SEMINARSKA NALOGA

PRI PREDMETU

ELEKTROMAGNETIKA

Testiranje "metode končnih elementov" za izračun potencialov na primerih koaksialnega kabla, dvovoda in izseka kondenzatorja

Avtor: Sergej Maršnjak

Kazalo

1. Uvod 3
2. Izdelava uporabniškega vmesnika 4
3. Urejanje mreže kontrolnih točk 5
4. 3D transformacija in rasterizacija (rendering) 9
5. Metoda končnih elementov 12
6. Komande in delo z aplikacijo 14
7. Zaključek 16

1. Uvod

Namen izdelave seminarske naloge je bil testiranje splošne metode končnih elementov za izračun potencialov na čim več primerih razporeditve znanih potencialov, prostih nabojev, mejnih pogojev in različnih dielektričnosti. Za samo preizkušanje pa so bili izbrani primeri koaksialnega kabla, dvovoda in izseka neskončnega kondenzatorja.

Zaradi prikazovanja razultatov na računalniškem zaslonu, metoda v treh dimenzijah (3D) ne pride v poštev, saj se bi težko prikazalo potencial znotraj nekega predmeta. Tudi sicer je težko prikazovati polne predmete. Pri 3D prikazu se v večini primerov omejimo le na površino predmetov, notranjost se pa zasilno lahko prikaže le s presekom skozi predmet.

Uporabljena metoda je zaradi opisanih težav izvedena v dveh dimenzijah (2D). Izračuna se potencial v kontrolnih točkah v poljubni mreži, vmesne vrednosti med točkami pa so linearno interpolirane. Mrežo kontrolnih točk lahko poljubno generiramo, saj lahko točke poljubno dodajamo in jih premikamo. Če vrednosti potenciala na zaslonu predstavimo z črno-belim ali barvnim gradientom, recimo preko mavrice, dobimo dvodimenzionalno sliko porazdelitve vrednosti potenciala.

Če pa sedaj vsako vrednost interpretiramo kot višino terena, lahko dvodimenzionalno sliko nagnemo in jo vidimo iz profila. Tako se dosti bolje prikaže potek potenciala, saj lahko vidimo, če pada linearno ali pa kako drugače. Iz čisto 2D prikaza poteka ne moremo tako nazorno videti.

Takšen prikaz v resnici je 3D prikaz, ker pa prikazujemo le površino brez notranjosti, nimamo nobenih težav.

Sicer pa tudi sama metoda končnih elementov ni uporabljena kompletno: nimamo možnosti za določanje Neumannovih robnih pogojev, imamo pa tudi le dve površini z znanima potencialoma (Dirichletova pogoja). V veliko primerih to zadošča.

Pri izdelavi seminarske naloge so bili tako uporabljeni koraki:
- izdelava uporabniškega vmesnika
- urejevalnik mreže kontrolnih točk
- 3D transformacija in prikaz (rendering)
- sama metoda končnih elementov

2. Izdelava uporabniškega vmesnika

S samo izdelavo uporabniškega vmesnika ni bilo veliko dela, saj je vmesnik zelo enostaven. Razporeditev grafičnih elementov je popolnoma standardna in tudi delo z vmesnikom je enostavno za uporabnike, ki so sicer navajeni operacijskih sistemov tipa Windows.

Večina komand, urejanje mreže kontrolnih točk in 3D prikaz se upravlja le z miško in njenima levim in desnim gumbom. Le vpis vrednosti je potreben s tipkovnico.

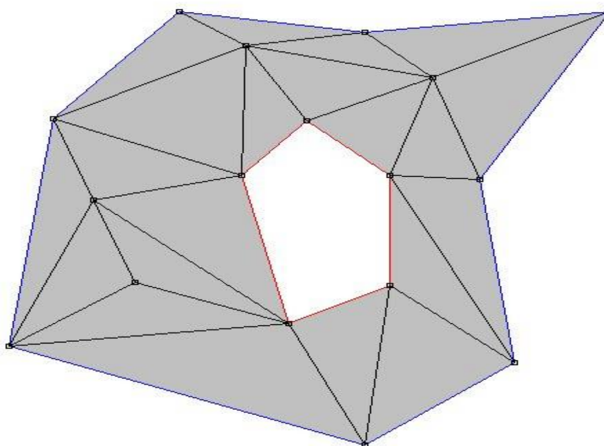
Okno aplikacije vsebuje:
- vrstico z menujem,
- komandno okno za določanje parametrov in izbiro komand in prikazom legende
- delovno okno, kjer urejamo mrežo kontrolnih točk in prikazujemo rezultate
- statusno vrstico, kjer prikazujemo razne podatke in opis posameznih komand

Slika 2.1: razporeditev grafičnih elementov uporabniškega vmesnika

3. Urejanje mreže kontrolnih točk

Urejanje mreže je v nekaterih segmentih relativno zahtevno, predvsem pri delitvi in obračanju robov.

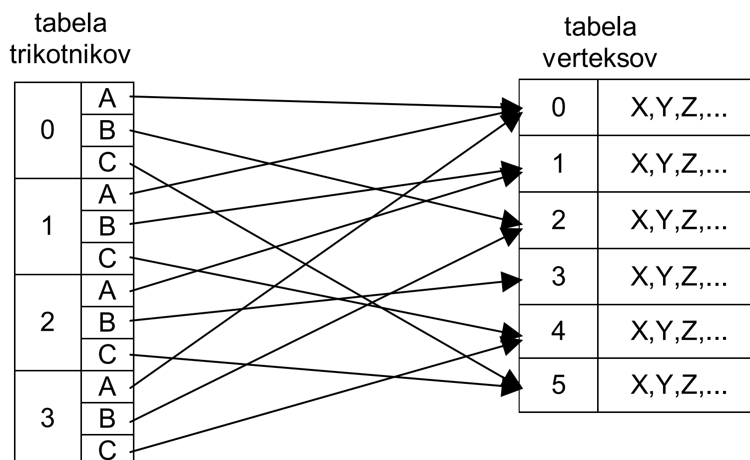
Ker je najbolj osnoven element v 2D trikotnik, saj je z njim računanje najlažje, je mreža kontrolnih točk v bistvu tudi mreža med seboj dotikajočih se trikotnikov, katerih oglišča so kontrolne točke. Vsak poljuben lik lahko razdelimo na končno število trikotnikov oziroma elementov (od tod ime metoda končnih elementov). Od števila trikotnikov je odvisna natančnost računanja, saj so vrednosti znotraj trikotnikov le linearna interpolacija vrednosti v ogliščih. Za večjo natančnost lahko trikotnike kadarkoli razdelimo na več manjših.



