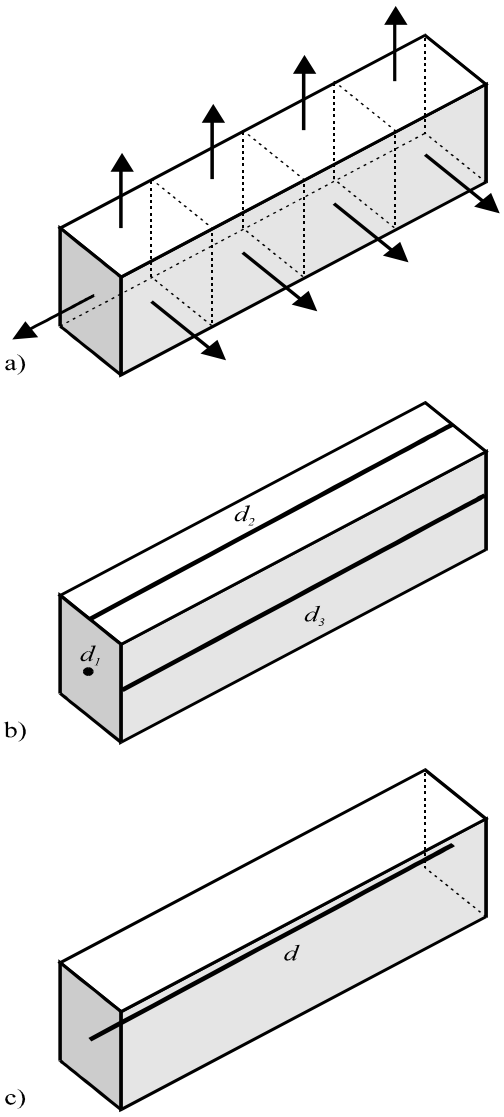
Slika 7 Vidna lica voksla

Preizkušali smo tudi različne aproksimacije šestkotnika, ki bi lahko pospešile senčenje. Tri implementirane zamisli prikazuje slika 8:

a) Voksel razdelimo na nekakšne enotske voksle, s čimer dosežemo, da je vsak enotski voksel dimenzij $1 \times 1 \times 1$. Projekcija posamezne ploskve enotskega voksla je piksel. Potem enostavno prikažemo vse piksle, v katere se projicirajo vidne ploskve enotskih vokslov (slika 8a).

b) Vsako vidno ploskev voksla aproksimiramo z daljico, potem pa prikažemo ustrezne daljice (eno, dve ali tri) – slika 8b. Za ploskev FRONT ali BACK, ki sta dimenzij 1×1, krajišči daljice (d1 na sliki) seveda sovpadata.

c) Ker sta daljici d2 in d3 s slike 8b vzporedni in zelo blizu skupaj (ploskvi FRONT in BACK sta dimenzij 1×1), sta zagotovo takšni tudi njuni projekciji. Zato lahko celoten voksel aproksimiramo z eno samo daljico (slika 8c).



Slika 8 Različne aproksimacije lic voksla

Najnatančnejši (in najpočasnejši) je prvi način, najslabši pa zadnji. Pri prvih dveh načinih lahko sliko še dodatno senčimo, saj lahko vsaki izmed šestih ploskev vokslov določimo različne parametre osvetljenosti. Pri tretjem načinu to ni možno, poleg tega pa bomo, v kolikor ga uporabimo, skoraj zagotovo dobili nezvezne površine oziroma porozne objekte (med dvema daljicama, ki prikazujeta dva sosednja voksla, bodo piksli, ki predstavljajo zrak).

Ne glede na to, ali senčimo posamezne vidne ploskve kot paralelograme, celoten voksel kot šestkotnik, ali pa uporabimo kakšno aproksimacijo, se je treba še odločiti, ali bomo voksle prikazovali neposredno na zaslon ali pa bomo risali v ozadju in potem prikazali celotno digitalno sliko (rezultat senčenja) naenkrat. Nedvomno je boljši drugi način, saj je barvanje posameznih pikslov zelo počasno opravilo. Pri risanju v ozadju lahko uporabimo 2D matriko, ki jo napolnimo z ustreznimi barvnimi indeksi ali kar z vrednostmi RGB ter nato prikažemo kot bitno sliko, lahko pa uporabimo priljubljeni programerski trik ter rišemo direktno z ukazi za risanje mnogokotnikov (npr. šestkotnika), vendar ne v kakšno vidno okno na zaslonu, pač pa na delovno površino (canvas v Borlandovem okolju oziroma kontekst naprave – DC v Microsoftovem okolju), ki ni dodeljena nobenemu aktivnemu oknu. Ob koncu risanja le še dodelimo takšno površino enemu izmed aktivnih oken aplikacije ter osvežimo vsebino okna. Počasnost direktnega risanja seveda izhaja iz nizkih časovnih zmogljivosti grafične kartice. Z gledišča samega algoritma je veliko bolj pomembno, kako določiti najprimernejši vrstni red prikazovanja vokslov iz VVL. Na voljo sta dve logični možnosti:

