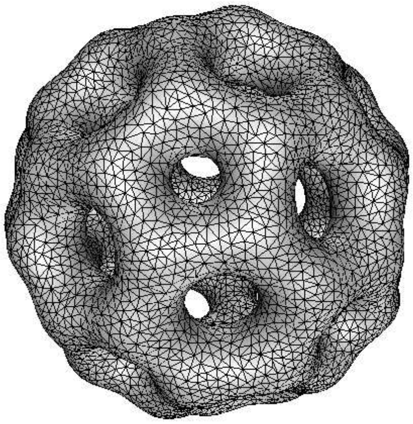
3D Visualisering

Jesper James Jensen



Juli 1995
Elektronisk Institut
Danmarks Tekniske Universitet

Abstract

Interaktiv visualisering af komplekse tre-dimensionale strukturer er en disciplin i stor udvikling. Dette skyldes ikke mindst den konstante udvikling der er inden for hardware til generering af tre-dimensional grafik.

Dog er udviklingen i de datamængder der ønskes visualiseret mindst lige så stor, da den større regnekraft giver mulighed for nøjagtigere dataopsamling og processering af større datamængder.

Der vil derfor stadigvæk være et behov for metoder der tillader en visualisering af meget komplicerede modeller og strukturer på udstyr der er begrænset i ydeevne. I denne rapport, vil metoder der muliggør dette dels ved hjælp af simplificering af polygon meshes, dels ved hjælp af tekstur mapping blive beskrevet. Disse metoder muliggør en afvejning mellem objekternes visuelle nøjagtighed og visualiseringens hastighed.

Indholdsfortegnelse

Kapitel 1

Volumen visualisering

Volumen visualisering er betegnelsen for metoder hvormed forskellige former for tre dimensionale data kan visualiseres. Volumen visualisering genererer billeder fra et volumen datasæt der indeholder information om en models egenskaber, ikke blot på overfladen, men også indeni. Disse datasæt er ofte opbygget af kubiske voxels (tre dimensionale volumen elementer). I dette afsnit vil forskellige metoder hvormed tre-dimensionale datastrukturer kan visualiseres blive diskuteret.

1.1 Skalære glyphs

Skalære glyphs er en teknik, hvor værdien for hvert datapunkt visualiseres med et simpelt objekt (kaldet *glyph*). Størrelsen af disse objekter bestemmes af data værdien. Objekterne kan farvelægges i forhold til samme datasæt, eller et andet datasæt kan bruges hvormed korrelationer kan undersøges. Da praktisk taget ingen udregninger skal foretages, er dette en beregningslet metode. Dog kræves meget af visualiseringsdelen, afhængigt af kompleksiteten af de brugte glyphs. Visualisering af vektorfelter foretages gerne med metoder lig denne, hvor vektorerne visualiseres med pile af forskellig længder og retninger.

1.2 Iso-flader

Denne teknik producerer polygonbaserede objekter i datasættet for iso-flader svarende til samme dataværdi. Iso-fladerne kan farvelægges i forhold til niveauet, eller de kan farvelægges i forhold til et andet datasæt med brug at tekstur teknikker. Brugen af tekstur giver mulighed for at finde korrelationer mellem forskellige datasæt. Til generering af iso-fladerne bruges gerne Marching Cubes algoritmen [LC87].

1.3 Snitflader

Denne teknik giver mulighed for at visualisere en tværsnit af et data-volumen med en snitflade. Snitfladen defineres som et rektangulært plan gennem volumenet. Dataværdier på et grid på dette snitplan findes med interpolation i de originale data.

1.4 Ray tracing

Denne teknik visualiserer ved at kaste en række 'stråler' ind igennem datasættet, for dermed at emulere den måde virkelige objekter bliver set på. Datasættet bliver visualiseret som værende semi-transparant, hvor værdierne i de respektive voxels definerer transparansen for den pågældende voxel. Denne metode kan give meget realistiske resultater, men er meget beregnings krævende. På maskiner der kan lave hardware tekstur mapping, er det muligt udnytte dette til ray-tracingen og få real-time visualisering. Denne metode har fordele fremfor visualisering med iso-flader da alle strukturer i volumet bidrager til visualiseringen. Billeder genereret med disse 'ægte' volumen visualiserings metoder, kan godt vøre svære at fortolke hvis alle intensitetsniveauer medtages. Dette kan afhjælpes ved kun at visualisere data-værdier indenfor bestemte interesse niveauer. Real-time volumen visualisering er uhøre beregningskrævende, og vil i praksis kun kunne udføres af udstyr i Silicon Graphics 'Reality Engine' klassen.

Kapitel 2

Virtual Reality

Virtual reality applikationer er baseret på en polygon-repræsentation af den verden og de objekter der skal visualiseres. I de følgende afsnit vil preformance relaterede aspekter omkring generering og visualisering af disse blive diskuteret.

2.1 Ydeevne

I forbindelse med interaktiv visualisering, er en af de vigtigste parametre at opretholde en så høj opdateringsfrekvens at en real-time illusion kan opnås. Dette vil for de fleste applikationer betyde at de skal være i stand til at generere mindst 10-15 billeder i sekundet. Hvis det objekt der skal visualiseres er opbygget af polygoner, og der benyttes hardware der er i stand til at generere 100.000 polygoner i sekundet, bør den visualiserede scene derfor ikke indeholde over 10.000 polygoner, for at kunne opretholde en høj nok opdaterings frekvens. I dette kaptiel vil kort blive beskrevet hvorledes man kan vurdere en given visualisering applikations ydeevne.

2.2 Flaskehalse

Der er flere potentielle flaskehalse i forbindelse med computer visualisering af tre dimensionale objekter. Følgende operationer er involveret i visualiseringen af tre dimensionale polygon objekter:

- Geometriske operationer:
  - koordinat transformationer

    - klipning / lys udregninger

- Skærm opdatering:
    - polygon fyldning
    - dybde buffering
- Applikations operationer

De tre hovedpunkter kan hver især være flaskehalse. Hvis en applikation opdaterer billederne hurtigere når dets vindue bliver gjort mindre, er applikationen *fill*-begrænset, dvs. at selve rasteriseringen af polygonerne ikke kan foretages hurtigt nok. Hvis vindues størrelsen ingen indflydelse har på opdaterings frekvensen, er applikationen *geometri*-begrænset eller *applikations*-begrænset. En fill-begrænset applikation kan optimeres ved at reducere dybde kompleksiteten af scenen, eller ved at bruge simplere grafik operationer hvor muligt (f.eks. ved ikke at bruge dybde buffering for baggrunde). En geometri-begrænset applikation kan accelereres ved at bruge simplere modeller f.eks. med færre vertices eller færre lyskilder.