Slika 3.1: Razdelitev poljubnega lika na trikotnike s skupnimi oglišči

Najpomembnejši korak je zasnovati tabele, ki bodo vsebovale trikotnike in točke, saj si lahko z dobro zasnovo olajšamo delo. Problem je zaradi medsebojne odvisnosti trikotnikov in točk, kar je težko zapisati v tabelah. Zato obstaja neodvisna tabela točk ("vertex", oz. v množini "vertices"), v kateri vsaka celica vsebuje koordinate točke, in tabela trikotnikov, ki pravzaprav vsebuje po tri indekse v tabelo točk.

S tem načinom tabeliranja je dosežena le enosmerna odvisnost točk in trikotnikov: za vsak trikotnik lahko vemo, katere vertekse vsebuje, ne vemo pa za konkreten verteks, kateri trikotniki si ga lastijo. Da bi dobili trikotnike, ki si tak verteks lastijo, bi jih morali iskati v tabeli, kar malce upočasni postopke. Za neko normalno količino trikotnikov (do 1000), je čas iskanja pravzaprav minimalen.



Slika 3.2: Povezava med tabelama verteksov in trikotnikov

Poleg koordinat pa tabela verteksov vsebuje tudi vrednost potenciala na njem, ki je lahko predefiniran z mejnim pogojem, ali pa izračunan, kar je odvisno od tipa verteksa, ki je tudi zapisan v tabeli. Za potrebe pri kasnejšem računanju z metodo končnih elementov, pa se tik pred računanjem ustvarita tudi pomožna polja za spisek sosednjih elementov in verteksov.

Zapis za posamezen verteks izgleda takole:

X	Y	Z	VertexType	Voltage	NumSubFaces	pSubFaceArray	NumSubVertices	pSubVertArray

kjer *"VertexType"* pomeni tip verteksa in sicer vrednost -1 pomeni, da se potencial v tej točki izračuna, vrednost 0 pomeni predefiniran nižji potencial, in 1 predefiniran višji potencial (oba predefinirana potenciala, oziroma Dirichletova mejna pogoja, se globalno določita v komandnem oknu).
"Voltage" pomeni vrednost potenciala, *"NumSubFaces"* pomeni število sosednjih trikotnikov (face), katerih indeksi v tabelo trikotnikov se nahajajo v dinamični (spomin se alocira po potrebi) tabeli, na katero kaže polje *"pSubFaceArray"*.
"NumSubVertices" pomeni število sosednjih verteksov, katerih indeksi v tabelo verteksov se nahajajo v dinamični tabeli, na katero kaže *"pSubVertArray"*.
Seveda se pomožni tabeli sosedov ustvarita šele pred računanjem potenciala.

Zapis za posamezen trikotnik (face) pa izgleda takole:

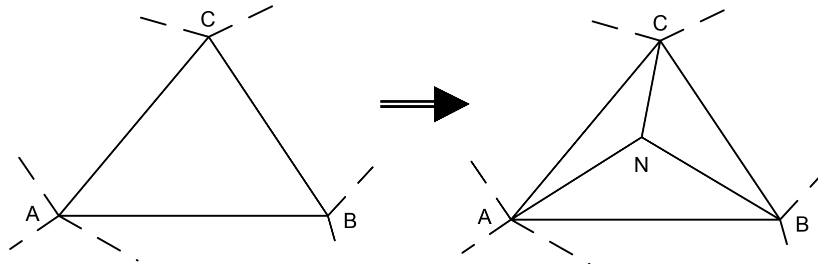
Vert[3]	EdgeType[3]	Charge	Dielectricity

kjer *"Vert[3]"* pomeni tri indekse v tabelo verteksov, ki so oglišča trikotnika, *"EdgeType[3]"* pa so tri vrednosti za tip robov; vrednosti so podobne kot pri verteksih: -1 za notranji rob (brez mejnih pogojev), 0 za rob na nižjem potencialu in 1 za višji potencial. Pravzaprav se tip verteksov ob njihovi kreaciji določi s pomočjo tipa robov iz tabele trikotnikov.
"Charge" je prostorska gostota naboja na trikotniku; vrednost je pravzaprav razmerje med gostoto naboja in ε_0, saj bi sicer morali vpisovati izredno majhna števila. Lahko je negativen ali pozitiven.
"Dielectricity" je relativna dielektričnost na posameznem trikotniku.

Zaradi čim lažjega dela lahko nove vertekse dodajamo le tako, da razdelimo nek trikotnik na tri manjše ali pa razdelimo rob in tako iz dveh trikotnikov dobimo štiri. Brisanje točk ni možno, prav tako pa ni komande "Razveljavi" (Undo). To sicer ni dobro, a bi vgraditev teh funkcij zelo zapletla izgradnjo in uporabo aplikacije. Pravzaprav pomotoma dodana nova točka pomeni le boljši rezultat in nič ne poslabša. Imamo pa še vedno možnost sprotnega shranjevanja in po želji nalaganja datotek.
Za komando Undo bi morali namreč shraniti celotno stanje oziroma kopijo mreže (za več nivojev Undo-ja torej več kopij mreže), kar ni preveč ekonomično.
Pri brisanju točke pa je problem, da s tem zbrišemo več trikotnikov in bi morali zgraditi nove, kar pa ni tako enostavno.

