3D visualisering av kartdata

Mastergradsoppgave i informatikk

Herman Kolås

13.12.2004

Høgskolen i Østfold
Avdeling for informasjonsteknologi

Sammendrag

Denne oppgaven tar for seg 3D-visualisering av kartdata, med data over Halden sentrum, som testgrunnlag. Målet med denne oppgaven er å generere en så god 3D-visualisering som mulig, basert på kartdata og flyfoto, og på den måten finne ut hvilke muligheter og begrensninger som finnes. Resultatet er et utvalg forskjellige visualiseringer av Halden Sentrum som demonstrerer effekten av forskjellige virkemidler i 3D-visualisering, slik som bruk av tekstur, detaljhåndtering og forskjellige modelleringsteknikker. Metoder som blir brukt for å visualisere kartdata, er import av kartdata på SOSI format, generering av terrengmodell ved hjelp av Delaunay triangulering, generering av bygninger som ekstrusjoner, bruk av flyfoto som tekstur og detaljhåndtering. De ulike aspektene og problemene ved disse metodene blir gjennomgått. Helt til sist drøftes ulike bruksområder for 3D-modeller basert på kartdata og hva slags type visualisering som egner seg best til ulike formål.

Innholdsfortegnelse

Figuroversikt

Eksempeloversikt

Forord

Jeg vil gi en stor takk til veileder Gunnar Misund som har bidratt med mye kunnskap om kartdata, algoritmer for behandling av kartdata og ideer for visualisering. Jeg vil også takke Teknisk etat i Halden Kommune som valgte å gjøre kartdata fritt tilgjenglig for dette prosjektet.

Kapittel 1. Introduksjon

1.1. Innledning

1.1.1. Bakgrunn

Visualisering av kartdata er et område med stor interesse. Digitale kartdata har allerede eksistert en god stund, men bare deler av disse har vært forsøkt brukt i VR-modeller. (VR står for Virtual Reality.) Et problem er ofte at kartdata er vanskelig tilgjengelig, enten fordi de er kostbare eller ikke er åpen for distribusjon. Spesifikasjonene til formatene som kartdata er lagret på, er ofte heller ikke tilgjengelige. Da Teknisk Etat i Halden Kommune bestemte at kartdata skulle regnes som offentlig informasjon, ga dette en gylden mulighet for et prosjekt med målsetting å visualisere kartdata. I tillegg til kartdata fikk man også tilgjengelig ortofoto for det samme området. Ortofoto er flyfoto, med nøyaktig posisjon og målestokk.

3D visualisering av store datamengder, som for eksempel kartdata, er et komplisert felt. Grunnen til at man ikke har sett flere 3D visualiseringer er stort sett fordi det krever stor datakraft, mye tid og ofte mye manuelt arbeid. Derfor har de modellene som blir generert, ofte veldig spesifikke oppgaver og er ofte ikke åpent tilgjengelig i likhet med de kartdata de ofte er basert på. Onemap prosjektet, som dette prosjektet er en del av, tar sikte på å gjøre kartdata for hele verden tilgjengelig over internett. Det kunne være ønskelig å gjøre det mulig å generere 3D visualiseringer basert på disse data og å distribuere disse over internett. En 3D visualisering basert på kartdata over Halden kan derfor være en test på dette konseptet.

1.1.2. Målsetting

Målet med dette prosjektet er å kunne produsere en best mulig visualisering av Halden, basert på tilgjengelige data. Det vil si å prøve å inkludere mest mulig av datasettet og å få til en så realistisk visualisering som mulig. Viktigst vil det være å kunne visualisere terreng og bygninger.

I tillegg til dette er det en målsetting å prøve å gjøre dette med minst mulig manuell tilpassing og modellering av modellen. Dette er viktig fordi hver tilpassing som skjer manuelt gjør det vanskeligere å bruke dette på andre datasett.

1.1.3. Hvordan oppgaven er organisert

Jeg har valgt å disponere oppgaven på denne måten: I kapittel 2 tar jeg for meg arbeid som har blitt gjort tidligere på dette området. I kapittel 3 vil jeg gjennomgå i mer detalj noen standarder og verktøy som er sentrale i prosjektet. I kapittel 4 tar jeg for meg de ulike problemstillingene i prosjektet, drøfter disse og vurderer ulike løsninger og alternativer. Kapittel 5 er en gjennomgang av hva som har blitt implementert i dette prosjektet. I Kapittel 6 blir resultatene som har blitt oppnådd presentert og evaluert. Til sist tar jeg for meg forslag til videre arbeid, gjennomgang av bruksområder og konklusjon.

1.2. Definisjoner

CAD
Se DAK

DAK
Data Assistert Konstruksjon. Forkortes på engelsk som CAD (Computer Assisted Development)

DTD
Document Type Definition. Format for å definere et sett med elementer og entiteter i XML.

DXF
Data Exchange File. Format støttet av de fleste PC-baserte DAK-verktøy.

ElevationGrid
En nodetype i VRML, mest brukt for å visualisere terreng.

Extrusion
En nodetype i VRML, man skaper en form som man trykker gjennom en form, omtrent som man lager pepperkaker.

GIS
Geographic Information System.

H-anim
En utvidelse av VRML standarden, for animasjon av menneskeskikkelser.

IndexedFaceSet
En nodetype i VRML, brukes for å kunne definere lage polygoner på et helt grunnleggende nivå.

Javamesh
Program for å demonstrere triangulering, med og uten beskrankinger.

LOD
Level of Detail, detaljnivå. Brukes i forbindelse med optimalisering.

Node
Et objekt i VRML.

Ortofoto
Flyfoto med nøyaktig posisjon og målestokk.

QTVR
QuickTime VR. Format for lagring og visning av 360° panoramabilder.

Rez
Program for å splitte opp terreng i mindre deler.

SOSI
Norsk standard for lagring av digitale geografiske data, utarbeidet av Statens Kartverk.

SVG
Scalable Vector Graphics. XML-kompatibelt format for å beskrive todimensjonal vektor/raster grafikk.

VR
Virtual Reality, på norsk virtuell virkelighet.

VRML
Virtual Reality Modeling Language. Standardformat for utveksling av VR-modeller over internett.

X3D
Extensible 3D. Dette format er etterfølgeren til VRML.

XML
Extensible Markup Language. Standard for dokument markup

Kapittel 2. Grunnlag for oppgaven og tidligere arbeid

Det har blitt utført flere prosjekter der man har lagd en visualisering av urbane områder ved hjelp av VRML. VRML er et format for utveksling av interaktive 3D modeller. Dette formatet blir beskrevet nøyere senere i oppgaven (Seksjon 3.2). Noen eksempler på byer der man har lagd en VRML modell er Bath (Bourdakis, 2001), vestkanten av London (Bourdakis, 2001) og Heidelberg (Schilling & Zipf, 2003) for å nevne noen. Det er flere tilnærminger for hvordan man kan visualisere en by eller deler av en by. Jeg har valgt å ta for meg noen av disse, for å kunne sammenligne med det som blir gjort i dette prosjektet. I tillegg har jeg også i noen tilfeller valgt å se nærmere på arbeid som disse prosjektene igjen baserer seg på.

Det første kriteriet er hvilke data man vil bruke eller har tilgjengelig. Kilder til data kan være CAD-data for bygninger, topografiske lasermålinger og komplette geodata. Alle formene har sine fordeler og ulemper. En CAD modell inneholder gjerne mye data, som ikke er nødvendig for visualiseringen. Lasermålinger gir en detaljert topografisk gjengivelse av terreng, og formatet er gjerne et regulært grid (Kanellopopoulos, 2001). Ulempen er at loddrette flater på bygninger ikke blir korrekte. Geodata er allerede ferdigprosessert og kan lettere brukes til å produsere en troverdig visualisering. Problemet er at disse ofte ikke er åpent tilgjengelig og/eller svært kostbare. Et annet problem er at det finnes flere forskjellige formater geodata kan være lagret på, og der mange av disse er proprietære. Dette prosjektet er tildels basert på at jeg har fått tilgjengelighet til geodata for visualiseringen. Like viktig er det også at formatet disse data er lagret på, er åpent tilgjengelig.

Det neste er hva slags teknologi man vil benytte. Et av aspektene man bør ta i betraktning er hva slags type data man har til rådighet. Men viktigere er hvilket bruksområde den skal ha og hvem som skal bruke den. En meget enkel form for 3D visualisering er Quicktime VR, som forkortes QTVR. Et panoramabilde som gir deg 360 graders utsyn fra et bestemt punkt. Hvis dette kan dette kombineres med et GIS system (Evans & Hudson-Smith, 2001). Men det er ingenting i veien for at man bruke dette i kombinasjon med en VRML modell. Man kan lage en referanse i VRML modellen på et bestemt sted til et QTVR med et panorama av det aktuelle stedet. Og på den måten virke som et supplement til modellen. VRML er best egnet for modeller som skal distribueres over internett eller som skal utveksles mellom plattformer.

2.1. Elementer i visualisering

Sentralt i visualisering av bymodeller er hvilke elementer som kreves for at modellen skal bli komplett og troverdig (Evans & Hudson-Smith, 2001). Artikkelen nevner tre kritiske elementer, som til sammen kreves for å skape en troverdig modell:

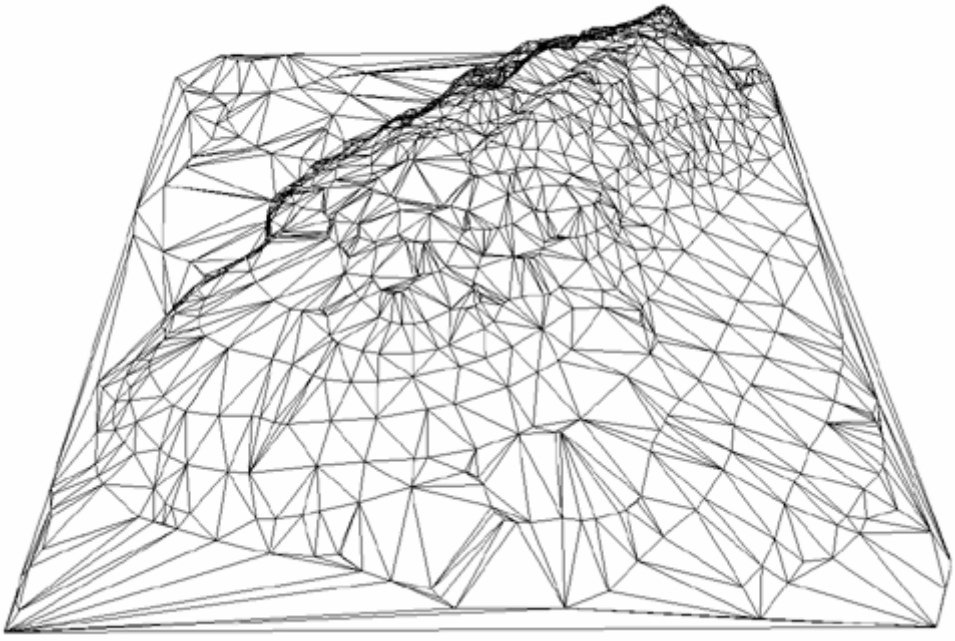
- Grunnflate for bygningsflate
- Takkonstruksjon
- Høydedata for terreng

Ved forprosjektet ble det lagd en enkel visualisering der bygningene var gjengitt med extrusion. Takkonstruksjonen ble derfor forenklet til et flatt tak. Dette var ikke noe stort problem for å gi et troverdig inntrykk av byen. Det som derimot var mer savnet, var fraværet av veier. Dette gjorde det vanskelig å orientere seg selv for en som var godt kjent i området som ble visualisert. Derfor kan det være på sin plass å legge til veier, visualisert på en eller annen måte til denne lista. Den enkleste måten å visualisere veier er å legge tekstur på terrenget. Dette er akseptabelt så lenge man ikke trenger å knytte noen ekstra informasjon til

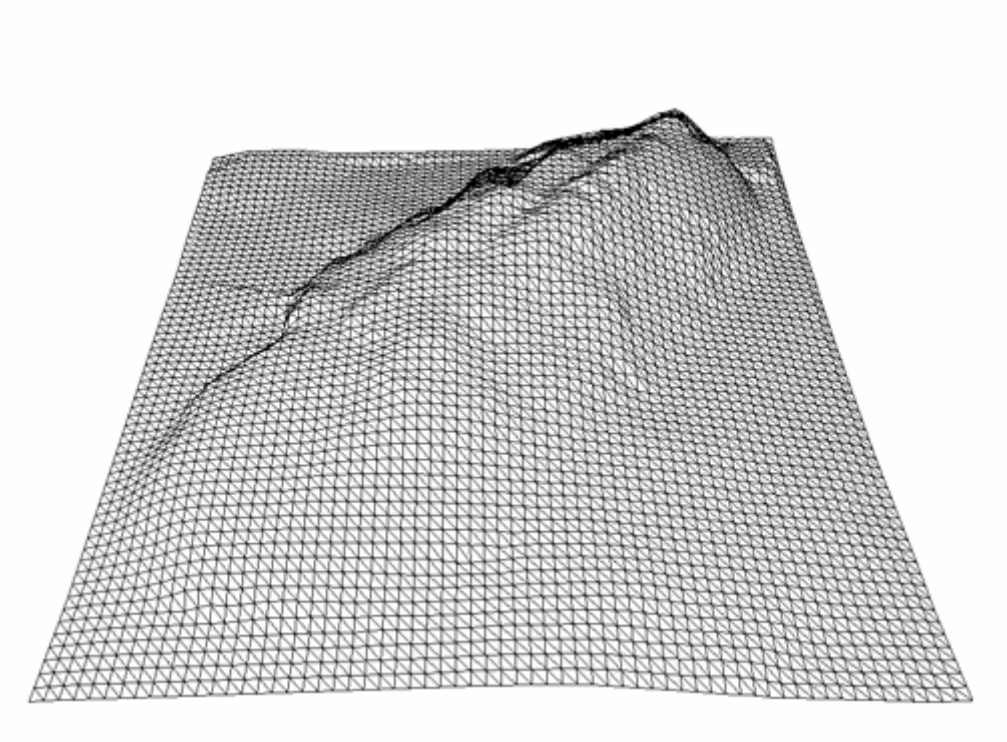
veiene. I så fall må man drapere veiene på terrenget. Dette har vist seg å være en mer komplisert affære enn bare å generere en terrengmodell alene (Schilling & Zipf, 2003).

2.2. Metoder for generering av terrengmodell

For selve genereringen av terrengmodell finnes det flere metoder. For en komplett gjennomgang kan man lese 'Survey of Polygonal Surface Simplification algorithms' (Heckbert & Garland, 1997). Generelt finnes det to forskjellige typer flater, parametriske og krummede. I en parametrisk flate må hvert eneste punkt ha en unik verdi i XY-planet. En krummet flate er en flate som kan ha punkter med felles verdi i XY-planet, men ulik høydeverdi. For å gjøre det litt enkelt er en krum flate tredimensjonal, mens en parametrisk flate er en todimensjonal flate med høydeverdier. Siden terrengdata er av parametrisk art vil jeg bare ta for meg disse. For parametriske flater kan sløyfe høydeverdiene. Det gjør at man kan bruke delaunay triangulering for å danne et nett av triangler. Denne metoden mest populær siden den maksimerer minimumsvinkelen på trianglene. Dette gjør at man unngår lange veldig tynne triangler (Figur 2.1. Eksempel på triangulering av terreng). I tillegg kan man bruke flere typer algoritmer for å implementere den. Alternativet til Delaunaytriangulering er regulær grid (Figur 2.2. Eksempel på et regulært grid 65x65 punkter). I et regulært grid danner punktene rutenett. Det er dette prinsippet som ElevationGrid noden i VRML benytter. Ulempen er at når man konverterer en triangulering til et regulært rutenett, kan man risikere at noen av punktene som beskriver de viktigste detaljene forsvinner. Men en av fordelene med regulære grid er at man enkelt kan bygge opp et kvadtre. Det vil si at man enkelt kan splitte opp et kvadratisk regulært grid i fire nye kvadrater. På denne måten kan man bygge opp et hierarki med forskjellige detaljnivåer. Håndtering av detaljnivåer er tatt for seg senere i oppgaven (Seksjon 4.5).



Figur 2.1. Eksempel på triangulering av terreng



Figur 2.2. Eksempel på et regulært grid 65x65 punkter

2.3. Navigasjon

Et viktig poeng i forbindelse med visualisering av kartdata er navigasjon, og hvordan dette håndteres. Begrepet navigasjon brukes om hvordan og hvor fort man beveger seg rundt i en VR-modell. Hvordan modellen skal navigeres påvirker også hvordan modellen bør utformes. For eksempel, hvordan bør bygninger være modellert i forhold til hvordan man navigerer modellen? Og Om man ser modellen direkte ovenfra, i fugleperspektiv eller på gateplan. Hver av disse perspektivene krever sitt eget detaljnivå. Når man ser modellen rett ovenfra, mister man dybdeperspektivet, og modellen blir da ikke mye forskjellig fra et tilsvarende todimensjonalt kart. Takkonstruksjon som stikker opp vegger blir uviktige. Derimot når man navigerer modellen på gateplan, blir veggene essensielle, mens takkonstruksjonen knapt synes. I fugleperspektiv derimot blir takkonstruksjonen svært tydelig, mens detaljer på veggene, som dører og vinduer, blir mindre synlige (Lal & Meng, 2001).

Et annet punkt når det gjelder navigasjon er hastigheten man navigerer rundt i modellen med. En gitt hastighet som er akseptabel når går rundt på gateplan, kan oppleves ulidelig tregt når man flyr over modellen. Mens det kreves enda større hastighet når man betrakter modellen fra en satellittbane, for å få samme følelse av flyt. Navigasjonshastighet er derfor noe som bør øke proporsjonalt med avstanden til modellen. Det er et typisk problem, når man visualiserer geografiske data. Skalaen gjør at en fast navigasjonshastighet er ikke tilstrekkelig.

2.4. Standard for oppbygging av VR bymodeller

Et aspekt ved generering av VR modeller er standarder for oppbygging av VR-modeller og da især bymodeller. Hvilke konvensjoner bør man følge og antagelser bør man gjøre? En artikkel (Bourdakis, 2001) tar for seg denne problemstillingen. Kart og tegninger for byer har en mengde konvensjoner og antagelser som er standard. For 3D modeller av byer må man bygge

opp et nytt sett konvensjoner og standarder. En tendens er at mange konvensjoner og metoder fra VR tas i bruk. Og dette er nødvendigvis ikke den beste løsningen for bymodeller.

Normalt vil valget av datakilde, være det første man vurderer. Siden grunnlaget for denne oppgaven er tilgjengeligheten av en bestemt datakilde, er dette ikke noen sentral problemstilling i denne oppgaven. Derimot er valget av programvare og eksportformat viktigere. To eksempler på aksepterte eksportformater er Data Exhange Format (DXF) og Virtual Reality Modeling Language (VRML). DXF er et format opprinnelig utviklet av Autodesk Inc. som eksportformat for deres programvare, AutoCAD. Dette formatet er ikke åpent, det vil si at spesifikasjonen for formatet ikke er åpent tilgjengelig, men har fått stor utbredelse innen DAK-programvare og på den måten blitt en defacto standard på dette området (Bourke, 1990). VRML har blitt et standard utvekslingsformat for 3D modeller over internett. Dette formatet er beskrevet i detalj senere i oppgaven (Seksjon 3.2).

Det neste punktet er bruken av koordinater og referansepunkter. For geodata benytter man ofte et nasjonalt referansepunkt, noe som gjør at koordinater for et gitt punkt blir svært store. Et GIS system er gjerne designet for å håndtere slike store tall, mens det i VR modeller kan føre til problemer, i form av avrundingsfeil, som igjen fører til problemer med Z-buffering. Z-buffering er en teknikk der man kalkulerer om et polygon ligger nærmere enn et annet, i forhold til observasjonpunktet, når disse ligger foran hverandre. Man må derfor som oftest finne et nytt nullpunkt. Ofte er det også slik at aksene er definert forskjellig. Kartdata bruker gjerne XY til å definere planet og Z-aksen til å definere høyden. VR fremviserer derimot bruker XZ til å definere planet og Y-aksen til å definere høyden. Derfor må man bytte om Z og Y aksen, for at man skal få en korrekt konvertering.

Detaljnivå er også noe man må ta i betraktning. Når man arbeider med kart, har man en klart definert målestokk som angir detaljnivået. VR derimot krever en annen tilnærming (Kada, 2002). Her må man dele opp modellen i blokker der hver blokk har flere detaljnivåer, relativ til avstand fra kamera. Her er et forslag til fem standard detaljnivåer for bymodeller og hva disse skal inneholde (Bourdakis, 2001):

- Bare terreng med tekstur.
- Simpel volumetrisk bygning med flatt tak. Veier, fortau og terreng er med. (under 200 triangler pr kvadrant)
- Distanse under 150m: hver bygning er modellert med vegger og tak og definert som et separat objekt i modellen. Trær innen radius er synlig.
- Distanse under 90m: vinduer, dører, rekkverk og frittstående murer og gjerder synlig.
- Distanse under 60 m: arkitektoniske detaljer som piper, ornamenter, etc. Vinduer og butikkfasader er teksturerte.
- Dette forutsatt at de nødvendige geodata og teksturer er tilgjengelig.

2.5. Fokusbasert visualisering av kartdata

Et eksempel på et tilsvarende prosjekt er 'Generation of VRML City Models for Focus Based Tour Animations' (Schilling & Zipf, 2003). Målsetningen med dette prosjektet, som tittelen sier, er å generere en VRML modell, som basis for fokusbasert tur animasjon. Trekk ved dette prosjektet er at de benytter seg av heterogene geodata, altså geodata som kommer fra forskjellige datakilder, som da både har forskjellig format og struktur. Det er da en utfordring å gi disse et felles referansesett og å identifisere korresponderende objekter i de forskjellige datasettene. Et eksempel på hvordan man benytter heterogene data i dette prosjektet, er

hvordan bygninger bygges opp i modellen. For å generere bygninger bruker de grunnflaten til bygningen som de ekstruderer oppover. For å beregne høyden på en bygning har de innhentet data over hvor mange etasjer en gitt bygning har og høyden over havet fra terrengdata.

Et annet trekk er at en sentral VR-server generer disse modellene på serversiden, basert på gitte data tilgjengelig. For å generere en VR-modell må følgende steg utføres:

- Ekstrudere 2D data til 3D
- Transformere data til et felles referanse system.
- Sammenføyning og tildeling (identifisering av korresponderende objekter fra forskjellige datakilder)

Hvis man har en homogen kilde til data, er det siste punktet overflødig. Men man må fortsatt transformere data til referansesystemet som modellen bruker. Man ønsker ikke nødvendigvis å bruke samme referansesystem for modellen som rådataene bruker.

Et annet trekk ved dette er at de bruker variabel geoskala for terrenget. Det vil si at de genererer en triangulering der trianglene blir gradvis større etter hvert som man beveger seg lenger bort fra det området man ønsker å fokuserer på. På den måten får man en glidende overgang fra områder med et høyt detaljnivå til områder i modellen med et lavere (Misund, 1997). Fokus i denne modellen vil følge en kurve som definerer ruten man vil animere. Langs denne ruten har man da det høyeste detaljnivået mens det blir gradvis lavere, når man går til hver side for denne kurven.

Det er noen punkter å påpeke i dette prosjektet. Det første er at man har hatt flere datakilder tilgjengelig og brukt dem. Bruk av heterogene datakilder gjør at man har flere muligheter når man skal generere en modell, men kompleksiteten og antall feilkilder øker tilsvarende. I tillegg blir det vanskelig å lage et generelt system som kan brukes på andre data.

Det andre punktet er bruken av variabel geoskala. Et alternativ til denne metoden er at man bygger opp et såkalt quadtree (Heckbert & Garland, 1997). Den går i korthet ut på man splitter området i kvadranter, der hver kvadrant igjen har et nytt nivå med fire kvadranter. Der hvert nivå har forskjellig detaljnivå. Metoden er beskrevet mer i detalj senere i oppgaven. Ulempen, som det påpekes, med denne metoden er at den skaper skarpe skillelinjer mellom detaljnivåene. Og gir derfor ikke en like pen visualisering av terrenget. På den annen side er den mye enklere å implementere enn variabel geoskala. En annen fordel er at man med quadtree ikke trenger å generere en helt ny triangulering hvis fokuset skifter. Derfor er den nok sannsynligvis bedre egnet hvis man skal la brukeren bevege seg fritt i modellen og ikke bare følge en fastlagt rute. Å generere terreng med variabel geoskala er akseptabelt når man på forhånd vet hvilke områder som er interessante og derfor må ha et høyere detaljnivå, og hvilke områder som ikke er det.

Hvis man sammenligner dette prosjektet med mitt prosjekt, er det i disse punktene forskjellene ligger. I mitt prosjekt er det bare en kilde til kartdata. Derfor har jeg til sammenligning bare en standard, et format og en datastruktur å forholde meg til. En annen vesentlig forskjell er at jeg tillater brukere å navigere fritt i modellen. Dette fører til at modellen må være like detaljert overalt.

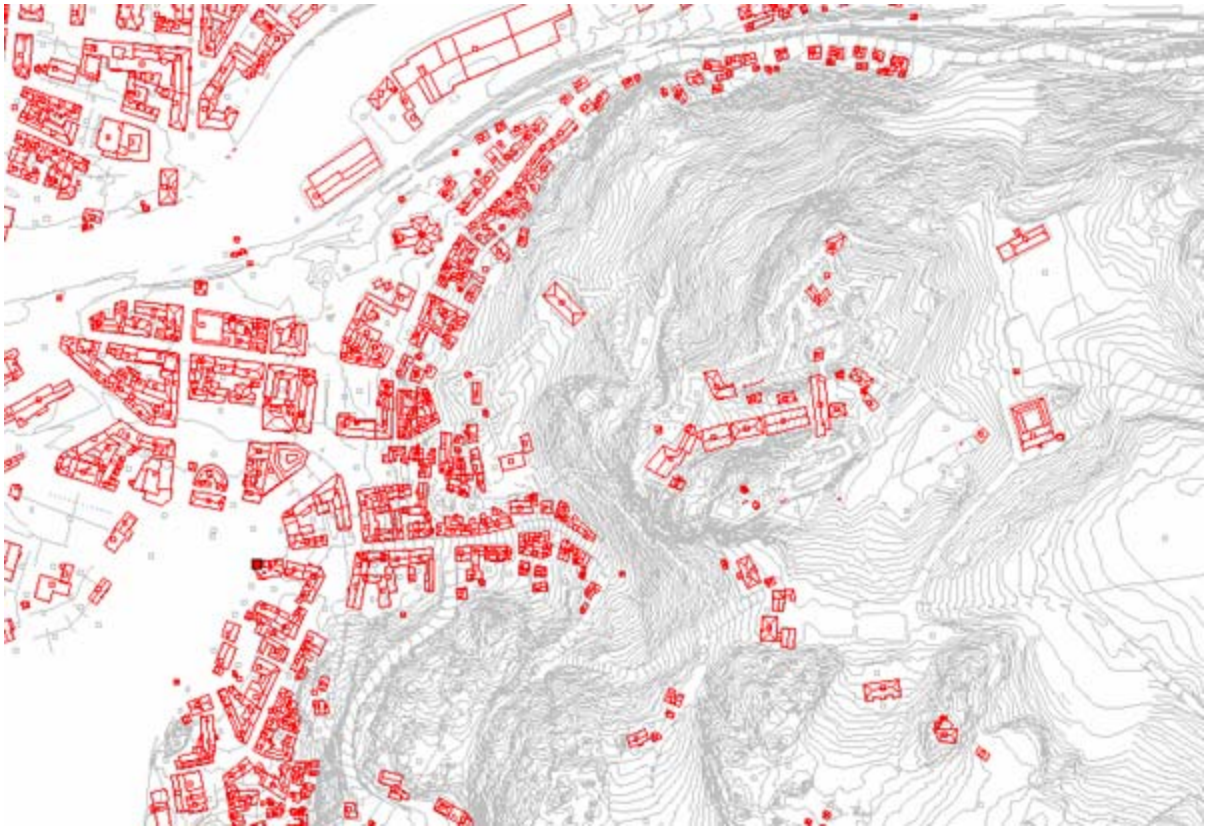
Kapittel 3. Formater og verktøy benyttet i oppgaven

3.1. SOSI formatet

Datasettet som brukes som grunnlag i denne oppgaven er lagret på SOSI format. SOSI formatet er et norsk format for å lagre geodata og relaterte data. Første versjon ble utgitt i 1987 av Statens Kartverk, som utarbeider formatet. Formatet brukes i mange statlige institusjoner og kommunene til å lagre geodata og økonomiske, administrative og persondata relatert til disse geodata. SOSI definerer et sett hvordan formatet skal beskrives geometri, topologi, kvalitet, koordinatsystemer og metadata (Statens kartverk, 1999). SOSI standarden er temmelig omfattende og tillater lagring av et vidt spekter av data. Jeg vil her derfor bare ta for meg de delene av standarden som er relevant.

3.1.1. Typer av data lagret i SOSI

Det er som sagt flere typer geografiske og relaterte data som kan lagres på SOSI format. For eksempel datasettet som danner grunnlaget til denne oppgaven, inneholder informasjon om anlegg (murer, gjerder, tanker, ol.), bygninger, jernbane, terreng, vann (bekker, elver og innsjøer) og veier.



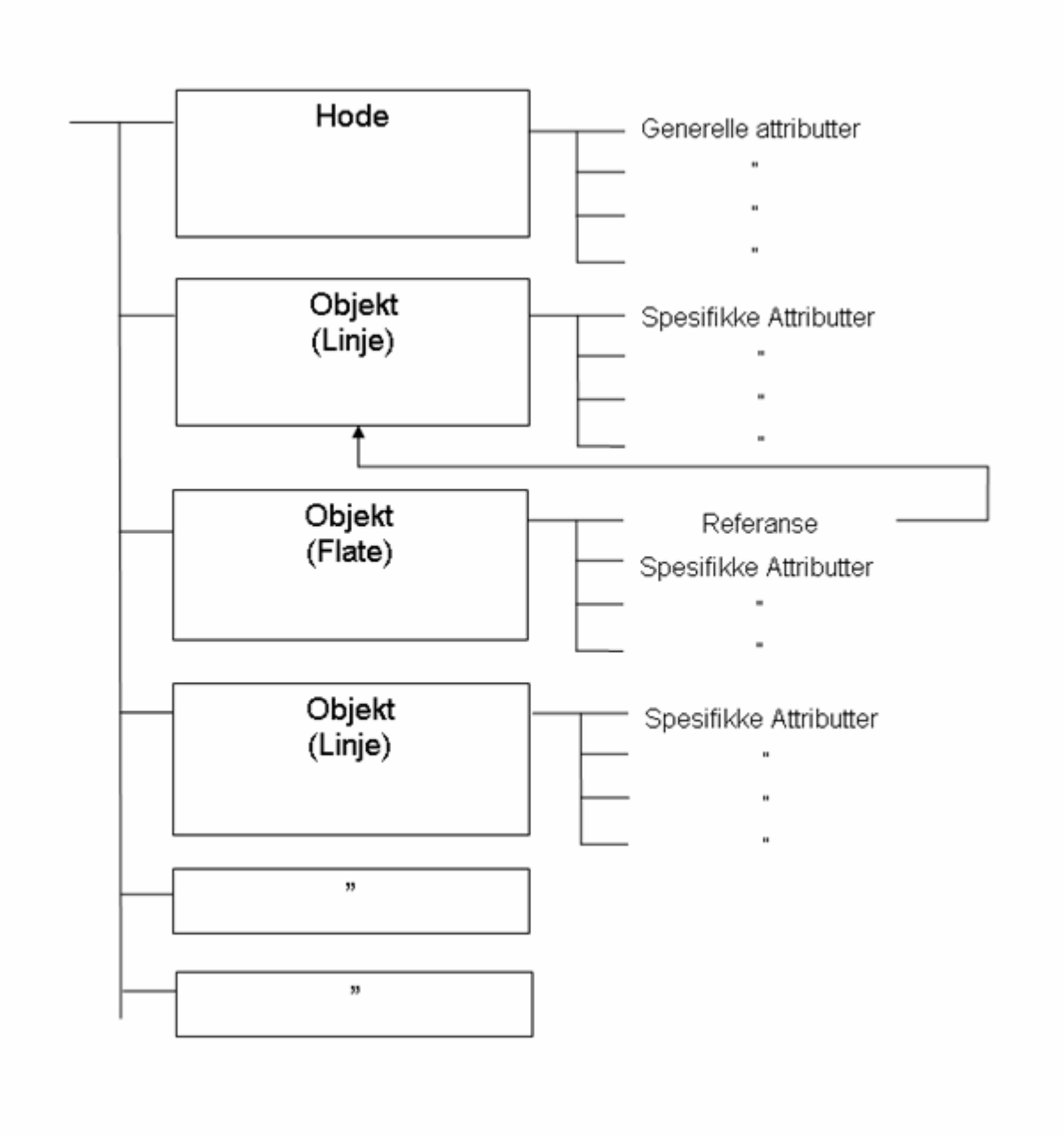
Figur 3.1. Illustrasjon av SOSI data fra SosiVis

Illustrasjonen (Figur 3.1. Illustrasjon av SOSI data fra SosiVis) er tatt fra programmet SosiVis v3.4 utarbeidet av Statens Kartverk. Utsnittet viser sydsiden i Halden sentrum og Fredriksten Festning. I dette utsnittet er data for bygninger og terreng tatt med. Terreng vises i grått og bygninger i rødt.