Hvis en applikation bruger det meste af tiden pr. billede på andre ting end at tegne, er programmet applikations begrænset. Og selve koden må optimeres med et profilerings værktøj.

2.3 Dobbelt-buffering

Brugen af *Dobbelt-buffering* kan også give anledning til performance relaterede problemer. Et system med dobbelt-buffering bruger to komplette farve buffere, den ene vises mens den anden genereres. Når opdateringen af en buffer er slut, byttes de to buffere. Dette gør at selve opdaterings processen ikke vises. Ud over at skifte buffere, vil dobbelt-buffer funktionen også vente indtil skærmens refresh er ovre, hvilket tillader den nye buffer at blive vist helt startende fra toppen. På et system med en 60 Hz skærmopdaterings frekvens, er den hurtigste opdatering fælgelig 60 billeder i sekundet. Hvis alle billeder fra applikation kan slettes og genereres på under 1/60 sekund vil applikationen kun være begrænset af skærm opdateringen. Hvad der ofte sker på et system med dobbelt buffering, er at billederne bliver så komplicerede at de ikke kan genereres på under 1/60 sekund. Således at hvert billede bliver vist mere end én gang. Den tid der bruges fra et billede er genereret færdigt til en ny skærmopdatering starter er dermed spildt.

Skærm opdateringen er konstant. Dette kan give uventede resultater med hensyn til applikationers hastighed. Hvis en monitor har en 60 Hz refresh rate, kan applikationer køre med 60 Hz, 30 Hz, 20 Hz osv. (60/1, 60/2, 60/3, ...). Dette betyder at en applikation kan springe fra en 60 Hz opdatering til 30 Hz, bare der er et par polygoner for meget i scenen. Et andet problem opstår hvis en applikation er tæt på en af de 'magiske' tider (1/60 s, 2/60 s, ...) i opdatering hastighed. I dette tilfælde kan tilfældige variationer gøre at nogle opdateringer bruger mindre tid og andre opdateringer bruger mere tid. Dette fører til en iregulær opdaterings frekvens, der visuelt vil være meget forstyrrende. I et sådant tilfælde kan løsningen være at indsætte et lille delay, der gør at alle opdateringer klares inden for samme langsommere antal skærm opdateringer.

Kapitel 3

Mesh simplificering

Siden introduktionen af standard metoder til iso-flade generering, har et af de største problemer været at håndtere det store antal trekanter (eller polygoner) der bliver genereret. I dette afsnit vil forskellige metoder hvormed denne reducering kan foretages blive gennemgået, og resultatet af implementeringen af en simpel trekants decimerings algoritme vil blive præsenteret.

3.1 Introduktion

Brugen af *Marching Cubes* (MC) algoritmen originalt foreslået af Lorensen og Cline [LC87] bliver anset for at være standard løsningen til problemet at generere isoflader fra et volumen datasæt. Marching Cubes er en meget praktisk og simpel algoritme, og mange implementeringer er tilgængelige både som del af kommercielle software pakker som f.eks. IDL, og som public domain software.

På trods af algoritmens store udbredelse, har standard Marching Cubes algoritmen nogle mangler. Grundet algoritmens lokale overflade rekonstruktion, genereres f.eks. ingen information om objekters interkonnektivitet. Den høje grad af data fragmentation kombineret med en meget stor datamængde, kan derfor forhindre interaktiv rendering når høj opløsnings datasæt skal visualiseres.

En af de største ændringer der er sket siden Marching Cubes blev introduceret i 1988 er den mængde data der skal processeres for at producere iso-flader. Medicinsk udstyr, der genererer datasæt med en størrelse på $512 \times 512 \times [\leq 512]$ voxels er i dag kommercielt tilgængeligt, og udviklingen vil snart bringe udstyr i stand til at producere datasæt med en størrelse på $1024 \times 1024 \times [\leq 1024]$ voxels, eller med andre ord 1 Gigavoxels pr. datasæt. Selv om en given isoflade ikke krydser enhver voxel, er det nemt at forstå hvorledes der kan blive genereret mere end en million trekanter pr. iso-flade. Selv dagens state-of-the-art hardware er ikke i stand til at manipulere sådanne datamængder med en hastighed der gør interaktivitet praktisk anvendelig. Disse hindringer har medført forskning rettet mod at reducere det antal trekanter der bliver produceret af Marching Cubes algoritmen.

Mange metoder er blevet foreslået til at foretage denne simplificering. En gruppe af disse metoder kan klassificeres som *adaptive* teknikker, hvor celle størrelsen lokalt ændres til iso-fladens form (Muller og Stark, [MS93]), eller datasættet bliver inddelt i høj- og lavinteresseområder, hvor flere primitiver bliver dannet i de udvalgte høj-interesse områder.

En anden klasse af metoder er *filtrerings* metoderne, der tager trekanterne i det dannede mesh, og filtrerer disse med henblik på at sammenlægge eller eliminere dem. Filtreringsmetoderne kan ydermere deles op i følgende grupper:

1. *coplanar facet merging*, hvor polygoner filtreres ved at søge efter, og sammenlægge coplanare nabopolygoner.

2. *eliminering af små trekanter*, hvor iregulariteten af fladerne reduceres ved at eliminere de mindste trekanter. Disse produceres hvor isofladerne passerer tæt på en vertex eller kanten af en kubisk celle. Reduceringen foretages ved at ændre på det genererede mesh, for en række udvalgte vertices, således at de små trekanter parvis vil degenerere til linier. Op til 40% reduktion af antallet af trekanter kan opnås på meshes genereret med Marching Cubes algoritmen med denne metode.