Dodajanje novega verteksa z razdelitvijo trikotnika na tri manjše poteka takole:
- poiščemo trikotnik, na katerega smo kliknili z miško
- zapomnimo si indekse verteksov za ta trikotnik, kot *A*, *B* in *C*
- v tabelo verteksov dodamo nov verteks *N* s koordinatami preračunanimi iz koordinat miške,
- dodamo nov trikotnik v tabelo trikotnikov in sicer z oglišči *C*, *A*, *N*
- dodamo nov trikotnik v tabelo trikotnikov in sicer z oglišči *B*, *C*, *N*
- originalnemu trikotniku spremenimo indeks verteksa *C* v *N*, tako dobimo *A*, *B*, *N*

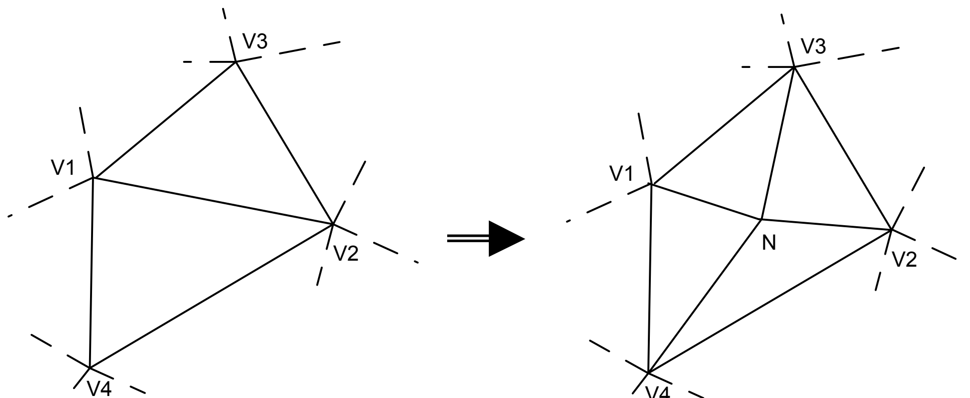
pri tem pa moramo paziti, da se tipi robov tudi po razdelitvi ohranijo. Notranji robovi tako vsi dobijo tip -1 (notranji), ostali robovi pri novih trikotnikih pa se kopirajo iz originalnega trikotnika.



Slika 3.3: K postopku razdelitve trikotnikov

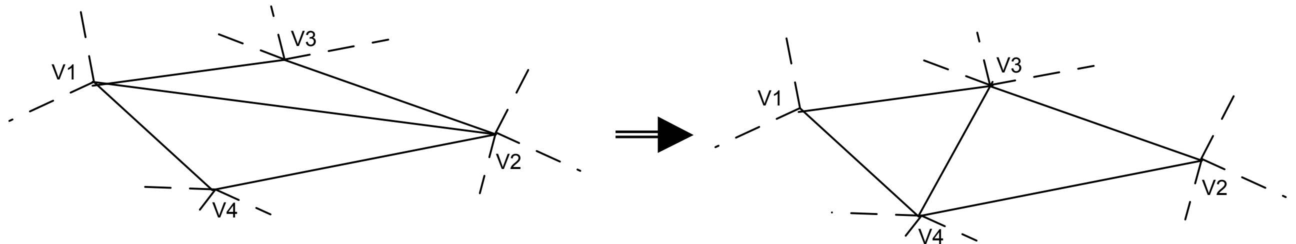
Dodajanje novega verteksa z razdelitvijo robov poteka takole:
- poiščemo rob na katerega smo kliknili, oziroma pravzaprav dva verteksa, ki tak rob tvorita. Označimo verteksa z *V1* in *V2*.
- poiščemo druga dva verteksa, ki z *V1* in *V2* tvorita štirikotnik (kot je razvidno na sliki 3.4 in jih označimo z *V3* in *V4*
- v tabelo verteksov dodamo nov verteks *N* s koordinatami preračunanimi iz koordinat miške,
- paziti moramo, kako so razporejeni ti štirje verteksi, saj so lahko v smeri urinega kazalca v vrstnem redu *V1, V3, V2, V4*, ali pa *V1, V4, V3, V2*. Od vrstnega reda je odvisno, kako bomo dodajali nove trikotnike.
- dodamo nov trikotnik v tabelo trikotnikov in sicer z oglišči *V3*-*N*-*V2* (oziroma *V4*-*N*-*V2*, če so štirje verteksi v vrstnem redu 1-4-3-2)
- spremenimo originalni trikotnik *V1-V2-V3* (oziroma *V1-V2-V4*) v *V1-N-V3* (oziroma *V1-N-V4*)
- dodamo nov trikotnik v tabelo trikotnikov in sicer z oglišči *V2-N-V4* (oziroma *V2-N-V3*)
- spremenimo originalni trikotnik *V1-V4-V2* (oziroma *V1-V3-V2*) v *V1-V4-N* (oziroma *V1-V3-N*)

Seveda pa moramo tudi tukaj paziti na tipe robov.
Krajni (notranji ali zunanji) robovi pa niso obdani s štirimi verteksi, ampak le tremi. Postopek delitve je pravzaprav podoben, le da verteksa *V3* oziroma *V4* nismo našli in tako opravimo le polovico postopka