3.1.2. Grunnleggende struktur

En SOSI fil inneholder vanligvis en eller annen form for geografisk informasjon, men kan også inneholde andre typer data. Den er tekstbasert og dermed menneskelig lesbar, men verken XML eller SGML kompatibel. En ny revisjon av standarden, som skal være XML-kompatibel er under utarbeidelse. Tegnsettet som benyttes kan være ASCII, ND7, DECN7, ISO8859-1, ISO8859-10 (Same) og DOSN8. Sistnevnte er den som er oftest brukt. Det er et 8-bit tegnsett med norske bokstaver, som opprinnelig ble brukt i MS-DOS, men er i dag lite brukt. Valg av tegnsett kan i dag virke underlig, og må anses for å være en anakronisme fra den gangen formatet opprinnelig ble utarbeidet.

Hver SOSI fil inneholder en samling med objekter. Det første objektet i fila, HODE, angir generell informasjon i fila. Den inneholder vanligvis informasjon om hva slags data filen inneholder, opphav, kilde, presisjon, tegnsett, datum, avgrensning, versjon og SOSI nivå. Deretter følger definisjoner, objekt definisjoner, før selve data objektene. Figuren viser hvordan objektene typisk er organisert (Figur 3.2. Strukturen til en SOSI fil).



Figur 3.2. Strukturen til en SOSI fil

3.1.3. Strukturen til dataobjektene

Det er fire typer objekter for å beskrive geometri i SOSI; flate, kurve, linje, og punkt. Alle disse typene har følgende attributter; type, dato og kvalitet. Objektet Punkt inneholder et koordinat, som definerer posisjonen til et punkt. Kurve og Linje inneholder et sett koordinater, som definerer henholdsvis kurven og linjen. I prinsippet er det ingen forskjell på datasettet mellom en kurve og linje. Forskjellen ligger i hvordan de skal tolkes. I et Kurveobjekt skal det være interpolasjon mellom punktene, mens ikke på et Linjeobjekt. Endepunktene kan være både åpne, lukket sammen slik at de danner en flate eller være knutepunkter til andre et annet Linje-, Kurve- eller Punkt-objekt. I tillegg til dette kan disse objektene inneholde informasjon om hva slags type informasjon de representerer. For eksempel om et linjeobjekt representerer mønekant på et hus eller en fortauskant.

Flateobjektet inneholder referanse til en eller flere Linje- eller Kurve-objekter, som definerer arealet til Flate-objektet. Hvis en referanse står i parentes skal ikke dette arealet regnes som en del av flaten, men blir en øy i området Flateobjektet definerer. Referansen har også fortegn. Denne angir om rekkefølgen av punktene går med eller mot klokka.

Her følger et eksempel på hvordan et SOSI datasett ser ut (Eksempel 4.1. Utklipp av SOSI data for terreng) .) Dette eksempelet viser to objekter, det første et linjeobjekt det andre et flateobjekt. Disse to til sammen definerer flaten på en bygning, i dette tilfellet en enebolig.

Eksempel 3.1. Utklipp av SOSI data for en bygning

```
.LINJE 26327:
..LTEMA 5001
..DATO 19950504
..KVALITET 22 13
..OBJTYPE TakKant
..NØ
12544883 3798715 3139 ...KP 1
..NØ
12544617 3798576 3134
12544873 3798069 3048
12545238 3798373 3081
12545254 3798499 3123
12545551 3798589 3122
12545769 3797976 3006
12546143 3798063 3006
12545974 3798675 3047
12545900 3798658 3136
12545884 3799081 3238
12546013 3799111 3238
12545958 3799904 3147
12545074 3799850 3147
12545099 3799491 3246
```

```
12544964 3799361 3272
12545160 3798706 3154
12544883 3798715 3139 ...KP 1
.FLATE 26328:
..FTEMA 5001
..DATO 20030220
..KVALITET 50 36
..KOMM 0101
..TILTAKSTATUS 2
..OBJTYPE BygningsEnhet
..BYGGTYP_NBR 111
..REF :26327
..NØ
12545600 3798700
```

Hver attributt ligger på egen linje. Linjen starter med en rad punkter, som angir hvor høyt attributtet ligger i hierarkiet. Deretter følger navnet på attributtet og til slutt verdien. Koordinater er unntaket fra denne regelen. Koordinatene for hvert punkt skal ligge separat på en linje.

Et objekt starter først med typenavnet på objektet, i eksempelet LINJE og FLATE fulgt av nummeret på objektet. Deretter følger de forskjellige attributtene. La oss ta for oss attributtene til flateobjektet. TEMA angir hva flaten representerer, i dette tilfellet en bygningsenhet. De neste attributtene angir dato når dataene for dette objektet ble laget, kommunenummer, kvalitet, tiltakstatus og beskrivelse av type objekt. Deretter følger en referanseattributt. Denne inneholder en eller flere referanser til linje- eller kurve-objekter som definerer flaten. Her er det en referanse med positivt fortegn, som betyr at rekkefølgen av punktene skal ligge i standard retning. Hvis det er flere referanser, skal disse ligge i rekkefølge slik at de danner et sluttet polygon. Hvert endepunkt skal da være knutepunkt til neste linjestykke. I eksempelet over det bare et linjestykke som beskriver flaten. NØ attributtet til dette linjestykket angir at koordinater følger på den eller de neste linjene. Den angir også himmelretningene koordinatene er oppgitt i, nord og østlig retning. KP 1 attributtet som følger første og siste koordinat angir at punktene er knutepunkter. I dette eksempelet er de knutepunkter til hverandre. Hvis verdien for KP attributtet hadde vært satt til 999, hadde det betydd at punktet hadde vært et åpent endepunkt. SOSI datasettet følger stort sett samme mønster som dette eksempelet.

3.2. VRML og X3D

Som tidligere nevnt i oppgaven er VRML et format med bred aksept, som standard utvekslingsformat for 3D-modeller over internett. Flere større applikasjoner kan importere og/eller eksportere til VRML. VRML står for Virtual Reality Modelling Language og ble unnfanget på den første World Wide Web konferansen i Geneve, våren 1994. Første versjon av spesifikasjonen ble utgitt i mai 1995. I 1996 ble versjon 2.0 ISO standard (ISO/IEC CD 14772). Denne versjonen ble, med mindre modifikasjoner av noen detaljer, senere omdøpt til VRML 97. VRML 97 har vært stabil siden (Web3D Consortium Inc., 1997a). I 1999 ble arbeidet på utkastet til X3D standarden påbegynt. Og dette arbeidet fortsetter. Forskjellene

mellom VRML 97 og X3D vil bli diskutert senere, men en av de viktigste nyvinningene med X3D, er XML-kompabilitet.

VRML ble opprinnelig inspirert av HTML-standarden og var et forsøk på lage noe tilsvarende for 3D modeller. Der HTML var et universelt format for utveksling av tekst over internett, skulle VRML være et tilsvarende universelt format for utveksling av 3D modeller. Formålet til den første versjonen var nettopp dette, beskrive 3D modeller. I andre versjon skiftet man fokuset fra at formatet skulle beskrive statiske 3D modeller, til å beskrive dynamiske 3D verdener (Carey, Carson & Puk, 1997). VRML ble utvidet fra bare å kunne beskrive form, farge og tekstur på 3D objekter, til også å kunne gjøre dem bevegelige, interaktive og med mulighet for utvidelser. To eksempler på utvidelser er h-anim og geovrml (Web3D Consortium Inc., 1999a).

X3D har blitt utarbeidet for å bøte på noen av de manglene, som etter hvert viste seg i VRML 97 standarden. Den første og mest opplagte var at VRML 97 ikke er XML-kompatibel. Et annet problem var at VRML 97 standarden er monolittisk, og de som vil implementere en fremviser for VRML 97 må implementere hele standarden. Dette skjedde i realiteten aldri, og resultatet ble at en VRML modells oppførsel og utseende kunne være temmelig annerledes avhengig av hvilken fremviser man brukte, i verste fall kunne man risikere at hele eller deler av modellen ikke fungerte. Og til sist, i VRML 97 hadde man ikke noen måte å gruppere utvidelser.

Web3D Konsortiet (Web3D Consortium Inc., 2000) har prøvd å løse disse problemene i X3D (Web3D Consortium Inc., 1997b). Først har de splittet opp den gamle VRML 97 standarden opp i flere komponenter, slik som kjernen, geometri, belysning, interpolasjon, osv. I tillegg er flere tidligere utvidelser som er bygd på VRML 97 standarden, med i X3D som egne komponenter. For eksempel er h-anim standarden, for animasjon av humanoider i VRML, nå Humanoid Animation Compononent i X3D. GeoVRML v1.0, en utvidelse av VRML 97 for geografiske data er nå GeoSpatial Component i X3D. Felles for alle komponentene i X3D er at de to har nivåer, nivå 1 grunnleggende funksjonalitet eller nivå 2 med full funksjonalitet, som VRML 97. En systemutvikler kan velge hvilke komponenter og nivå han ønsker å implementere og kan lettere dokumentere hvilke deler av standarden som er implementert. Kanskje det viktigste er at X3D støtter en XML-kompatibel syntaks og har utarbeidet en DTD som spesifiserer hvordan VRML 97 noder skal representeres i XML. En bør riktignok merke seg at støtte for XML ikke er påkrevd i X3D og kan anses for å være en tilleggskoding (Web3D Consortium Inc., 2002).

3.2.1. Grunnleggende konsepter innen VRML

VRML har fire sentrale konsepter; header, scene graph, prototypes og event routing. Headeren identifiserer at det er en VRML fil, hvilken versjon av VRML og hvilket tegnsett den bruker. Scene graph er hierarkiet som inneholder alle nodene som beskriver VRML verdenen eller modellen. Prototypen gir brukeren mulighet å lage egne utvidelser av VRML standarden. Event routing gjør det mulig å sende en hendelse fra en node til en annen. For eksempel kan en sensornode sende en event til en annen node, hvis den blir utløst.

3.2.2. VRML noder og filstruktur

En typisk VRML fil består av følgende, først en header som beskriver versjon og tegnsett, deretter eventuelle referanser til eksterne prototyper eller implementering av interne prototyper. Tilslutt følger selve nodehierarkiet, som normalt er den mest omfattende delen av en VRML fil. En VRML fil kan ha null eller flere rotnoder i hierarkiet. Rotnodene er gjerne

grupperingsnoder som igjen inneholder flere noder. Det finnes flere forskjellige type node, og de kan grovt kategoriseres slik:

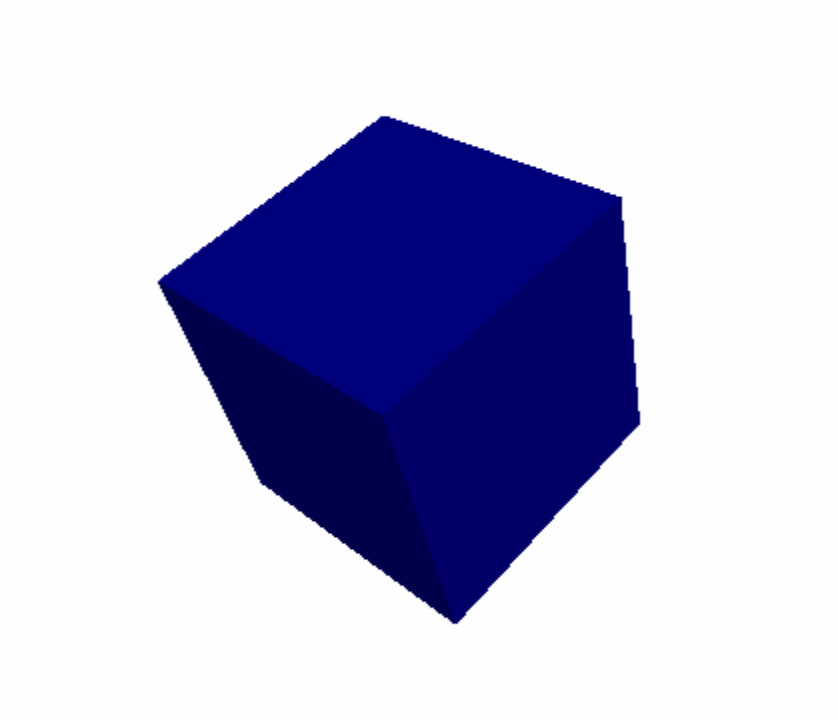
- ***Grupperings noder*** kjennetegnes ved at de kan inneholde en eller flere undernoder.
- ***Geometri noder*** beskriver formen på et objekt.
- ***Geometrisk attributt noder*** beskriver geometriske attributtene til et objekt.
- ***Utseende noder*** beskriver utseendet på et objekt.
- ***Sensor noder*** brukes til å oppdage handlinger utført av brukeren i modellen.
- ***Interpolasjon nodenes*** hensikt er å skape en jevn overgang fra en verdi til en annen. Disse er typisk brukt i animasjoner.
- ***Bundne noder*** kan defineres flere ganger i en VRML fil, men bare en kan være aktiv om gangen.
- ***Spesielle noder*** kan sees på som noder som peker til andre noder.
- ***Andre noder*** kan ikke grupperes sammen med noen av de andre nodene. Disse omfatter noder for, blant annet, lyd, lys og skript.

Et hierarki av noder består gjerne av en grupperingsnode med flere noder under seg i hierarkiet. Disse kan igjen være nye grupperingsnoder eller Shape noder som beskriver form og utseende på et objekt. En Shape node består av to noder; Appearance noden, som beskriver material og tekstur på objektet og Geometry noden, som beskriver formen på objektet. Dette kan best illustreres med et eksempel. (Eksempel 3.2. VRML kode for en blå terning))

Eksempel 3.2. VRML kode for en blå terning

```
#VRML V2.0 utf8

Group {
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0.0 0.0 0.7
        }
      }
      geometry Box {
        size 1.0 1.0 1.0
      }
    }
  ]
}
```



Figur 3.3. Bilde av blå terning.

I en VRML fremviser dette eksempelet vise en enkel blå terning (Figur 3.3. Bilde av blå terning.).) En node har flere attributter som kan settes. Hvis en attributt ikke blir satt, vil standardverdien bli brukt. I eksempelet over blir bare en attributt i Material noden satt, nemlig diffuseColor. Material noden til sammenligning har denne definisjonen (Eksempel 3.3. En VRML node).)

Eksempel 3.3. En VRML node

```
Material {
  exposedField SFFloat ambientIntensity  0.2       # [0,1]
  exposedField SFColor diffuseColor     0.8 0.8 0.8 # [0,1]
  exposedField SFColor emissiveColor    0 0 0      # [0,1]
  exposedField SFFloat shininess        0.2        # [0,1]
  exposedField SFColor specularColor    0 0 0      # [0,1]
  exposedField SFFloat transparency     0          # [0,1]
}
```

Hver attributt har fire egenskaper. Den første, 'exposedField' angir at denne attributt kan bli koblet til andre noder via routing; i motsetning til 'field', som ikke har denne muligheten. Den neste definerer hva slags type en attributt er, 'diffuseColor' er for eksempel en enkel farge. Den er av typen SFColor. De to første bokstavene SF angir at det er en skalar verdi, mens MF hadde angitt at det hadde dreid seg om et array. 'Color' betyr at det er en farge, som er angitt av tre tall for henholdsvis rød, gul og blå og disse tallene kan ha en verdi mellom 0 og 1. Etter typedefinisjonen følger navnet og til slutt standardverdien. Både 'exposedField' og 'field' må ha en standardverdi definert. Dette er mest viktig når man definerer sine egne skript noder eller prototyper.

Hvis man har behov for det, kan man ved hjelp av PROTO utvide VRML, med sine egendefinerte noder. Prototyper sammen med Script noder gjør at man kan implementere temmelig kompleks interaktivitet i VRML (Web3D Consortium Inc., 1997b).

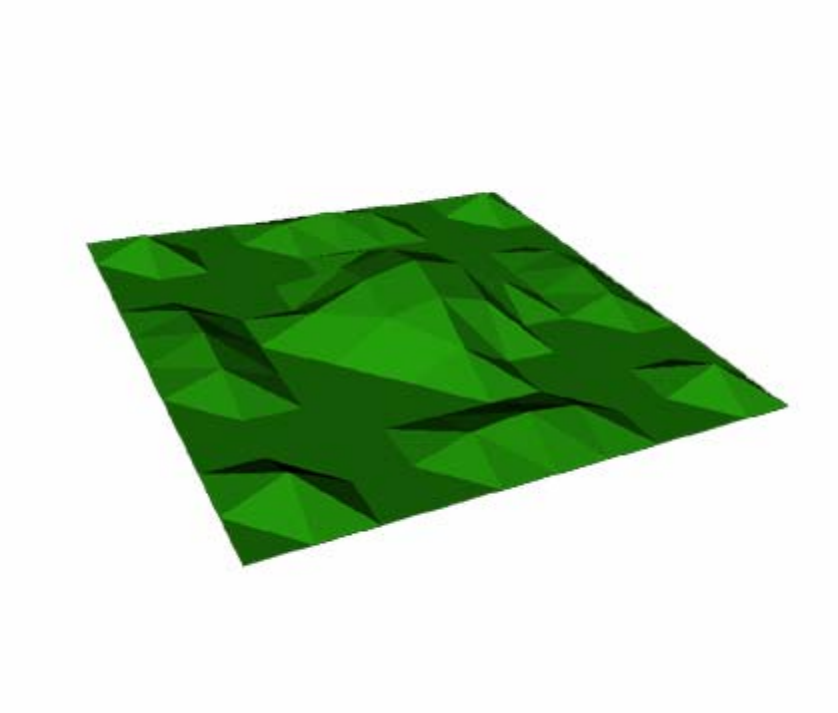
3.2.3. ElevationGrid, Extrusion og IndexedFaceSet nodene

Av de ulike nodene for å definere geometri er elevationGrid, Extrusion og IndexedFaceSet de mest sentrale for visualisering av geografiske data. Derfor kan det være nyttig å gå litt mer i detalj på disse.

ElevationGrid er sannsynligvis den noden, som er definert mest med tanke på visualisering geografiske data og da spesielt høydedata. Det er det opplagte valget, når man skal visualisere terreng. ElevationGrid noden har denne definisjonen (Eksempel 3.4. Definisjon av ElevationGrid noden.)

Eksempel 3.4. Definisjon av ElevationGrid noden

```
ElevationGrid {
  eventIn      MFFloat  set_height
  exposedField SFNode   color          NULL
  exposedField SFNode   normal         NULL
  exposedField SFNode   texCoord       NULL
  field        MFFloat  height         []
  field        SFBool   ccw            TRUE
  field        SFBool   colorPerVertex TRUE
  field        SFFloat  creaseAngle    0
  field        SFBool   normalPerVertex TRUE
  field        SFBool   solid          TRUE
  field        SFInt32  xDimension     0
  field        SFFloat  xSpacing       1.0
  field        SFInt32  zDimension     0
  field        SFFloat  zSpacing       1.0
}
```



Figur 3.4. Bilde av et elevationGrid.

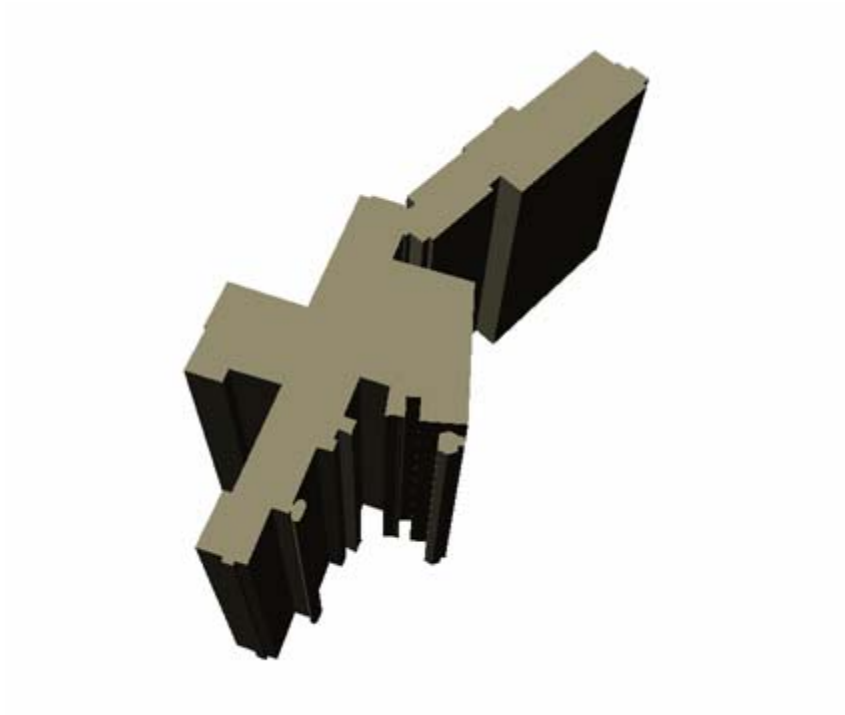
Bildet viser et eksempel på en enkel elevationGrid (Figur 3.4. Bilde av et elevationGrid.) Den geometriske formen som et ElevationGrid er et rektangulært rutenett der hvert punkt i

rutenettet har angitt høyde. Høyde for hvert punkt settes i et endimensjonalt array, height. Attributtene xDimension og zDimension angir hvor mange punkters utstrekning rutenettet har, i henholdsvis x og z aksen. Mens xSpacing og zSpacing angir hvor stor avstand det skal være mellom hvert punkt. Disse er de viktigste attributtene, som definerer formen på et elevationGrid. For å få korrekt visualisering må verdiene for xDimension og zDimension være korrekt satt i forhold til arrayet, height, som er generert.

Extrusion noden fungerer slik at man bestemmer formen på et objekt ved å definere en søyle og tverrsnittet denne skal ha. Enkelt forklart, fungerer det slik at man her lager en pepperkakeform og bruker den til å trykke ut en figur ut av en deig. Tykkelsen på deigen bestemmer høyden eller figuren. Definisjonen for denne noden ser slik ut (Eksempel 3.5. Definisjon av Extrusion noden.)

Eksempel 3.5. Definisjon av Extrusion noden

```
Extrusion {
  eventIn MFVec2f    set_crossSection
  eventIn MFRotation set_orientation
  eventIn MFVec2f    set_scale
  eventIn MFVec3f    set_spine
  field   SFBool     beginCap      TRUE
  field   SFBool     ccw           TRUE
  field   SFBool     convex        TRUE
  field   SFFloat    creaseAngle   0
  field   MFVec2f    crossSection  [ 1 1, 1 -1, -1 -1,
                                     -1 1, 1  1 ]
  field   SFBool     endCap        TRUE
  field   MFRotation orientation   0 0 1 0
  field   MFVec2f    scale         1 1
  field   SFBool     solid         TRUE
  field   MFVec3f    spine         [ 0 0 0, 0 1 0 ]
}
```



Figur 3.5. En extrusion form

Bildet (Figur 3.5. En extrusion form) viser et eksempel på en Extrusion. For orden skyld, tverrsnittet i dette eksempelet flaten til Halden Sykehus og Halden Sykehjem, mens lengden på søylen er bygningens høyde over havet. Eksempelet illustrer også hva slags geografiske data noden egner seg til, nemlig bygg og anlegg. Attributten 'crossSection' er et array bestående av todimensjonale vektorer og definerer tverrsnittet for objektet. Elementet 'spine' er et array bestående av tredimensjonale vektorer og definerer søylen som tverrsnittet skal strekkes ut langs. 'Spine' attributten i eksempelet over definerer en lineær søyle, men den kan ha andre former. For eksempel kan den være rund, spiralformet, tvinnet eller en annen form.

Hvis ingen av de andre geometriske nodene gir en god løsning på hvordan et objekt skal defineres, må man bruke IndexedFaceSet. IndexedFaceSet noden definerer nøyaktig hvordan hvert eneste polygon i et objekt skal se ut og kan brukes til å definere hvilket som helst 3D objekt. Grunnen til at man har de andre nodene, er at den også er den mest kompliserte noden å bruke (Eksempel 3.6. Definisjon av IndexedFaceSet Noden).

Eksempel 3.6. Definisjon av IndexedFaceSet Noden

```
IndexedFaceSet {
  eventIn      MFInt32 set_colorIndex
  eventIn      MFInt32 set_coordIndex
  eventIn      MFInt32 set_normalIndex
  eventIn      MFInt32 set_texCoordIndex
  exposedField SFNode  color          NULL
  exposedField SFNode  coord          NULL
  exposedField SFNode  normal         NULL
  exposedField SFNode  texCoord       NULL
  field        SFBool  ccw            TRUE
  field        MFInt32 colorIndex     []
  field        SFBool  colorPerVertex TRUE
  field        SFBool  convex         TRUE
  field        MFInt32 coordIndex     []
  field        SFFloat creaseAngle    0
  field        MFInt32 normalIndex    []
  field        SFBool  normalPerVertex TRUE
  field        SFBool  solid          TRUE
  field        MFInt32 texCoordIndex  []
}
```

Alle eksemplene over kan i prinsippet defineres ved hjelp av IndexedFaceSet. De to mest sentrale attributtene er coord og coordIndex. coord er en node i seg selv og inneholder koordinatsettet for objektet. coordIndex er et array, som brukes til å definere hvilke koordinater hvert polygon i formen skal bestå av (Web3D Consortium Inc., 97).

3.3. GeoVRML/GeoSpatial komponent

GeoVRML er en utvidelse av VRML 97. Den ble utarbeidet for å kunne lagre diverse geografisk informasjon i VRML 97. Selv om VRML 97 har en noe støtte for å visualisere terreng, har den ikke noen grunnleggende støtte for geografisk informasjon. Det er dette GeoVRML går inn for å bøte på.

GeoVRML gjør det mulig å referere til lengde og breddegrad og bruk av UTM-koordinater (Universal Transverse Mercator) direkte i en VRML modell. I tillegg gjør den det mulig å

benytte høyere presisjon i en VRML modell. Begrensningen i vanlig VRML gjør at man ikke kan presentere geografiske data med en større presisjon enn 10 meter på globalt nivå. Et annet problem er navigasjon når man beveger seg fra et lokalt til et globalt nivå. En hastighet som vil oppleve som mer enn rask nok på gatenivå, vil oppfattes som stillestående når man beveger seg opp i satellittbane (Reddy & Iverson, 2002). GeoVRML utvider VRML med disse nodene; GeoCoordinate, GeoLocation, GeoElevationGrid, GeoLOD, GeoMetadata, GeoPostionInterpolator, GeoTouchSensor, GeoViewpoint og InlineLoadControl. Av disse er GeoCoordinate, GeoLocation, GeoElevationGrid og GeoLOD de mest interessante; GeoCoordinate noden tillater definisjon av koordinater spatial referanse, GeoLocation noden knytter en georeferanse til en ordinær VRML modell og GeoElevationGrid tillater elevationGrid med spatial referanse. GeoLOD noden står litt i en særstilling, denne noden har innebygd den funksjonaliteten som kreves for håndtering av detaljnivå ved hjelp av kvadtrær (Seksjon 4.5.2).

GeoVRML er i seg selv i dag å anse som et dødt format, siden prototypene krever fremviser som støtter Java. Cosmoplayer er den eneste fremviseren som støtter dette, og denne har ikke blitt utviklet siden 1998. VRML 97 er på vei til å bli overtatt av X3D, derfor kan man ikke forvente at noen ny blir utviklet. Derimot lever standarden videre stort sett uendret, som GeoSpatial Komponent i X3D (Web3D Consortium Inc., 1999b).

3.4. Javamesh

Javamesh er en java-applet utviklet av Hsuan-Cheng Lin, som en del av hans masteroppgave i teknologi. Kort fortalt tar dette programmet et sett med punkter og gjør en Delauney triangulering mellom dem. Man har også muligheter til legge inn linjer som begrenser triangulering. Applikasjonen var en del hans masteroppgave og ble lagd for å demonstrere prinsippene ved triangulering og også med tanke på praktiske applikasjoner.

Algoritmen som blir benyttet i denne applikasjon, følger nøye den presentert av Shewchuk(1996), som inkorporerte Rupperts Delaunay Forbedrede algoritme. Kjernen i denne algoritmen inneholder Delauney triangulering og Delaunay triangulering med beskrankninger; som tidligere nevnt (Seksjon 2.2). Programmet hadde som mål å demonstrere at dette var den mest hensiktsmessige, robuste og effektive algoritmen for triangulering (Lin, 1997).

Dette programmet var det eneste åpent tilgjengelig programmet som hadde implementert en algoritme for Delaunay triangulering. Noe som er nødvendig, hvis man ønsker å generere en 3D modell av terrengdata. Alternativet hadde vært å implementere algoritmen fra bunnen av, noe som ville vært altfor tidkrevende. Det faktum at Javamesh i seg selv er en masteroppgave, burde si sitt.

At denne applikasjonen også kunne triangulere med beskrankninger, er et viktig poeng i denne sammenhengen. Det muliggjør definisjon av områder som ikke skal trianguleres, når man genererer en 3D visualisering.

Til sist kan det være verdt å merke seg at Delaunay triangulering er en ren 2D metode for triangulering og dette programmet er heller ikke egentlig beregnet på 3D data. Men ved bare å bruke xy-projeksjonen for hvert punkt, kan man gjøre en datauavhengig triangulering (Heckbert & Garland, 1997).

3.5. Rez

Rez er et open source prosjekt av Chris Thorne, som enkelt sagt splitter opp en terrengmodell i flere detaljnivåer. Det er en filparser som tar et rutenett på et gitt splitter og lager et hierarki med flere detaljnivå, definert av brukeren. Den kan skrive ut resultatet på flere forskjellige typer filformat. Hovedformålet med Rez er å kunne håndtere store terreng datasett, bygge modeller ubegrenset i størrelse og som kan bli vist over med akseptabel hastighet.

Rez kan lese en rekke formater for geografiske data. Disse er ArcView ASCII eksport fil, ArcView BIL fil, DTED fil, VRML97 ElevationGrid, GeoVRML GeoElevationGrid, ETOPO5, Gunta ASCII CSV og Gtopo30 DEM. Som utskriftsformat har man mulighet til å velge mellom VRML elevationGrid, hierarki av JGP konturbilder, X3D elevationGrid og binærformat (Thorne, 1999).

Dette programmet er nyttig når man har et stort regulært grid (Seksjon 2.2) som skal splittes opp i et hierarki av flere detaljnivåer for å gjøre den lettere å kjøre.

Kapittel 4. Beskrivelse av sentrale problemstillinger

For et prosjekt som dette, har man en rekke valg og problemstillinger. Hva slags datamateriale skal man ta utgangspunkt i? Hvilket format skal man benytte på den ferdige visualiseringen? Hvilke metoder skal man benytte for å transformere fra 2D-geodata til 3D-geodata? Skal man benytte åpne standarder eller proprietære standarder? Er det mest hensiktsmessig å lage en generell metode, som fungerer på flere typer data, eller gir en spesialtilpasset metode et bedre resultat? Hvordan optimaliserer man visualiseringen på den mest hensiktmessige måten? Jeg har prøvd å kategorisere problemstillingene i oppgaven på denne måten:

- Valg av kilde for kartdata. Hvilke kilder skal/kan man benytte? Hvor mye av datasettet skal man benytte? Hvor mye er det mulig å benytte?
- Valg av format for visualisering. Hvilke formater er best egnet? På hvilken måte skal man bruke formatet?
- Metode for transformering. Hvilke metoder skal man benytte for å konvertere fra rå kartdata til ferdig 3D-visualisering? Hvordan skal forskjellige objekter behandles?
- Metode for detaljhåndtering og optimalisering.