3. *approximeret overflade fitting*, baseret på en afvejning mellem data reduktion og præcision af den genererede repræsentation, hvor fejlkriterier bruges til at måle brugbarheden af den approximerede flade.

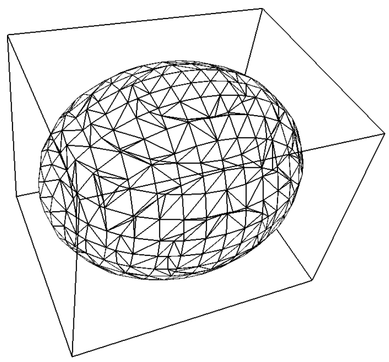
Schroeder et al. [SZL92] foreslår en algoritme baseret på flere filtrerings iterationer, der lokalt analyserer geometrien og topologien af et trekants mesh, og fjerner vertices der opfylder et minimums afstands- eller krumnings kriterium. Fordelen ved denne fremgangsmåde, er at et vilkårligt reduktions niveau kan opnås, hvis et tilstrækkeligt grovt approximations kriterium bliver sat. Schroeder har opnået op til 90% reduktion med en approximations fejl på under én voxels størrelse.

En lignende fremgangsmåde bruges af Hoppe et al. [HDD+93], hvor mesh-optimeringen opnås ved at evaluere en energi funktion over meshet, og derefter at minimere denne funktion, ved enten at flytte vertices, eller at kollapse/bytte kanter. Begge disse to fremgangsmåder kræver en topologisk repræsentation af det mesh der skal decimeres eller en approximation af denne.

I de følgende afsnit vil den implementerede rutine til at decimere et trekants mesh med blive beskrevet. Rutinen er en filtrerings metode baseret på parvis eliminering af trekanter udvalgt efter forskellige kriterier.

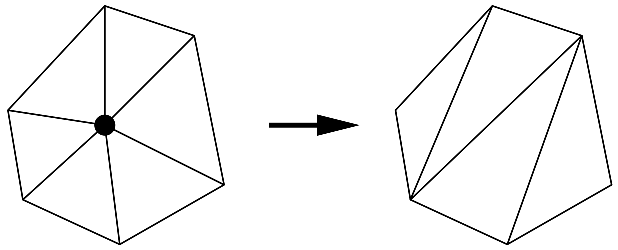
3.2 Decimering

Et typisk mesh genereret med en Marching Cubes algoritme, kan se ud som på figur 3.1. Som det ses er alle trekanterne af samme størrelses orden, og den gennemsnitlige sidelængde er ca. lig voxel sidelængden. Det ses at en relativ stor del af trekanterne meget små i forhold til voxel størrelsen. Ved blot at eliminere disse 'små' trekanter, kan opnås en stor reduktion af meshet uden væsentlig reduktion af den visuelle kvalitet.



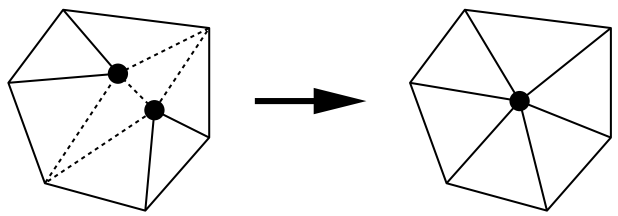
Figur 3.1: Mesh genereret af Marching Cubes.

I praksis er der to måder at simplificere et sådant mesh på. Disse baserer sig på eliminering af enten vertices eller trekanter. Den i litteraturen mest omtalte metode beskrevet af Schroeder et al. [SZL92] er vertex baseret. Denne udfører en række gennemløb af det givne mesh, og fjerner vertices der opfylder specificerede afstands- og krumningskriterier. De resulterende huller udfyldes derefter med en trianguliserings process. Dermed bliver antallet af trekanter og vertices reduceret. Processen kan eventuelt gentages iterativt med ændrede grænseværdier indtil den ønskede kompleksitet af meshet er opnået. På figur 3.2 er vist hvorledes en vertex kan fjernes og hvorledes det resulterende hul i meshet kan re-trianguliseres. I det viste eksempel bliver antallet af trekanter reduceret fra seks til fire. Trianguliseringen foretages af en tre-dimensional rutine således at der vælges den optimale overflade forhold til den fjernede vertex' position.



Figur 3.2: Fjernelse af en vertex og re-triangulisering.

Den fremgangsmåde der er blevet implementeret, decimerer med en metode baseret på parvis eliminering af trekanter. Denne reducering kan foretages ved brug af de samme kritierier som ved fjernelsen af vertices, eller klassificeringen kan baseres på trekanternes opfyldelse af opstillede grænseværdier for f.eks. størrelse og krumning. På figur 3.3 ses hvorledes to vertices kan slås sammen til én, og to trekanter elimineres. Den nye vertex kan placeres midt imellem de to gamle, eller normalerne for de to sammenlagte vertices kan bruges til at finde en position der bedre passer til det oprindelige mesh i forhold til krumningen af dette.



Figur 3.3: Sammenlægning af to vertices.

Metoden med vertex-sammenlægning er valgt, grundet simplere implementering, og forventet hurtigere ekseveringtid. Alle de i litteraturen beskrevne re-trianguliserings metoder har en $O(n^2)$ kompleksitet i antallet af vertices, hvorimod den implementerede algoritme på grund af krydsrefererende datastrukturer har en $O(n)$ kompleksitet.

3.3 Datastrukturer

For at reducere decimeringtiden, har det været nødvendigt med opbygningen af komplekse datastrukturer til at beskrive meshets vertices og trekanter med. Strukturerne er baseret på krydsreferering mellem alle vertices og alle trekanter. Dermed undgås søgninger gennem meshet når der skal findes tilstødende vertices og trekanter. Opbygningen af disse datastrukturer er for store meshes mere tiddskrævende end selve decimeringen, men nedsætter den samlede beregningstid betragteligt. For alle vertices gemmes information om:

- Koordinater
- Normalvektor
- Tilstødende vertices
- Tilstødende trekanter
- index
- Kant/loop flag

Da det maksimale antal tilstødende trekanter og vertices er ukendt, bliver alle vertices initialiseret med plads til et vurderet maksimalt antal. Brugen af denne datastruktur, kræver dermed over 200 bytes lager for hver vertex. For hver trekant gemmes information om:

- Hjørne vertices
- Normalvektor
- Tilstødende trekanter
- Sidelængder
- Tilhørende objekt

Disse informationer gør, at det er let at fjerne en given trekant fra meshet, da alle de trekanter og vertices der bliver berørt af ændringen umiddelbart kan findes og opdateres. Ydermere kan de genererede isoflader ved hjælp af disse informationer nemt inddeles i seperate objekter der kan behandles individuelt.