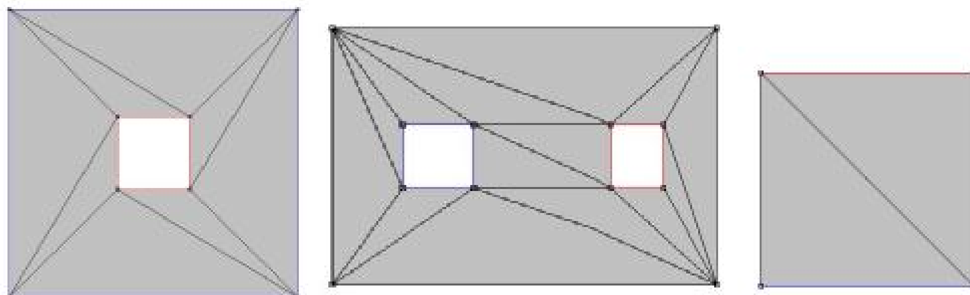
Slika 3.4 K postopku delitve robov

Potrebna je tudi tehnika obračanja robov, saj nam zelo podolgovate trikotnike spremeni v bolj enakomerne. Včasih pa, kadar hočemo določen verteks premakniti, pa je sploh potrebno obrniti rob, saj bi sicer dobili napačno usmerjene trikotnike (njihova normala bi bila usmerjena navzdol oziroma v zaslon), torej ne več trikotnik ABC ampak ACB.
Aplikacija nas na napačno usmerjene trikotnike opozarja tako, da jih obarva z rdečo barvo. Takrat moramo ali premakniti problematičen verteks, ali pa obrniti rob.
Postopek obračanja robov je zelo podoben postopku delitve robov, le da ne dodamo novih verteksov in trikotnikov, pač pa le spremenimo trikotnika *V1-V2-V3* in *V1-V4-V2* (oziroma *V1-V2-V4* in *V1-V3-V2*) v trikotnika *V1-V4-V3* in *V4-V2-V3* (oziroma *V1-V3-V4* in *V3-V2-V4*).



Slika 3.5: K postopku obračanja robov

S pomočjo teh treh postopkov delitve in obračanja lahko zgradimo poljubno mrežo kontrolnih točk (in seveda trikotnikov).
Ker pa nimamo možnosti iz nič ustvariti trikotnikov, lahko le delimo že obstoječe, se ob kreiranju novega dokumenta (ukaz “Novo” v menuju “Datoteka”) ustvari minimalna mreža za tri tipe problemov (odvisno katerega izberemo iz menuja): koaksialni kabel, dvovod in kondenzator. Ob vstopu v aplikacijo se ustvari mreža za koaksialni kabel.
Zato lahko (na žalost) testiramo metodo računanja potencialov le na teh treh problemih. Ostale primere bi bilo treba vprogramirati.



Slika 3.6: Osnovne mreže za koaksialni kabel, dvovod in kondenzator

Za lažje delo pri premikanju in ustvarjanju novih kontrolnih točk, pa lahko uporabljamo tudi pomožno mrežo (grid), katere velikost oziroma raster lahko spreminjamo v menuju. Mrežo lahko prikažemo ali skrijemo.
Lahko pa izberemo, da se pomožna mreža upošteva ob premikanju in ustvarjanju novih točk. Tako se točka ne ustvari ali premakne na mesto klika oziroma premika miške, ampak v najbližje presečišče (vozel) v mreži.

4. 3D transformacija in rasterizacija (rendering)

3D transformacija je celo področje zase in bi se o njej dalo napisati celo knjigo, zato bomo o njej govorili bolj na kratko. Sam 3D pogon je bil razvit v druge namene in zato vsebuje tudi elemente, ki v tem primeru niso uporabljeni. Pa tudi sicer bi bila podrobna razlaga izvorne datoteke za 3D pogon prezahtevna in predolga.

Tudi pri 3D transformaciji je najbolj osnoven element trikotnik, zaradi česar je zelo enostavno transformirati mrežo kontrolnih točk potenciala.
Tri točke v trikotniku se vsaka zase pretransformirajo v točke na zaslonu. Sam transformacija je linearna in kot taka transformira linijo v linijo. Zato ni treba transformirati vsakega piksla na zaslonu, ampak le omenjena tri oglišča trikotnika.

Transformacija je seveda odvisna tudi od smeri in oddaljenosti gledanja, torej mora transformacijska enačba te podatke upoštevati.

Pri postopku se uporablja matrika dimenzije 4x4, ki nam 4D vektor (dimezije X, Y in Y, z dodano dimenzijo W, ki pa je ponavadi vedno enaka 1). 4x4 je zato, da lahko ustvarimo tudi premike, saj sicer točke (0,0,0), nikakor ne bi mogli premakniti.
Ko vektor množimo z matriko, dobimo ortogonalno projekcijo, ki pa je za nekatere namene tudi ustrezna. V našem primeru pa je projekcija v perspektivi, zaradi česar je potrebno uporabiti tudi deljenje treh transformiranih koordinat s četrto.

V resnici se uporabljata dve matriki: ena vsebuje le podatke o projekciji, druga pa podatke o pogledu (oddaljenost in smer), imenovana “modelview matrix”. Pri transformaciji se tako uporablja kar produkt teh dveh matrik. Dve matriki sta zato, ker je projekcijska večinoma vedno enaka, druga pa se lahko spreminja, kadar opazovani predmet spreminjamo.