For hver av disse vil det være en rekke underordnede problemstillinger.

4.1. Oversikt over sentrale problemstillinger

4.1.1. Kilde for kartdata

Det første man må stilling til er hvilke data som er tilgjengelig og hvilke av disse man bør benytte. Dette prosjektet bygger i stor grad på at man har fått tilgjengelig data over Halden Sentrum på SOSI format. I tillegg har man også fått tilgang til flyfoto over samme område i flere oppløsninger. Derfor kan man si at valg av datakilde er gitt på forhånd. Data på SOSI format vil i dette prosjektet være kilden for kartdata. Derimot når man ser nærmere på kildematerialet man har tilgjengelig, er ikke alle svarene like opplagte. SOSI datasettet er delt opp i syv deler:

- ***Anlegg*** omfatter informasjon om forskjellige anlegg. Dette omfatter demninger, flaggstenger, gjerder, idrettsanlegg, murer og tankanlegg.
- ***Bane*** omfatter informasjon om jernbane.
- ***Bygg*** inneholder informasjon om alle bygningsenhetene. Dette omfatter både bolighus, kommersielle og industrielle bygningsenheter, driftsbygninger og uthus. Bygninger som var under oppføring, da data ble innhentet er også med. For hver bygningsenhet er det informasjon om bygningsflate og takkonstruksjon.
- ***Høyde*** omfatter all informasjon om terrenget i området. Dette er høydekurver, forsenkningskurver, terrenglinjer og bruddlinjer. I tillegg er terrenglinjene som omkranser bygg tatt med her.
- ***Tekst*** er tekststrenger som angir høyder og gatenavn.
- ***Vann*** inneholder informasjon om ferskvann. Det vil si bekker, dammer, elver og innsjøer.
- ***Veg*** inneholder informasjon om alle veier og gater åpen for motorferdsel. Dette omfatter senterlinje, vegavgrensning, fortau, gang- og sykkelveg og vegrekkverk.

Ut fra denne lista skulle man tilsynelatende ha den informasjonen man trenger og litt i tillegg. Noen deler av informasjonen er mer sentral enn andre for at man skal kunne være i stand til å produsere en troverdig visualisering. Som nevnt tidligere er det tre elementer som er sentrale for å lage god visualisering av en by. Det er bygningsflatene, takkonstruksjonene og terrenget. Derfor er det data for bygg og terreng som det er naturlig å begynne med. Informasjon om jernbane, vann og veg er også informasjon som man kan vurdere å ta med. Imidlertid kan bruk av ortofoto som tekstur være tilstrekkelig erstatning for disse. Ortofoto er flyfoto med nøyaktig geografisk posisjon og målestokk. Dette brukes vel og merke hvis man ikke har behov for mer informasjon om for eksempel gater og deres beliggenhet. Data om anlegg inneholder data om svært forskjellige gjenstander. Derfor varierer det veldig hvor viktig denne informasjonen er for en god visualisering. For eksempel er tankanlegg store strukturer, som fort kan bli savnet i en visualisering, mens flaggstenger neppe vil være sentral i en visualisering. Tekstdata vil neppe være viktig for noen visualisering, og den vil sannsynligvis være vanskelig å bruke av andre grunner. Men igjen; bygninger og terreng er de mest sentrale data.

4.1.2. Hvordan skal man behandle SOSI formatet

Når man har bestemt seg for hvilke typer data som er de mest sentrale kommer man til hvordan man best kan behandle SOSI data. Det er flere tilnærminger til hvordan man kan lese inn formatet. Det enkleste alternativet er å lese SOSI-data direkte for generering av 3D-visualisering. Formatet er tekstbasert og har en enkel form. Det er derfor relativt enkelt å implementere noe som leser SOSI-data direkte. En ulempe med dette er at man risikerer å lage et program som er for spesialisert. Den vil bare være i stand til å lese SOSI data. Og siden SOSI formatet spenner så vidt, kan man risikere at den ikke vil lese andre typer SOSI data eller SOSI data fra andre områder, slik at datasettet man tar utgangspunkt i, er det eneste som kan behandles av systemet.

Et annet alternativ kan være å først konvertere SOSI data, til noe som oppfyller kravene til velformethet innen XML. Det vil i prinsippet si at man tar alle objektene og attributtene og konverterer dem til XML-tager. Fordelen med dette er at man da kan bruke ferdige biblioteker for å lese XML og slipper implementere dette helt fra bunnen av. På den måten kan man kanskje unngå at systemet blir for spesialisert og lettere kan brukes på andre måter. Det må nevnes i denne sammenheng at Statens Kartverk, i skrivende stund, arbeider med en revisjon av SOSI-standard, og XML-kompabilitet er et av arbeidsområdene.

Det siste alternativet er å først konvertere SOSI data til GML. Og deretter benytte disse på GML videre for generering av modellen. GML-formatet er en åpen internasjonal standard for kartdata, basert på XML. Fordelen med dette er at systemet da vil kunne lese hvilke som helst data på GML-format, noe som kan gi systemet større nytteverdi. Ulempen er at det vil kreve en del ekstra arbeid å implementere dette. Konverteringen kan også bli en ekstra feilkilde, hvis den ikke blir korrekt implementert.

4.1.3. Valg av format for visualisering

Som tidligere nevnt i oppgaven, fins det flere alternativer å velge mellom som format for selve visualiseringen. Kriteriene for valg av format bør være at formatet har stor utbredelse, er bredt akseptert og at det er åpent. Det er viktig at det finnes verktøy som er i stand til behandle formatet. Og at det er egnet til oppgaven.

VRML kan være det naturlige valget. Men jeg vil ta for meg noen alternativer først. DXF-formatet er tidligere nevnt. Det har bred aksept og er mye brukt og mange applikasjoner kan

lese formatet. Men av mange grunner er ikke dette formatet egnet. For det første er formatet proprietært. Spesifikasjonen er ikke åpent tilgjengelig. Men viktigst av alt formatet er egentlig beregnet på konstruksjonstegninger og ikke VR. Data på DXF-format er derimot ofte brukt som kildemateriale for VR-modeller.

VRML er opprinnelig beregnet på 3D-visualisering over internett. Den er åpen og er til og med en ISO-standard (ISO/IEC 14772.) Men det er andre alternativer til VRML man bør vurdere. Det første alternativet er GeoVRML. GeoVRML er en utvidelse av VRML standarden. Den adresserer noen av manglene ved VRML, når det gjelder visualisering av terreng. Disse er mangelen på geokoordinater og geografisk informasjon, flyttall med for dårlig presisjon og navigering av geografiske modeller. Problemet med GeoVRML er at det er få verktøy som støtter denne utvidelsen. Og standarden er i prinsippet et dødt format.

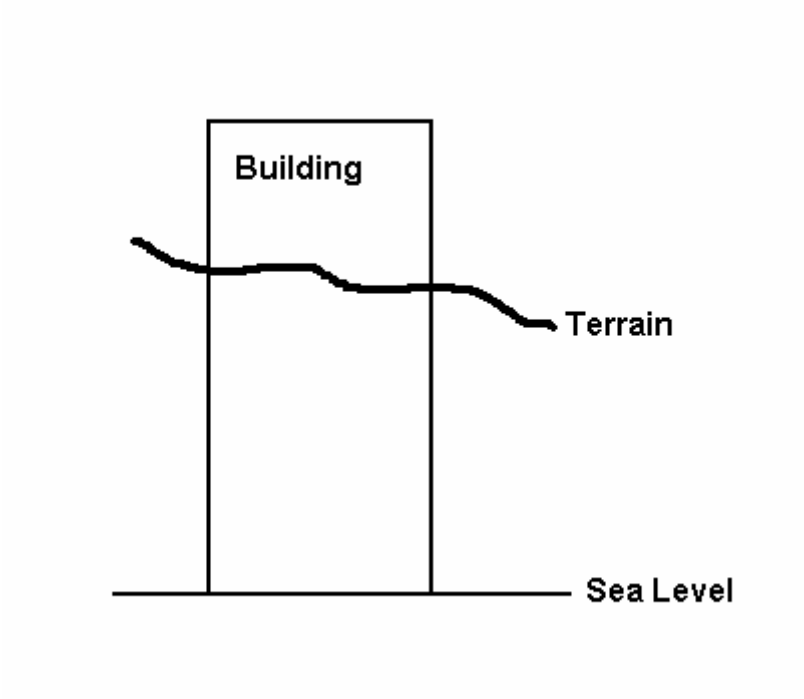
Det andre alternativet er X3D. Dette formatet har tatt over for VRML 97. Den inkorporerer flere av utvidelsene som ble utviklet til VRML, deriblant GeoVRML. Den skal også være XML-kompatibel. Ulempen med X3D er at standarden er relativt ny, og det finnes for øyeblikket få verktøy som støtter denne standarden. XJ3D som er verktøyet Web3D selv utvikler, er fortsatt under utvikling og første versjon er ennå ikke lansert.

Derfor kan det i første omgang være det enkleste å basere seg på VRML 97. Mange applikasjoner er i stand til å både importere og eksportere til dette formatet. X3D vil være bakoverkompatibel, og derfor skulle det ikke være for vanskelig senere å konvertere eller oppdatere fra VRML97 til X3D. Et annet punkt er at områdene som vil bli visualisert er relativt små. Derfor vil ikke geokoordinater, datum og geografisk navigasjon være noe stort problem. Disse kan også legges til i ettertid, hvis det er behov for det.

4.1.4. Hvordan skal man visualisere forskjellige typer data

En annen problemstilling er at forskjellige typer data, har forskjellige former for visualisering, som er mest optimal. Det er også et spørsmål hvilke deler av datasettet man bør ta med. To ytterpunkter, når det gjelder visualisering er terreng og bygninger. Terreng visualiseres gjerne best med elevationGrid. ElevationGrid er et rutenett der hvert punkt har angitt en høydeverdi. Terreng kan stort sett uttrykkes som en ujevn flate. Bygninger derimot er en samling enkeltstående objekter. Å bruke elevationGrid på disse vil ikke gi en troverdig visualisering. Tilslutt har man data for veier og bruer. Hvordan skal disse best visualiseres? Skal de visualiseres som en del av terrenget eller for seg?

Hvordan kan man kombinere disse i en god visualisering? For eksempel hvordan kombinerer man en terrengvisualisering og visualisering av bygninger i samme område? Terreng visualiseres best som tilsynelatende sammenhengende flater. (Mer om dette under detaljnivå.) Mens bygninger visualiseres best som enkeltstående objekter. For en gitt bygning finnes det informasjon i datasettet, om blant annet husets grunnflate, grunnmur og takkonstruksjon. Det viktige i denne sammenhengen er tverrsnittet der terrenget skjærer bygningen. Kort sagt at bygningen står på bakken i visualiseringen. Det er to måter å løse dette på, man kan la bygningsobjektet skjære ned gjennom terrenget til for eksempel havnivå. Z-buffering i fremviseren vil sørge for at det som er under terrenget ikke synes. Prinsippet er illustrert under. (Figur 4.1. Illustrasjon av hvordan terreng skjærer bygning)

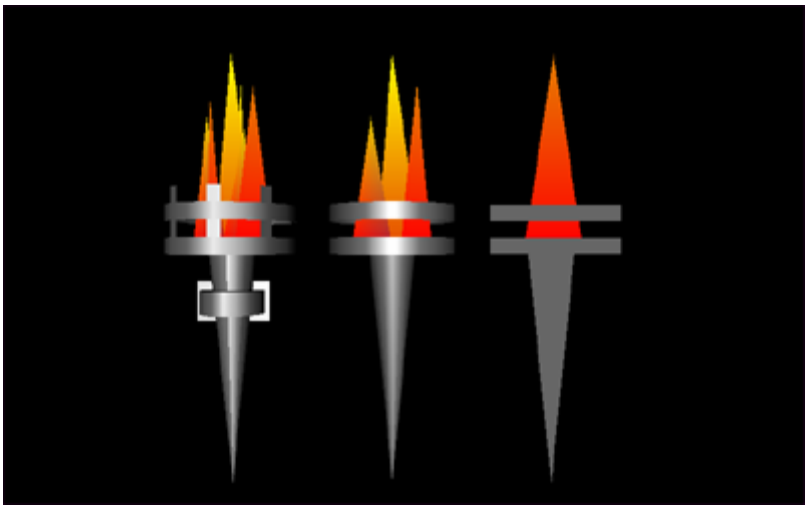


Figur 4.1. Illustrasjon av hvordan terreng skjærer bygning

En metode for å kombinere disse forskjellige objektene er å inkludere grunnflaten til bygningene når man skal generere terrengvisualiseringen. På denne måten skulle man i prinsippet få en mer jevn overgang mellom bygning og terreng. En siste mulighet er å generere terreng og bygninger sammen og bruke enkle polygoner (IndexedFaceSet). Men med tanke på håndtering av detaljnivå, vil nok dette være upraktisk.

4.1.5. Optimalisering av modellen

Når man har generert en visualisering av kartdata, er den som oftest for tung til at navigasjon er praktisk mulig. Derfor må man på en eller annen måte redusere detaljmengden. I en 3D-modell er det ofte gjenstander eller detaljer som er for små, for langt unna, eller skjult bak andre gjenstander. Å gjengi disse er derfor bortkastet og tar opp unødvendige ressurser. En vanlig metode er å ha flere versjoner av samme objekt, der hver versjon er gradvis mer detaljert. Hvilken versjon av objekt som vises, bestemmes av hvilken avstand man observer objektet. Denne metoden betegnes gjerne med forkortelsen LOD (Level Of Detail). Illustrasjonen viser et eksempel (Figur 4.2. Eksempel på LOD.) Et objekt er modellert i 3 versjoner gradvis mindre detaljnivå.



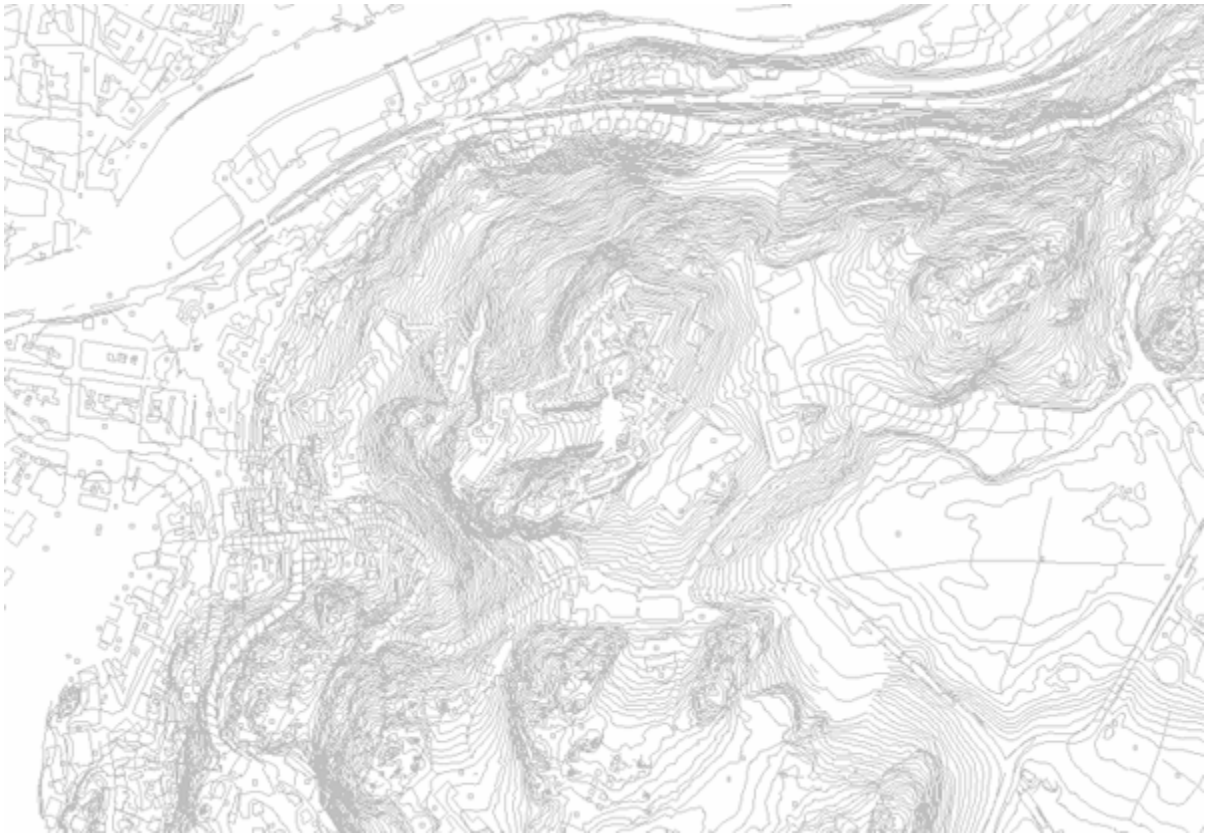
Figur 4.2. Eksempel på LOD

Håndtering av detaljnivå i geografiske modeller er svært viktig, siden disse har stor utstrekning. Avstanden et objekt observeres på, kan variere svært mye og derfor også behovet for detaljer. En annen særegenhet ved geografiske modeller, er at terrenget er et kontinuerlig objekt med utstrekning over hele modellen. Det betyr at den delen av terrenget som ligger nærmest observasjonspunktet, vil være et objekt på nært hold. Mens terrenget når man gradvis fjerner seg fra observasjonspunktet, vil være et objekt sett på stadig større avstand. Terrenget lengst unna observasjonspunktet vil danne horisonten. Behovet for detaljer vil gradvis avta etter hvert som man fjerner seg fra observajonspunktet.

En tommelfingerregel for detaljhåndtering av terreng er følgende: Den relative størrelsen til et triangel i terrenget skal være like stor ute i horisonten som et triangel på nært hold. Det vil si at hvis trianglene i en terrengvisualisering gradvis blir mindre når man ser utover terrenget fra et gitt observasjonspunkt, betyr det at terrenget har for mange detaljer. Omvendt betyr det at man har en for aggressiv detaljhåndtering som fjerner for mange detaljer. Detaljhåndtering er tatt for seg i detalj senere i dette kapittelet (Seksjon 4.5).

4.2. Import av data fra SOSI format

Illustrasjonen under (Figur 4.3. Illustrasjon av høydedata på SOSI format) er en 2D visualisering av terreng data fra en SOSI-fil. Disse data må tilpasses en 3D-visualisering. Det første steget i denne prosessen, er å importere datasettet.



Figur 4.3. Illustrasjon av høydedata på SOSI format

4.2.1. Svakheter ved SOSI datasettet

Som grunnlag til en 3D visualisering, har SOSI datasettet en del svakheter. Den største svakheten er mangelen på referanser. Hvis man tar utgangspunkt i data for bygninger, har man flere objekter som til sammen utgjør de samlede data for en bygning. Men dessverre har ikke disse objektene noen referanse som forteller hvilken bygning de tilhører, og de ender dermed opp som en samling løse linjer som ved et tilfelle ligger i nærheten av hverandre. Flateobjektet som definerer grunnflaten til en bygningsenhet er det eneste unntaket. Dette er ikke noe problem ved en 2D visualisering som den vist over (Figur 3.1. Illustrasjon av SOSI data fra SosiVis), men når man skal lage en 3D visualisering, der er flatene mellom linjene er mer sentrale enn linjene som danner dem, holder ikke dette.

Et eksempel, kanskje trivielt, men som viser tydelig at SOSI dataene ikke er lagd med tanke på bruk i 3D, er balkongene. Hus som har balkong har disse avmerket. En balkong på et hus er et eget linje- eller kurveobjekt. Dette objektet inneholder ingen referanse som kan fortelle hvilket hus det tilhører, heller ikke finnes det noen objekter som refererer til dette objektet. Balkong regnes normalt ikke til grunnflaten av en bygning, og derfor har heller ikke flateobjektet for den aktuelle bygningen noen referanse til balkongobjektet. Det mest tydelige tegnet på at dette er 2D data, er tilfellet der en blokk på flere etasjer har flere balkonger under hverandre, der vil bare den øverste regnes med.

4.2.2. Mangler ved SOSI datasett og hvordan kompensere for disse.

Som nevnt i over har SOSI datasettet en god del svakheter, når man skal bruke det som grunnlag for en 3D visualisering. Den mest åpenbare er mangelen på referanser mellom dataobjektene. En kan riktignok argumentere med at koordinatene i seg selv er en form for

referanse. I SOSI er det slik at objekter, som har en eller flere koordinater felles, refererer til hverandre. Disse koordinatene er markert som knutepunkter og binder linje eller kurveobjekter sammen. I tillegg er det en god del koordinater som har felles verdi om X og Y aksen. Dette gjelder spesielt for bygningsdata.

Eneste mulighet til å knytte disse objektene som tilhører for eksempel en bygning er å utføre et geometrisk søk på alle objektene i datasettet. Dette er en tung og tidkrevende prosess. Hvis man ved hjelp av denne metoden finner alle objektene som hører til, har man fortsatt oppgaven med å sette dem sammen korrekt.

4.2.3. Manglende eller feilaktig rådata

Feil og mangler ved rådata kan også skape problemer. Disse kan være opplagte, slik som at datasettet for kystlinjen mangler, eller de kan være mer snikende og som først kanskje viser seg når man får en 3D visualisering. Et eksempel på det siste har man her (Eksempel 4.1. Utklipp av SOSI data for terreng.)

Eksempel 4.1. Utklipp av SOSI data for terreng

```
.KURVE 29721:
..LTEMA 2001
..DATO 19950504
..KVALITET 60
..NØ
12484402 3892799 600
12484347 3893926 600
12484851 3893996 600
.KURVE 29722:
..LTEMA 2001
..DATO 19950504
..KVALITET 60
..NØ
12484717 3892998 700
12484400 3893874 700
12484851 3893996 700
.KURVE 29723:
..LTEMA 2001
..DATO 19950504
..KVALITET 60
..NØ
12484932 3893137 800
12484467 3893844 800
12484851 3893996 800

[...]
```

```
.KURVE 38347:
..LTEMA 2001
..DATO 19950504
..KVALITET 60
..NØ
12483872 3893475 10500 ...KVALITET 22 21 ...KP 1
..NØ
12484325 3893979 500
12484851 3893996 500 ...KP 999
```

Dette er et utsnitt av en SOSI-fil. Utsnittet viser fire KURVE objekter. LTEMA 2001 er betegnelsen for høydekurver. Koordinatene angir posisjon i nordlig og østlig retning, samt høyde over havet. Enheten er i centimeter. Dette utsnittet kan hjelpe til å belyse flere problemer med SOSI. For det første, den siste koordinaten i alle fire kurvene er like, med unntak av høyden. Her kan man spørre seg hvilken høyde som er korrekt å bruke i en visualisering. En annen ting er at disse punktene ikke heller er markert som noe knutepunkt, i KURVE 38347 har koordinaten angivelsen, KP 999, som betyr at det er et åpent endepunkt. KP 1 derimot, som angitt etter den første koordinaten i samme kurve, betyr at linjen kurven er knyttet til, er en annen kurve eller linje via denne koordinaten. Dette er typisk for mye av datasettet for terrenget.

Det som gjør dette eksemplet ekstraordinært er høydeangivelsene. Legg merke til første koordinat i KURVE 38347. Den oppgir høyden, for dette punktet til å være 105 meter over havet, mens de andre punktene har en høyde på mellom 5 og 8 meter. Høydekurvene som ligger rundt, har en høyde på mellom 100 og 120 meter. Hvis dette er korrekt, skulle det bety at det på dette stedet befinner seg et hull på over hundre meter. Man kan med selvsyn på stedet slå fast at dette absolutt ikke er tilfelle. Denne feilen i datasettet er usynlig i en 2D visualisering, men i en 3D visualisering blir det svært tydelig.

4.2.4. Mulige forbedringer av SOSI data

Forbedringer av disse problemene ville være referanser som viser at dataobjekter hører sammen. For å ta et eksempel; bygningsdata. Her kunne hvert objekt ha inneholdt en referanse til hvilken bygning den tilhører, bygningsnummeret kunne for eksempel være referansenøkkel. I tilfellet med eiendomsdelelinje som markerer skillet mellom to bygninger som deler vegg, kan den ha referanse til begge bygningene den skiller. Rekkefølgen referansene kommer i kan angi hvilken side av skillelinjen de respektive bygningene ligger. Enda bedre, referansen oppgir spesifikt hvilken side av linjen, bygningen ligger.

En annen mulighet er at to objekter deler et punkt, det vil si at begge dataobjektene inneholder samme koordinat. I visse tilfeller er dette markert som knutepunkt, noe som gjør det lettere å knytte disse sammen. Men det kunne også være ønskelig å ha en referanse til hvilke objekter som deler koordinater og dermed hvilke de henger sammen med.

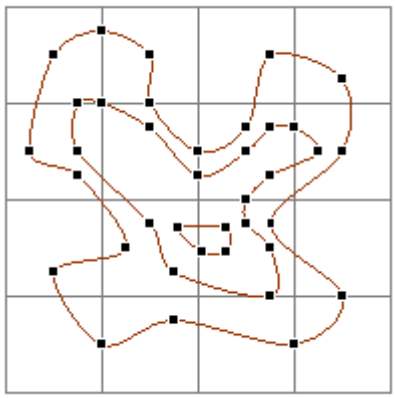
4.3. Modellering av terreng

4.3.1. Overføring av vektorbaserte data til elevationGrid

Rådataene er vektorbaserte data, med linjer og punkter for å definere formen på terrenget, mens elevationGrid er et regulært rutenett, med lik avstand mellom punktene. ElevationGrid har i så måte mye til felles med rasterbaserte data, som utformet som rader av uniforme celler, der celler er kodet i overensstemmelse med dataverdier.

Dette temaet er beslektet med problematikk rundt spredte data. Når man innhenter for eksempel geografiske data, hender det at målepunktene er ujevnt fordelt, og man må foreta en approksimasjon mellom dem. Man har da to alternativer for approksimasjon, et regulært grid eller en triangulering (Fremming, Hjelle & Tarrou, 1997).

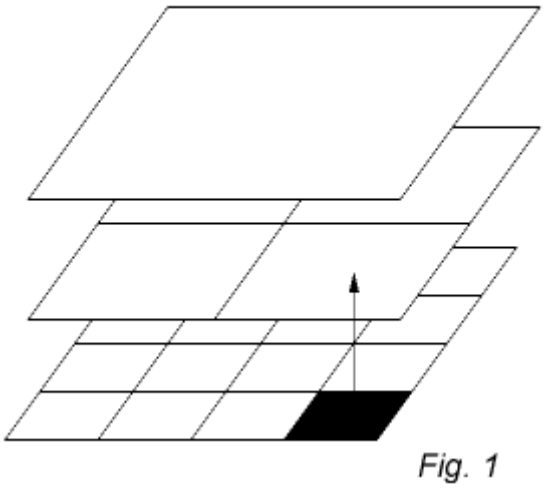
Bildet under visualiserer prosessen fra vektorbaserte data til rutenett eller elevationGrid (Figur 4.4. Illustrasjon av vektorbaserte Kartdata.)



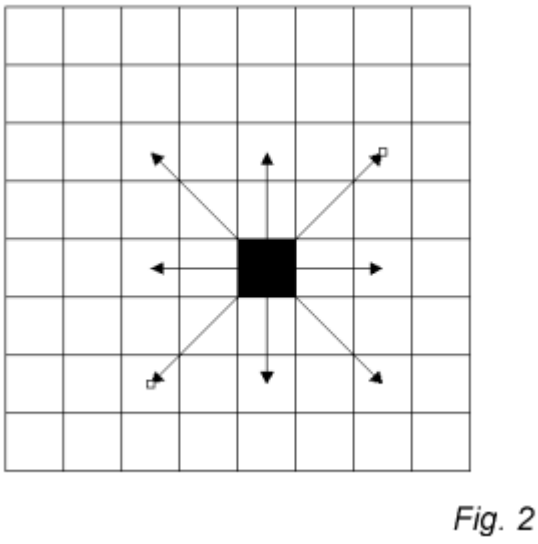
Figur 4.4. Illustrasjon av vektorbaserte Kartdata

Et sett kurver, linjer og punkter definerer terrenget. Over er det lagt et rutenett. Hvert punkt i rutenettet skal ha en høydeverdi, som korresponderer med vektordatasettet (USGS Eastern Region Geography, 2003). En simpel metode for å få til dette er å legge et rutenett, som på illustrasjonen over, og så finne hvilke punkter som faller innenfor de aktuelle rutene. Til slutt tar man gjennomsnittverdien for punktene innenfor hver rute. Denne metoden gir en omtrentlig korrekt høyde for hvert punkt i rutenettet, gitt at rådatasettet har et eller flere punkter definert innenfor den aktuelle ruten.

Dette fungerer best, jo høyere tettheten i datasettet er, relativt i forhold til størrelsen på rutene i rutenettet. Kupert terreng uten skrenter, der man ikke har bratte stup eller skrenter, er terrenget der det fungerer best. Et annet kriterium er at terrenget er omtrent like kupert over hele området. Hvis man har større ruter i rutenettet er sjansen større for at det eksisterer et punkt innenfor et gitt rutefelt. Problemer oppstår med denne metoden, når man ikke har et punkt gitt innenfor en rute. Man vil ikke ha noe utgangspunkt for å beregne høyden for dette punktet i rutenettet, man vil da få et hull i rutenettet. Dette oppstår der det er lav tetthet av data, for eksempel store flate områder, der det er langt mellom kotelinjene.

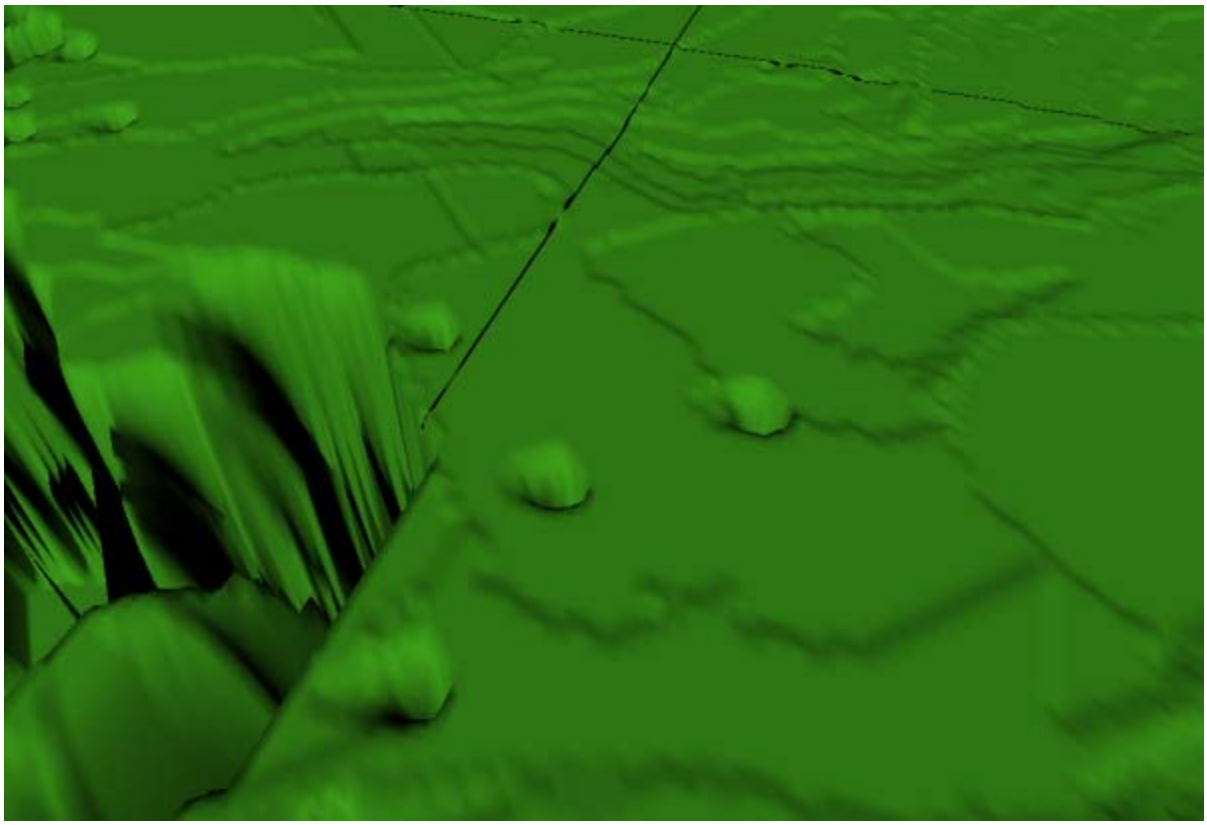


Figur 4.5. Interpolasjon Figur 1



Figur 4.6. Interpolasjon Figur 2

En måte å bøte på dette på, er å interpolere mellom omkringliggende punkter (Figur 4.6. Interpolasjon Figur 2), eller hente verdi fra ruten som ligger høyere i LOD-hierakiet (Figur 4.5. Interpolasjon Figur 1). Den siste metoden krever at man har bygd opp et slikt hierarki av kvadranter med forskjellige detaljnivåer (Seksjon 4.5.2). Ingen av disse metodene gir noen garanti for at man får en verdi for en gitt rute. Dessuten vil man uansett aldri få helt korrekte verdier for hvert punkt i rutenettet. Spesielt ved bruka av den første metoden får man verdier som man tydelig kan se er malplasserte.