- prikazovanje vokslov od najbolj oddaljenega do najbližjega projekcijski ravnini (dobro poznan algoritem za risanje[8]);
- prikazovanje vokslov v obratnem vrstnem redu, torej od najbližjega do najbolj oddaljenega.

Obe rešitvi zahtevata urejanje vokslov glede na njihove oddaljenosti od projekcijske ravnine. Hitro urejanje, ki ga danes največ uporabljamo za urejanje kakršnih koli podatkov, se izkaže kot katastrofalno počasno, kadar je normalni vektor približno vzporeden z osjo z. Takrat so namreč voksli v vhodni matriki V pa tudi v seznamu VVL že urejeni po razdaljah od projekcijske ravnine (ta je v tem primeru približno vzporedna z rezinami), kar predstavlja enega od najslabših možnih vhodov za hitro urejanje. Zato rajši uporabimo svoj algoritem urejanja, ki smo ga poimenovali navihano urejanje (angl. smart sort). Razdaljo

računamo na preprost način z uporabo obrazca iz prejšnjega poglavja.

Prva rešitev je preprostejša, saj ne zahteva dodatnih testov. Voksli, ki so bližje projekcijski ravnini, enostavno prekrijejo že prikazane bolj oddaljene voksle ali dele teh. Očitna slabost tega pristopa pa je, da običajno prikazuje tudi veliko pikslov, ki jih kasneje prekrijemo, saj jih ne potrebujemo v zaključni sliki.

Druga rešitev najprej prikaže najbližji voksel. Ko obravnavamo naslednji voksel, preverimo za vsak piksel njegove rasterizirane projekcije, ali je bil morebiti že uporabljen za prikaz prvega voksla. Postopek ponavljamo, dokler niso obravnavani vsi voksli, oziroma dokler je v matriki še kaj neuporabljenih pikslov. Tukaj očitno ne moremo uporabiti ukaza za prikaz mnogokotnika, ampak je treba programsko poskrbeti za njegovo rasterizacijo. Ker moramo posamezne piksle, ki jih testiramo, identificirati v pomnilniku, je ta metoda uporabna zgolj v povezavi z risanjem v ozadju.

Ob koncu poglavja o senčenju še nekoliko osvetlimo vlogo vektorja u. Projekcijska ravnina je seveda neskončna, zato lahko le majhen del prikažemo v oknu oziroma v vidni odprtini. Ko preslikamo realne koordinate vidne odprtine v koordinate zaslona, dobimo pravokotno območje, v katerem so lahko objekti poljubno orientirani. Z definiranjem vektorja u uporabnik nedvoumno izbere eno izmed neskončno veliko orientacij. Pravzaprav uporabnik ne poda tega vektorja direktno, ampak z definiranjem dodatne točke p_2. Vektor u potem dobimo kot vektor, usmerjen iz točke p_1 v smeri projekcije p_2 v projekcijsko ravnino. Pravzaprav nam ni treba računati projekcije točke v ravnino, pač pa si pomagamo s pomožnim vektorjem $v = n \times (p_2 - p_1)$, ki ga nato uporabimo za izračun $u = n \times v$.

Tudi točka p_1, ki sicer določa položaj projekcijske ravnine v realnem koordinatnem sistemu, ima velik pomen pri transformaciji v koordinatni sistem izhodne naprave. Ta točka določa središče vidne odprtine v realnem koordinatnem sistemu in se preslika v središče okna. Razmerje med merskimi enotami v realnem koordinatnem sistemu in koordinatnem sistemu naprave je 1 : 1.