Transformacija poteka takole:

$$\begin{bmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{bmatrix} \bullet \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} X' \\ Y' \\ Z' \\ W' \end{bmatrix}$$

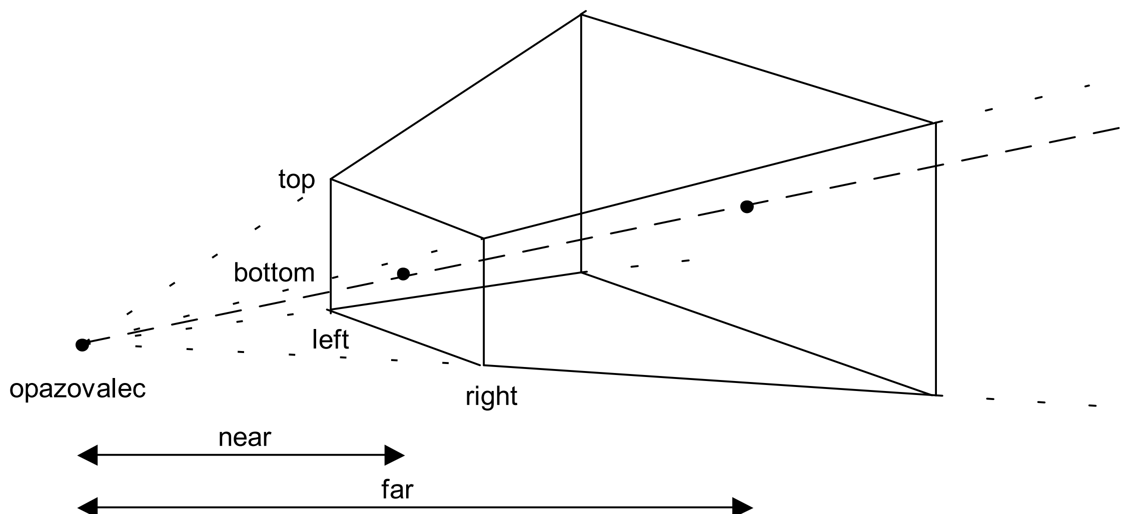
$$X'' = X' / W'$$
$$Y'' = Y' / W'$$
$$Z'' = Z' / W'$$

kjer so *m* vrednosti matrike, ki je produkt projekcijske in “modelview” matrike, X, Y, Z koordinate točke v 3D prostoru, *X’, Y’, Z’* transformirane točke pred perspektivnim deljenjem. *X”* in *Y”* je koordinata piksla na zaslonu, kamor se je pretransformirala prostorska točka. *Z”* ne pomeni globine piksla (oddaljenost od zaslona) ampak pomeni pomožno vrednost, ki se uporablja pri tako imenovanem Z bufferju (medpomnilnik globine, ki ga bomo spoznali kasneje) in pomaga pri risanju, saj bližnji piksli prekrijejo bolj oddaljene, oziroma se bolj oddaljeni niti ne rišejo.
Oddaljenost od zaslona (globino) v resnici predstavlja koordinata *W’*.

Projekcijsko matriko zapolnimo z naslednjimi vrednostmi:

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

kjer so vrednosti *n* (near), *f* (far), *l* (left), *r* (right), *b* (bottom) in *t* (top) v bistvu koordinate prisekane piramide (frustum), lepo razvidne na sliki 4.1



Slika 4.1: Prisekana piramida, ki pomaga pri transformaciji

Točke, ki so v notranjosti piramide, so po transformaciji vidne na zaslonu, zunanje pa ne. Torej je *l* levi rob zaslona, *r* desni, *b* spodnji in *t* zgornji. Točke ki so bližje kot *n* ali dlje kot *f*, se ne vidijo.
Iz slike se tudi vidi, da je bolj oddaljen pravokotnik večji kot bližnji. Zato so bolj oddaljeni predmeti videti manjši. To pa je glavna lastnost perspektivne projekcije.

Če pa hočemo spremeniti kot pogleda ali pa oddaljenost, pa moramo zapolniti še drugo matriko, že prej omenjeno "modelview matrix". Privzeto so vse matrike do spremembe enotske matrike. Rotacijo, translacijo (premik) in skaliranje dosežemo z množenjem matrike z drugo začasno matriko, ki vsebuje vrednosti, ki omogočajo tako rotacijo, translacijo in skaliranje. Množimo lahko kolikokrat hočemo in s tem dosežemo zapletenejše gibe. Seveda pa množenje matrik ni komutativno, zato ni vseeno, če najprej opravimo translacijo in nato rotacijo ali obratno.

Translacijska matrika izgleda takole:

$$\begin{bmatrix} 1 & 0 & 0 & Xt \\ 0 & 1 & 0 & Yt \\ 0 & 0 & 1 & Zt \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

kjer so vrednosti *Xt, Yt* in *Zt* koordinate vektorja premika, oziroma povedo za koliko se v posamezni smeri premaknemo.

Skalirna matrika izgleda takole:

$$\begin{bmatrix} Xs & 0 & 0 & 0 \\ 0 & Ys & 0 & 0 \\ 0 & 0 & Zs & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

kjer so *Xs, Ys* in *Zs* konstante, s katerimi skaliramo posamezne koordinate. Če so vse tri vrednosti enake, je skaliranje simetrično, krog ostane krog, sicer bi pa postal elipsoid.

Rotacijska matrika izgleda takole:

$$\begin{bmatrix} & R & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = u * u^T + \cos\theta * (I - u * u^T) + \sin\theta * S$$

$$u = \frac{v}{|v|} = \begin{bmatrix} Xr' \\ Yr' \\ Zr' \end{bmatrix}$$

$$v = \begin{bmatrix} Xr \\ Yr \\ Zr \end{bmatrix}$$

$$S = \begin{bmatrix} 0 & -Zr' & Yr' \\ Zr' & 0 & -Xr' \\ -Yr' & Xr' & 0 \end{bmatrix}$$

kjer so *Xr, Yr* in *Zr* koordinate vektorja, okoli katerega se zavrtimo za kot *θ*. *u* je normirani vektor, tako kot so *Xr', Yr'* in *Zr'* normirane koordinate vektorja. *I* je enotska (identity) matrika 3 * 3.

Vidne transformirane točke imajo koordinati *X''* in *Y''* v rangu -1..1 in koordinato *W'* v rangu 0..1.

3D preslikavo si lahko predstavljamo takole: potegnemo premico med točko, ki jo hočemo transformirati in točko opazovalca in pogledamo, kje ta premica seka bližnjo ravnino. Točka presečišča je torej torej transformirana točka.