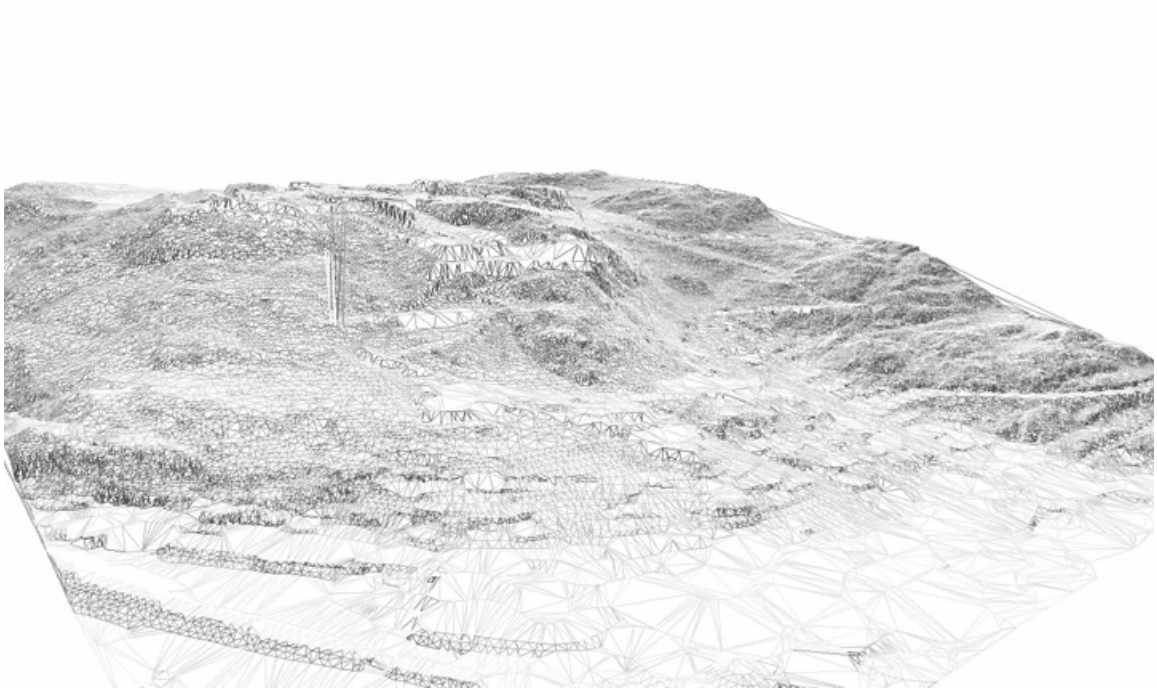
Figur 4.7. Eksempel mangelfullt rutenett for terreng.

I illustrasjonen over (Figur 4.7. Eksempel mangelfullt rutenett for terreng.) ser man et tydelig eksempel på manglene ved dette. Til venstre ser man at mangel på data for dette området har ført til et stort hull. I tillegg kan man se et par mindre hull i forgrunnen. En annen svakhet er at for de punktene der man får en noenlunde fornuftig verdi, er denne ikke alltid korrekt. Resultatet bli at man får brå overganger mellom kotelinjene. På illustrasjonen over kan man tydelig se kotelinjene i terrenget. Derfor bør man bruke en bedre form for approksimasjon.

En bedre, men mer komplisert løsning på problemet er å lage en triangulering mellom punktene. Dette betyr at man bygger opp triangler mellom punktene i datasettet. Eventuelt med linjene og kurvene som beskrankninger. Deretter brukes denne trianguleringen til å approksimere verdiene i elevationgridet. Denne metoden er betydelig mer komplisert og har sine egne problemer. I tillegg krever den mye mer datakraft.

4.3.2. Triangulering av terreng

Visualisering av terreng krever flere steg. Prosessen kan inndeles i følgende steg, triangulering og approksimasjon. Det første steget i visualiseringen er å lage en triangulering mellom punktene som man har importert fra SOSI formatet. Ved å bruke Delaunay trianguleringer får man et nett av triangler. Når man trianguleret kan velge å bruke linjene i datasettet som beskrankninger. Det vil si at linjene som blir skapt i prosessen ikke får lov til å krysse disse linjene. Alternativt kan man velge å bruke bare punktene enkeltvis, uten informasjon om linjene mellom dem. Resultatet bør uansett tilsvare noe som vist i figuren under. (Figur 4.8. Trianguleringsgrid av Fredriksten Festning)



Figur 4.8. Trianguleringsgrid av Fredriksten Festning

Det kan oppstå et problem. Delaunay triangulering er en strengt todimensjonal algoritme. Hvis to punkter befinner seg på samme sted i XY-planet, men har forskjellig høyde, bryter trianguleringsprosessen sammen. Dette skal normalt ikke oppstå, hvis datasettet er korrekt.

4.3.3. Approksimering av rutenett

Når man fått lagd en komplett triangulering av terrenget, kan man begynne å lage et regulært rutenett. Rutenettet lager man ved å approksimere hvert punkt i rutenettet. Dette gjøres ved å først finne ut hvilke triangler punktet faller innenfor rammen av. Rammen til et triangel finner man ved å ta høyeste og laveste verdiene både for X og Y aksen. For den eller de trianglene, som punktet er innenfor rammen til, regner man ut de barysentriske koordinatene for det gitte triangelet. De barysentrisk koordinatene kan regnes ut etter følgende formler:

u = areal(P, B, C)/areal(A, B, C)

v = areal(A, P, C)/areal(A, B, C)

w = areal(A, B, P)/areal(A, B, C)

A, B og C er punktene i triangelet. P er punktet man skal finne ut om ligger innenfor omkretsen av triangelet.

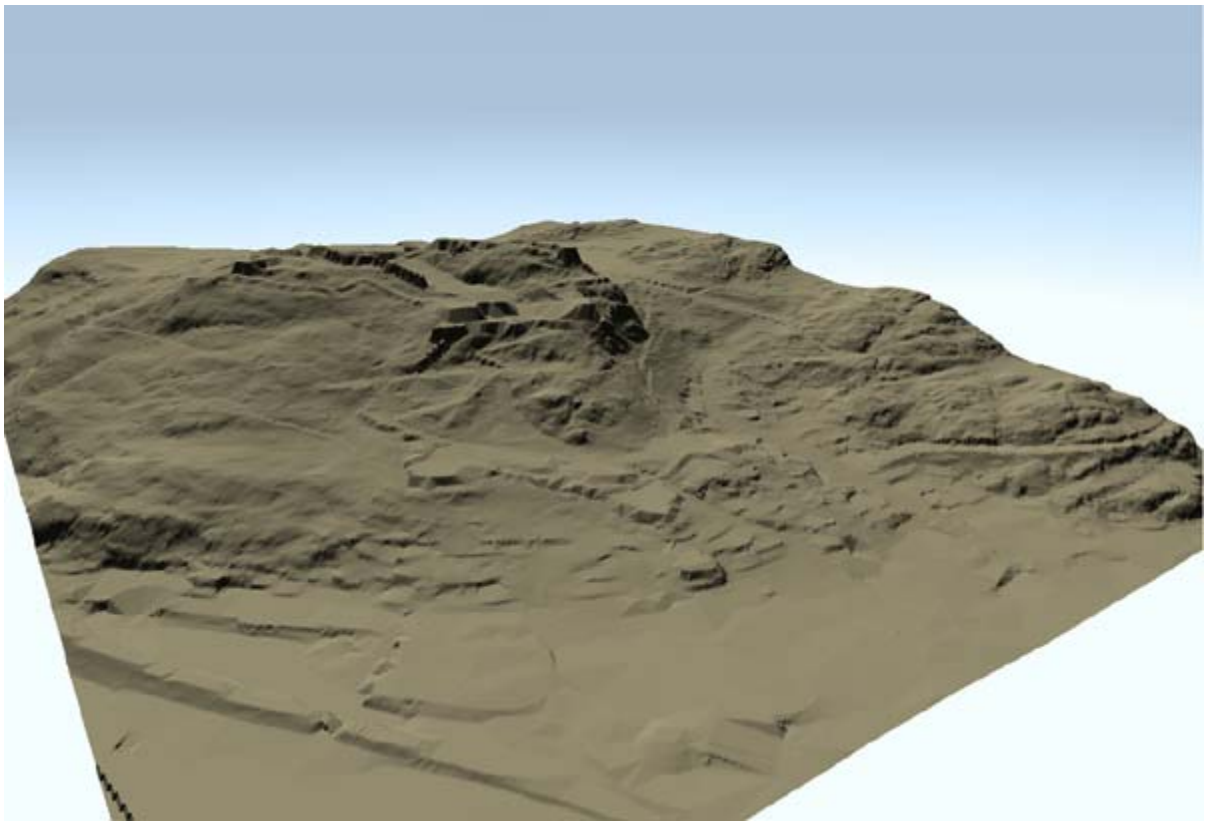
Arealet regnes ut etter formelen:

areal(A, B, C) = ($A_xB_y + B_xC_y + C_xA_y - A_yB_x - B_yC_x - C_yA_x$)/2

Hvis alle de barysentriske koordinatene er positive, betyr det at punktet ligger innenfor trianglets omkrets. Det neste steget er i så fall å regne ut høyden på punktet etter formelen:

$P_z = u\ A_z + v\ B_z + w\ C_z$

Når man har utført denne beregningen for hvert punkt i det regulære rutenettet, får man høyden for hvert av punktene i elevationgridet. Som sluttresultat får man rutenett, som man kan bruke som et elevationGrid. Bildet under (Figur 4.9. Elevationgrid for Fredriksten Festning) viser et elevationGrid approksimert fra triangulering av punkter.

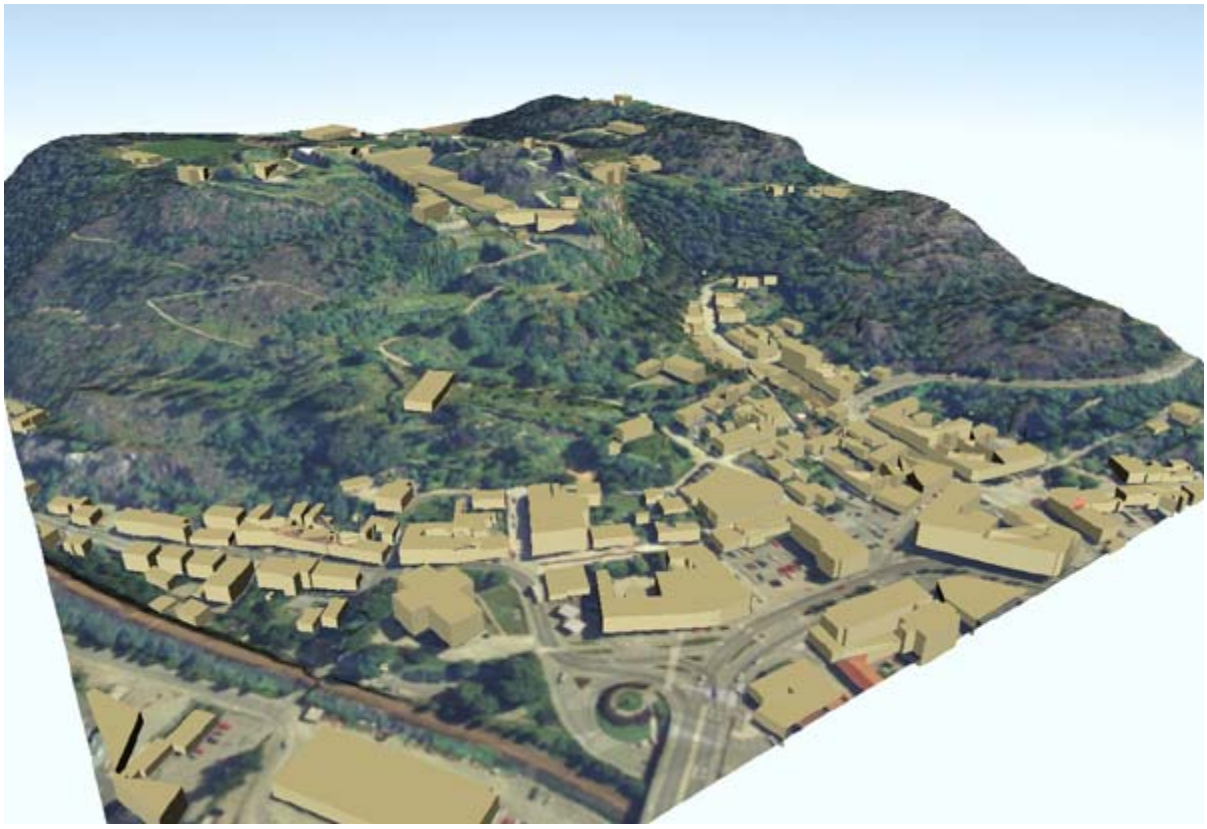


Figur 4.9. Elevationgrid for Fredriksten Festning

4.4. Modellering av bygninger

4.4.1. Metoder for å modellere bygninger

Visualisering av bygninger krever en annen form for visualisering. Bygninger i motsetning til terreng er en mengde distinkte objekter. Bygninger har også i de fleste tilfeller, loddrette vegger og egner seg derfor ikke som en del av et elevationGrid. En enkel men effektiv måte å visualisere bygninger på, er å bruke Extrusion. Man tar da grunnflaten til bygget og ekstruderer nedover. En ordinær bolig med fire vegger, blir da en grå boks med flatt tak. Illustrasjon under viser et eksempel på dette (Figur 4.10. Festningen med bygninger). Dette er begrensningen med Extrusion. For næringsbygg der taket faktisk er flatt i virkeligheten, er dette et mindre problem.



Figur 4.10. Festningen med bygninger

Som nevnt tidligere i kapittelet (Seksjon 4.1.1) er en metode å ekstrudere ned til havnivå. Dette fungerer fordi flaten til bygningen er definert av takkanten til bygningen. Hvis arealet til bygningen hadde vært på bakkenivå, måtte man ekstrudert nedenfra og oppover. Og dermed kjenne høyden på huset. Å ekstrudere ned til havnivå er ikke alltid ideelt. Hvis bygningen ligger høyt over havet, blir ekstrusjon et svært langt objekt der bare toppen synes. Dette kan igjen føre til avrundingsfeil, problemer med z-buffering og nedsatt ytelse. En bedre tilnærming er bare å ekstrudere ned til det laveste punktet der bygget skjærer terrenget. Dette krever imidlertid at man må hente inn terrengdata, noe som gjør det mye tyngre å generere bygninger.

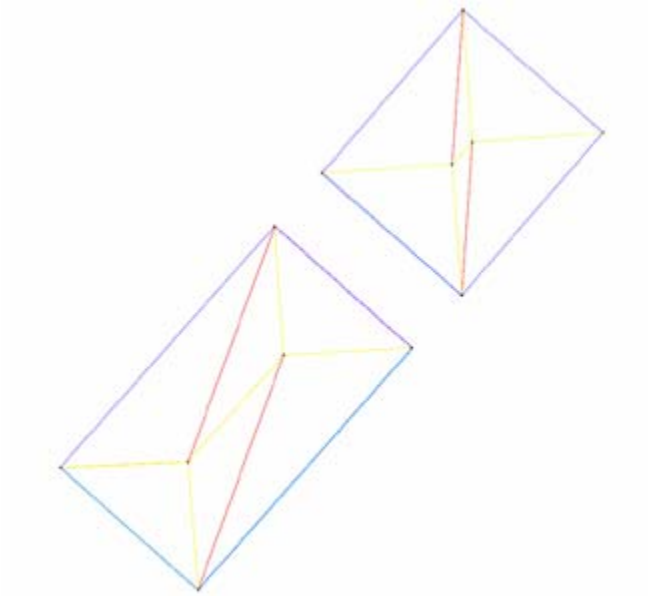
Et annet alternativ til å bruke extrusion noden er å bruke indexedfaceset noden for å modellere en bygning. Med indexedfaceset står man helt fritt til å definere formen på en bygning. Ulempen er at den er litt mer komplisert å bruke. Men skal man ha mer detaljerte bygninger, kommer man ikke utenom. For eksempel hvis de forskjellige veggene på en bygning skal ha teksturer, kan man ikke bruke extrusion.

Men en mulighet er å kombinere disse. Man kan bestemme om en extrusion node skal ha endestykker og hvis man velger å ikke ha endestykke, kan erstatte dette med en indexedfaceset node som modellerer taket på bygningen. Modellering av tak er på mange måter et kapittel for seg.

4.4.2. Modellering av tak

Den enkleste måten å visualisere tak på er bare å bruke en enkelt flate som i ekstrusjon. Dette er ofte ikke tilfredsstillende. En forbedring er å ta et utklipp fra et ortofoto av taket og legge dette på flaten. Mer detaljert modellering av tak byr på en rekke utfordringer og problemer. Og SOSI datasettet er stort sett kilden til de fleste av dem. Men generelt sett er det ikke helt

trivielt å lage en generell algoritme som fungerer korrekt i alle tilfeller, selv om datasettet skulle være korrekt. Takkonstruksjonstyper varierer fra de enkleste der taket er helt flatt, via det typiske bolighus med skråtak og en enkelt mønekant, til mer komplekse konstruksjoner med valmet tak (Figur 4.11. To hus med valmet tak), med takutspring og tilbygg med taksprang. En fremgangsmetode kunne være å lage kategorier for type tak, slik som flatt tak, skrått tak, valmet tak og lignende. Og lage en metode for å visualisere de forskjellige typene optimalt. Men forskjellige objektene for å angi takkonstruksjon er brukt i stort alle tenkelige kombinasjoner. Dette er grunn til at dette ikke går. Datasettets oppbygning gjør det også vanskelig å kategorisere på denne måten.



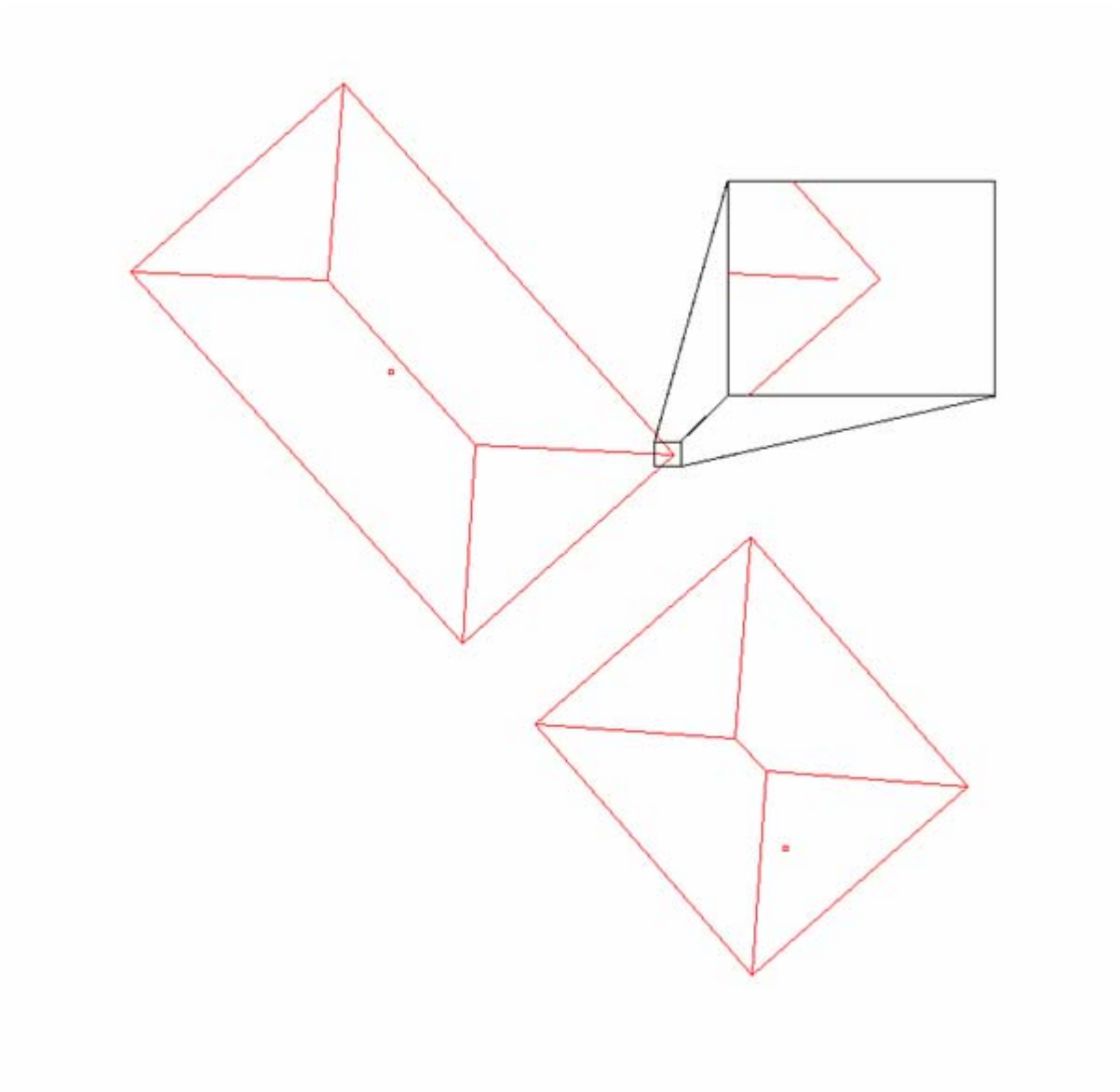
Figur 4.11. To hus med valmet tak

Svakhetene ved oppbyggingen av SOSI-datasettet er mest tydelig når det gjelder takkonstruksjon. Datasettet har flere svakheter som gjør det vanskelig om ikke umulig å visualisere takkonstruksjon. Den første og mest åpenbare er mangelen på referanser til bygning. Som nevnt tidligere (Seksjon 4.2.1) mangler SOSI-datasettet referanser, i dette tilfellet referanse til bygning. Derfor er eneste måten å finne ut hvilken bygning et gitt linjestykke tilhører, å gjøre et geometrisk søk. Man søker igjennom alle bygningsflatene for å se hvilken flate linjestykket faller innenfor og dermed tilhører. Dette er en temmelig tung prosess og med større mengder data temmelig tidkrevende. I tillegg gir den av andre grunner, som vi skal se, ikke noen garanti for suksess.

En annen svakhet som rett og slett gjør det umulig å lage en helt korrekt visualisering av takkonstruksjoner, er bygningslinjer og bygningsdelelinjer. Disse objektene markerer ofte loddrette flater. Høydekoordinatetene angir de øverste punktene av denne flaten, men ikke de nederste. Man vet dermed hvor flaten begynner, (Sett ovenfra,) men ikke hvor den slutter. Dette er typisk trekk som røper at dette datasettet ikke var beregnet på 3D visualisering. Det eneste man kan gjøre i dette tilfellet, er å godta at visualiseringen ikke vil bli korrekt.

Det siste problemet er noe som ikke kan karakteriseres som noe annet enn direkte feil. Problemet kan best illustreres med et par eksempler. Det første eksempelet er et hus med valmet tak. (Figur 4.12. Hus med Valmet tak, utsnitt av hjørnet) Takkonstruksjonen på dette huset, defineres av et par linje objekter, som definerer mønekanten. Hvis man studerer et av hjørnene, ser man at linjen som definerer mønekanten ikke er korrekt knyttet sammen med linjen som definerer takkanten. Skulle det vært korrekt, burde det ha vært et felles knutepunkt.

Det som er verre er at endepunktet for mønelinjen kan ligge på utsiden av linjen for takkant. Dette er ikke bra av flere grunner.



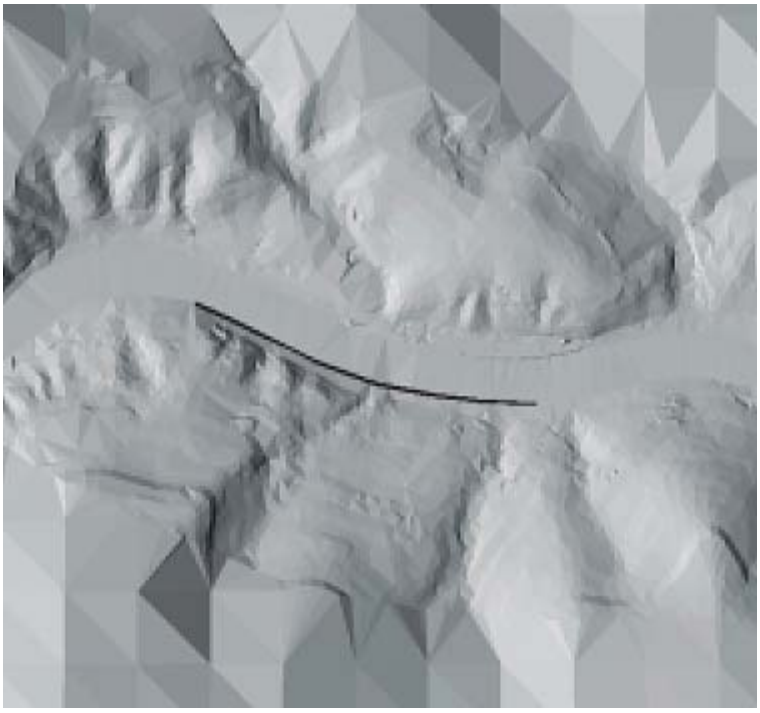
Figur 4.12. Hus med Valmet tak, utsnitt av hjørnet

4.5. Håndtering av detaljnivå - LOD

Håndtering av detaljnivå er viktig for at visualiseringen skal gå så jevnt som mulig. Selv om maskinene blir stadig raskere, er det fortsatt begrenset hvor mye informasjon man kan vise på en gang. Derfor er det viktig at man ikke viser høyere detaljnivå enn nødvendig. Detaljnivå, eller Level of Detail forkortes gjerne LOD og denne forkortelsen er gjerne mest brukt i omtale av dette feltet. LOD, generelt, går i prinsippet ut på at man har flere versjoner av samme objekt, men der de forskjellige versjonene har forskjellig detaljnivå. Avhengig av hvilken avstand man ser objektet på vises de ulike versjonene. Det er ingen grunn til å vise detaljer som er for langt borte til å synes. Det finnes to forskjellige metoder for håndtering av detaljnivå. Den ene er en variabel geografisk skala og den andre er hierarki oppbygd av kvadranter. Den sistnevnte metoden betegnes ofte med forkortelsen geoLOD (geographic Level of Detail).

4.5.1. Flater med variabel skala

Grunntanken bak teknikken med flater med variabel skala er at man kombinerer to eller flere forskjellige skaleringer på samme flate. Prinsippet er slik at man for de områdene som er av spesiell interesse, har et høyt detaljnivå, mens områdene rundt får gradvis lavere detaljnivå, til man har ytterkantene av flaten, der bare et fåtall triangler visualiserer trekkene ved terrenget (Misund, 1997). Illustrasjonen viser et eksempel på en slik flate. (Figur 4.13. Eksempel terrengflate med variabel skala.)

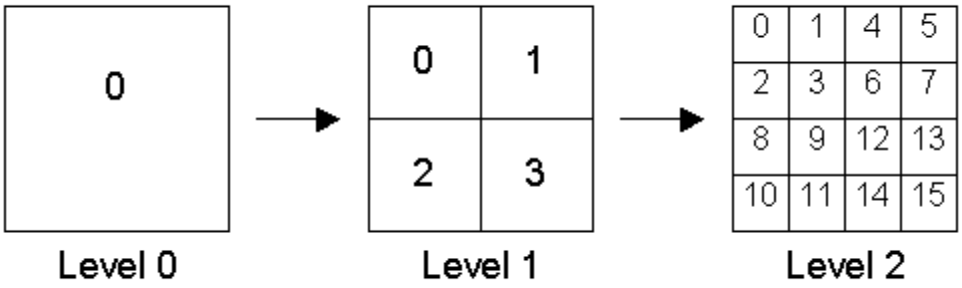


Figur 4.13. Eksempel terrengflate med variabel skala (Schilling & Zipf, 2003).

Fordelen med denne metoden er at man får helt jevne overganger fra et detaljnivå til et annet. Man slipper skarpe kanter når man splitter terrenget opp i kvadranter. Ulempen er at denne metoden er temmelig komplisert å implementere. I tillegg er detaljnivået statisk. Derfor fungerer bare denne metoden hvis man bare har begrensede navigasjonsmuligheter, slik at man hele tiden vet hvilke områder det vil være fokus på.

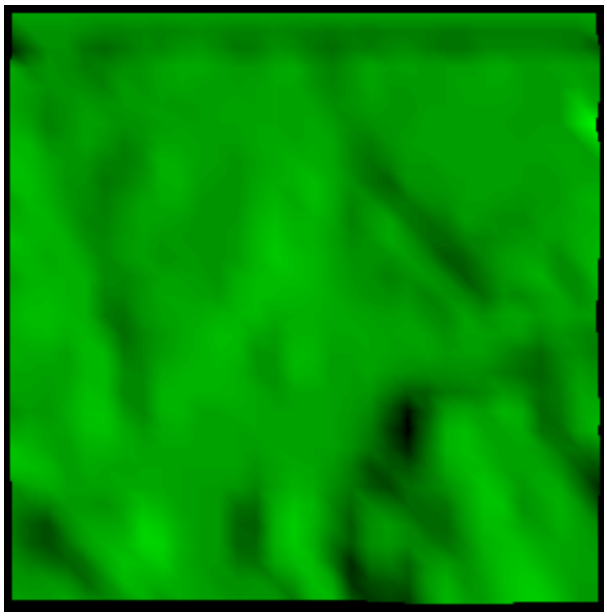
4.5.2. Håndtering av detaljnivå ved hjelp av kvadtrær

En annen metode for detaljhåndtering av terreng er kvadtrær. Et gitt område blir splitt opp i fire mindre kvadrater, der hvert av disse har et høyere detaljnivå enn nivået over. Disse kvadratene blir igjen splittet opp i nye fire kvadrater. (Figur 4.14. Oppsplitting av terreng i mindre deler)



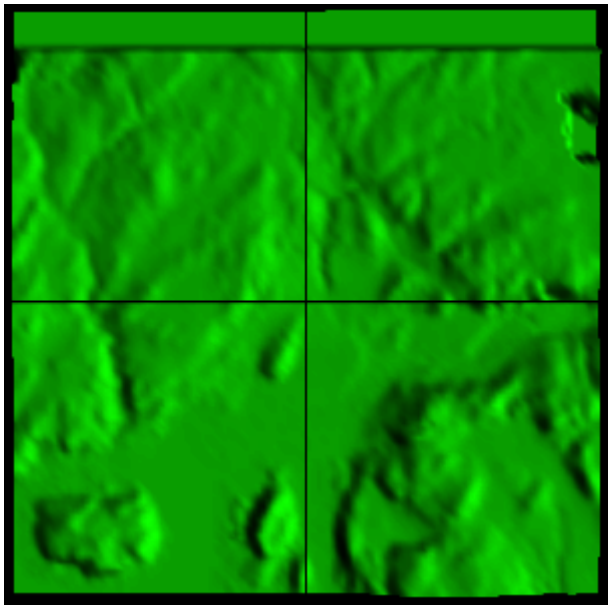
Figur 4.14. Oppsplitting av terreng i mindre deler

På denne måten bygger de et hierarki der toppnivået har et lavt detaljnivå, mens nivåene nedover i hierarkiet har et gradvis høyere detaljnivå for et mindre område, til man til slutt har det laveste nivået der man har det høyeste detaljnivået. For hvert nivå man går ned, splittes området fra nivået over opp i fire nye deler. Prinsippet kan illustreres med et eksempel.