3.4 Klassifikation

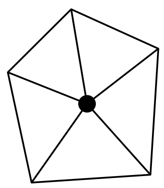
En decimering af et trekant mesh baseres normalt på en klassifikaiton af elementerne i meshet. Trekanter og vertices bliver klassificeret på to niveauer, dels ud fra deres konnektivitet og placering i meshet, dels ud fra en vurdering af meshets karakter på det givne sted i meshet.

Schroeder deler vertices i et mesh op i fem klasser: simple, komplekse, ende, indre kant og hjørne vertices. Klassificeringen af trekanterne kan efterfølgende foretages ud fra klassificeringen af deres vertices. Decimeringen kan derefter foretages med rutiner der kan tage hensyn til de forskellige klasser af vertices og trekanters specielle interkonnektivitet.

3.4.1 Klassifikation vha. konnektivitet

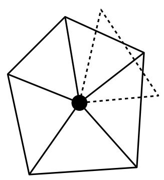
Ved at undersøge konnektiviteten i meshet, kan flere forskellige typer vertices der skal tages hensyn til identificeres:

- En *simpel vertex* er fuldstændigt omkranset af en cyklus af trekanter, hvor hver af disse trekanter støder op til to andre trekanter i cyklen. En simpel vertex kan både sammenlægges og elimineres.



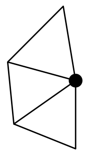
Figur 3.4: Simpel vertex.

- En *kompleks vertex* er en simpel vertex der ydermere bliver brugt af mindst én trekant der ikke er i cyklen af omkransende trekanter. En kompleks vertex kan ikke elimineres, men kan i de fleste tilfælde sammenlægges med en anden.



Figur 3.5: Kompleks vertex.

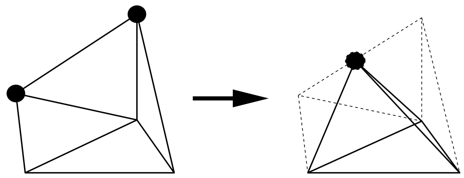
- En *ende vertex* er kun omkranset af en halv-cyklus af trekanter, men er ellers simpel. En ende vertex kan hverken elimineres eller sammenlægges med en anden, uden at ødelægge den kant den er en del af.



Figur 3.6: Ende vertex.

- Andre *Ikke flytbare vertices* er vertices der ved en eliminering eller sammenlægning med andre vil producere degenererede trekanter. Degenererede trekanter er her trekanter der bruger den samme vertex mere end én gang, og dermed degenererer til linier eller punkter.

  Et vertice-par der ved en sammenlægning genererer en anden vertex med mindre end fire tilhørende trekanter, kan i uheldige tilfælde medføre overlappende trekanter som på figur 3.7. Yderligere reducering af en vertex der kun har tre nabo-trekanter, vil resultere i degenererede trekanter og fejl i meshet.

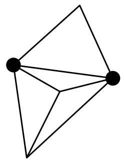


Figur 3.7: Overlappende trekanter

  På figur 3.8, ses to vertices der har tre vertices som fælles naboer. En sammenlægning af to af de tre trekanter vil resultere i at to andre trekanter degenerer til linier. I praksis undersøges hver vertex par således for om de har mere end to nabo-vertices til fælles. Hvis dette er tilfældet, kan de ikke sammenlægges.

3.4.2 Klassifikation vha. topografi

Ved at undersøge meshets topografiske karakter omkring de enkelte trekanter og vertices, kan yderligere klasser af vertices defineres. Denne klassificering er grundlaget for selve filtreringsdelen af decimeringen, da denne klassifi-



Figur 3.8: Ikke flytbare vertices

cering kan graduere de forskellige trekanter og vertices betydning for selve iso-fladens form og struktur:

- En *indre kant vertex* er en vertex der bruges af to *struktur-kanter* men ellers er simpel. En struktur-kant eksisterer de steder hvor to tilstødende trekanter har en vinkel der er større end en specificeret værdi.
- En *hjørne vertex* er en simpel vertex der bruges af én, tre eller flere strukturkanter.
- En *lille trekant* er en trekant hvor den mindste sidelængde der er mindre end en referenceværdi, evt. defineret i forhold til trekanternes middel-sidelængde eller voxel-størrelsen.

3.5 Iterativ decimering

Selve decimeringen af meshet er en iterativ process, baseret på den foretagne klassificering af trekanter og vertices. Den implementerede rutine bruger kun regler for trekanternes mindste sidelængde i forhold til middel-sidelængden som selve decimerings regel. En mindste sidelængde vælges ud fra den gennemsnitlige sidelængde for alle trekanter i meshet. Alle trekanter der der har sidelængder under denne, vil blive forsøgt elimineret. Efter hver iteration forøges grønseværdien gradvist, indtil det ønskede antal trekanter er nået.

Dette er en meget simpel måde at decimere på, men er for de omfattende og langt fra optimale skanner-data der skal visualiseres, tilstrækkelig til at generere meget vellignende simplificerede modeller af de originale iso-flader.

Hvis information om meshets topologiske karakter ved de respektive trekanter bruges til decimeringen, vil der kunne genereres modeller der er mere nøjagtige på bekostning af en nedsat reduktionsgrad. Dette kan foretages ved at sammenligne normalvektorer mellem trekanter og vertices, og iterativt eliminere de trekanter der ligger hvor meshet har mindst krumning. Således vil små strukturer, kanter og hjører blive bevaret. Denne fremgangsmåde er ikke afprøvet (størrelses baseret decimering gav gode nok resultater).