Problem pa nastane, če je točka zadaj za opazovalcem, recimo na levi strani. Premica skozi točko in opazovalcem seka ravnino na desni strani. Tako ugotovimo, da bi se točka, ki je ne bi smeli videti (saj je za opazovalcem), vidi na zaslonu. Torej lahko točko narišemo le, kadar je njena transformirana *W'* koordinata večja od nič.
Kaj pa, če hočemo risati daljico med dvema točkama, od katerih pa je ena za opazovalcem? V tem primeru bi dobili napačen rezultat.
Zato se uporablja tehnika imenovana "viewport clipping" in pomeni razrez (obrez) likov, da se lahko pravilno narišejo. Transformirani trikotnik (pred perspektivnim deljenjem) najprej prerežemo z ravnino W' = near (seveda le, če ravnina seka trikotnik), nato pa z ravnino W' = far. Šele nato opravimo perspektivno deljenje.
Za deljenjem trikotnik še prerežemo z ravninami X'' = -1, X'' = 1, Y'' = -1 in Y'' = 1.

Na koncu tako dobimo lik, ki ima največ 7 oglišč, je pa cel na zaslonu. Seveda lahko trikotnik tudi izpade iz risanja pri clippingu, če je cel na napačni strani rezalne ravnine.

Lik moramo nato rasterizirati oziroma risati na zaslon točko za točko. Lahko ga tudi teksturiramo, kar pa v seminarski nalogi ni uporabljeno, čeprav 3D pogon to podpira. Lahko pa vsakemu oglišču določimo barvo, ki jo nato preko lika na zaslonu interpoliramo. Interpolacija ni linearna in je preobsežna za opis v tem poročilu.

Pri rasterizaciji zelo pomaga omenjeni Z oziroma globinski medpomnilnik (buffer). Ima ravno toliko polj, kot je pikslov v oknu, kamor rišemo 3D objekte. Za vsak narisani piksel, v Z buffer zapišemo pomožno vrednost globine Z'' iz transformacijske enačbe (pomožna zato, ker ni prava globina, saj ne poteka linearno z oddaljenostjo; zato danes nekatere 3D grafične kartice raje uporabljajo W buffer, ker W' pa je linearen z oddaljenostjo). Novi piksel narišemo samo v primeru, da je bližje kot je vrednost v Z bufferju. Privzeto (pred risanjem) zapolnimo Z buffer z najbolj oddaljenimi vrednostmi. Če torej najprej na določeno točko zaslona narišemo bolj oddaljen piksel, nato pa bližjega, tako bližji prekrije bolj oddaljenega. Če pa najprej narišemo bližjega, pa se bolj oddaljen sploh ne bo narisal, saj se izloči na podlagi Z bufferja. Tako so vedno vidni najbližji piksli, kar pa je tudi prav.

V aplikaciji sta koordinati X in Y za posamezno točko kar koordinati v mreži kontrolnih točk, Z koordinata pa je linearno sorazmerna s potencialom v tej točki. Faktor multiplikacije Z-ja določimo z drsnikom "Izraženost terena" v komandnem oknu. Tako so predeli z višjim potencialom videti višji kot predeli z nižjim potencialom. S tem dobimo občutek terena, da torej gledamo hribe in doline in tako imamo boljši vpogled v potek potenciala. Seveda lahko cel graf rotiramo in zoomiramo okoli njegove osi in ga tako lahko vidimo iz vseh smeri.

V tej aplikaciji se uporablja paleta 256 barv, ki predstavlja mavrico. Indeks 0 v paleti pomeni modro barvo, indeks 255 pa rdečo barvo. Vmesne vrednosti prehajo od modre, preko zelene in rumene v rdečo. Zato so oglišča trikotnikov obarvana z sivimi odtenki od 0 do 255, ki se nato s pomočjo palete izrišejo v mavričnih odtenkih.

Ista mavrica je izrisana tudi v komandnem oknu kot legenda. Poleg nje so zapisane vrednosti potenciala, torej, na katerem potencialu je posamezna barva.

V osnovi je toliko o 3D grafiki dovolj. Omeniti bi bilo potrebno le še to, da danes večino 3D transformacije in rasterizacije opravi grafična kartica. V tej aplikaciji ta strojno pospešen sistem ni uporabljen zaradi kompatibilnosti in prenosljivosti, saj deluje na praktično vsakem računalniku tipa PC.

5. Metoda končnih elementov

Ta metoda je zelo dobro obdelana v knjigi prof. dr. Antona Sinigoja, ELMG polje in se zato tukaj vanjo ne bomo preveč spuščali. Potrebno je le razložiti način uporabe metode s podatki o izdelani mreži kontrolnih točk. Razlika je tudi v tem, da tukaj leva in desna stran nista množeni z ε_0, pač pa se gostota prostega naboja razume kot kvocient med gostoto in ε_0, saj bi sicer za naboj morali pisati izredno majhna števila (okoli 10^{-12}). Tako se uporabljajo le relativne dielektričnosti.

Ker je kontrolnih točk manj kot je vseh točk v mreži, saj so nekatere na predefinarih potencialih, je treba najprej prešteti te točke. Število označimo z *n*. Nato alociramo (rezerviramo) spomin za matriko *m* in vektor *p*, kot sta označena v knjigi.

Sedaj moramo matriko in p napolniti s pravilnimi vrednostmi. S »for« zanko nad *i* gremo od 0 do n-1 preko vseh kontrolnih (nepredefiniranih) točk. Za vsako točko izračunamo integral s prostim nabojem in integral z gradienti med *i*-tim verteksom in vsemi njegovimi predefiniranimi sosednjimi trikotniki. Integral z mejnim Neumannovim pogojem se ne uporablja. Vsoto integralov zapišemo na *i*-to mesto v vektor *p*.
Nato za vsako točko izračunamo integrale z gradienti med *i*-tim verteksom in vsemi njegovimi nepredefiniranimi sosedi in jih zapišemo v *i*-to vrstico v matriko *m*.
Na koncu rešimo sistem po Gaussovi eliminacijski metodi in vsakemu nepredefiniranemu verteksu pripišemo zanj izračunano vrednost.