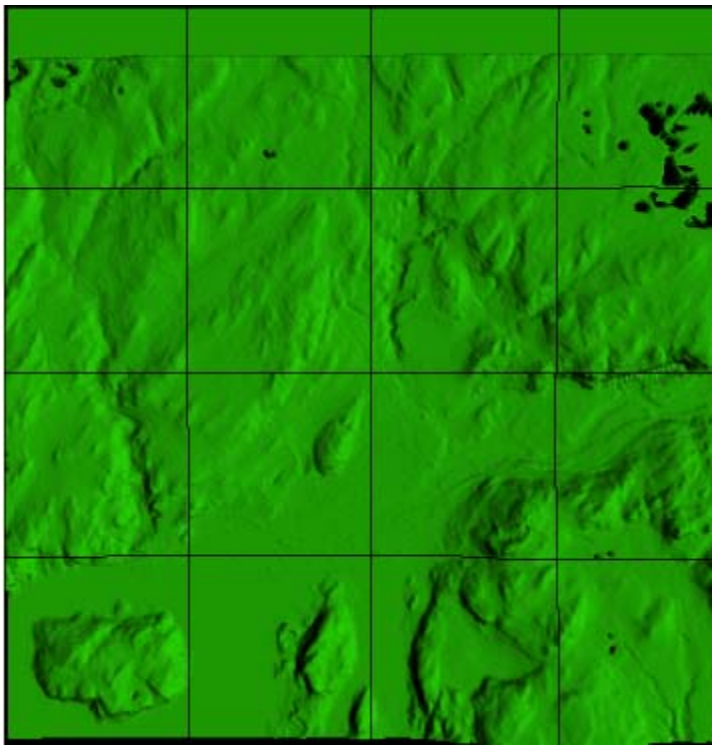
Figur 4.15. Terreng med lav oppløsning

Det øverste nivået er et enkelt kvadrat, med lavt detaljnivå. Denne vises når man ser terrenget på lang avstand (Figur 4.15. Terreng med lav oppløsning).



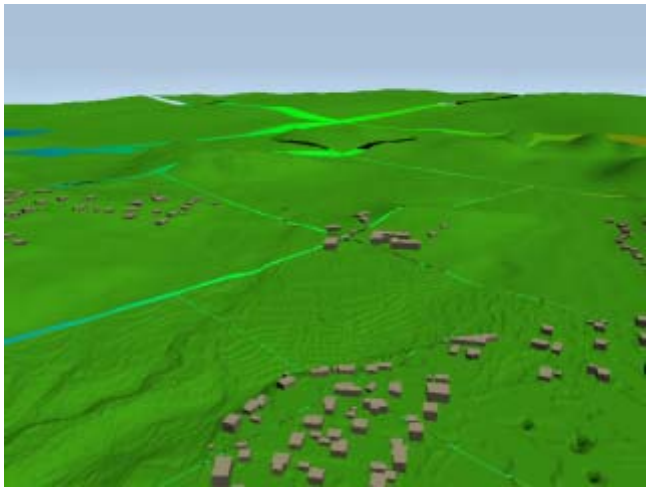
Figur 4.16. Terreng med middels oppløsning

På det neste nivået er terrenget splittet opp fire like store kvadranter. Disse har et høyere detaljnivå enn det forrige. Disse vises når man ser terrenget fra en litt mindre avstand (Figur 4.16. Terreng med middels oppløsning).



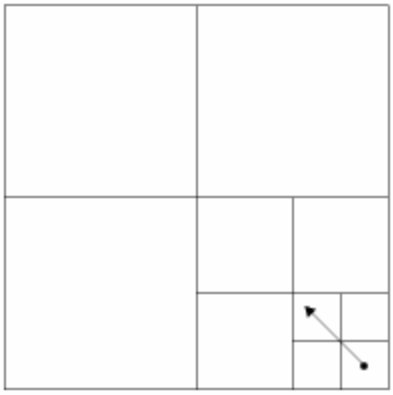
Figur 4.17. Terreng med høy oppløsning

Det høyeste detaljnivået der kvadrantene fra nivået igjen er splittet opp i nye fire nivåer. Det opprinnelige terrenget er nå splittet opp i 16 deler. Dette nivået er på nært hold (Figur 4.17. Terreng med høy oppløsning).



Figur 4.18. Eksempel på effekt av geoLOD

Illustrasjonen (Figur 4.18. Eksempel på effekt av geoLOD) viser hvordan det kan se ut i praksis. Nærmest har man de minste kvadrantene av terrenget med det høyeste detaljnivået. Lenger unna har man gradvis større kvadrater med lavere detaljnivå. Lengst unna, i horisonten har man de største kvadrantene, og det aller laveste nivået. Figuren under (Figur 4.19. Oppdeling av et terreng avhengig av utsiktspunkt) viser illustrer tydeligere hvordan oppdelingen foregår. Pilen viser synsretningen.



Figur 4.19. Oppdeling av et terreng avhengig av utsiktspunkt

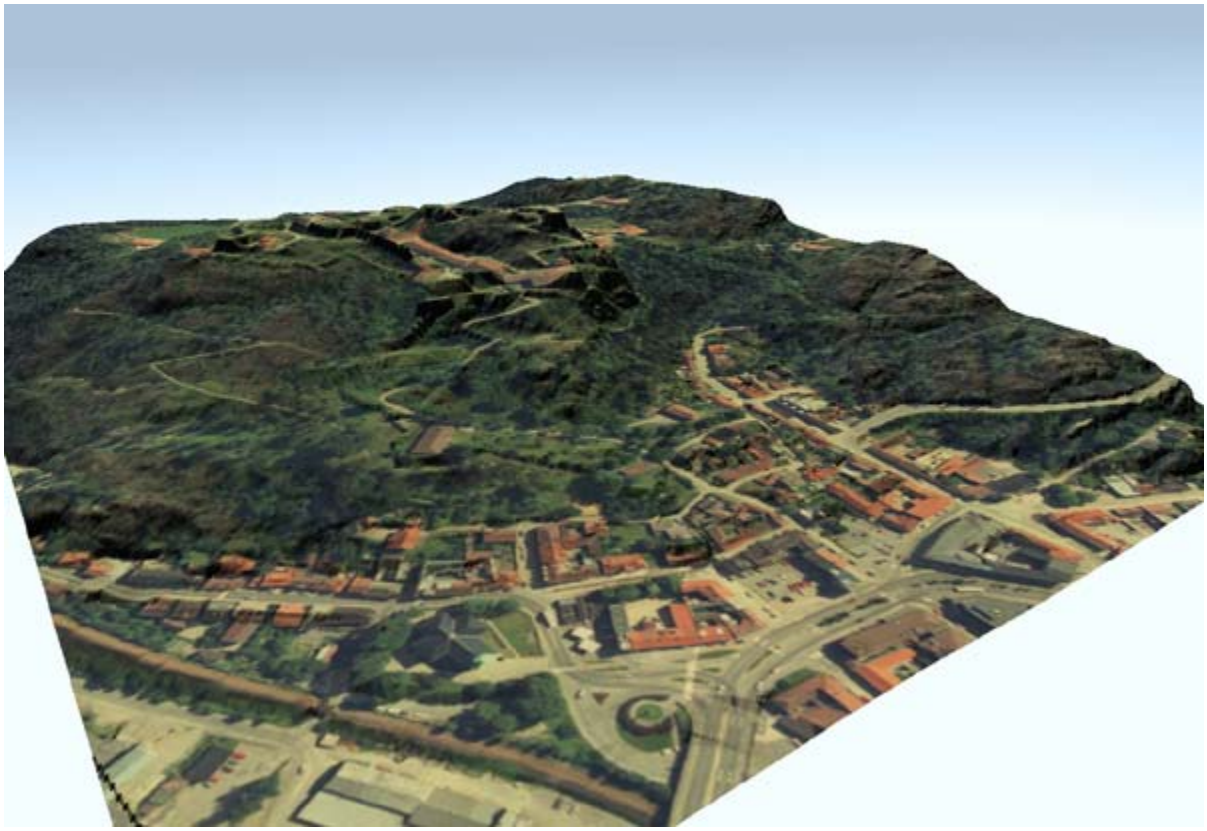
Fordelen med å splitte opp terrenget i forskjellige detaljnivåer, er at man ikke viser flere detaljer enn nødvendig. En annen fordel er at det er lettere for en fremviser å vise flere små deler av terrenget enn å vise en stor modell av hele terrenget. Det betyr også at hele modellen ikke bør være i minne hele tiden, noe som ytterligere øker ytelsen. Ulempen er at man får skarpe skillelinjer mellom kvadrantene. Men denne løsningen er mer fleksibel og fungerer bedre når man skal navigere fritt i modellen, og man ikke kan på forhånd bestemme hvilket område som skal være i fokus.

4.6. Bruk av teksturer

For en troverdig visualisering må man ha tekstur. Teksturer kan man bruke fra flere kilder. Det kan være ortofoto, bilder tatt på stedet, eller generisk tekstur.

For terreng er ortofoto best egnet. Dette er flyfoto tatt direkte ovenfra, i hvert fall skal de ideelt sett være det. I virkeligheten vil man alltid få litt avvik, spesielt i ytterkant av bildene. For å tilpasse teksturen til terrengmodellen er det viktig at geokoordinatene for ortofotoet og terrengmodellen stemmer overens, og at disse har samme referansepunkter og datum.

Fordelen med bruk av ortofoto, hvis teksturen er korrekt plassert, vil vann, veier og jernbane bli visualisert. (Figur 4.20. Festningen med ortofoto som tekstur)



Figur 4.20. Festningen med ortofoto som tekstur

Ortofoto kan også være nyttig når man skal legge tekstur på tak. Man kan da ha et utklipp av ortofotoet som tilsvarer taket på bygningen, og på den måten få lagt tekstur på taket.

Derimot som tekstur på loddrette flater, som for eksempel husvegger, er ortofoto helt uegnet. Her kan man bruke bilder tatt av det aktuelle objektet, slike bilder er stort sett for tidkrevende å innhente, i tillegg vil det være svært komplisert å implementere en rutine som legger bilder på veggene som tekstur.

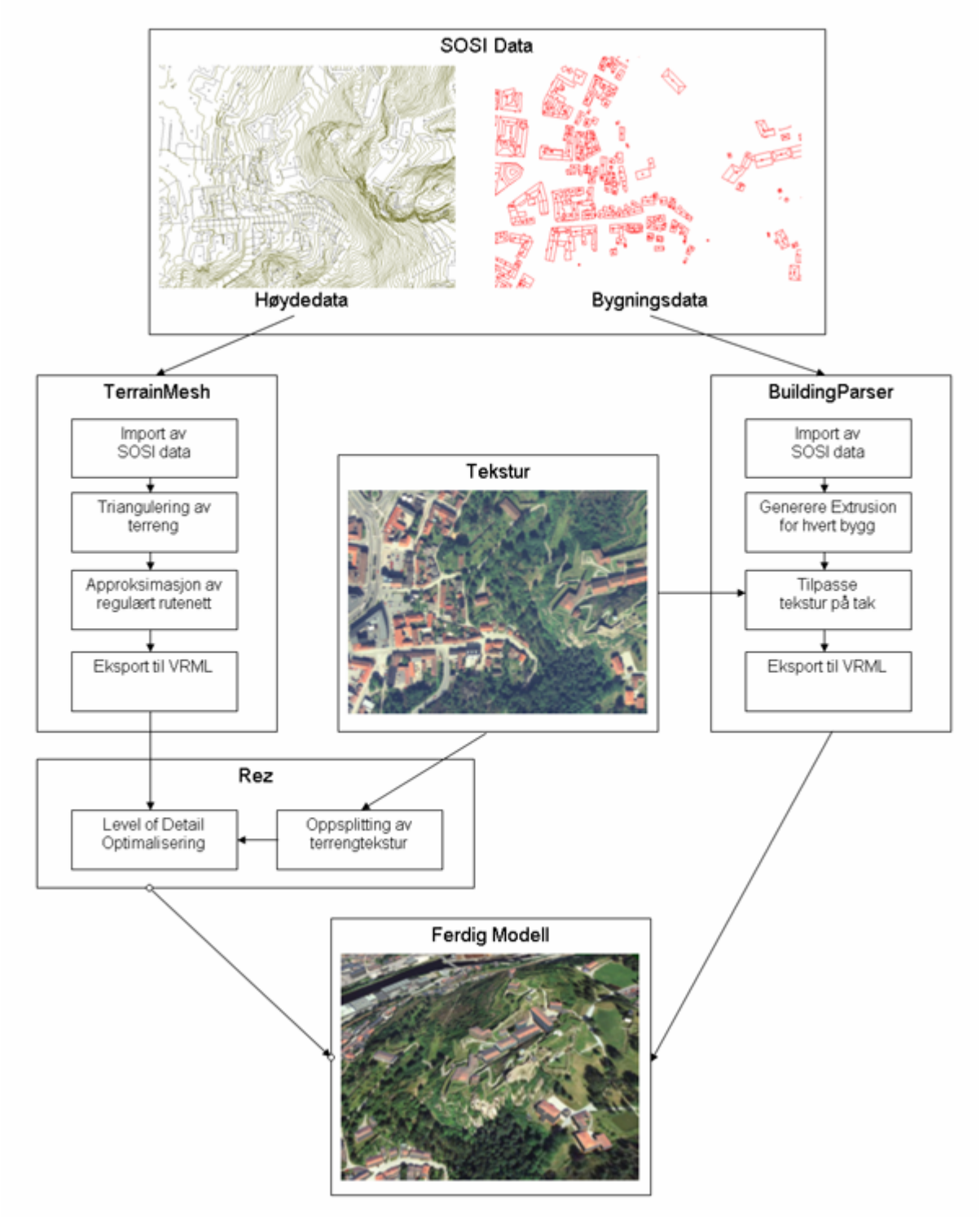
Et alternativ kan være å lage et sett med generiske teksturer. Disse teksturene kan fungere slik at man har en tekstur for en bestemt bygningstype, og på den måten gjøre modellen litt mer troverdig, uten at man behøver å ta bilde av hver eneste vegg som er med i modellen. Generiske teksturer er det selvsagte valget når man kommer til å legge inn detaljer, som gjerder og murer.

Kapittel 5. Implementasjon

I dette kapittelet vil jeg ta for meg implementasjon og resultatene av disse. Som programmeringsspråk for dette prosjektet valgte jeg å bruke Perl og Java. Jeg valgte Perl i første omgang av to grunner. Den første grunnen er at det er raskt å lage små prototyper for å teste ut teorier. Den andre grunnen er at Perl er svært godt egnet til å behandle tekst. Dette er viktig når både SOSI datasettet og VRML er tekstbaserte formater. For større implementasjoner valgte jeg å bruke Java. Grunnlaget for dette valget ble mye diktert av at både Javamesh og Rez er implementert i Java. I tillegg til det faktum at Java skalerer bedre enn Perl.

5.1. Overordnet beskrivelse

Generelt består visualiseringen av to deler, terreng og bygninger. Det ble naturlig å behandle disse hver for seg og som siste steg føye dem sammen. Når man gjør det på denne måten er det viktig at man bruker samme referansepunkter for begge typer data om man vil unngå problemer i siste steg av prosessen. Figuren under viser den generelle arbeidsflyten mellom de forskjellige komponentene i dette prosjektet (Figur 5.1. Oversikt over komponentene i prosjektet).

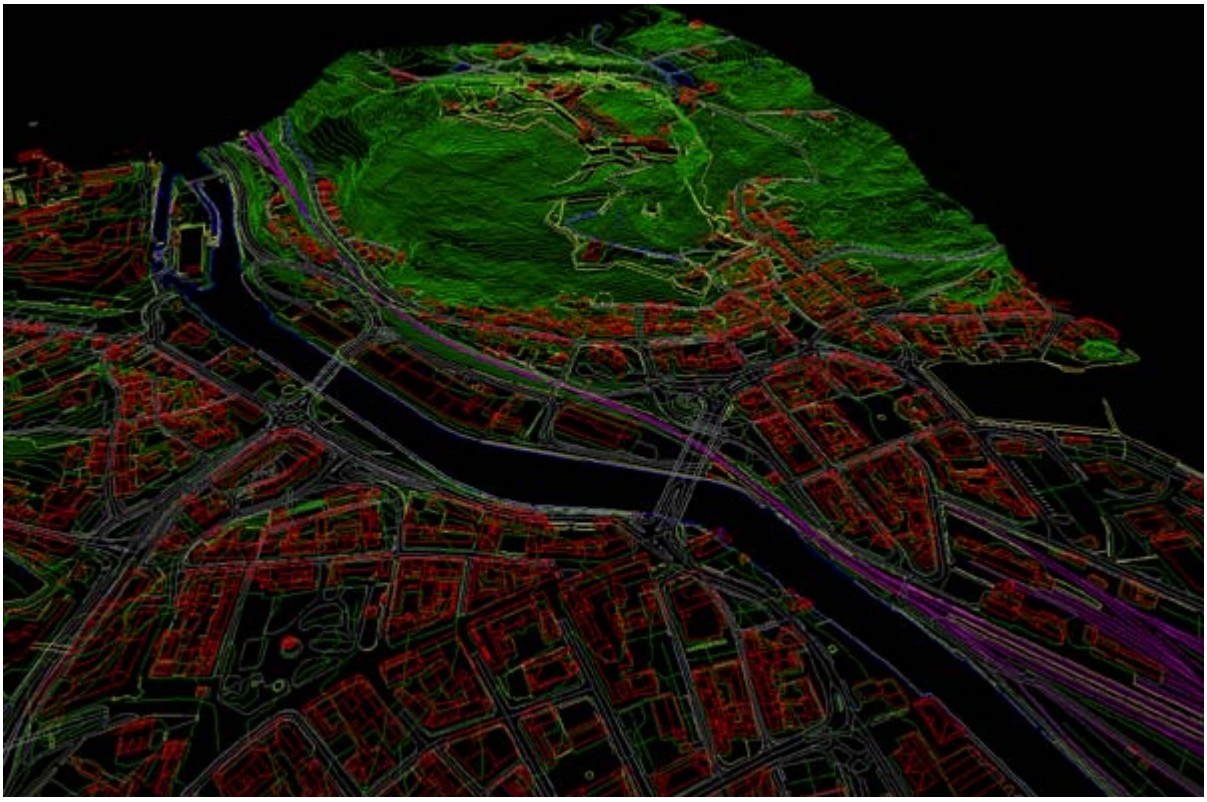


Figur 5.1. Oversikt over komponentene i prosjektet

Kort fortalt fungerer det på denne måten: Man tar utgangspunkt i SOSI data og tekstur basert på ortofoto. Høydedata og bygningsdata blir behandlet av henholdsvis TerrainMesh og BuildingParser. TerrainMesh genererer ferdig terrengmodell som deretter får lagt på tekstur og blir optimalisert av Rez. Denne blir tilslutt satt sammen med bygningsmodellen som blir generert av BuildingParser.

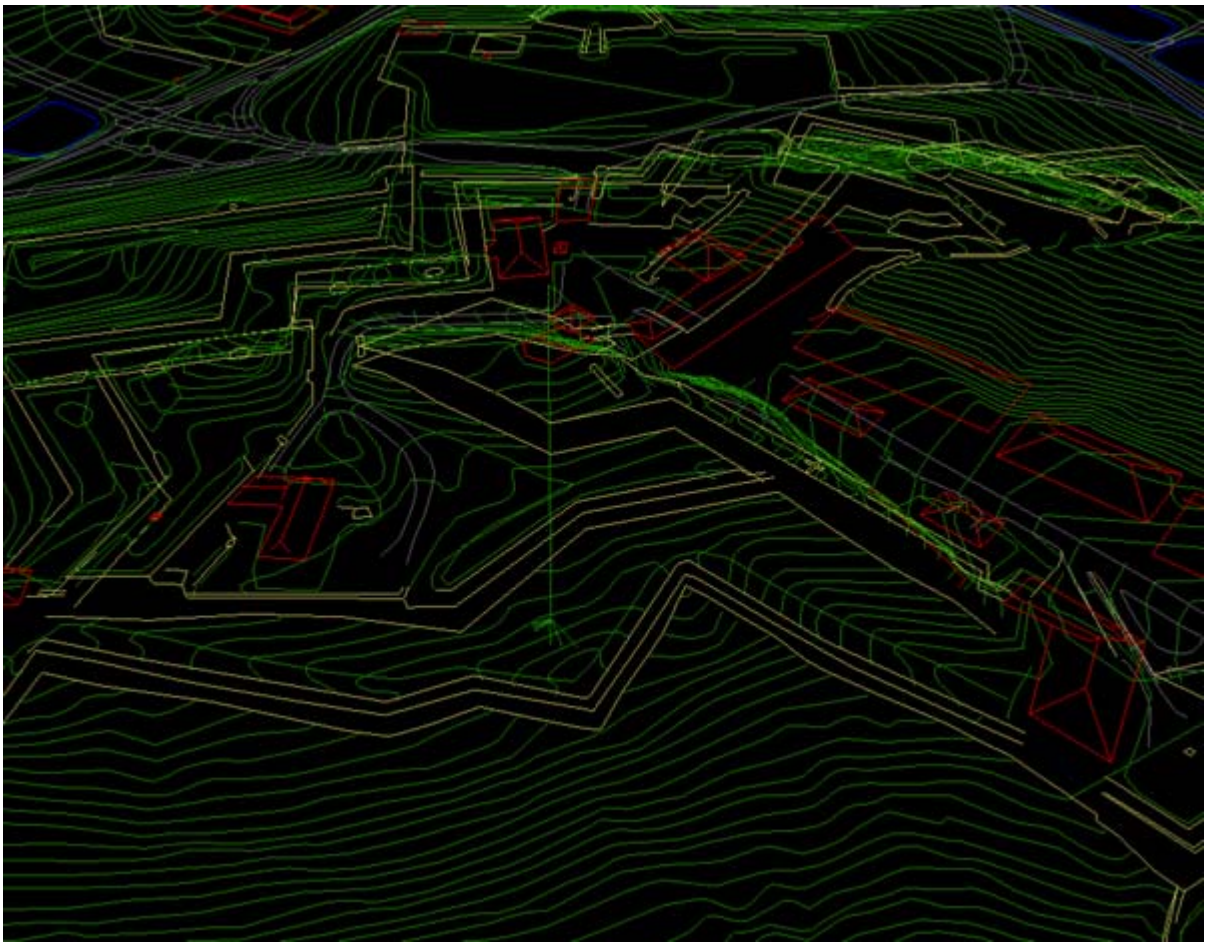
5.2. Script for generere trådmodell

Et av scriptene som ble lagd i dette prosjekt hadde til hensikt å lage en trådmodell. Det vil si en 3D modell som bare består av linjer og punkter og ingen polygoner. Siden SOSI data består av linjer og punkter, kan man relativt enkelt konvertere disse direkte til VRML. Det er ikke noe behov for approksimasjon, triangulering eller andre krevende algoritmer for konvertering. Hensikten med dette var at man dermed enkelt kunne få en 3D visualisering direkte av SOSI data, i motsetning til 2D visualisering, som man bare har mulighet til med SosiVis (Seksjon 3.1.1). Siden det på denne måten ikke heller er behov for å ta hensyn til egenskapene til de forskjellige datatypene, kunne jeg visualisere alle typene data jeg hadde fått tilgjengelig.



Figur 5.2. Trådmodell av Halden sentrum

Illustrasjonen over viser et bilde av en trådmodell over Halden sentrum (Figur 5.2. Trådmodell av Halden sentrum). Jeg har valgt å gi linjene farge etter hva slags type data de representerer; grønt for terrenglinjer, rødt for bygningslinjer, grått for veier, gult for anlegg, blått for vann og fiolett for jernbane. På denne måten får man visualisert hva dataene egentlig representerer. En interessant egenskap ved denne modellen er at den avslører ting som ikke er synlig i en 2D visualisering.



Figur 5.3. Trådmodell av Fredriksten Festning

På illustrasjon over (Figur 5.3. Trådmodell av Fredriksten Festning) kan man se en terrenglinje som skjærer på tvers av alle de andre. Grunnen til dette er at høydeverdiene for noen av punktene i linjen, er feil med 100 meter. Denne feilen er umulig å oppdage i en 2D visualisering, mens den her er opplagt. En slik visualisering kan på denne måten være nyttig, men selv om man er i stand til kjenne igjen stedet som blir visualisert, kan man neppe kalle dette en realistisk visualisering. Så for en visualisering, der målet etter beste evne er å gjenskape virkeligheten, kommer denne til kort.

5.3. Tilpassing av Javamesh

Det ble tidlig klart at for å kunne generere en god terrengmodell måtte man bruke triangulering, og at Delaunaytriangulering var den metoden som var best egnet til formålet. En implementasjon av Delaunaytriangulering fra bunnen av, ville være for krevende og ta mye fokuset bort fra det egentlige målet med denne oppgaven. Det eneste tilgjengelige implementasjon av Delaunaytriangulering med beskrankninger var Javamesh (Seksjon 3.4). Dette programmet var en demoapplikasjon for å demonstrere prinsippene ved metoden og ikke et bibliotek. Normalt ville et programbibliotek med et ferdig og klart grensesnitt, være å foretrekke, men i dette tilfellet var dette det eneste alternativet. Derfor ble det mye jobb med å analysere og tilpasse Javamesh til mitt formål.

Det første jeg måtte gjøre var å analysere de ulike klassene og finne ut hvilken oppgave i programmet de hadde, og om de var nødvendige for selve trianguleringsalgoritmen. Generelt bestod programmet av klasser for datastruktur, grafisk brukergrensesnitt, visualisering og

klassene som inneholdt selve trianguleringsalgoritmen. For meg var kun algoritmen og datastrukturen interessant. Den første jobben var derfor å ta vekk alle klasser som hadde med brukergrensesnitt og visualisering å gjøre, slik at man bare hadde igjen klassene som utførte beregningene og klassene for datastrukturen. Det viste seg at disse klassene ikke kunne fungere uavhengig av grensesnittet. Datastrukturen og brukergrensesnittet var ikke klart nok atskilt og noen steder temmelig sammenblandet. Det krevde at en ganske detaljert gjennomgang av programmet for å ordne dette. Det neste problemet var mer forståelig. Javamesh var ikke beregnet på 3D data og hadde derfor ikke implementert dette. Dette krevde noen endringer i datastrukturen og noen andre steder i programmet som befattet seg med kalkulering av selve punktene.

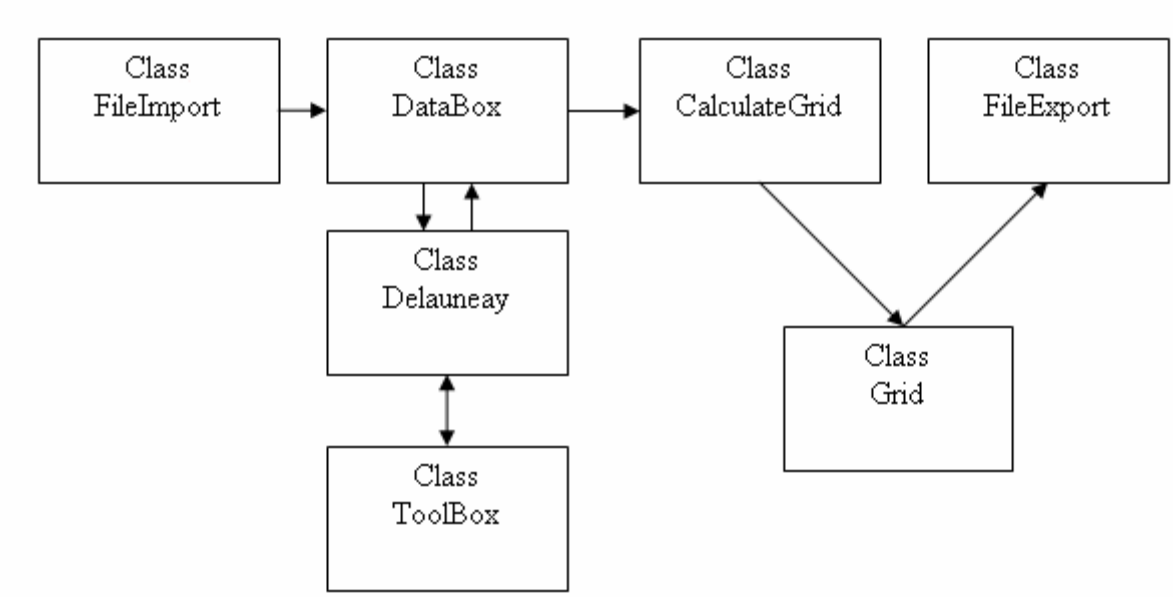
5.4. Implementasjon av TerrainMesh

For å kunne teste om trianguleringsklassen fungerte måtte man finne en ny metode for importere og eksportere data. Det naturlig var derfor å implementere to nye klasser for import og eksport av data. For import av data hadde man følgende muligheter. Man kunne importere data direkte fra SOSI, man kunne konvertere disse slik at de oppfylte kravene velformethet i XML. Eller man kunne konvertere SOSI dataene til GML før man importerte dem. Siden den minst kompliserte løsningen var å implementere en importklasse for å importere SOSI data, ble denne valgt.

For eksport av data hadde man også flere alternativer. Ren tekst, SVG, VRML. Ren tekst er den enkleste formatet å eksportere til, men det kan være vanskelig å få et inntrykk av om dataene er generert. For å kunne enkelt verifisere at trianglene ser ut til å være korrekt generert, er SVG et nyttig format. I SVG får man et tydelig oversikt over trianglene og kan raskt oppdage om noe er galt. En annen fordel er at SVG er i henhold XML til standarden. Ulempen med SVG er at den ikke er tredimensjonal. Dermed er det vanskelig å se om høydedata er korrekte. For å kunne visualisere trianglene i 3D er VRML det beste alternativet som eksportformat.

5.4.1. Oppbygging av TerrainMesh

Selv om trianglene blir overført til VRML, ble de visualisert som en enkelt IndexedFaceSet. På den ene siden er dette den mest effektive måten å vise en komplett visualisering, med fullt detaljnivå. Ulempen er at denne formen å lagre data på, gjør det vanskelig å bearbeide dem videre. Spesielt i forbindelse med generering av flere detaljnivå som er nødvendig for at visualiseringen skal gå optimalt. For dette formålet er det mer hensiktmessig å lage en elevationGrid. For å kunne lage et elevationGrid må man foreta en approksimasjon i forhold trianglene. Slik sett var det hensiktsmessig med en ny klasse som utførte denne operasjonen. Under vises hvordan de forskjellige klassene er avhengig av hverandre i TerrainMesh (Figur 5.4. Skjema over klassene i TerrainMesh.)



Figur 5.4. Skjema over klassene i TerrainMesh

5.4.2. Import av SOSI data

En generisk importrutine for alle mulige typer data på SOSI format ville bli en svært komplisert oppgave. Men oppgaven her var kun å importere terrengdata på SOSI format. Og dette er en mer overkommelig oppgave. Dataobjekter som inneholder terrengdata er av typen, KURVE, LINJE eller PUNKT. De forskjellige temaene som er brukt er:

- HøydeKurve, Bygg (LTEMA 2001)
- ForsenkningKurve (LTEMA 2003)
- TerrengLinje, Bygg(LTEMA 2206)
- TerrengPunkt (PTEMA 2001)
- ToppPunkt (PTEMA 2002)
- ForsenkningPunkt (PTEMA 2003)

Disse dataobjekter og tema var dermed de man måtte ta hensyn til når man skulle importere terrengdata fra SOSI og inn i TerrainMesh. For disse objektene hadde man noen valg å ta. For eksempel skal tas med som en del av terrenget, eller skal de brukes som beskrankninger, der det innenfor flatene disse defineres ikke skal foretas triangulering? Skal høydekurve og forsenkningskurve behandles likt? Hvordan skal punktobjektene håndteres?

De fleste høydekurvene i modellen har ikke åpne ender, men har knutepunkter til andre objekter.