Primer uporabe: vizualizacija človeške glave

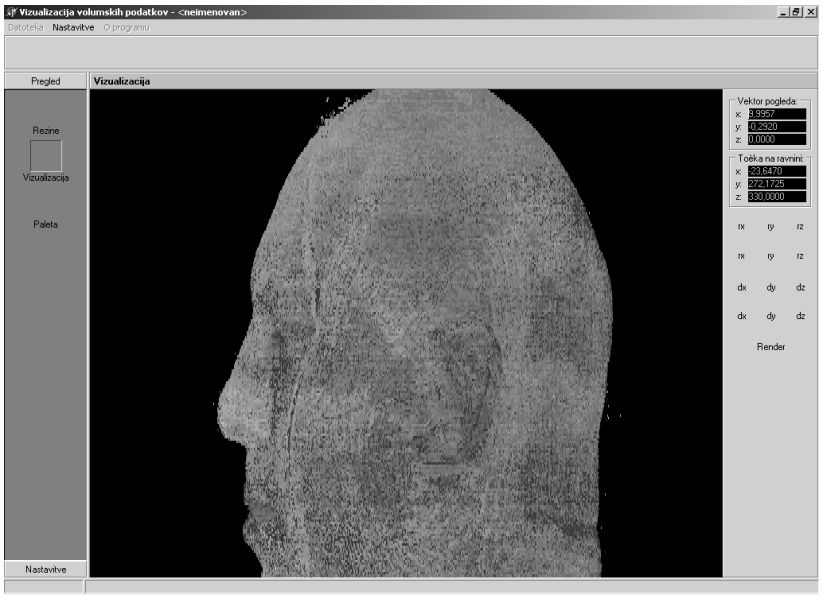
V tem poglavju predstavljamo nekaj praktičnih rezultatov naše implementacije in časovnih karakteristik programa. Uporabili bomo medicinske podatke: prostorski model človeške glave. Model je pridobljen iz 220 rezin velikosti 512×512 pikslov. Rezine so zaporedno zajete od vrha glave ($z = 0$) proti dnu ($z = -(n-1) \cdot r$), kot je običajno v računalniški tomografiji. Razdalja med dvema sosednjima rezinama je $r = 3$.

Na sliki 9 je prostor projiciran v navpično projekcijsko ravnino, nekoliko rotirano iz ravnine yz. Enaka orientacija projekcijske ravnine je uporabljena tudi na sliki 10, kjer pa je ravnina vzporedno premaknjena, tako da smo dobili prerez skozi sredino glave ($p_1=(255,255,110)$). Na sliki 11 je projekcijska ravnina bolj nagnjena, orientirana diagonalno. Središče vidne odprtine predstavlja ista točka $p_1 = (255, 255, 110)$ kot v prejšnjem primeru.

V tabeli 1 so meritve časov procesiranja v sekundah za vse tri predstavljene primere.

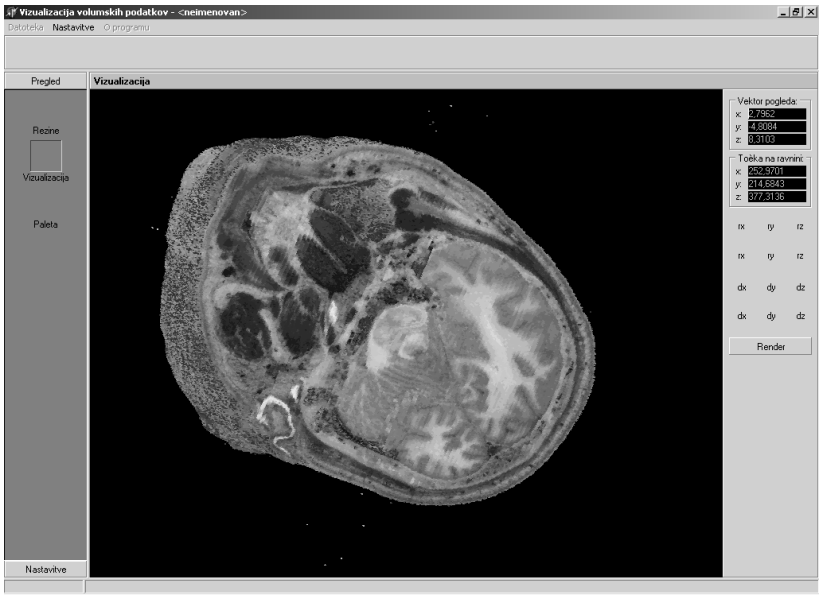
Tabela 1 Izmerjeni procesorski časi

Operacija	Slika 9	Slika 10	Slika 11
Izločanje zraka in notranjosti	1,9 s	1,9 s	1,9 s
Izločanje zakritih ploskev in zaledja	2,4 s	2,4 s	2,0 s
Dodajanje prereza	0,0 s	0,6 s	0,8 s
Hitro urejanje	2,0 s	1,4 s	1,6 s
Prikaz	3,2 s	2,4 s	2,6 s
SKUPAJ	9,5 s	8,7 s	8,9 s