Samo računanje integralov pa je pravzaprav kar enostavno, saj so enačbe zaradi uporabe vektorjev relativno enostavne. Nikjer ne nastopajo koreni in trigonometrične funkcije.

Pri vseh integralih nastopajo površine trikotnikov. Površina trikotnika z oglišči *V1, V2* in *V3*, od katerih uporabimo samo X in Y koordinati (brez Z, ker uporabljamo 2D metodo), se izračuna takole:

$$S = 0.5 * ((V1_X - V2_X) * (V1_Y + V2_Y) + (V2_X - V3_X) * (V2_Y + V3_Y) + (V3_X - V1_X) * (V3_Y + V1_Y))$$

Integrali s prostimi naboji so tako enostavno vsota produktov naboja na sosednjih trikotnikih s površino teh trikotnikov (in deljeni s 3).

Integrali z gradienti pa so vsota členov za vse sosednje trikotnike in njihova oglišča (s predefiniranimi ali nepredefiniranimi potenciali, odvisno od tega, ali polnimo matriko *m* ali pa vektor *p*). Posamezen člen se izračuna takole:

$$\varepsilon_k * S_k * (\mathbf{e}_i * \mathbf{e}_j) / (l_i * l_j) = \varepsilon_k * S_k * Cos / (I1 * I2)$$

$$Cos = (V1b_X - V1a_X) * (V2b_X - V2a_X) + (V1b_Y - V1a_Y) * (V2b_Y - V2a_Y)$$

$$I1 = (V1_X - V1a_X) * (V1b_Y - V1a_Y) - (V1_Y - V1a_Y) * (V1b_X - V1a_X)$$

$$I2 = (V2_X - V2a_X) * (V2b_Y - V2a_Y) - (V2_Y - V2a_Y) * (V2b_X - V2a_X)$$

kjer je *k* indeks sosednjega trikotnika in *j* indeks sodednjega verteksa, ki je v lasti *k*-tega trikotnika. ε_k je relativna dielektričnost za *k*-ti trikotnik, tako kot je S_k površina tega trikotnika. V1 je *i*-ti verteks, V1a in V1b sta druga dva, od *i*-tega sosednja verteksa v *k*-tem trikotniku, V2 je *j*-ti verteks, V2a in V2b sta druga dva, od *j*-tega sosednja verteksa v *k*-tem trikotniku.

Zanimiva je tudi linearna intepolacija potenciala v notranjosti posameznega trikotnika:

$$\text{Voltage} = K1 * \text{Voltage1} + K2 * \text{Voltage2} + K3 * \text{Voltage3}$$

$$K1 = \frac{(V2_Y - V3_Y) * (X - V2_X) + (V3_X - V2_X) * (Y - V2_Y)}{(V2_Y - V3_Y) * (V1_X - V2_X) + (V3_X - V2_X) * (V1_Y - V2_Y)}$$

$$K2 = \frac{(V3_Y - V1_Y) * (X - V3_X) + (V1_X - V3_X) * (Y - V3_Y)}{(V3_Y - V1_Y) * (V2_X - V3_X) + (V1_X - V3_X) * (V2_Y - V3_Y)}$$

$$K3 = \frac{(V1_Y - V2_Y) * (X - V1_X) + (V2_X - V1_X) * (Y - V1_Y)}{(V1_Y - V2_Y) * (V3_X - V1_X) + (V2_X - V1_X) * (V3_Y - V1_Y)}$$

kjer so *Voltage1, Voltage2* in *Voltage3* potenciali na ogliščih trikotnika, *V1, V2* in *V3* so oglišča trikotnika, *X* in *Y* pa sta koordinati, kjer računamo interpolirani potencial. V tej aplikaciji *X* in *Y* dobimo kot koordinate miške, kadar potujemo preko trikotnika. Tako se nam v komandnem oknu izpisuje potencial na točki pod miško.

6. Komande in delo z aplikacijo

Aplikacija je bila izdelana z jezikom “Watcom C/C++ v11.0” in sicer za platformo Win32, kamor spadajo Windows 95, 98, NT, Me, 2000 in XP.

Delno je bil uporabljen jezik C/C++, delno pa (predvsem za časovno kritične rutine v 3D pogonu) tudi Assembler.

Za delovanje aplikacije ni potrebna posebna oprema, le računalnik naj bo kolikor toliko hiter (recimo vsaj Pentium 600) in grafična resolucija naj bo vsaj 1024x768 točk. Aplikacija tudi ni požrešna s spominom, saj ga v povprečju porabi manj kot 4MB.

Kot je že bilo omenjeno, aplikacija vsebuje menuje in komandno okno.

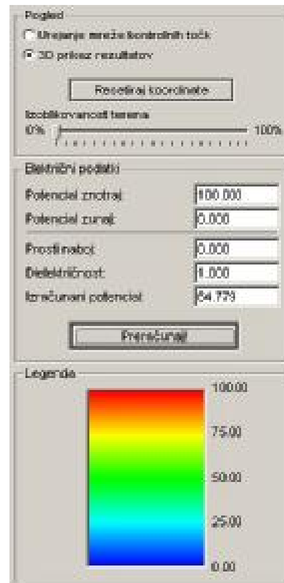
V menuju *Datoteka* so komande:
- *Novo*, ki zavrže do sedaj izgrajeno mrežo in ustvari novo in sicer odvisno od pod-izbire kot koaksialni kabel, dvovod ali kondenzator,
- *Odpri*, ki odpre shranjeno datoteko z diska,
- *Shrani*, ki shrani datoteko na disk (ob prvem shranjevanju nas vpraša za ime datoteke),
- *Shrani kot*, s katero imamo možnost shraniti datoteko pod novo ime
- *Tiskaj*, ki pa zaenkrat še ni implementirana, bi pa natisnila vsebino okna na papir
- *Izhod*, ki predstavlja izhod iz aplikacije

poleg teh pa se v tem menuju pojavijo tudi štiri nazadnje odprte datoteke. Tako jih lahko hitreje odpremo, saj nam ni treba brskati po disku.