3.6 Decimations rutinen

Ved hjælp af de ovenfor beskrevne metoder, er en iterativ rutine til decimering af trekants mesh blevet implementeret. Rutinen kan overordnet beskrives med følgende iterative skridt:

1. Generér datastrukturer for meshet, med konnektivitets information.

   Alle vertices og trekanter klassificeres som flytbare/ikke flytbare ud fra konnektivitets informationerne.

2. Evaluér decimations kriterierne (mindste sidelængde) og udfør punkt 3 for for alle trekantpar.

3. Hvis kriterierne opfyldes og begge trekanter er flytbare, sammenlægges de, og alle datastrukturer opdateres med de foretagne ændringer. Ellers foretages ingen ændringer.

4. Hvis kompleksiteten af meshet ønskes reduceret yderligere, kan trin 2 og 3 udføres med ændrede værdier for decimations kriterierne.

Ændringerne i datastrukturerne i trin 3 indebærer også en re-evaluering af alle vertices og trekanter der er berørt af sammenlægningen. Dette kan godt medføre at en vertex eller en trekant får ændret status fra flytbar til ikke flytbar eller omvendt. Således bliver alle nabo vertces og trekanter til et elimineret trekant-par evalueret igen.

Kapitel 4

Polyreduce programmet

I dette kapitel, vil brugen af det udvilkede program, `polyreduce` blive beskrevet. Programmet er skrevet i C. Makefiler er lavet således at programmet kan compiles og afvikles på Silicon Graphics, HP og PC-Linux maskiner.

4.1 Anvendelighed

Som det før er blevet omtalt, vil de datamængder der bliver produceret af medicinsk udstyr være af et ganske betragteligt omfang, F.eks. består analyze billedet `mirSINGLE` af ca. 4.5 millioner voxels. Den implementerede marching cubes algoritme er baseret på floating point data, og vil derfor kræve omkring 20 MB hukommelse. Dette sætter naturligvis visse krav til det computerudstyr programmet kan køres på.

I forbindelse med polygon reduceringen arbejdes der med relativt store datastrukturer for hver enkelt vertex og trekant i meshet. I praksis har genereringen og reduceringen af meshes med omkring 200.000 vertices og 400.000 trekanter kunnet udføres på en Silicon Graphics Indigo^2 med 128 MB RAM inden for et realistisk tidsrum (omkring 2 min.).

4.2 Funktionalitet

Programmet `polyreduce` er et stand alone program der ud fra et tre dimensionalt data-array, kan generere en polygon repræsentation af et iso-flade niveau i dette data-array dvs. en kontur i rummet. Følgende funktioner bliver udført på et givent data-array:

Indlæsning af data-array
↓
[Filtrering]
↓
[Nedsampling]
↓
Iso-flade generering
↓
[Polygon reducering]
↓
Udlæsning af polygoner

hvor funktionerne i klammer er optionelle. Data-arrayet kan indlæses fra flere forskellige formater, disse inkluderer analyze, HDF, MultiStruct og rå byte formater. Udskrivningen af polygoner kan foretages i tre forskellige ASCII formater, og andre formater kan nemt implementeres.

4.3 Kørsel af `polyreduce`

Programmets kørsels-parametre opgives alle på kommandolinien som:
`polyreduce [options] <filetype> <infile> <threshold>`

4.3.1 Options

`-o <name>` Output `off` fil. Navnet (uden extention) på den geomview `off` fil de genererede trekanter skal skrives til.

`-n <name>` Output `nff` fil. Navnet (uden extention) på den World Toolkit `nff` fil de genererede trekanter skal skrives til.

`-i <name>` Output `iv` fil. Navnet (uden extention) på den Inventor `iv` fil de genererede trekanter skal skrives til. Enten `-o`, `-n` eller `-i` skal specificeres

`-f <x y z>` Filtrering. Denne parameter bestemmer hvilken størrelse filterkerne input-arrayet skal filtreres med. Der bruges et almindeligt middelværdifilter med kernestørrelse [`x`×`y`×`z`] voxels. Gyldige værdier for parametrene `x`, `y` og `z` er ulige heltal: $1, 3, 5, \ldots$

`-F <n>` Samme funktion som `-f`, blot bruges der et kubisk middelværdi filter med størrelse [`n`×`n`×`n`]. Gyldige værdier for `n` er ulige heltal: $1, 3, 5, \ldots$

-d <x y z> Nedsampling. Denne parameter bestemmer hvor meget input-arrayet skal nedsamples med i hver retning. Størrelsen på det nedsamplede array bliver den originale størrelse divideret med de opgivne faktorer i de respektive retninger. Der bruges tri-lineær interpolation. `x`, `y` og `z` opgives som floating point tal.

-D <n> Samme funktion som `-d`, blot nedsamples der med denne funktion med samme faktor `n` i alle tre retninger.

-r <n> Polygon reduktion. Denne parameter bestemmer hvor mange trekanter der maksimalt må være i det reducerede objekt. Programmet vil prøve at reducere indtil dette antal er nået. I ekstreme tilfælde hvor antallet ikke kan nås, vil programmet returnere med det mindste antal Trekanter der kan nås.

-m <n> Minimalt antal trekanter i et objekt. Polyreduce programmet deler de genererede trekanter op i uafhængige objekter. Objekter med under `n` trekanter vil blive fjernet. Default værdi for `n` er 150 trekanter.

-p Verbose kørsel. Med dette flag slået til, vil der blive skrevet mere information om kørslen på skærmen. Dette inkluderer information om kørselstid, hvis `-DTIME` flaget er brugt under compileringen.

4.3.2 Input fil-formater

Alle input filnavne opgives uden extension, det antages at filerne har default ekstensions dvs. `img/hdr`, `hdf` og `f3f` for hhv. analyze, HDF og MultiStruct formaterne.

-ana Analyze format. Indlæser fra en analyze fil med enten 8 eller 16 bit heltals data.

-hdf NCSA HDF format. Indlæser fra en `.hdf` fil med et tre dimensional SDF datasæt. Kræver at NCSA HDF bibliotekerne er linket med i programmet.

-ms MultiStruct format. Indlæser fra en 'MultiStruct' fil. MultiStruct er et format der bruges internt på Elektronisk Institut på DTU.