Slika 9 Stranski pogled na glavo

Slika 10 Navpični prerez skozi središče glave



Slika 11 Diagonalni prerez skozi središče glave

Zaključek

V delu je predstavljena metoda za senčenje prostorskih podatkov v cenenem računalniškem okolju – na osebnem računalniku. Metoda je zmožna vizualizirati množice podatkov, ki so dovolj velike za rabo v medicinskem okolju. Vizualizacija je pospešena s hitro izločitvijo vokslov, ki niso zanimivi za uporabnika (t.i. beli voksli oz. zrak), kar se izvede v treh korakih:

1. Izločitev vokslov zraka in notranjosti objektov.
2. Izločitev vokslov, ki niso vidni glede na izbrano operacijo gledanja.
3. Tvorba prereza.

Posamezni koraki predstavljajo elementarne algoritme računalniške grafike in geometrijskega modeliranja, originalna pa je njihova povezava v učinkovit algoritem vizualizacije, ki jo omogoča uporaba originalne podatkovne strukture – seznama vidnih vokslov (VVL), v katerem je vsak voksel opremljen z atributi vidnosti posameznih mejnih ploskev. Algoritem precej oklesti VVL, katerega elemente lahko potem veliko hitreje vizualiziramo kot pa vse voksle originalne 3D matrike vokslov V. Rezultat okoli 10 s (tabela 1) za vizualizacijo $220 \times 512 \times 512$ vokslov na zmerno zmogljivem osebnem računalniku s 512 MB delovnega pomnilnika in s 600 MHz procesorjem ni slab, še posebej ne, ker gre za naš prvi poskus na področju prostorskega upodabljanja, in ker so redke obstoječe rešitve praviloma omejene na rezine manjših ločljivosti. Vendar nas čaka še precej dela, preden bo algoritem resnično učinkovit. Rezerve so še predvsem pri izločanju zakritih ploskev in še posebej pri sami vizualizaciji, pa tudi hitro urejanje ne predstavlja vedno najboljše izbire pri urejanju že precej urejenih geometrijskih podatkov. Poleg tega bo treba upoštevati še delno prosojnost, pa tudi sam uporabniški vmesnik bo treba prilagoditi zahtevam morebitnih uporabnikov.

Literatura

1. Dong F, Krokos M, Clapworthy G: *Fast volume rendering and data classification using multiresolution min-max octrees*. Eurographics 2000, Computer Graphics Forum, 19 (3): 359-366.
2. Fruhaus M: *Volume visualization on workstations: image quality and efficiency of different techniques*. Computer & Graphics, 15 (1), 1991, 101-1991.
3. Poston T, Serra L, Solaiyappan M, Heng PA: T*he graphics demands of virtual medicine.* Computer & Graphics, 1996, 20(1), 61-68.

4. Zuffo MK, Grant AJ, Lopes RD, Santos ET, Zuffo JA: *A programming environment for high-performance volume visualisation applications*. Computer & Graphics, 1996, 20(3), 385-393.
5. Lopez J, Tost D, Puig A, Navazo I:. *VOLDMI: an open system for volume modeling and vusualization.* Compuer & Graphics, 1996, 20 (5), 703-712.
6. Levoy M: *Display of surfaces from volume data.* IEEE Computer Graphics and Applications, 1988,8 (3), 29-37.
7. Žalik B, Clapworthy G, Oblonšek Č: *An Efficient Code-Based Voxel-Traversing Algorithm.* Computer Graphics Forum, 1997, 16 (2), 119-128.
8. Foley JD, Van Dam A, Feiner SK, Hughes JF: *Computer graphics – principles and practice.* Addison Wesley, 1990.
9. Samet H: *Applications of Spatial Data Structures*. Addison – Wesley, 1989.
10. Muller H, Stark M: *Adaptive generation of surfaces in volume data*. The Visual Computer, 1993, 9 (4), 182-199.
11. Hopgood RA, Duce DA: *A Primer for PHIGS*. John Wiley & Sons, 1991.