V menju *Pogled* so komande:
- *Prikaži mrežo*, s katero lahko vklopimo ali izklopimo pomožno mrežo, ki pomaga pri orientaciji
- *Upoštevaj mrežo*, s katero lahko vklopimo oziroma izklopimo tako imenovan “snap to grid”, kar pomeni, da bo premikana točka skočila v najbližje presečišče v pomožni mreži
- *Velikost mreže*, s katero določimo raster pomožne mreže. Pri tem moramo vedeti, da na zaslonu koordinate po Y gredo od -1 do 1, po X pa sicer različno, odvisno od razmerja med širino in višino okna

V menuju *Pomoč* so komande:
- *Jezik*, s katerim lahko izberemo pogovorni jezik v aplikaciji kot angleški ali slovenski. Privzeto je jezik slovenski
- *Vizitka*, ki prikaže osnovne podatke o aplikaciji in njenem avtorju

Komadno okno na sliki 6.1 vsebuje konkretne podatke v zvezi s prikazom in delom z mrežo kontrolnih točk.



Slika 6.1: Izgled kontrolnega okna

Komande so:

- *Urejanje mreže kontrolnih točk*, s katero izberemo, da v delovnem oknu urejamo in prikazujemo mrežo
- *3D prikaz rezultatov*, s katero izberemo prikaz reliefa v 3D načinu. Tukaj ne moremo premikati kontrolnih točk.
- *Resetiraj koordinate*, s katero nastavimo pogled v 3D načinu na privzete vrednosti, to je na pogled iz vrha brez terena. Le s takimi vrednostmi je enostavno meriti potencial na posamezni točki, saj bi sicer morali izvajati težje transformacije. Kakor hitro torej spremenimo pogled, v 3D prikazu ne moremo več meriti interpoliranega potenciala, dokler spet ne uporabimo tele komande
- *Izoblikovanost terena*. Z njim določimo faktor multiplikacie Z koordinate in s tem izbočenost terena. Vrednost 0% pomeni brez reliefa.
- *Potencial znotraj* in *Potencial zunaj*. Imeni sicer nista povsem ustrezni, saj izhajata iz osnovnega primera koaksialnega kabla, sicer pa pomenita višji in nižji potencial, ki sta v mreži označena z rdečo in modro črto.
- *Prosti naboj* nam odčitava gostoto prostega naboja na posameznem trikotniku, kadar gremo z miško čezenj.
- *Dielektričnost* nam odčitava relativno dielektričnost na posameznem trikotniku, kadar gremo z miško čezenj.
- *Izračunani* potencial nam odčitava interpolirani potencial na posameznem trikotniku, kadar gremo z miško čezenj.
- *Preračunaj* nam izvede izračun potencialov po metodi končnih elementov

Poleg tega pa je v komandnem oknu tudi legenda, ki pove, kakšno vrednost predstavlja določena barva.

Kako pa premikamo vertekse, delimo trikotnike in ostalo? Vse lahko opravimo z miško. Komande so:

- enojni klik levega gumba v bližini roba nam ta rob obrne
- če z levim gumbom kliknemo verteks in nato premikamo miško s pritisnjenim gumbom (drag and drop), s tem premikamo verteks po zaslonu
- dvojni klik levega gumba na površino trikotnika nam odpre pogovorno okno, kamor vpišemo gostoto prostega naboja in relativno dielektričnost za ta trikotnik
- enojni klik desnega gumba v bližini roba nam ta rob razdeli
- enojni klik desnega gumba v trikotniku (ne v bližini roba) nam ta trikotnik razdeli

7. Zaključek

Zaključki, kot sem jih lahko izvlekel iz preizkusov so (večina jih potrjuje že znano teorijo):

- Potek potenciala v kvadratnem koaksialnem kablu ni linearen (v okroglem bi bil logaritmičen), potek med vogaloma žile in oklopa pa je še bolj nelinearen, saj ekvipotencialne ploskve postajajo proti notranjosti vedno bolj okrogle. Torej narava »teži« k odstranjevanju ostrih robov.
- Če del kondenzatorja v smeri plošč zapolnimo z drugačnim dielektrikom, tam (kot sledi že iz mejnega pogoja za normalno komponento za jakost električnega polja) nastane skok jakosti polja in s tem drugačen nagib poteka potenciala, ki sicer v posameznem dielektriku poteka linearno.
- V dvovodu se lepo vidi potek ekvipotencialnih ploskev, ki so pravokotne na silnice električnega polja, tako da vidimo znano sliko.
- Prosti naboji (tako negativni kot pozitivni), ki jih vnesemo v mrežo, generirajo svoje električno polje, kar je lepo razvidno v poteku potenciala. »Hribi«, ki nastanejo, imajo obliko hiperbole, kar tudi izhaja iz teorije.

Kot literaturo lahko navedemo samo:

– “ELMG Polje”, prof. dr. Antona Sinigoja,
– in internetsko specifikacijo (URL naslova se ne spominjam več): “The OpenGL Graphics System: A Specification (Version 1.1)”, avtorja Mark Segal in Kurt Akeley

Ostale literature se ne spominjam več, saj sem znanje o programiranju in 3D grafiki pridobival dolga leta, tako da sem v fizični in elektronski obliki ohranil le posamezne odseke in citate. Večino znanja sem pridobil kot samouk, nekaj pa tudi iz člankov na internetu.

Za konec prilagam še nekaj tako imenovanih “screen-shot”-ov za različne primere.