-raw <x y z> Råt byte format. Indlæser fra en rå byte fil, indeholdende et data-array på [$x \times y \times z$] enheder, med data i `x` retningen varierende hurtigst, fulgt af `y` og `z` retningen.

4.3.3 Default værdier

Som default vil `polyreduce` programmet ikke udføre hverken filtrering, nedsampling eller polygon reducering. Dog vil alle objekter der består af under 150 trekanter bilve fjernet. Derfor vil en kørsel af `polyreduce` hvor der blot specificeres en input- og outputfil, blot svare til at køre marching cubes algoritmen på input arrayet, og udlæse de genererede objekter der består af over 150 sammenhængende trekanter.

4.4 Output formater

Som output format kan vælges enten Geomview `.off`, World ToolKit `.nff` eller Inventor `.iv` format. Alle tre formater er simple læsbare ASCII formater, der lister vertices og konnektivitet, med små forskelle i headerformatet. Alle tre formater giver mulighed for en mere kompleks beskrivelse af objekter, med information om bla. farver og materialer, Disse muligheder udnyttes dog ikke af dette program.

4.4.1 `.off` formatet

Geomview `.off` formatet er et simpelt ASCII format, der for polygonfiler har følgende struktur:

```
OFF
<#vertices> <#polygons> <#edges>
<first vertice>
...
<last vertice>
<first polygon>
...
<last polygon>
```

Hvor en vertex har formatet: `<x> <y> <z>`,
og en polygon formatet: `<#vertices> <vertices>`

4.4.2 `.nff` formatet

World Toolkit `.nff` formatet, minder i dets simpleste form meget om `.off` formatet, fil-headeren er blot en smule anderledes:

```
nff
version 2.1
```

```
<objectname>
<#vertices>
<first vertice>
...
<last vertice>
<#polygons>
<first polygon>
...
<last polygon>
```

Hvor en vertex har formatet: `<x> <y> <z> N`
og en polygon formatet: `<#vertices> <vertices>`

4.4.3 `.iv` formatet

Inventor `.iv` formatet bruges i dets simpleste form. Et polygon baseret objekt beskrives som:

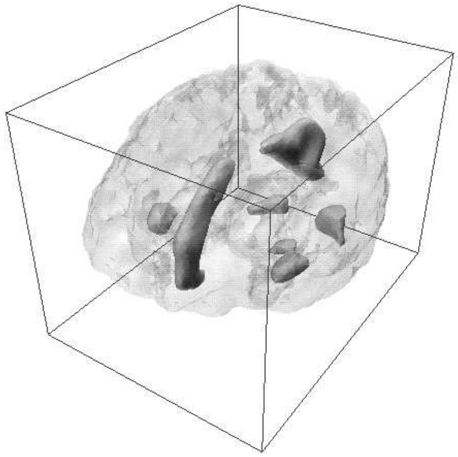
```
#Inventor V2.0 ascii

Separator {
  ShapeHints {
    shapeType SOLID
    creaseAngle 3.1416
  }
Coordinate3 {
  point [ <first vertice>,
          ...
          <last vertice> ]
  }
NormalBinding { value DEFAULT }
IndexedFaceSet {
   coordIndex  [ <first polygon>,
                 ...
                 <last polygon> ]
               }
}
```

Hvor en vertex har formatet: `<x> <y> <z>`
og en polygon formatet: `<first vertice>,...,<last vertice>,-1`

4.5 Geomview

Geomview formatet er inkluderet således at visualisering også bliver muligt på maskiner, der ikke har World ToolKit. Geomview er et public-domain program der kører på de fleste platforme. Disse inkluderer Silicon Graphics, HP og Linux maskiner.

Geomview programmet er meget fleksibelt og giver mulighed for interaktivt (med en mus) at manipulere alle aspekter af visualiseringen. Flere objekter kan hentes og vises samtidigt som på figur 4.1, hvor en iso-flade fra et såkaldt saliency-map er vist indlagt i en semi-transparent hjerne overflade, genereret fra et MRI-skan.



Figur 4.1: Kombinerede iso-flader genereret med Geomview.

4.6 World ToolWit

World ToolKit (WTK) er et såkaldt API (Application Programmers Interface) udviklet specielt til Virtual Reality. Toolkittet giver adgang til et C-bibliotek med over 500 funktioner til håndtering af grafiske objekter og styring af in- og output devices. De fleste Virtual Reality orienterede devices såsom head mounted devices (HMD), head trackere og 6-D input devices er der direkte support for. Tool kittet er tilgængeligt på en lang række platforme, incl. PC og Silicon Graphics.

4.7 Inventor

Inventor er ligeledes et objekt-orienteret bibliotek med objekter og metoder beregnet til udviklingen af interaktive tre-dimensionale applikationer. Grafik delen af inventor benytter Open-GL bibliotekerne. Inventor bliver i takt med udbredelsen af Open-GL mere udbredt på andre platforme end SGI. Herunder PC/Windows og Linux.

Inventor har ikke support for så mange in- og output devices som f.eks. WTK, men enkelte 6-D input devices understøttes. Mere Virtual Reality orienterede devices understøttes på nuværende tidspunkt ikke.

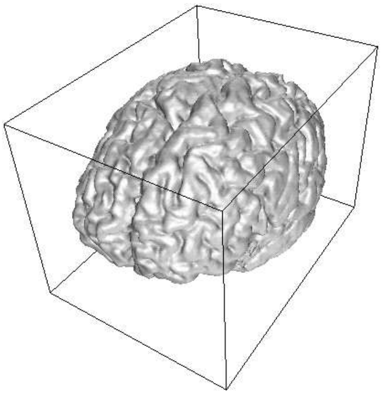
Kapitel 5

Resultater

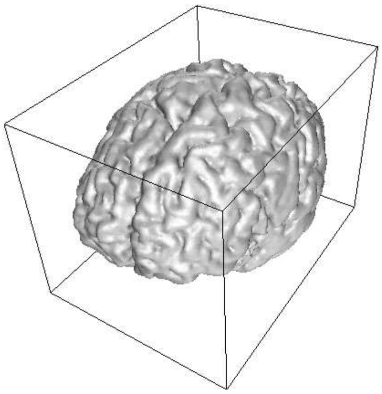
I dette afsnit, vil de resultater der er opnået med `polyreduce` programmet blive beskrevet. De resultater der vil blive sammenlignet er reduktionsgrad og reduktionshastighed. Alle kørsler er foretaget på en Silicon Graphics Indigo^2 med en 200 MHz MIPS-R4400 processor. Geomview er brugt til at generere de viste billeder. Alle de viste billederne er viualiseret ved hjælp af GeomView. Der er brugt gouraoud-shading