5.4.3. Internt lagringsformat i TerrainMesh

Javamesh hadde opprinnelig tre klasser, som definerte objektene i Javamesh. Disse var Mypoint, Edge og Element. Disse er beholdt. Mypoint objektet som definerte et punkt i planet, ble modifisert til å definere et punkt i planet. Edge definer en linje i et triangel. Objektet inneholder referanse til to punkter, og de to elementene på hver side av linje.

Element objektet definer et triangel.. Hvert element objekt inneholder et punkt til linjene (Edge) og punktene (MyPoint), som definerer objektet.

Javamesh har en egen klasse for å lagre alle verdiene, DataBox. Datastruktur som generelt er benyttet i denne klassen for å lagre verdier er enkle array. Det kunne kanskje vært ønskelig med et annet lagringsformat, men siden det å innføre en ny datastruktur hadde krevd for mye endringer på den originale koden for triangulering, valgte man å beholde den originale datastrukturen.

For å lagre rutenettet benytter Grid klassen seg av hashmap. Denne datastrukturen hadde muligens vært mer effektiv også andre steder i programmet, men dette ville som sagt krevd for mange endringer.

5.4.4. Approksimasjon av rutenett

Klassen Grid er klassen som tar seg av approksimasjon av rutenettet. For hvert punkt går den igjennom alle trianglene for å finne hvilket triangel denne faller innenfor. Dette gjør den først ved å lage en boundingboks, og deretter regner den ut barysentriske koordinater, hvis punkt er på innsiden av boundingboksen. Deretter blir det foretatt en sjekk mot barysentriske koordinatene om punktet er på innsiden av triangelet, hvis så er tilfellet, blir høyden approksimert. Prosessen gjentas for hvert punkt i modellen.

Bruk av boundingboks gjør at prosessen går lettere, man slipper dermed å regne ut de barysentriske koordinatene ved hver iterasjon. Bortsett fra denne optimaliseringen er bruker algoritmen rå kraft.

5.4.5. Eksport av triangulering og rutenett

Eksportklassen eksporterer tre forskjellige formater, SVG, VRML IndexedFaceSet og VRML elevationGrid. De to første er ment som debugging. Mens den siste er det formatet vi er interessert i, for videre bruk.

SVG er nyttig for å se om trianglene har blitt korrekt generert. Dessuten er den på XML kompatibel. Problemet med SVG er at den bare kan vise 2D. Et annet problem er at plug-in fra Adobe, som brukes for å se på SVG filer, har dårlig minnehåndtering når den skal vise store filer med mange triangler og linjer.

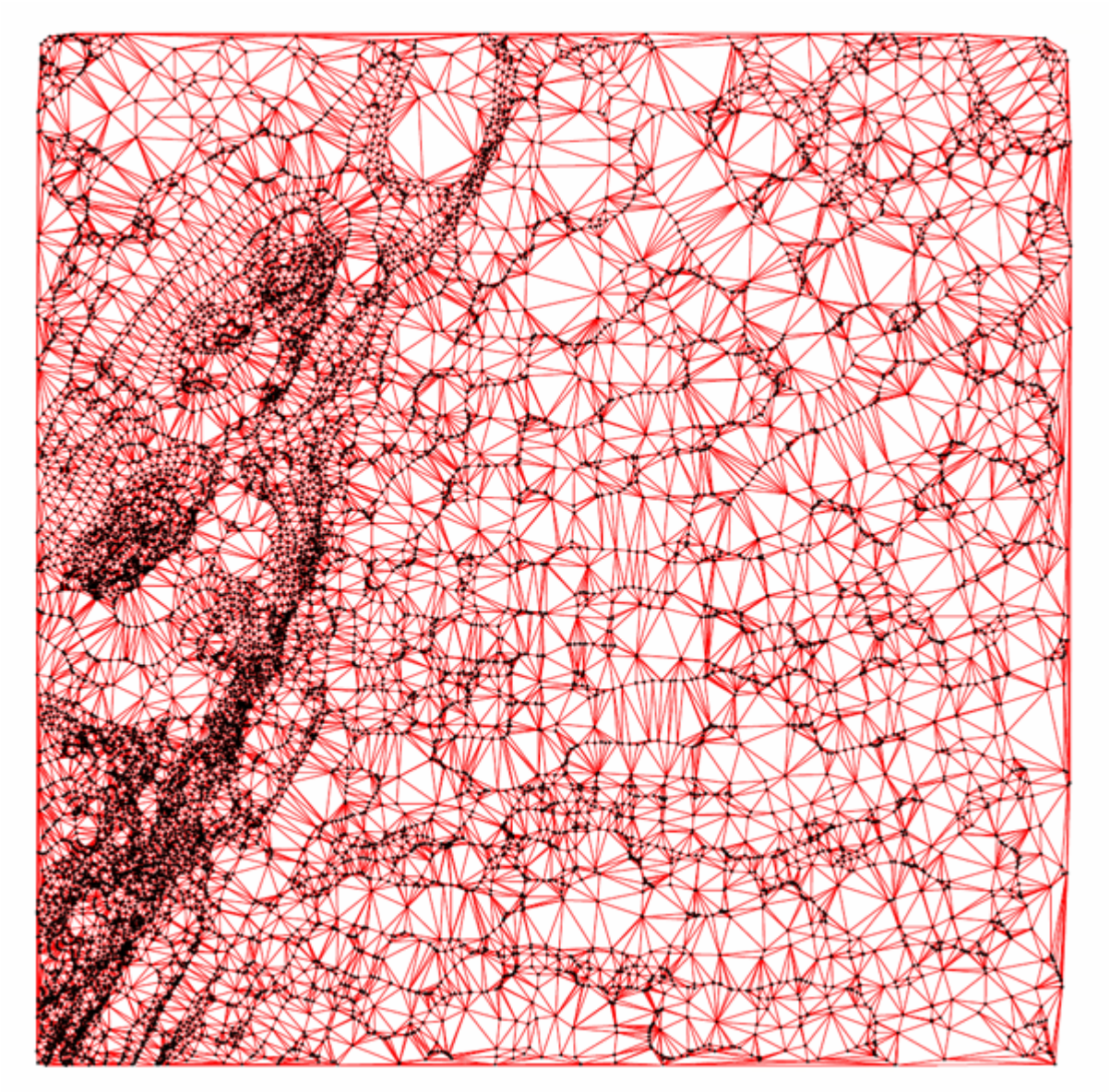
VRML indexedFaceSet er fin til å vise 3D visualisering av terrenget. Denne er god når man skal vise hvordan trianguleringen ser ut 3D. Eventuelle problemer med høydeverdiene som ikke vil synes i SVG, vil bli avslørt her.

VRML elevationGrid er sluttformatet som TerrainMesh eksporterer. Dette er det formatet vi er interessert for videre bruk. Problemet med dette formatet er hvis datamengden er stor, må den splittes opp i mindre håndterbare deler slik at den blir praktisk å navigere i en fremviser.

5.4.6. Resultat av terrenggenerering

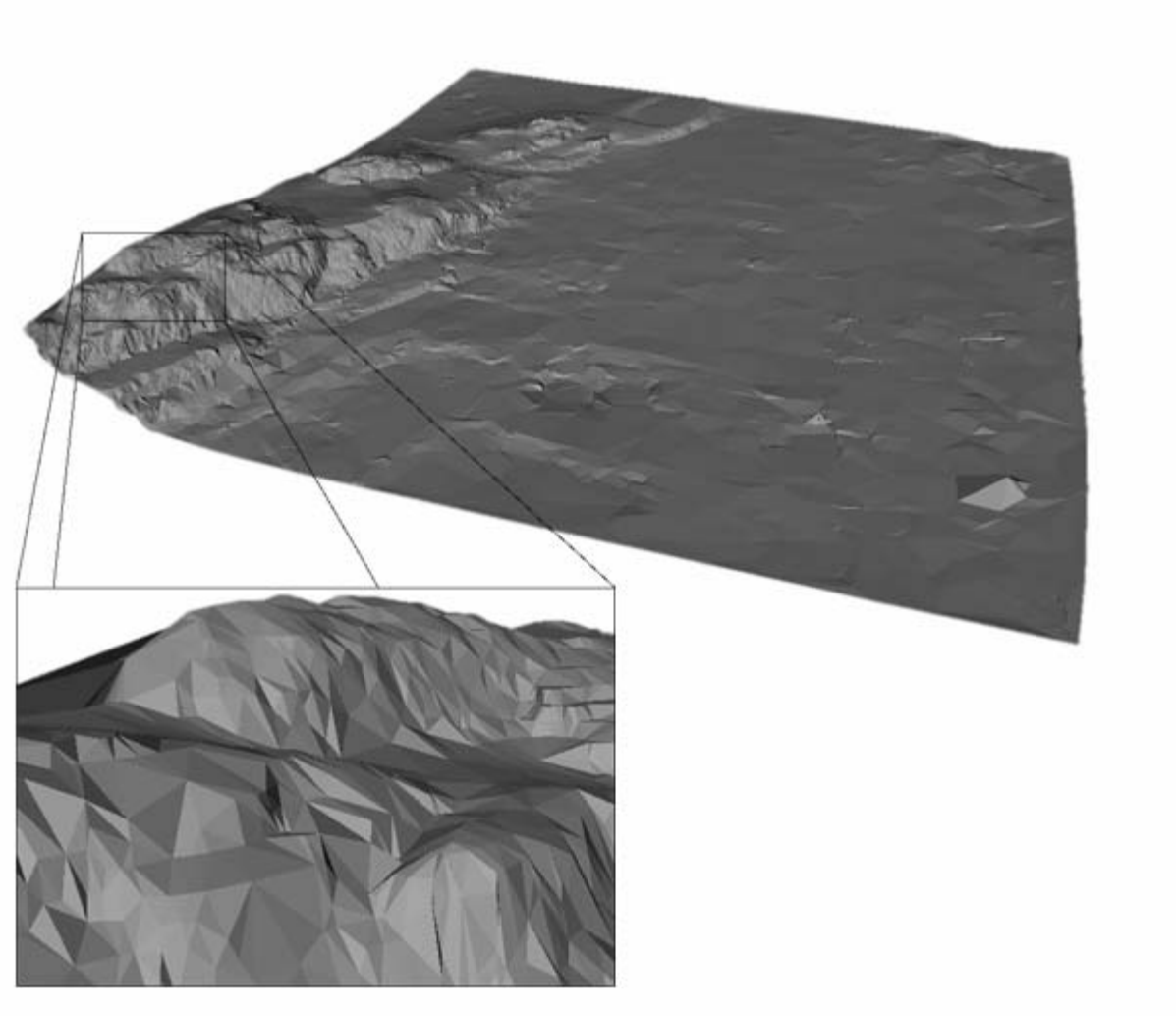
En vellykket gjennomkjøring av TerrainMesh skal resultere i en triangulering og et elevationGrid for det området som på forhånd er definert. Illustrajonen (Figur 5.5. Eksempel triangulering vist på SVG format) under viser resultatet av en triangulering på SVG format. Her ser man klart de forskjellige punktene og linjene som danner trianglene. Dette var et nyttig verktøy for å kunne sjekke at trianguleringen var riktig utført, eller identifisere eventuelle feil eller feilkilder i de tilfellene der noe gikk galt. Det som ikke blir vist er

høydeverdiene. Derimot kan man tydelig se store forskjeller i tettheten av trianglene alt etter hvor kupert terrenget er.



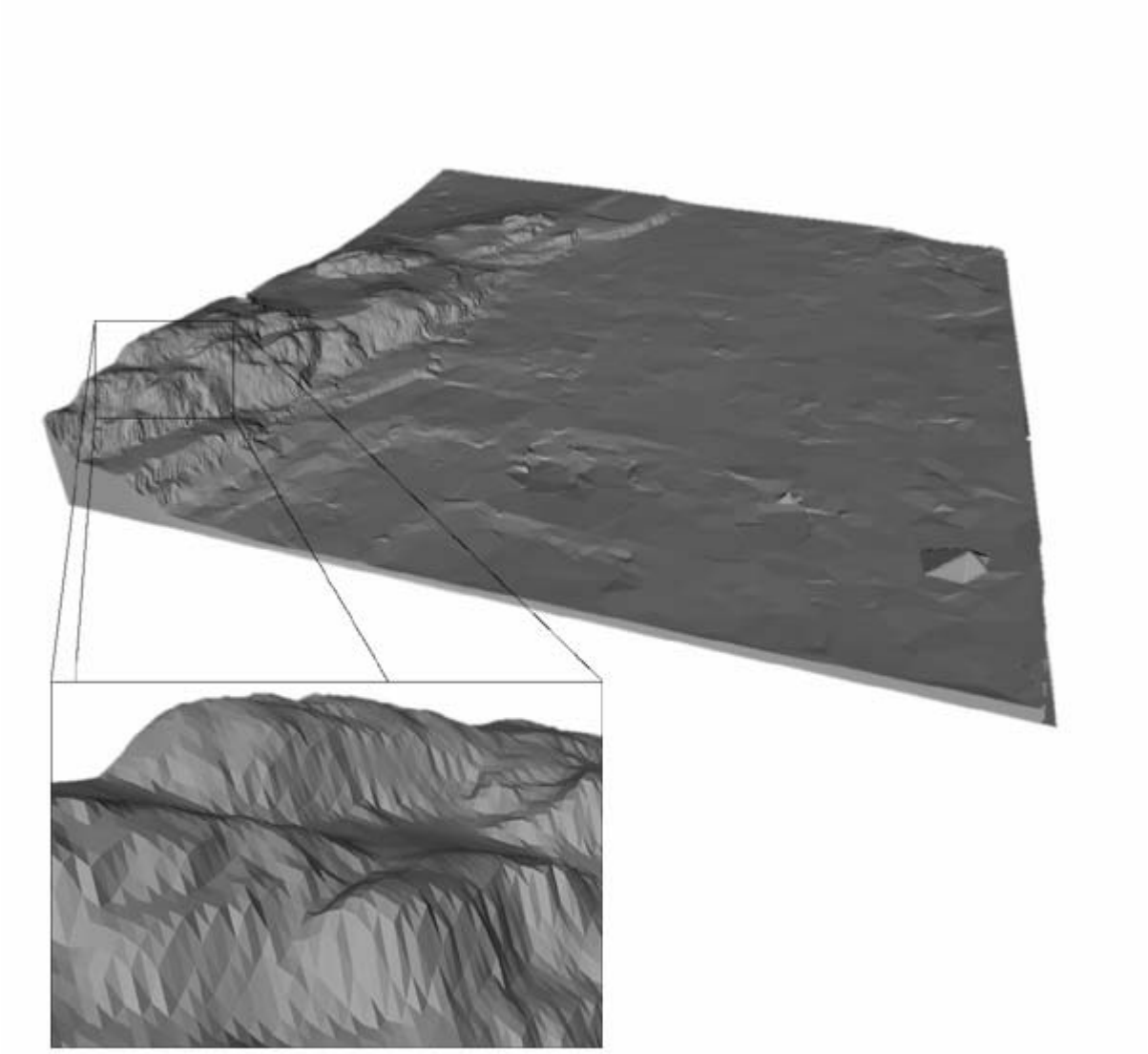
Figur 5.5. Eksempel triangulering vist på SVG format

Denne illustrasjonen viser samme (Figur 5.6. Eksempel triangulering vist på VRML format) triangulering bare på VRML format, med polygoner. Dette er i utgangspunktet en fullgod terrengvisualisering, men den egner seg ikke for optimalisering. Datamengden varierer alt etter hvor kupert terrenget er og blir derfor vanskelig å forutsi. Den egner seg heller ikke for oppsplitting i mindre kvadranter. For dette formålet er det bedre med et regulært rutenett.



Figur 5.6. Eksempel triangulering vist på VRML format

Det regulære rutenettet som er illustrert under (Figur 5.6. Eksempel triangulering vist på VRML format) er en approksimasjon basert på denne trianguleringen. For en slik approksimasjon er det viktig at oppløsningen på rutenettet er optimal. For store ruter i rutenettet vil føre til at detaljer i terrenget forsvinner, mens for små ruter vil føre til en unødvendig stor datamengde. Hvis man har en riktig oppløsning på rutenettet, skal man kunne se mindre forskjell på denne visualiseringen og visualiseringen av den originale trianguleringen. Med dette elevationGrid har man et godt utgangspunkt for optimalisering og teksturering.



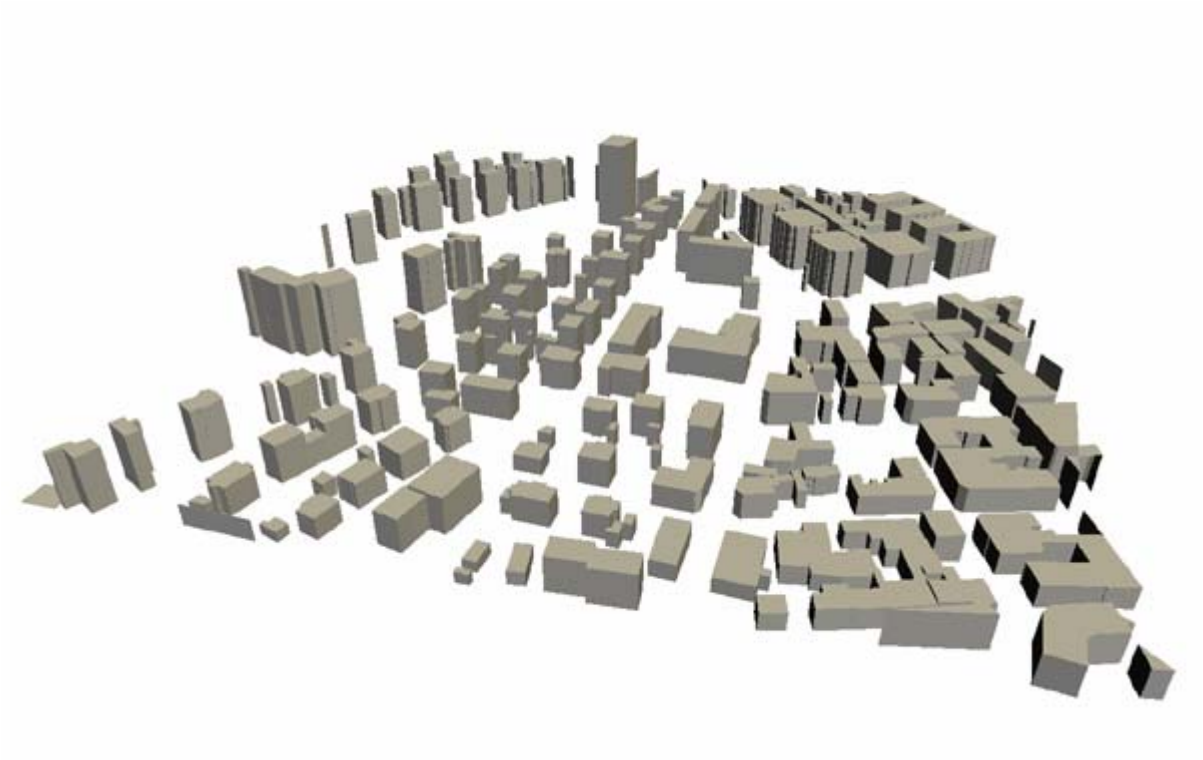
Figur 5.7. Eksempel på elevationGrid

Det var flere problemer med terrenggenerering. Problemet var at når man prøvde å generere triangulering for større områder, brøt programmet sammen. En kilde til et av problemene ble identifisert i datasettet. Noen av punktene hadde nøyaktig samme koordinater i planet, men hadde forskjellig høydeverdi. Trianguleringsalgoritmen tillater ikke dette, derfor måtte det legges inn en sjekk i importklassen slik at dette ikke inntraff. Et annet problem var selve størrelsen på datasettet. Den opprinnelige datastrukturen, var beregnet på en demoapplikasjon med begrenset mengde data. Den var derfor ikke optimalisert for større mengder data. Dette førte til at applikasjon bare klarte å prosessere deler av datasettet uten å få problemer med for høyt minneforbruk. Den største biten av datasettet programmet klarte å prosessere, var størstedelen av Halden Sentrum, en bit på 1200x1600 meter, med en oppløsning på 1 meter. Denne delen tok det 3 dager å prosessere på den maskinen jeg hadde tilgjengelig.

5.5. BuildingParser, script for å generere bygninger

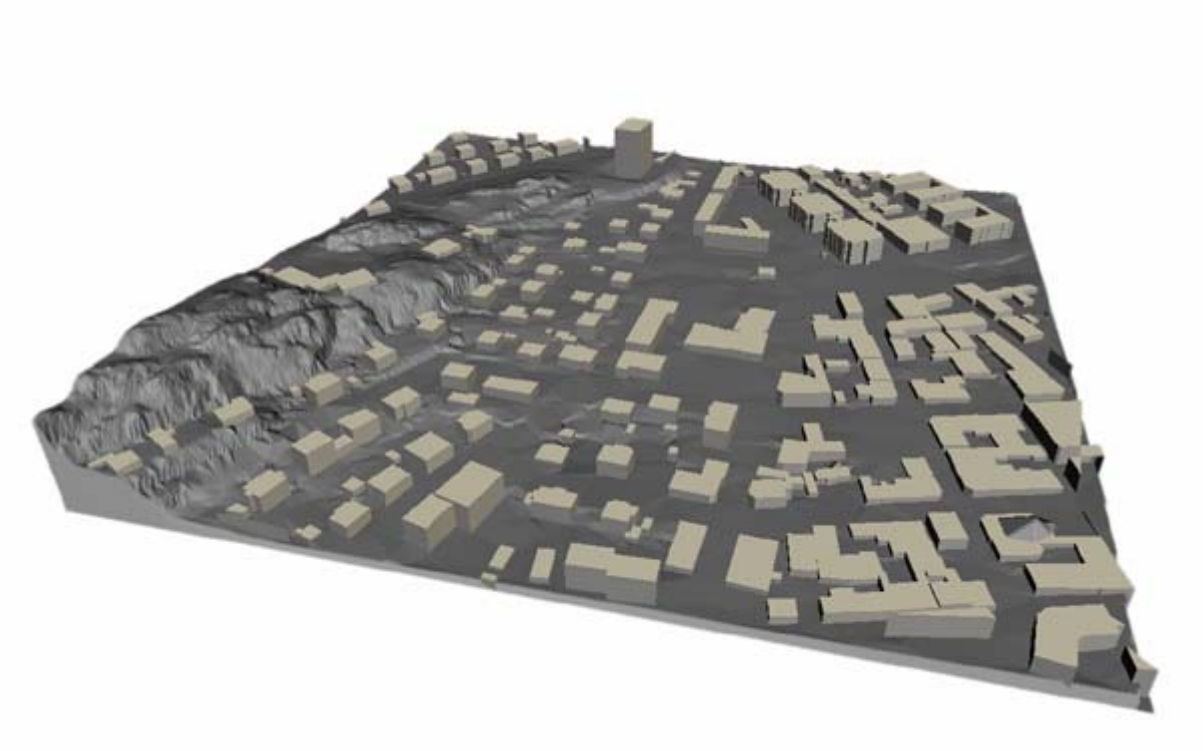
En enkel, men effektiv visualisering av bygninger krevde ikke noen større beregninger, i motsetning til terreng som krever en tung trianguleringsprosess. Ved å lage et script som genererer en extrusion for hver bygning, fikk jeg et resultat som fungerte.

Måten scriptet fungerer på er at det først parser SOSI datasettet for bygninger. Hver bygning i SOSI datasettet har et antall linjer som definerer flaten for denne. Denne flaten blir brukt som utgangspunkt for å generere en extrusion. Høyden på denne extrusion blir da satt fra takkant til null, altså havnivå. Høyden til takkanten er en beregnet verdi. Denne beregnes ved å ta gjennomsnittet av alle punktene for bygningen. Muligens kan medianverdien være bedre, men uansett vil det være en tilnærmet verdi. Jeg valgte å sette en LOD node på hver enkelt bygning. Denne har bare et nivå og fungerer ganske enkelt slik at bygninger som er så langt unna at de ikke synes, ikke blir vist. Dette hjelper til å øke ytelsen på modellen.



Figur 5.8. Bygninger generert av scriptet

Illustrasjonen (Figur 5.8. Bygninger generert av scriptet) over viser resultatet av en gjennomkjøring av scriptet for å generere bygninger. Høyden til bygningene er satt fra havoverflaten og opp til takkanten på de respektive bygningene. Bygninger som ligger høyt i terrenget vil derfor bli unormalt høye.



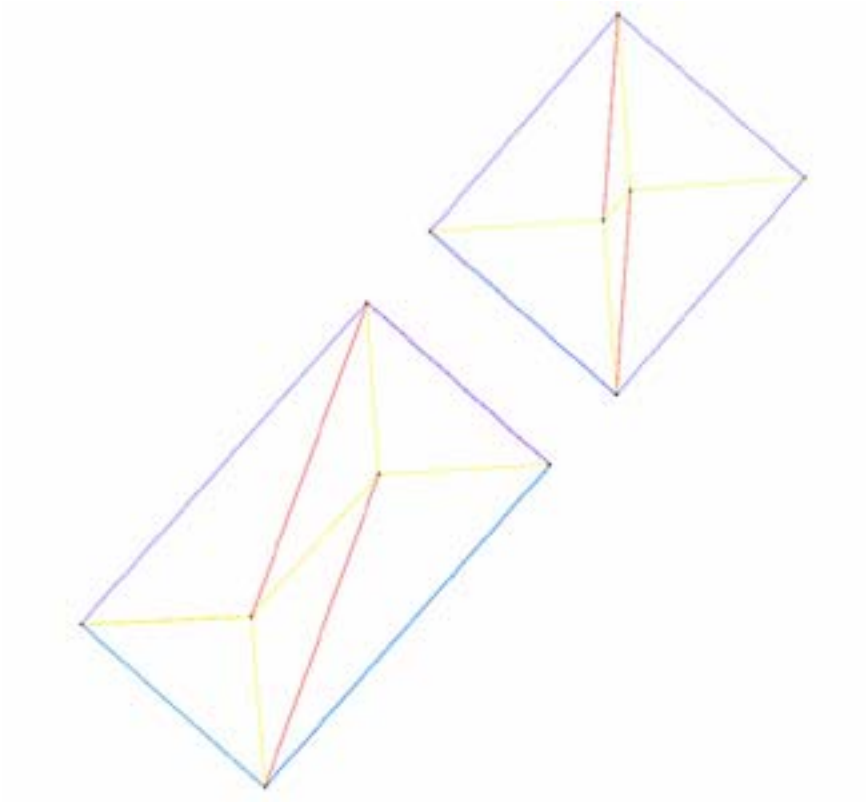
Figur 5.9. Bygninger kombinert med terreng

Illustrasjonen (Figur 5.9. Bygninger kombinert med terreng) over viser resultatet, når man kombinerer bygningene med terrenget for det samme området. Bygninger som ligger høyt, får nå sin korrekte høyde. Z-buffering sørger for at den nedre delen av bygningen ikke vises.

5.6. Implementasjon av BuildingMesh

I et forsøk på å modellere takkonstruksjoner lagde jeg en modifisert utgave av TerrainMesh. Jeg valgte å kalle denne BuildingMesh. Algoritmen til Javamesh støttet muligheten til å definere linjer, som begrenset hvor delaunaytrianguleringen skal foregå. I tillegg kan man definere linjer på forhånd som skal ligge fast, uansett om andre trianguleringer er mer optimale. Jeg benyttet ikke dette i TerrainMesh, dels fordi det ville gjøre kalkulasjonene enda tyngre, og dels fordi det ville krevd en mer komplisert importrutine. Det eliminerte også en feilkilde.

Ideen med å lage en variant av denne var at man kunne utnytte det faktum at Javamesh originalt kan triangulere med linjer som begrenser områder hvor det skal trianguleres. Siden man har et flateobjekt for hver bygning, kan man bruke denne til å avgrense områder som skal trianguleres. Dermed får hver bygning sitt sett med triangler som visualiserer takkonstruksjonen. Figuren under (Figur 5.10. To hus med valmet tak) illustrerer hvordan dette fungerer. I eksempelet har man to hus med valmet tak. De blå linjene indikerer flaten til huset. Disse avgrenser området hvor det skal trianguleres. Det er viktig at disse linjene til sammen danner en sluttet flate. De gule linjene er forhåndsdefinerte linjer fra datasettet og er de som definerer takkonstruksjonen. I trianguleringsprosessen kan ikke disse linjene fjernes eller krysses. De røde linjene er dannet i trianguleringsprosessen.



Figur 5.10. To hus med valmet tak

Denne ideen har i utgangspunktet en svakhet. Som tidligere nevnt markerer visse linjer toppen av loddrette flater. I disse tilfellene vil ikke visualiseringen bli helt korrekt. Likevel vil dette gi en bedre visualisering enn et flatt tak, med unntak av de tilfellene selvfølgelig, der taket faktisk er flatt.

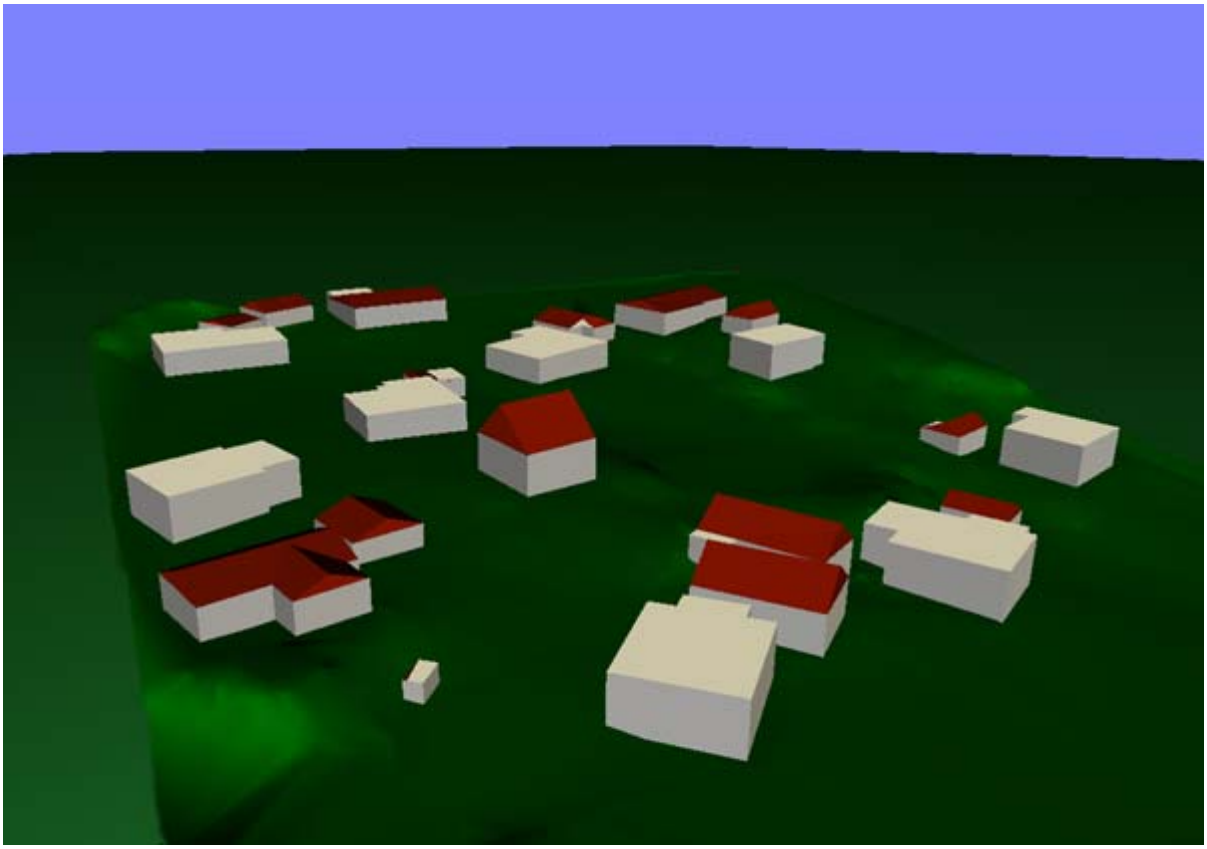
5.6.1. Modifikasjoner gjort i Buildingmesh

Det første jeg gjorde var å skrive om importklassen til å importere bygningsdata fra SOSI datasettet. Denne måtte også kunne lese linjene og flatene i tillegg til punktene. Linjer som definerte flaten på en bygning ble markert som grense, som det ikke trianguleres utenfor. Jeg valgte å kutte ut siste steg i trianguleringsprosessen. Dette var nødvendig for å få trianguleringen til å virke.

