Sammendrag

Terrengmodellering i 3D er i dag en mye brukt måte å fremstille landskap på. Slike modeller kan man se i utallige dataspill, animasjonsfilmer, og geologiske modeller. Den vanligste måten å fremstille et 3D terreng på er ved bruk av kotekart. Mønsteret i en treplate kan minne mye om slike koter, og ved segmentering av mønsteret kan dette resultatet behandles på samme måte som et ordinært kart. Denne rapporten beskriver hvordan en finerplate kan bli ført fra treplate til trelandskap ved hjelp av generelle metoder innenfor området for bildebehandlig og 3D modellering. For sluttproduktet blir to løsninger fremstilt. Den ene er et selvlaget program kalt 3Dtre som viser modellen ved hjelp av Delauney triangulering, den andre er basert på bilblioteket SIM scenery og programvare fra Systems in Motion.

Forord

Dette dokumentet er masteroppgaven til Ingrid Marie Hjellset Larsen, utført våren 2005 som en del av studiet i Master i teknologi i datateknikk ved NTNU. Hun vil rette en takk til Torbjørn Hallgren og Knut Ragnar Holm for veiledning, og Yngve Zakarias for ideen bak oppgaven. I tillegg rettes en takk til Systems in Motion for bruk av SIM Scenery.

Oppgavebeskrivelse

Norwegian Wood Innovation

Kunstneren Yngve Zakarias vil høsten 2005 ha en stor utstilling i Trondheim kunstmuseum. Denne utstillingen vil han først og fremst basere på sitt nye prosjekt Virtual Woodscape. Ideen bak prosjektet går ut på å modellere en finerplate som et terreng i 3D. Utstillingen vil bestå av store trykk, animasjoner og visning i en stereoskopisk virtuell omgivelse.

Målet med denne masteroppgaven blir å sette seg inn i kunstnerens problemstillinger og prøve å finne metoder for hvordan ideene hans kan realiseres.

De metodene som det vil være behov for, er sammenfallende med generelle metoder som brukes på tilsvarende typer av bilder for å syntetisere geometriske modeller i sin alminnelighet.

En finerplate viser et mønster som er karakteristisk for treslaget den er laget av. Furu viser et kraftig mønster som ligner på kotene på et topologisk kart. Oppgaven vil gå ut på å segmentere ut kurver på grunnlag av finerplatens mønster. Neste trinn blir å tolke kurvene som koter og deretter etter regler som må lages, bygge opp en terrengmodell i 3D. Her kommer mange av de problemstillingene som er aktuelle i terrengmodellering, inn.

Innhold

1 Introduksjon **1**
- 1.1 Mål . . . 1
- 1.2 Begrensninger . . . 1
- 1.3 Utfordringer . . . 1
- 1.4 Rapportens struktur . . . 2

2 Bakgrunn **3**
- 2.1 Bildeanalyse . . . 3
- 2.2 Terrengmodellering . . . 3

3 Metoder **5**
- 3.1 Segmentering . . . 5
  - 3.1.1 Terskling . . . 5
  - 3.1.2 Jevning . . . 6
  - 3.1.3 Kantdeteksjon . . . 6
  - 3.1.4 Kantknytting . . . 8
  - 3.1.5 Hough transformasjon . . . 8
  - 3.1.6 Aktive konturer . . . 9
- 3.2 Morfologiske metoder . . . 10
  - 3.2.1 Erosjon og Dilasjon . . . 10
  - 3.2.2 Åpning og lukking . . . 11
  - 3.2.3 Tynning . . . 11
- 3.3 Terrengmodellering . . . 11
  - 3.3.1 Finne punkter . . . 13
  - 3.3.2 Polygonisering . . . 13
  - 3.3.3 Level of Detail . . . 16
  - 3.3.4 Teksturmapping . . . 19
  - 3.3.5 Lyssetting . . . 21
- 3.4 Stereoskopisk projeksjon . . . 22

4 Implementasjon **25**
- 4.1 Segmentering . . . 25
  - 4.1.1 Terskling . . . 25
  - 4.1.2 Kantdeteksjon . . . 25
  - 4.1.3 Hough transformasjon . . . 27
  - 4.1.4 Aktive konturer . . . 28
- 4.2 Morfologiske metoder . . . 30
  - 4.2.1 Tynning . . . 30
  - 4.2.2 Åpning og Lukking . . . 30
- 4.3 Alternative metoder . . . 31
- 4.4 Terrengmodellering . . . 32
  - 4.4.1 Triangulering . . . 34