5.1 Reduktion af `mriSINGLE.img`

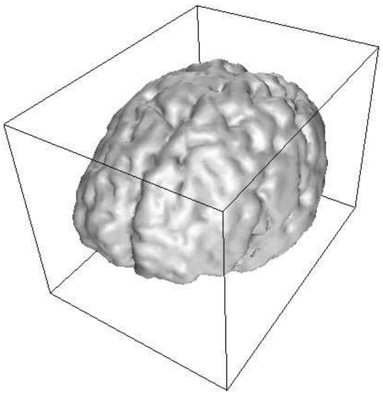
Dette volumen er et MRI scan af et patient hoved. Størrelsen på datasættet er [151×195×151] voxels. Før iso-flade genereringen er datasættet filtreret med et middelværdifilter på 3×3×3 voxels, for at nedsætte den tilfældige støj, og dermed undgå at generere en masse overflødige polygoner. Isofladeniveauet er valgt således at der genereres en isoflade svarende til hjernebarken. Dette niveau er eksperimentelt fundet til 100. På figur 5.1 til 5.3 ses de genererede iso-flader for forskellige reduktionsgrader. Kun billedet for 93.5% reduktion er visuelt ringere end det originale. Kørslestiderne for de forskellige reduktions grader er vist på figur 5.4 og 5.5.



Figur 5.1: Uden reduktion.



Figur 5.2: Reduceret 75%.



Figur 5.3: Reduceret 93.5%.

Volumen: `mriSINGLE`			
Størrelse: 151×195×151 voxels			
Threshold: `100`			
Options: `-F 3`			
Reduce option	No. trekanter	% decimeret	Tid/s
-	375096	-	109.1
`-r 100000`	93532	75.0	124.9
`-r 25000`	24324	93.5	128.2

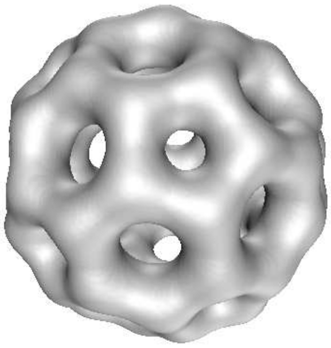
Figur 5.4: Polygon reduktion af `mriSINGLE.img`.

Funktion	Tid/s	Tid/%
Filtrering	12.8	10.1
Iso-flade gen.	15.8	12.5
Initialisering	62.9	49.8
Polygon reducering	34.8	27.6

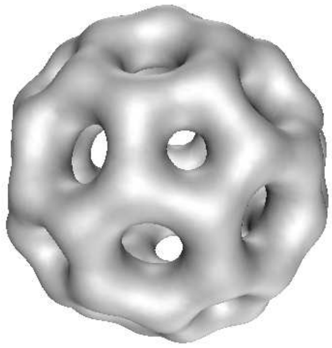
Figur 5.5: Reduktion af `mriSINGLE.img`, Kørselstid for `-r 25000`.

5.2 Reduktion af `bucky128.hdf`

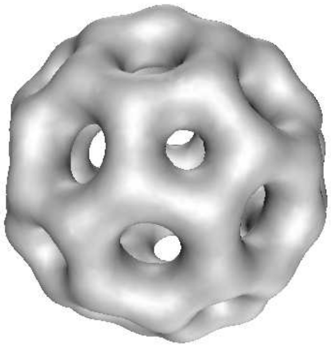
Dette volumen er et computersimuleret repræsentation af den forventede elektron densitet i et c-60 'fodbold'-molekyle. Iso-flade niveauet er valgt således at der fås en god visualisering af molekylets stuktur. Alle objekter med under 5000 trekanter er fjernet, da dele af de tilstødende molekyler ellers vil komme med i billederne. størrelsen på datasættet er [128×128×128] voxels. På figurerne 5.6 til 5.9 er vist de genererede iso-flader ved forskellige reduktionsgrader. Det ses at denne type objekt, med meget glatte overflader får et meget pænt udseende, selv ved meget høje reduktions grader.



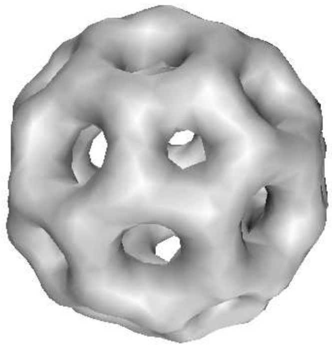
Figur 5.6: Uden reduktion.



Figur 5.7: Reduceret 75.7%.



Figur 5.8: Reduceret 87.8%



Figur 5.9: Reduceret 97.5%.

Volumen: `bucky128.hdf` **Størrelse:** 128×128×128 voxels **Threshold:** `0.1` **Options:** `-m 5000`			
Reduce option	No. trekanter	% decimeret	Tid (sec)
-	194352	-	40.5
`-r 50000`	47256	75.7	43.5
`-r 25000`	23944	87.7	44.0
`-r 5000`	4850	97.5	45.1

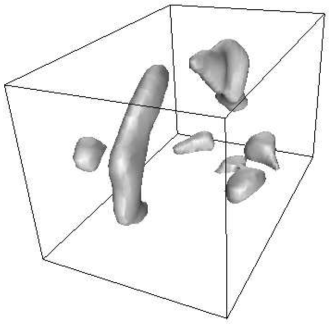
Figur 5.10: Polygon reduktion af `bucky128.hdf`.