Klassen som genererte det regulære rutenettet ble tatt vekk, slik at programmet bare eksporterte trianguleringen, henholdsvis på SVG og VRML format. Disse modifikasjonene var tilstrekkelig til å test om dette konseptet fungerte.

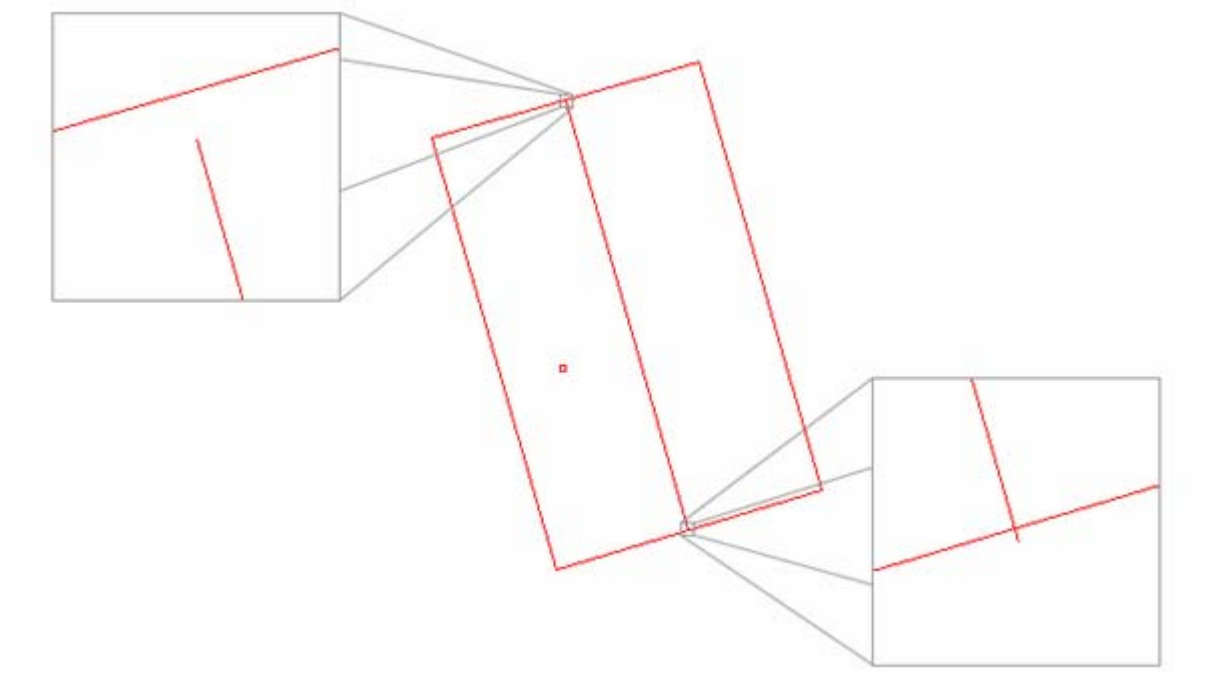
5.6.2. Resultatet av forsøk på å modellere tak

For å teste metoden ble det tatt utgangspunkt i en liten del av datasettet. Som utgangspunkt ble det valgt et lite boligfelt med et utvalg forskjellige hus og takkonstruksjoner; deriblant rekkehus og hus med og uten takutspring. Figuren under (Figur 5.11. Visualisering av et boligfelt med hustak.) illustrerer resultatet. Noen av husene fikk helt korrekt visualisering av taket, mens andre fikk en delvis korrekt visualisering. På andre hus igjen, ble ikke taket visualisert i det hele tatt.



Figur 5.11. Visualisering av et boligfelt med hustak.

Grunnene til at visse hus ble visualisert feil eller ikke visualisert i det hele tatt, kan spores tilbake til SOSI datasettet. Den mest graverende feilen med datasettet, og dette må karakteriseres som en feil, er at endepunkter på linjer ikke er knyttet sammen når de burde være det. Et annet problem er at visse endepunkter ender midt på en annen linje, uten at denne har et tilsvarende knutepunkt.



Figur 5.12. Hus med skråkant der mønekanten krysser takkanten.

Et typisk eksempel er bolighus med skråtak som vist i figuren over. (Figur 5.12. Hus med skråkant der mønekanten krysser takkanten.) I datasettet er disse definert med to linjeobjekter. Det ene definerer takkanten rundt hele huset med fire punkter som definerer plasseringen av hvert av husets fire hjørner. Det andre objektet definerer mønekanten og består av to punkter. I eksempelet over kan man se at mønekanten på et sted krysser takkanten, mens den på den andre siden slutter før linjen. Begge deler er feil, men det er den første som er fatal. Hvis man skulle forsøke å triangulere dette huset, ville de to linjestykkene som krysser hverandre bli eliminert. Siden det en linjestykket er avgrensning for området som skal trianguleres, vil flaten ikke bli fullstendig omsluttet. Dette igjen fører til at dette huset ikke blir triangulert og at det ikke blir noen visualisering av taket for dette huset. I det andre tilfellet der mønekanten slutter innenfor takkanten, vil trianguleringen fungere, selv om det ikke er korrekt. Her vil man få generert et ekstra triangel mellom takkanten og endepunktet på mønelinjen.

I et korrekt datasett burde man aldri tillate at to linjestykker krysser hverandre. Linjene burde i så fall være delt med et felles knutepunkt. For å ta tilfellet med eksempelet over burde linjeobjektet som definerer takkanten hatt to ekstra punkter, og disse burde være knutepunkter mellom mønekanten og takkanten.

5.7. Tekstur på terreng og bygninger

På illustrasjonen over ser man Fredriksten Festning. Her er terrenget teksturert, mens bygningene er extrusion noder, lagt oppå terrenget. Med tekstur på terrenget blir mangelen på tekstur på bygningene framhevet. Det neste steget kan enten være å legge på flere detaljer på bygningen eller legger på tekstur. Det siste er det mest nærliggende (Figur 5.13. Festningen med bygninger og taktekstur.)

Figur 5.13. Festningen med bygninger og taktekstur

Ved å definere at en Extrusion ikke skal ha en toppflate og i stedet sette inn et IndexedFaceSet Node for å definere taket, kan man legge tekstur på denne flaten. Man klipper da ut en liten bit av teksturen for terrenget, tilsvarende utsnittet av huset. På denne får man lagt tekstur på taket. Metoden er enkel, men effektiv, på den måten kan man ved å bruke en enkelt tekstur, legge tak på samtlige bygninger i modellen. Metoden er riktignok ikke uten svakheter. Det er svært begrenset hvor detaljert teksturen på hvert tak kan være, siden man bruker en svært liten bit av bildet. For å få god detaljoppløsning på teksturen må bildet være svært stort, og dette er uhensiktsmessig. Dessuten er det grenser for hvor detaljerte ortofoto man har tilgang til som kilde til tekstur. Det at man bruker ortofoto som taktekstur, er ikke uten svakheter. Bortsett fra at detaljoppløsningen er for lav, har man også et problem med at fotoet er tatt fra stor høyde og ikke nødvendigvis er tatt loddrett i forhold til taket. Dermed blir teksturen forskjøvet i forhold til taket.

Et annet problem er at det kan være gjenstander som hindrer sikten på bildet. For eksempel kan store trær, som står tett inntil bygningen, skygge for sikten, slik at treet blir projisert ned i taket på den ferdige modellen. På den annen side er det ikke mange gode alternativer for skaffe egnet tekstur. Man kan manuelt rette opp feil som oppstår på grunn av perspektivet. Men ellers har man ikke så mange andre alternativer enn å ta spesifikke bilder av hver bygning, noe som utvilsomt vil være en svært tidkrevende jobb og komplisert for taktekstur.

Figur 5.14. Bygninger med tekstur på veggene.

For loddrette husvegger kan man ikke bruke ortofoto. Her er eneste mulighet å ta bilder for hver eneste bygning og manuelt tilpasse dem bygningen. Bildet over (Figur 5.14. Bygninger med tekstur på veggene.) viser et eksempel på nettopp dette. Teksturene på veggene her, er hentet fra en tidligere modell av Fredriksten Festning (IFE98). Dette er som sagt tidkrevende og vil i de fleste tilfeller være uaktuelt. Man kan derimot tenke seg at visse sentrale bygninger

i modellen tilpasses på denne måten. Men uansett vil det være en manuell jobb og markerer vel på den måten grensen for hva man kan visualisere med geografiske data og ortofoto alene.

5.8. Bruk av Rez

For å øke ytelsen til et brukbart nivå for de større visualiseringene, var man helt avhengig av å bruke LOD. Dette gjaldt spesielt for terreng. Det regulære rutenettet som terrainmesh produserer, har en nøyaktighet på 1 meter. Dette er et detaljnivå som kun er nødvendig når man befinner seg på helt nært hold. Et annet problem er at rutenettet blir for stort. For Halden Sentrum er det originale rutenettet for terrenget 1200 x 1600. Dette var stort nok til at den ikke lot seg laste inn i VRML-fremviseren.

I stedet for å implementere detaljhåndtering selv, valgte jeg å bruke Rez for å optimalisere terrengmodellen. Rez importerer regulære rutenett, på et utvalg forskjellige formater, deriblant VRML ElevationGrid. Resultatet kunne enten eksporteres som et hierarki av ElevationGrid eller GeoElevationGrid noder. Siden jeg har valgt å bare bruke ren VRML, valgte jeg å bruke elevationGrid.

Rez lar deg sette flere parametere, det viktigste av disse er antall nivåer og størrelsen på hver kvadrant. I tillegg kan man sette skalering av høyde. Et problem med Rez var at den opererte kun med heltall. Dette førte til at høydeverdiene ikke kan ha bedre nøyaktighet enn 1 meter. Det originale datasettet har en nøyaktighet på 1 cm, derfor var ikke dette tilfredsstillende. Løsningen ble å skalere alt opp 100 ganger, og dette gjorde nytten. Riktignok har jeg en mistanke om at dette er en av grunnene til problemene med avrundingsfeil i z-buffering i VRML-fremviseren, men har ikke hatt mulighet til etterprøve dette.

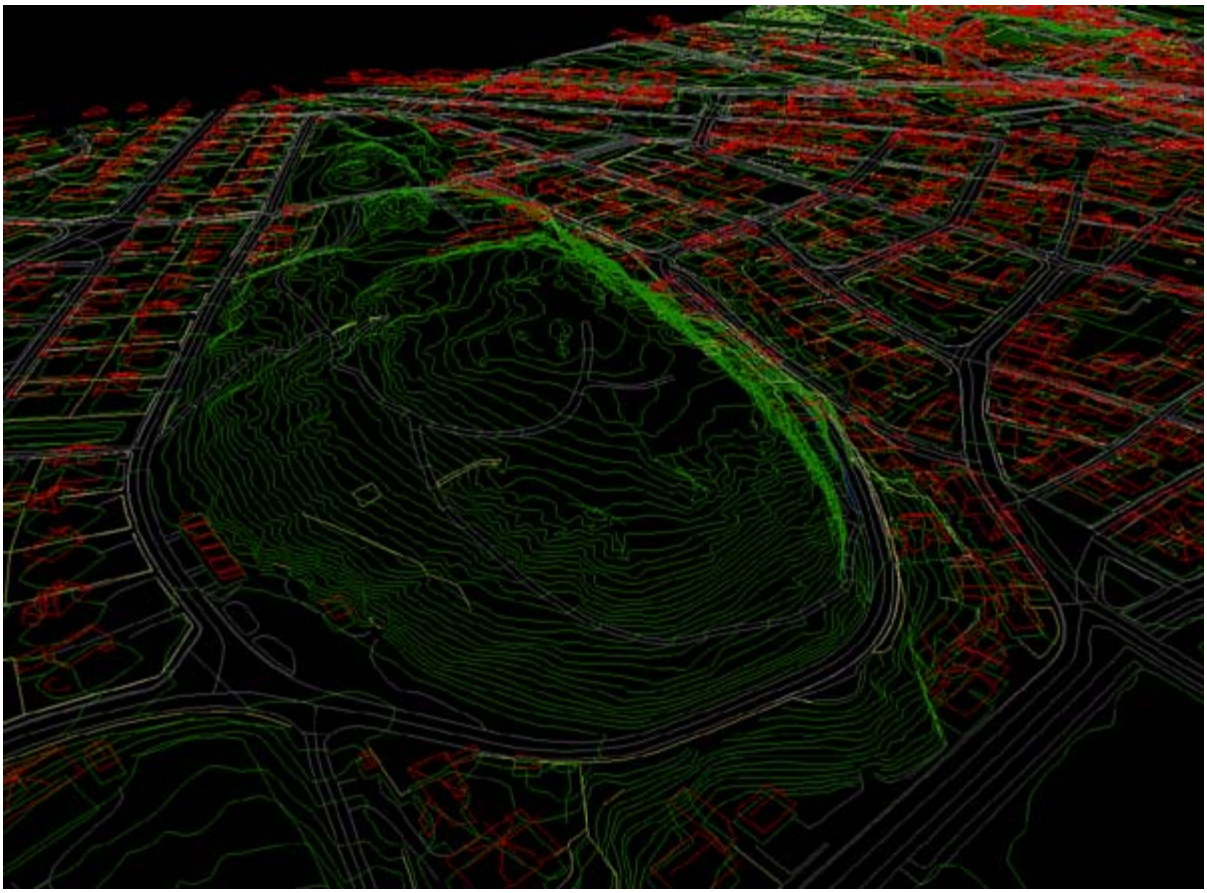
Kapittel 6. Resultater og evaluering

6.1. Resultater

I dette kapittelet vil jeg vise noen eksempler på resultatene som ble oppnådd i dette prosjektet. Alle illustrasjonene i dette kapittelet er tatt direkte fra modellene som ble lagd i dette prosjektet.

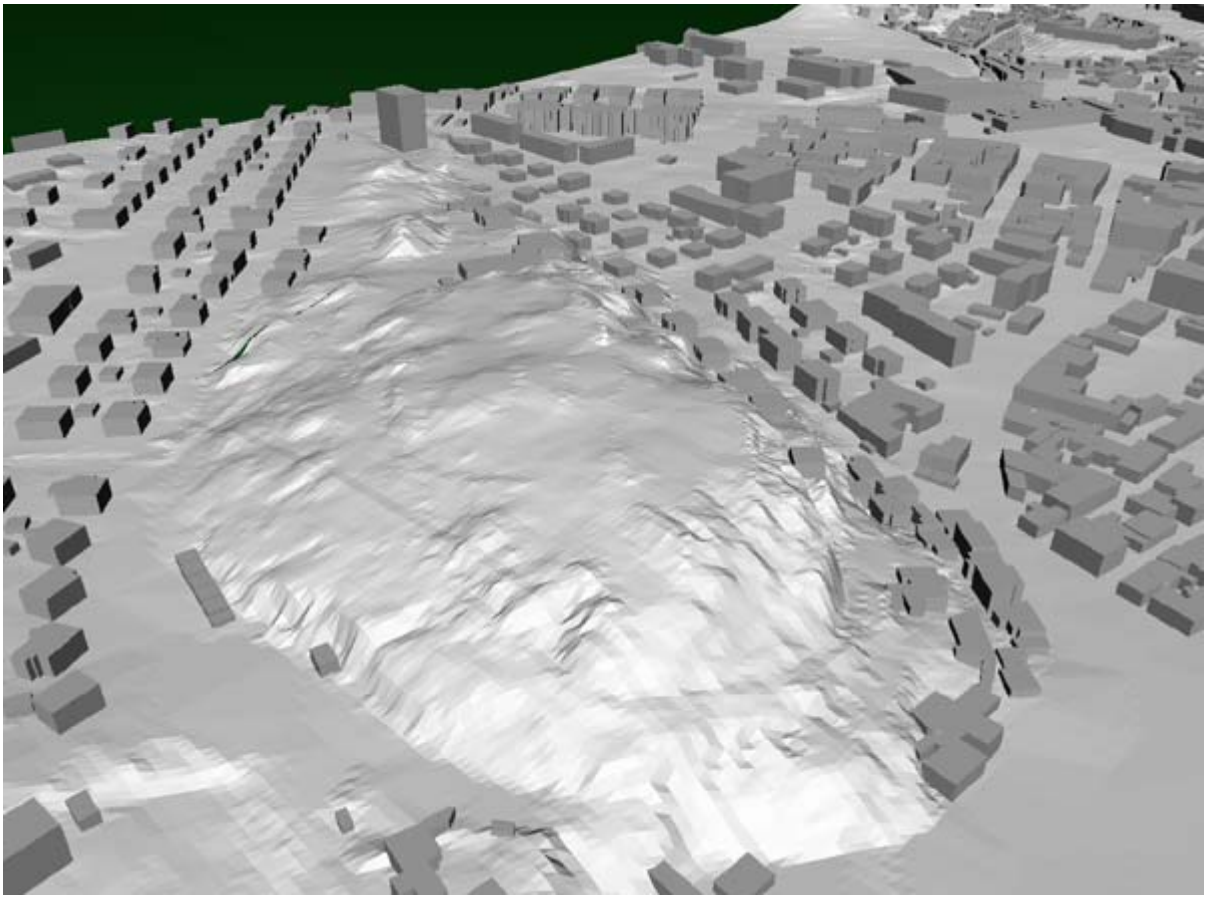
6.1.1. Oppbygging av modellen

I dette prosjektet ble det prøvd ut flere forskjellige former for visualisering og hvilken effekt disse hadde. Jeg vil her presentere de forskjellige formene for visualisering som ble gjennomført i prosjektet.



Figur 6.1. Linjemodell av Rødsfjellet sett sydfra

Illustrasjonen over (Figur 6.1. Linjemodell av Rødsfjellet sett sydfra) er en linjemodell over Rødsfjellet. Denne viser tydelig strukturen til det originale datasettet som modellen er basert på. Terrenglinjene gir et godt inntrykk av hvordan terrenget ser ut, mens bygningene er vanskeligere å få noe godt inntrykk av.



Figur 6.2. Modell av Rødsfjellet sett sydfra uten tekstur

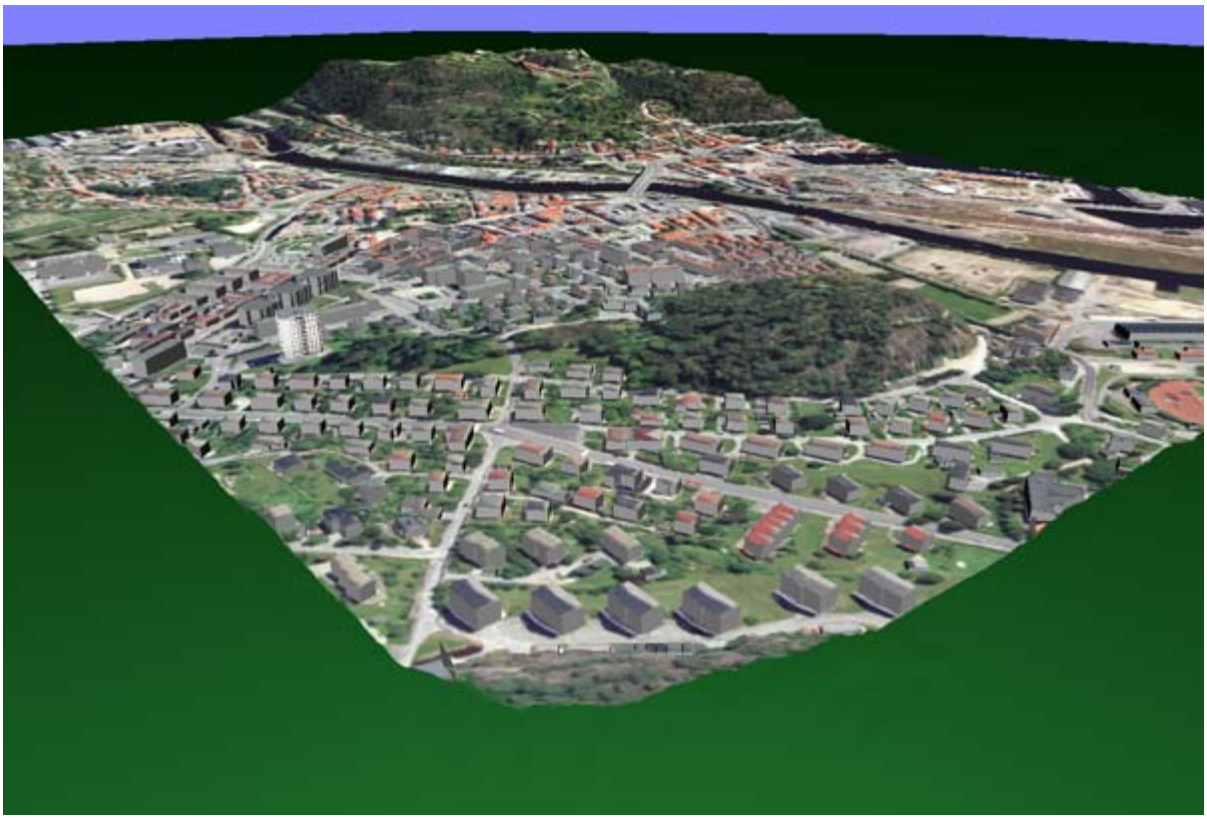
Går man et steg videre i prosessen og gjør linjebaserte data om til polygoner, får man dette resultatet (Figur 6.2. Modell av Rødsfjellet sett sydfra uten tekstur). Nå ser man tydeligere bygningenes utstrekning og får et bedre inntrykk av størrelsen. Man kan også lettere kjenne seg igjen. Likevel får man ingen følelse av realisme.

Figur 6.3. Modell av Rødsfjellet sett sydfra med tekstur

Det siste steget er å legge på teksturer (Figur 6.3. Modell av Rødsfjellet sett sydfra med tekstur). Det er først på dette stadiet at man kan begynne å anse dette for en virkelig realistisk visualisering.

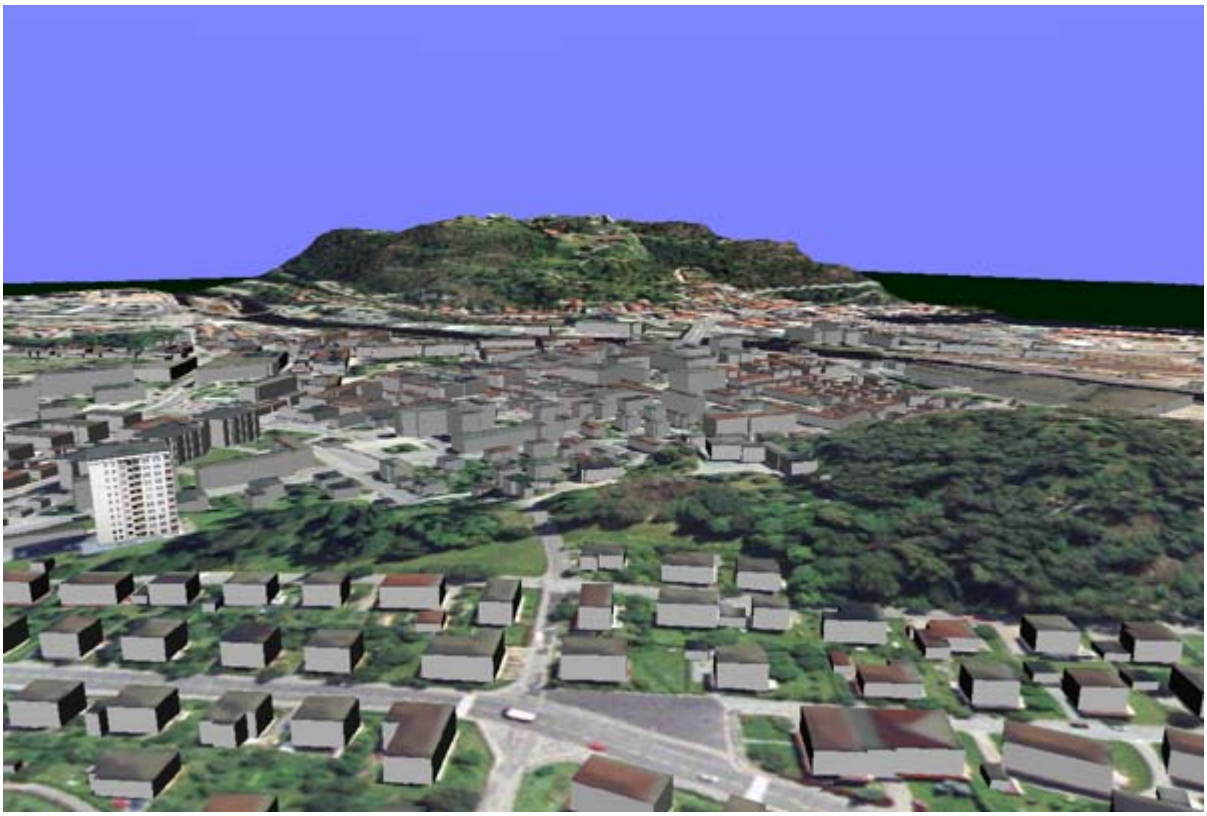
6.1.2. Eksempel på Detaljhåndtering

På grunn av store datamengder vil det alltid være behov for optimalisering. Illustrasjonene under viser et eksempel på en overflygning over Halden sentrum fra nordvest mot festningen, og med disse vil jeg forsøke å illustrere hvordan detaljhåndtering fungerer.



Figur 6.4. Halden Sentrum fra nordvest på avstand

Denne illustrasjonen (Figur 6.4. Halden Sentrum fra nordvest på avstand) viser Halden Sentrum på avstand, når man nærmer seg fra nordvest. Her kan man se de nærmeste bygningene, mens de nede i selve sentrum ikke blir vist. I det fjerne kan man se terrenget til festningen, som bare delvis blir vist.



Figur 6.5. Halden Sentrum fra nordvest mellomdistanse

Her hae man beveget seg nærmere (Figur 6.5. Halden Sentrum fra nordvest mellomdistanse), og alle bygningene i sentrum er nå synlige, men bygningene i festningen er ennå ikke synlige.

Figur 6.6. Halden Sentrum fra nordvest på nært hold

På det siste bildet (Figur 6.6. Halden Sentrum fra nordvest på nært hold) har man nærmet seg såpass mye at bygningene i festningen nå kan skimtes. Når man optimaliserer, må man gjøre et kompromiss mellom krav til ytelse og krav til detaljnivå. I en perfekt optimalisert modell skal det ikke synes at man skifter detaljnivå, men dette går utover ytelse. Derfor setter man ofte grensene mellom hvert detaljnivå slik at detaljer forsvinner før de er ute av syne, og skiftene blir derfor merkbare som i illustrasjonene over. Dette er som sagt for å bedre ytelsen.

6.1.3. Resultater med bruk av tekstur

Figur 6.7. Boligblokk med tekstur

Hensikten med denne illustrasjonen er å demonstrere effekten av tekstur (Figur 6.7. Boligblokk med tekstur) på bygninger. Alle bygningene i modellen har tekstur på taket, mens bare et par utvalgte bygninger har tekstur på veggene. Grunnen til dette er at tekstur til tak enkelt kan hentes fra flyfoto, mens tekstur på vegger må hentes inn enkeltvis. Men hvis man først har fått tekstur på vegger og tak, slik som boligblokken i forgrunnen, har man oppnådd et detaljnivå som skulle være tilstrekkelig for de fleste formål, i hvert fall i geografisk sammenheng. Illustrasjonen under viser andre bygninger med tekstur (Figur 6.8. Festningen med utsikt over Halden sentrum).

Figur 6.8. Festningen med utsikt over Halden sentrum

Når man ser modellen fra satelittperspektiv, får man kanskje den beste visuelle effekten av modellen. Illustrasjonene under viser et overblikk over festningen (Figur 6.9. Oversiktsbilde av festningen) og Halden sentrum (Figur 6.10. Oversiktsbilde over Halden Sentrum).

Figur 6.9. Oversiktsbilde av festningen

Figur 6.10. Oversiktsbilde over Halden Sentrum

6.2. Evaluering av effekt og ytelse av de forskjellige visualiseringene

Den visuelle effekten av modellene varierer svært mye alt etter hvilke data som er tatt med, og hvordan de blir visualisert.

Linjemodellen er en enkel form for visualisering. Den krever bare en enkel konvertering fra SOSI data til VRML modell. Ytelsen til en linjemodell er god, selv om man ikke benytter LOD, sammenlignet med en tilsvarende modell, med polygoner. Det gjør at den tåler en større tetthet av data enn andre, selv om det er en øvre grense også her. Når man ser denne modellen, ser man tydelig det originale datasettets oppbygning og eventuelle feil ved datasettet. Denne visualiseringen fungerer best på terreng og veier. For bygninger fungerer den heller dårlig. Det eneste som vises av bygningen er takkonstruksjonen, og det krever ofte litt fantasi for å kunne forestille seg hvordan denne ser ut i virkeligheten. Det som kanskje visualiseres best ved bygninger, er problemene med datasettet på dette området. Tatt i betraktning av hvor enkel denne visualiseringen er, så er den overraksende effektiv, men noen følelse av realisme får man ikke.

Den andre formen for visualisering er bruk av polygoner. Polygoner krever mer kraft, og modellen blir derfor en god del tyngre sammenlignet med en linjemodell. Derfor er det helt nødvendig at man for større modeller bruker LOD, for at modellen i det hele tatt skal være mulig å navigere. Det kreves også i tillegg mange utregninger å konvertere fra SOSI data til ferdig modell. Man må også i mye større grad ta hensyn til datatypene når man konverterer. Fordelen med denne formen for visualisering er at man får et mye bedre inntrykk av masse, spesielt gjelder dette for bygninger. Selv om man bare benytter enkle extrusion, uten takkonstruksjon, vil inntrykket av masse være korrekt. Man får også litt mer realisme i forhold til en linjemodell.

Det er først når man legger på teksturer, at man virkelig får en følelse av realisme i modellen. På de stedene der jeg også fikk lagt på teksturer på veggene, var det svært lett seg å kjenne igjen. Teksturer og lyssetting setter krav til grafikkortets ytelse, i motsetning rene polygoner, som krever mer rå datakraft. En av erfaringene jeg hadde med bruk av teksturer var problemer med Z-buffering. På grunn av avrundningsfeil, fikk jeg flimring i skjæringspunktet mellom bygning og terreng, spesielt på avstand. Likevel er tekstur helt essensielt for en realistisk visualisering.

Kapittel 7. Videre arbeid og konklusjon

7.1. Videre arbeid

7.1.1. Forbedre visualisering av bygninger.

Et av stedene der det virkelig er rom for forbedring, er visualisering av bygninger. Både fordi en del informasjon i rådataene ikke blir brukt, og fordi det ikke blir korrekt tolket. Slik som det er nå, blir hver bygning definert av et flatt grunnriss og høyden, som er en gjennomsnittsverdi av punktene som definerer takkanten til bygningen. Siden høyden takkanten på en bygning ikke er nødvendigvis lik for bygget, blir dette ikke alltid en helt korrekt visualisering av bygningen.

Det første steget i en forbedringsprosess ville være å finne hvilke geometriske dataobjekter som hører til en gitt bygning. Slik kunne man da bygge opp et bygningsobjekt, som da ville inneholde referanser til alle dataobjektene som hører til dette bygningsobjektet. Hvert dataobjekt ville da ha en referanse til det bygningsobjekt det tilhører. I tillegg ville det også ha en referanse til objektet som det måtte dele punkter med. Bygningsdelelinjer må i dette tilfellet referere til bygningen den deler.

Å knytte alle dataobjektene til en bygning ville kreve et geometrisk søk, for å sjekke hvilken bygning hvert objekt tilhører. Utenpåliggende strukturer, som verandaer, vil sannsynligvis kreve spesiell oppmerksomhet, siden disse ligger utenfor området som er definert av flateobjektet.

En annen liten optimalisering vil være å knytte høydekurveobjektet som ligger rundt hver bygning til bygningen. Slik det er nå blir høyden på hver bygning satt fra takkanten og ned til havoverflaten. Utseendemessig har det liten betydning, siden terrenget skjuler den delen av objektet som er under bakken, men for å få en mer optimal modell vil det være ønskelig om høyden på bygningen ikke går lenger ned en til litt under det laveste punktet til høydekurven som ligger under bygningen.

7.1.2. Bruk av geografiske koordinater i modellen.

Bruk av geografiske koordinater vil være en av de tingene som gjør 3D visualisering til en geografisk visualisering. Hvis man først har fått opp en visualisering basert på geografiske data, burde det ikke være altfor komplisert å legge inn geografiske koordinater som referanse. Grunnen til at det ikke har vært brukt, er at det krever bruk av GeoVRML eller X3D med geospatial komponent. Problemet er å finne gode fremvisere som støtter dette. Derfor har jeg foreløpig latt dette ligge. Men geospatiale koordinater vil absolutt øke nytteverdien av modellen. Det gjør det mulig blant annet å gjøre geografiske søk i modellen. Det vil si at man søke etter steder i modellen ved hjelp av bredde og lengdegrader. En annen mulighet er å kunne få oppgitt geografisk posisjon på stedet man befinner seg i modellen.

7.1.3. Integrere data om veier og gater i visualiseringen

En ting som bør vurderes, er hvordan gater, veier og bruer kan integreres med resten av visualiseringen. Er det i det hele tatt nødvendig? Ved å bruke ortografiske foto, som tekstur på terrenget, kan man på en enkel og effektiv måte visualisere gater og veier. Er det da nødvendig å øke datamengden ytterligere, for å få en helt korrekt gjengivelse? Hvordan kan disse data best integreres med allerede terrengdata, som allerede blir brukt i visualiseringen?

7.1.4. Lage avgrensninger for terreng ved kystlinje

En annen mangel er kystlinje som avgrenser terrenget. Datasettet som er tilgjengelig for øyeblikket har ikke dette. En forbedring av TerrainMesh ville være å kunne legge inn denne som begrensning for hvor triangulering skal foretas. Diskusjon om hvordan elver og bekker skal integreres i modellen, vil på de fleste praktiske områder bli lik den om veier, nemlig om tekstur er nok til visualisere elver og bekker, eller om man trenger å hente disse dataene for å bruke dem til visualisering.

7.1.5. Anlegg, gjerder, flaggstenger, lyktestolper ol.

Tilslutt er det også geodata tilgjengelig, som beskriver det som med samlebegrep kalles anlegg. Dette er gjerder, murer, flaggstenger, lyktestolper, kumlokk og lignende. Disse er det også mulig å ta med i en visualisering. For lyktestolper og flaggstenger kan man bruke et generisk objekt, eller prototyp. Det vil si at man legger et 3D objekt for eksempel, en flaggstang og bruker den på alle stedene, der det er markert at det finnes en flaggstang. Samme teknikk kan brukes for lyktestolper. Bildet (Figur 7.1. Eksempel på bruk av et objekt (flaggstang).) under viser et eksempel på dette. Her en generisk modell av en flaggstang lagt inn i modellen.

Figur 7.1. Eksempel på bruk av et objekt (flaggstang).

Murer og gjerder kan også legges inn som enkle objekt. Murer og gjerder er kan enkelt visualiseres som en enkel loddrett flate. Denne kan man legge en generell tekstur på. Siden det også er angitt hva slags typer gjerder og murer det er snakk, kan man legge på tekstur alt etter hva slags mur eller gjerde det dreier seg om.

Alt dette er relativt enkelt å implementere, men i forhold til terreng, bygninger og veier er dette små objekter, og de blir først interessante når man lager en visualisering på gateplan. Men da bør først terreng, bygninger og veier være på plass. Dette er kort sagt prioritert ned.

7.1.6. Bruk av GML

I et system som på mer generell basis kan visualisere geografiske data, vil muligheten til å importere og tolke GML, være sentral. Ved å bruke GML kan men utvide systemet til å visualisere data, utover det som blir brukt i dette prosjektet.

7.1.7. Legge generisk tekstur på bygninger, ut i fra hva slags bygningstype, størrelse det er snakk om

Et problem ved visualisering av bygninger er at man ikke har noen teksturer for veggene på bygninger. Arbeidet med å anskaffe teksturer for alle veggene for området det er snakk om, vil også være for tidkrevende. En mulighet kan være å bruke et sett med generell tekstur, der bygningstypen angir hvilket sett med tekstur som blir brukt. For eksempel en for murhus, en for trehus, en for fabrikk, en for kjøpesenter osv.

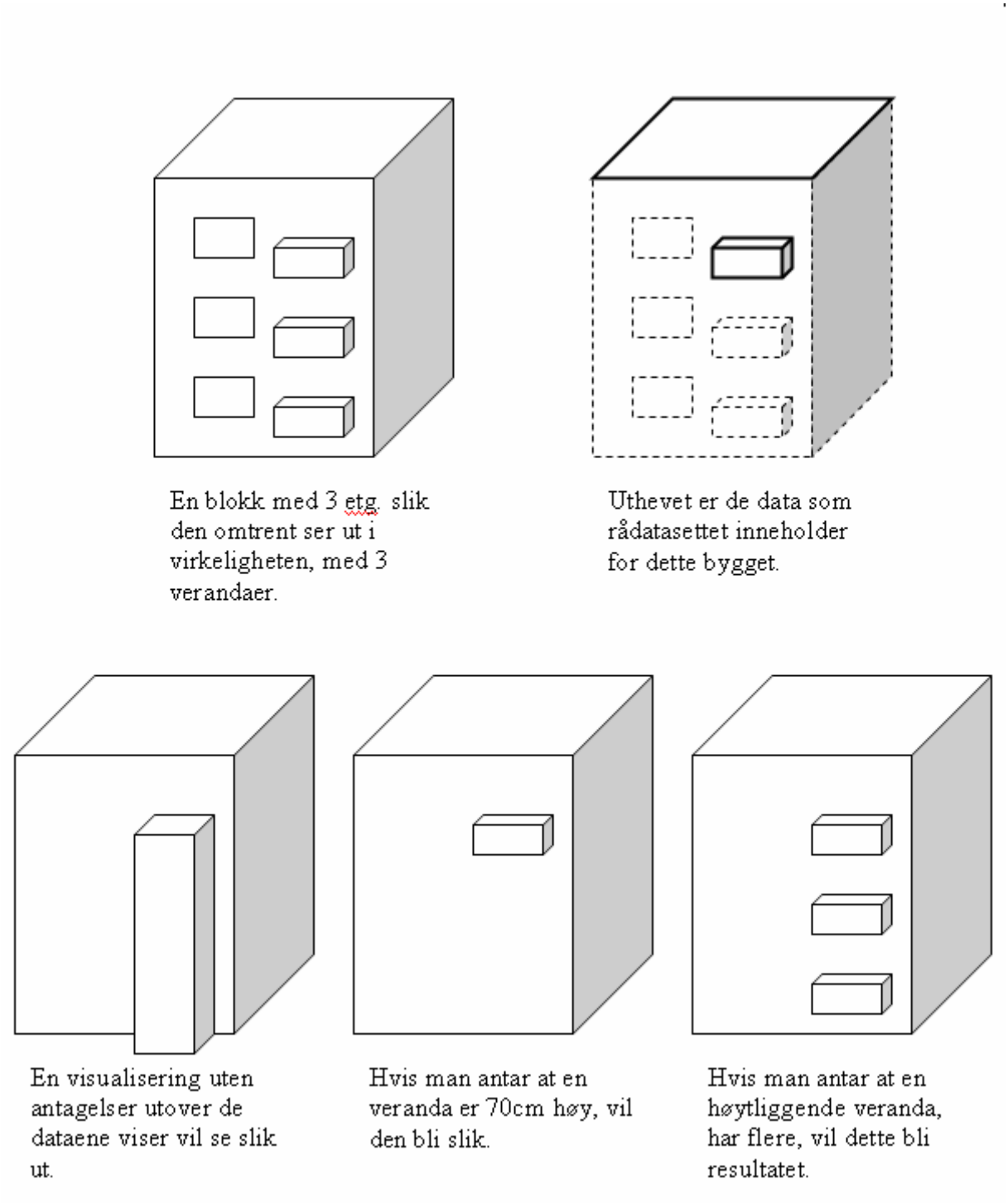
7.1.8. Bruke SOSI data for murene på Fredriksten Festning når man genererer terreng

Fredriksten Festning er et slags spesialtilfelle i denne visualiseringen og ganske atypisk for terrenget i resten av Halden sentrum. Festningen ligger i det mest kuperte terrenget og har et sett med murer. Disse murene ligger separat lagret i forhold til resten av terrenget. En mulighet kunne være å lese inn disse data når man genererer terrenget. Dette gjør forhåpentligvis terrenget mer korrekt for dette spesielle området.

7.1.9. Om geografiske 2D data

Det er klart at datasettet som man hadde tilgang til har flere begrensninger. Informasjonen er i utgangspunktet 2D, der hvert punkt inneholder en attributt som definerer høyde. Selv om man finner ulike løsninger for å løse svakheter i rådatasettet, vil dette alltid være et faktum, i hvert fall så lenge man jobber med denne typen data. Det betyr at alt som ser er skjult under et annet objekt, ikke kan visualiseres. Det vil si veier og elver som går under broer, eller verandaer som ligger rett under en annen veranda. Et annet problem er at man heller ikke vet utstrekningen på slike objekter. Ut fra disse data vet man høyden, men ikke tykkelsen på en bro.

På en veranda sier dataene noe om hvor høyt gelenderet ligger i terrenget, men ikke hvor høyt gulvet ligger. Skal man lage en visualisering av en veranda uti fra de data man har, må man bruke generellere antagelser. For eksempel kan man anta at et gelender på en veranda er omtrent 70 cm høyere enn gulvet og ut i fra dette generere en troverdig visualisering. Man kan gå enda lenger i sine antagelser. For eksempel beregne høyden til en veranda i forhold til terrenget. Hvis høyden tilsier at verandaen ligger i andre, tredje eller i en etasje enda høyere, kan man anta at dette er øverste veranda i en blokk, som har flere verandaer under hverandre og da visualisere flere verandaer under hverandre, tilsvarende antall etasjer. Man må da også anta at en etasje har en standard høyde på for eksempel 270 cm. (Figur 7.2. Visualisering av veranda)



Figur 7.2. Visualisering av veranda

Problemet er nå at man gjør en rekke antagelser, som kan være, eller ikke være korrekt. Det vil høyst sannsynlig være minst et tilfelle der antagelsene man har gjort, vil være helt feil. Poenget er hvor mange antagelser er man villig til å foreta for å få en god visualisering? Vil man ikke godta noen antagelser og bare generere objekter, slik som rådatasettet tilsier, vil dette føre i tilfellet med verandaer til at alle verandaer vil bli gå fra sin definerte høyde ned til bakkenivå. Eller man kan godta enkelte antagelse og si at et en normal veranda er 70 cm høy. Eller man skal man prøve å gjøre så mange man kan? Jo flere antagelser man gjør, jo flere feil vil de også føre til i visualiseringen. En annen side er at de også blir vanskeligere å implementere, og gjør systemet mer komplisert og med flere feilkilder. Derfor bør man avveie

om visualiseringen vil bli merkbart bedre, hvor mange feil som vil oppstå på grunn av den og hvor komplisert den er å implementere.

7.2. Formål og konklusjon

7.2.1. Forslag til formål for modellene

Hvilke formål kan man bruke 3D visualisering til? Å gjenskape virkeligheten til minste detalj er ofte ikke det som er mest interessant. Det som ofte er mer interessant er å visualisere det som ikke eksisterer lenger, eller det som ikke eksisterer ennå. En av fordelene med en VR-modell er at den er relativ enkel å modifisere. Et av formålene med en VR-modell kan derfor være å gjenskape en bygning eller et landskap som av en eller grunn ikke lenger eksisterer. Enda viktigere kan det være å visualisere bygninger eller byggverk, som enda ikke er bygd. Hvordan vil slike passe inn i det allerede eksisterende bybilde eller landskap? En annen ting som kan være viktig å visualisere, er ting som eksisterer, men som er delvis eller helt skjult. En av fordelene med visualisering av kartdata, er at man enkelt kan skille data fra hverandre. På denne måten kan man for eksempel enkelt visualisere terreng i byer som normal er skjult under bebyggelse. Det kan også være at formålet ikke er å visualisere byen, men selve datasettet, for eksempel i forbindelse med verifisering og validering.

Personlig ser jeg på visualisering av nybygg som et av de mest nærliggende formålene for en slik VR-modell. Dette vil være spesielt ved større nybygg, slik som for eksempel oppføring av bygninger for institusjoner. Disse har en tendens til å være store og sentrumsnære og kan derfor ha en stor innvirkning på bybildet, i noen tilfeller katastrofale. Hvis man da har en eksisterende visualisering i byen, kan man sette planen for nybygget i denne. For en slik visualisering er det viktig at inntrykket av masse kommer riktig og tydelig frem. Blir bygget for høyt, sperrer det utsikten, eller er det for massivt? Modellen behøver nødvendigvis ikke ha så stor realisme som mulig, derfor er det ikke nødvendig med teksturer, som faktisk kan virke forstyrrende for dette formålet. Hvis man derimot skulle ønske å bruke en slik visualisering i reklameøyemed, vil det være absolutt nødvendig å ha teksturer, effekter og tilbehør som øker realismen. En av fordelen med 3D visualisering til dette formålet, er at man kan få verifisert at byggverket føyer seg inn i bybildet slik man ønsker. En annen fordel og kanskje den viktigste i denne sammenhengen, er at man veldig enkelt kan prøve ut et uttall løsninger, som for eksempel i en arkitektkonkurranse. Virkelig store bygg kan påvirke lokale vindforhold. Hvis man kobler en slik visualisering opp mot en simulator for vind, kan man på forhånd forutsi om et nybygg vil påvirke vindforhold på en uheldig måte.

Et annet bruksområde for visualisering av kartdata kan være til bruk i simulasjon av forskjellige scenarier. Typisk er når man bruker VR for simulering av hendelser, om disse er av en slik art at de ikke kan gjenskapes i virkeligheten uten store konsekvenser. Dette kan være naturkatastrofer, slik som jordskjelv, flom, skred eller lignende, eller det kan være store ulykker, slik som store industriutslipp, nedsmeltning av kjernekraftverk eller flystyrt. Flysimulatorer har lenge brukt 3D visualisering. I slike tilfeller kan det være interessant å legge andre typer data på toppen av modellen, gjerne da ting som normalt ikke er synlig, slik som for eksempel utbredelse av forurensing eller radioaktivitet.

Verifisering og validering av selve datasettet, er et område hvor det kan være nyttig med en 3D visualisering. I dette prosjektet ble det avslørt direkte feil ved datasettet. En av disse var at noen av høydedataene var opplagt feil. Dette er feil som er umulig å oppdage ved 2D visualisering, mens de er opplagte i en 3D visualisering. For dette formålet vil det i så fall være mest hensiktsmessig med visualisering som er så ren som mulig og som visualiserer selve dataene framfor hva de faktisk representerer. I dette tilfellet vil en enkel linjemodell

være det mest hensiktsmessige, gjerne med mulighet til å kunne gå inn på hvert objekt og undersøke hvilke data som er registrert for dette.

For historiske formål kan det være interessant å prøve å gjenskape hvordan byen så ut på en viss tid. Hvis man vil skal gjenskape noe langt tilbake i tid, gjerne der alt er forandret, krever dette at all modellering skjer fra bunnen av. Likevel kan for eksempel terrengdata fortsatt være nyttig. Derimot hvis man ønsker å gjenskape noe fra nyere tid, kan det hende at mer av datasettet er brukbart. Prosessen vil da ha en del likheter med visualisering av nybygg, bare omvendt. Man fjerner da byggverk som ikke eksisterte på det tidspunkt man ønsker å visualisere og modellerer byggverk som i dag er borte. Data om disse må da hentes fra andre kilder. Jo lenger tilbake i tid man ønsker å visualisere og jo flere forandringer som har inntruffet, jo mer krevende vil denne oppgaven være. En visualisering vil også av en annen grunn være komplisert. Realismen i en slik modell bør være så høy som mulig, da hensikten med en slik visualisering er å gjenskape atmosfæren som eksisterte. Derfor bør denne være fullt teksturert, noe som kan være en utfordring når man gjerne modellerer byggverk som ikke lenger eksisterer.

7.2.2. Konklusjon

I dette har det blitt vist at man kan lage en visualisering basert på kartdata på SOSI format. Men det har også blitt avdekket en rekke problemer og begrensninger. Noen av dem er mulig å løse med bedre teknikker for konvertering, mens andre vil kreve endringer i det originale datasettet. Det har også blitt demonstrert at man kan lage modeller, som er kjørbare på ordinære PC-er og som kan lastes ned over internett, selv med begrenset båndbredde. Et lite poeng til sist er at i dette prosjektet har det kun vært benyttet åpne formater og fri programvare. Det har ikke vært nødvendig å benytte noen proprietære løsninger for å lage disse visualiseringene.

Jeg mener at når man skal lage en VR-modell av en by eller et område, bør kartdata være fundamentet som man bygger modellen på. Kartdata sammen med ortofoto kan gi en temmelig god visualisering av en by. Men disse har sine begrensninger når man ønsker flere detaljer og økt realisme. For å oppnå dette kan det nok tenkes at man bør benytte andre kilder som er mer detaljerte. En god VR-modell av en by optimalisert og med høy realisme vil nok fortsatt innebære en god porsjon manuelt modelleringsarbeid.

Appendix A Web-siden for oppgaven

3D Visualisering Av Kartdata, Modeller og Programeksempler

Herman Kolås

Sammendrag

Denne siden inneholder informasjon om de forskjellige komponentene i dette prosjektet. Det er to kategorier VRML modellene som er resultatet av dette prosjektet og programeksempler av programmene som ble brukt for å generere disse modellene.

Generelle Systemkrav

Kravet til systemet varier litt for de forskjellige komponentene. Men generelt anbefales følgende:

- PC: Intel Pentium 4 2.4GHz, AMD Athlon XP2800+ 2.083 GHz eller raskere
- Minne: Minimum 512MB
- Grafikk: 3D akselerator med minimum 64MB
- OS: Windows 2000 eller nyere

Programeksemplene kan også sannsynligvis kjøres på Linux, men dette har ikke blitt testet.

VRML Modeller

Gjennomgang av de forskjellige VRML modellene i dette prosjektet.

Systemkrav

For å kunne vise VRML modellene, må du ha Internet Explorer med Blaxxun Contact vrml plugin installert. Hvis du ikke har denne pluginen kan du hente den her.

Modell over Halden Sentrum

Dette er den største modellen man fikk generert med grunnlagsdata. Denne modellen finnes i flere versjoner. De er inkludert slik at man kan vurdere effekten av de forskjellige enkeltelementene som er inkludert.

Komplett Versjon - Denne versjon inkluderer alle elementene. Terreng med tekstur, bygninger med tekstur og i tillegg har noen av bygningene i festningen fått lagt teksturer på veggene for hånd. En animert modell av flagget på dronningens bastion er lagt til.

Full versjon med teksturer - Denne versjoner inkluderer alle elementer, som terreng med tekstur og bygninger med tekstur, men detaljene som er lagt til for hånd, som tekstur på veggene er utelatt.

Versjon med bygninger uten tekstur - I denne versjonen er takteksturen på alle bygningene utelatt.

Versjon med terreng og bygninger uten tekstur - I denne versjonen er all tekstur, både på bygninger og terreng, fjernet.

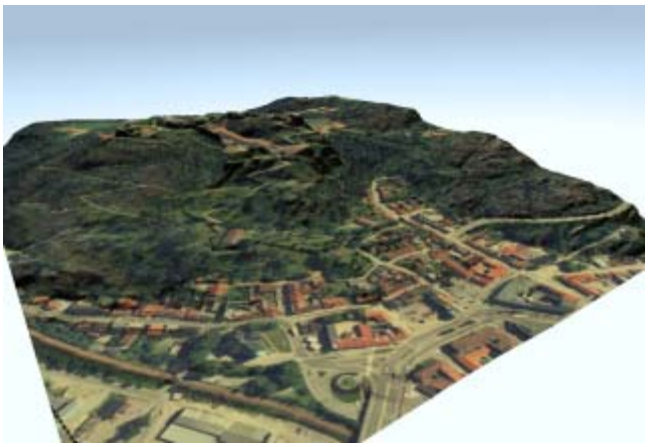
Versjon uten bygninger - I denne versjonen er bare terrenget med tekstur inkludert.

Versjon uten bygninger og tekstur - I denne versjonen er bare terrenget uten tekstur inkludert.

Tabellen under viser hva de forskjellige versjonene inkluderer og ikke inkluderer:

Versjon	Terreng	Terrengtekstur	Bygninger	Bygningstekstur	Håndlagde Detaljer
Komplett	✓	✓	✓	✓	✓
TerrengMTByggMT	✓	✓	✓	✓	
TerrengMTByggUT	✓	✓	✓		
TerrengUTByggUT	✓		✓		
TerrengMT	✓	✓			
TerrengUT	✓				

Modell over Fredriksten Festning



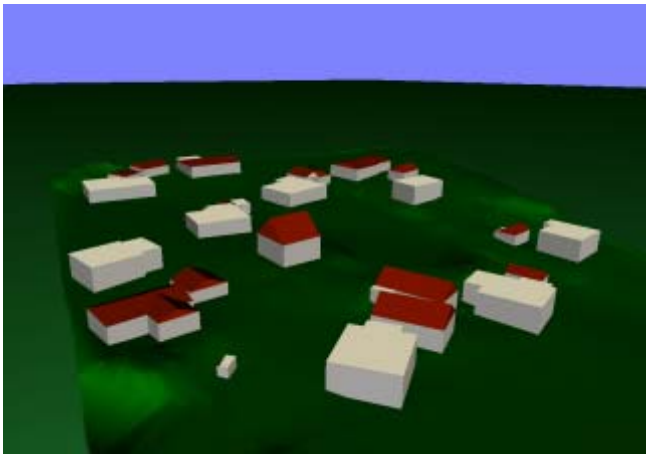
Denne modellen er en visualisering av Fredriksten Festning og området rundt. Denne modellen er inkludert både med og uten håndtering av detaljnivå (LOD). Slik at man kan se betydningen av detaljhåndtering og hvor viktig dette er for ytelse. Denne modellen er den største som det er praktisk å vise uten detaljhåndtering. I tillegg er det inkludert en modell av den originale triangluering, som er grunnlaget for beregningen av det regulære rutenettet for terrenget for denne modellen.

Festningen med LOD og tekstur - Denne modellen har håndtering av detaljnivå av terreng.

Festningen uten LOD, med tekstur - Denne modellen har ikke detaljnivå av terreng. **NB!: Denne modellen er stor. Regn med lang innlastingstid.**

Original triangulering - Trianguleringering som er utgangpunktet for terrengmodellen.

Boligområde som test for generering av tak



Dette er resultatet av forsøket på å legge inn tak på bygningene. På noen av bygningene ble ikke tak generert.

Modellen over et boligområde med tak.

Trådmodell over Halden Sentrum

Nærmest som en test av grunnlagsdata, ble det gjort et forsøk på å visualisere Halden Sentrum bare med bruk linjer. Dette er i prinsippet bare en 3-dimensjonal versjon av SosiVis.

Linjene er i forskjellig farge alt etter hvilke data de representerer. Fargekoden er som følger:

- Grønn: Terreng
- Rød: Bygninger
- Blå: Vann
- Fiolett: Jernbane
- Grå: Vei
- Gul: Gjerder, Murer, Anlegg

Legg merke til at på et bestemt område i festningen er feil i terrengdata.

Linjemodellen over Halden Sentrum

Programeksempler

Generelt om programeksemplene

Den enkleste måten å kjøre programeksemplene på er å pakke ut programeksempler.zip et sted på harddisken. Programeksemplene er implementert i Java og Perl. For å kunne kjøre Perlscript på pc må du har ActivePerl for windows installert. Installasjonspakken for denne kan hentes her. For å kunne kjøre javaeksemplene må du ha en java virtual machine installert. Dette gjelder også eksempelet med Rez.

Script for generering av bygninger

For kjøre scriptet som generer bygninger gå inn i katalogen; Programeksempler\perlscript. Og kjør filen parseBygg_texture.pl. Følg instruksjonene i programmet og velg hvilket område som skal genereres. Resultatet blir lagt i Programeksempler\output.

Script for generering av linje modeller

For kjøre scriptet som generer bygninger gå inn i katalogen; Programeksempler\perlscript. Og kjør filen sosiparser.pl. Følg instruksjonene i programmet og velg hvilket område som skal genereres. Resultatet blir lagt i Programeksempler\output.

Program for generering av terreng

For å kjøre programmet for generering av terreng; gå til katalogen Programeksempler\java. Og kjør terrainmesh.bat. I dialogboksen som kommer opp velger du hvilket område som skal genereres. Du kan også velge hvilken datafil du vil bruke. Denne skal være hoyde.sos. I tillegg kan du velge hvor den ferdige modellen skal ligge. Hvis du ikke definerer noe annet blir modellen lagt i Programeksempler\output. **Merk:** Hvis du velger å generere et større område, slik som Fredriksten festningen eller Halden Festning vil det ta **svært lang tid** å kjøre programmet.

Program for generering av tak

For å kjøre programmet for generering av tak: Gå til katalogen Programeksempler\java. Og kjør buildingmesh.bat. I dialogboksen som kommer opp velger du hvilket område som skal genereres. Du kan også velge hvilken datafil du vil bruke. Denne skal være bygg2.sos. I tillegg kan du velge hvor den ferdige modellen skal ligge. Hvis du ikke definerer noe annet blir modellen lagt i Programeksempler\output. **Merk:** Programmet har så langt bare klart å produsere et resultat for de to minste områdene ('To bolighus med valmet tak' og 'Et boligområde for test av tak'). Hvis du prøver å generere hustak for 'Fredriksten festning' og 'Halden Sentrum' vil du neppe få et fornuftig resultat. (Se rapporten om hvorfor.)

Testeksempel for Rez

I tillegg til egne programmer har jeg inkludert et testeksempel for Rez. Dette programmet brukes til å splitte terrenget opp i mindre biter for LOD'ing. Terrengdata for Fredriksten Festning er lagt for å kunne teste Rez.

For å kjøre Rez: Gå til katalogen, Programeksempler\RezTestExample. Og kjør run.bat.

Disse parameterene må være som følger:

- First Tree Level: 0
- Final Tree Level: 4
- Sampling Increment: 0
- Minimum Tile Dimension: 30
- Maximum Tile Dimension: 40
- z/latitude translation: 0.0
- y/height translation: 0.0
- x/longitude translation: 0.0
- Height Scale: 1.0
- Horizontal Scale: 1.0
- Source Data directory: worldSingleEG/

For å se det ferdige resultatet kan man åpne fila:
\MasteroppgaveCD\Programeksempler\RezTestExample\WorldSingleEG\festningen\Display.wrl.

Herman Kolås, 2004

Bibliografi

Bourdakis, V. (2001). On Developing Standards for the Creation of VR City Models [WWW Document] URL http://www.hut.fi/events/ecaade/E2001presentations/15_01_bourdakis.pdf (Sist besøkt 2004, 14. desember)

Bourke, P. (1990). Minimum Requirements for Creating a DXF File of a 3D Model [WWW Document] (Rev. September 1993) URL http://astronomy.swin.edu.au/~pbourke/geomformats/dxf/min3d.html (Sist besøkt 2004, 14. desember)

Carey, R., Carson, G. S., & Puk, R. F. (1997). The Development of the VRML 97 International Standard [WWW Document] URL http://www.web3d.org/aboutus/historydevelopment.htm (Sist besøkt 2004, 12. mars)

Evans, S., & Hudson-Smith, A. (2001). Information Rich 3D Computer Modeling of Urban Environments [WWW Document] URL http://www.casa.ucl.ac.uk/paper35.pdf (Sist besøkt 2004, 14. desember)

Fremming N. P., Hjelle, Ø., & Tarrou, C. (1997). Surface Modelling from Scattered Geological Data. I M. Dæhlen & A. Tveito (Eds.), Numerical Methods and Software Tools in Industrial Mathematics (s. 317-328). Birkhäuser Boston.

Heckbert, P. S., & Garland, M. (1997). Survey of Polygonal Surface Simplification Algorithms [WWW Document] URL http://graphics.cs.uiuc.edu/~garland/papers/simp.pdf (Sist besøkt 2004, 14. desember)

Kada, M. (2002). Automatic Generalisation of 3D Building Models [WWW Document] URL http://www.ifp.uni-stuttgart.de/publications/2002/245_kada_paper_ottawa.pdf (Sist besøkt 2004, 14. desember)

Kanellopopoulos, I., Stein, A., & Turatto, M. (2001). Visualisation of Geographic Information in a Dynamic 3-Dimensional Environment [WWW Document] URL http://wwwlmu.jrc.it/Workshops/7ec-gis/papers/pdf/kanellopoulos.pdf (Sist besøkt 2004, 14. desember)

Kristiansen, D., Østby, C. A., & Louka, M. (1998). Fredriksten Fortress [WWW Document] URL http://www2.hrp.no/vr/vrml/fredriksten/ (Sist besøkt 2004, 14. desember)

Lal, J., & Meng, L. (2001). Rules and Constraints for 3D Generalization of Urban Area [WWW Document] URL http://129.187.92.218/publications/raheja/3d_generalization_ag_2001.pdf (Sist besøkt 2004, 10. april)

Lin, H. (1997). Javamesh - a two dimensional triangular mesh generator for finite elements [WWW Document] URL http://www.steven.pop.net.tw/javamesh/my_html/project.html (Sist besøkt 2004, 20. januar)

Misund, G. (1997). Varioscale Surfaces in Geographic Information Systems. I M. Dæhlen & A. Tveito (Eds.), Numerical Methods and Software Tools in Industrial Mathematics (s. 317-328). Birkhäuser Boston.

Reddy, M., & Iverson, L. (2002). GeoVRML 1.1 Specification [WWW Document] URL http://www.geovrml.org/1.1/doc/ (Sist besøkt 2004, 14. desember)

Schilling, A., & Zipf, A. (2003). Generation of VRML City Models for Focus Based Tour Animations [WWW Document] URL http://www2.geoinform.fh-mainz.de/~zipf/Zipf-Schilling-Web3D-2003-final.pdf Integration, Modeling and Presentation of Heterogenous Geo-Data Sources. (Sist besøkt 2004, 14. desember)

Statens Kartverk. (1999). SOSI Standarden v3.4 [WWW Document] http://www.statkart.no/standard/sosi/html/sosi.htm (Sist besøkt 2004, 14. desember)

Thorne, C. (1999). *Rez* [WWW Document] URL http://www.surak.com.au/~chris/vrml/RezIndex.html Open source framework and tools for translating terrain data and images to different formats including multiresolution versions optimised for web browsing. (Sist besøkt 2004, 14. desember)

USGS Eastern Region Geography. (2003). Geographic Information Systems [WWW Document] URL http://erg.usgs.gov/isb/pubs/gis_poster/ (Sist besøkt 2004, 14. desember)

Web3D Consortium Inc. (1997). Extensible 3D (X3D) ISO/IEC FDIS 19775:200x Part 1: Architecture and base components [WWW Document] URL http://www.web3d.org/x3d/specifications/ISO_IEC_PDAM_19775_1_Am1/index.html (Sist besøkt 2004, 14. desember)

Web3D Consortium Inc. (1997). VRML 97 ISO/IEC 14772-1:1997 The Virtual Reality Modeling Language [WWW Document] URL http://www.web3d.org/x3d/specifications/vrml/vrml97/vrml97specification.pdf (Sist besøkt 2004, 14. desember)

Web3D Consortium Inc. (1999). History of the VRML Specification [WWW Document] URL http://www.web3d.org/aboutus/historyspec.htm (Sist besøkt 2004, 21. februar)

Web3D Consortium Inc. (1999). Extensible 3D (X3D) ISO/IEC FDIS 19775:200x Part 1: Architecture and base components, 25 Geospatial Component [WWW Document] URL http://www.web3d.org/x3d/specifications/ISO_IEC_PDAM_19775_1_Am1/Part01/components/geodata.html (Sist besøkt 2004, 14. desember)

Web3D Consortium Inc. (2000). Web3D Consortium FAQ [WWW Document] URL http://www.web3d.org/fs_faq.htm (Sist besøkt 2004, 21. februar)

Web3D Consortium Inc. (2002). X3D (Extensible 3D) Frequently Asked Questions (FAQ) [WWW Document] URL http://www.web3d.org/x3d/faq/index.html (Sist besøkt 2004, 14. desember)