Funktion	Tid/s	Tid/%
Iso-flade gen.	6.5	14.5
Initialisering	22.0	49.1
Polygon reducering	16.3	36.4

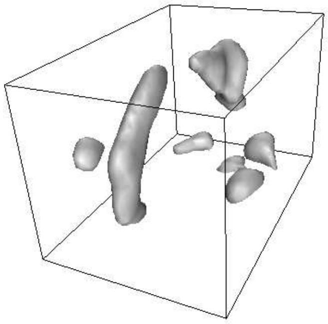
Figur 5.11: Reduktion af `bucky128.hdf`, Kørselstid for `-r 5000`.

5.3 Reduktion af `asa_sal.img`

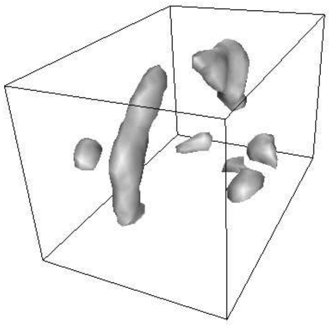
Dette volumen er en saliency-map på [65×87×30] voxels. Som iso-flade niveau er valgt en værdi på 0.011. Det ses at dette volumen også består af forholdsvis jævne overflader, og at den visuelle kvalitet bibeholdes selv ved høje reduktionsgrader.



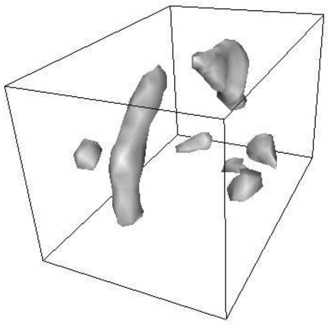
Figur 5.12: Uden reduktion.



Figur 5.13: Reduceret 65.1%.



Figur 5.14: Reduceret 82.7%



Figur 5.15: Reduceret 88.4%.

Volumen: asa_sal.img Størrelse: $65\times87\times30$ voxels Threshold: 0.0011			
Reduce option	No. trekanter	% decimeret	Tid (sec)
-	5588	-	0.61
-r 2500	1950	65.1	0.73
-r 1000	964	82.7	0.75
-r 750	650	88.4	0.75

Figur 5.16: Polygon reduktion af asa_sal.img.

Funktion	Tid/s	Tid/%
Iso-flade gen.	0.14	19.2
Initialisering	0.32	43.8
Polygon reducering	0.27	37.0

Figur 5.17: Reduktion af asa_sal.img, kørselstid for -r 750.

Kapitel 6

Information på nettet

I dette afsnit listes nogle WWW og FTP sites, der alle er koncentreret omkring emnerne tredimensional grafik og visualisering. Brugen af disse er en meget bekvem måde hvormed der kan indhentes information om formater og algoritmer, samt adgang til public domain programmer og source kode.

6.1 WWW sites

`http://www.dataspace.com/WWW/vlib/comp-graphics.html`
The World-Wide Web Virtual Library (Computer Graphics), Database over alle publikationer inden for computer grafik. Returnerer søgninger med BibTex lister.

`http://www.lightside.com/3dsite/`
3D site, Web-site dedikeret til 3D grafik. Mange links og referencer.

`http://www.hitl.washington.edu/projects/knowledge_base/onthenet.html`
Internet Resources in Virtual Reality. Mange informationer og links til/om VR relaterede emner.

6.2 FTP sites

`nic.funet.fi/pub/graphics/packages/`
Distributioner af de fleste public domain program pakker og biblioteker til visualisering og grafisk manipulation. Mange af disse er af meget høj kvalitet.

`avalon.viewpoint.com/pub/objects/`
Stor samling af 3D objekter i mange forskellige formater.

6.3 Usenet newsgrupper

De nedenfor listede newsgrupper, diskuterer alle computergrafiske emner:

```
alt.3d
alt.graphics
comp.graphics
comp.graphics.algorithms
comp.graphics.visualization
sci.virtual-worlds
comp.graphics.api.opengl
comp.graphics.opengl
```

Kapitel 7

Konklusion

I denne rapport er beskrevet udviklingen af et program til simplificering af tre dimensionale polygon baserede objekter blevet beskrevet. Resultaterne af det implementerede program er blevet præsenteret. Det er vist at de polygon objekter der genereres med marching cubes algoritmen på f.eks. medicinske datasæt kan reduceres meget i kompleksitet uden at det går ud over objektets visuelle kvalitet. Reduktioner på op til 98% er opnået på meget store datasæt med en simpel decimerings algoritme. Disse metoder har stor relevans i forbindelse med Virtual Reality, hvor antallet af polygoner i en simulering i de fleste tilfælde er bestemmende for hastigheden af den pågældende applikation.

Bibliografi

[BE92] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. Tech. Report CSL-92-1, Xerox PARC, Palo Alto, CA, 1992.

[Dur88] M.J. Durst. Letters: Additional references to "marching cubes". *ACM Computer Graphics*, 22(3):72–73, 1988.

[DZ91] Michael De Haemer, Jr. and Michael J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers and Graphics*, 15(2):175–184, 1991.

[HB94] C. T. Howie and E. H. Blake. The mesh propagation algorithm for isosurface construction. In *Computer Graphics Forum*, volume 13, pages 65–74. Eurographics, Basil Blackwell Ltd, 1994. Eurographics '94 Conference issue.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, August 1993.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[Mon94] C. Montani, R. Scateni, R. Scopigno. Discretized marching cubes. Technical report, I.E.I. Consiglio Nazionale delle Ricerdhe, Via S. Maria 46, 56126 Pisa, Italy, 1994.

[MS93] H. Muller and M. Stark. Adaptive generation of surfacees in volume data. *The Visual Computer*, 9(4):182–199, 1993.

[NH91] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In *Visualization '91*, pages 83–91, 1991.

[Rai92] E. Raible. 13 ways to say nothing with scientific visualization. Technical report, NASA Ames Reasearch Center, Febuary 1992.

[SBhD86] Francis J. M. Schmitt, Brian A. Barsky, and Wen hui Du. An adaptive subdivision method for surface-fitting from sampled data. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 179–188, August 1986.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[BE92] [Dur88] [DZ91] [HDD$^+$93] [HB94] [LC87] [Mon94] [NH91] [Rai92] [SBhD86] [SZL92] [Tur92]