4.4.2 Culling . . . 34
4.4.3 Tekstur . . . 34
4.4.4 Lyssetting . . . 35
4.5 Steroskopisk projeksjon . . . 36
4.6 Andre Ting . . . 36

5 Testing og evaluering **37**
5.1 Testing . . . 37
5.1.1 Finne punkter . . . 37
5.1.2 Delauney triangulering . . . 38
5.1.3 Modellen i sanntid . . . 38
5.1.4 Bakside culling . . . 39
5.2 Evaluering . . . 39
5.2.1 Segmentering . . . 39
5.2.2 3Dtre . . . 40
5.2.3 SIM Scenery . . . 40

6 Diskusjon **43**

7 Konklusjon **45**

8 Videre arbeid **47**

A Bilder **53**

B Bruksanvisning 3Dtre **65**
B.1 Hva som må være klart . . . 65
B.1.1 Kotekart . . . 65
B.1.2 Teksturer . . . 65
B.1.3 Eventuelt . . . 66
B.2 Kjøring av programmet . . . 66
B.2.1 Fra kurvekart til høydebilde . . . 66
B.2.2 Fra høydebilde til regulære punkter . . . 67
B.2.3 Fra kurvekart til irregulær polygonmodell . . . 68
B.3 Navigering . . . 69

C Bruksanvisning SIM basert program **71**
C.1 Hva som må være klart . . . 71
C.1.1 Punkter . . . 71
C.1.2 Tekstur . . . 71
C.1.3 Database . . . 71
C.2 Kjøring av programmet . . . 72

Figurer

2.1 Kunstig landskap laget i Terragen 4

3.1 Adaptiv segmentert terskling.(Privat foto av Hufvudstaden i Göteborg) 6
3.2 En typisk maske for jevning 6
3.3 Kant 7
3.4 Hough transform 8
3.5 Strukturelement 10
3.6 Fra punkter til trekanter 14
3.7 Voronoy punkter er også tatt med 15
3.8 Koter med utvalgte voronoypunkter. 15
3.9 Uønskede oscillasjoner. 15
3.10 Level of Detail 16
3.11 Hull 17
3.12 T-kryss 17
3.13 Geomorfing 17
3.14 Mipmap pyramide 19
3.15 Tekstur mapping 20
3.16 Mapping fra mellomstasjon 20
3.17 Antialiasing 21
3.18 Mipmapping av en trekant 21
3.19 Fysisk modell 23
3.20 Stereopar (figurer funnet på http://www.flibberkids.com) 23
3.21 Rød-cyan stereoskopi 24

4.1 Problemer med terskling 26
4.2 Horisontale linjer til problem 26
4.3 Hough transformasjon utført på testbilde 27
4.4 Hough transformasjon utført på koter 28
4.5 Aktive konturer rundt en hullete sirkel 29
4.6 Aktive konturer med et fast kontrollpunkt 29
4.7 Aktive konturer 29
4.8 Masker for tynning 30
4.9 Tynning utført på kotene 30
4.10 Åpning og lukking utført på kotene 31
4.11 Rød/blå teknikk 32
4.12 Varianter av teksturmapping på et lite stykke av finerplaten 35
4.13 Rød-cyan 36

5.1 Problemer med bakside culling 39

8.1 Alternativ punktalgoritme 48

A.1 Finerplaten 54

A.2 Resultatet fra adaptiv terskling 55
A.3 Kotekartet 56
A.4 Høydebilde 57
A.5 Delauney triangulering 58
A.6 Forside culling 59
A.7 Porcelainscape 60
A.8 Lystekstur 61
A.9 Modell med tåke 61
A.10 Oversikt over hele terrenget med lystekstur 62
A.11 Nærbilde av et lite utsnitt med høy oppløsning fra 3Dtre 63
A.12 Nærbilde av et lite utsnitt i SIM Scenery 63

B.1 Teksturer som er tilpasset 65
B.2 Tekstfil med teksturer 66
B.3 Høydebilde 67
B.4 Tekstur 69

C.1 hdr fil 71
C.2 SIMScenery 72

Kapittel 1

Introduksjon

Kunstneren Yngve Zakarias (http://www.zakart.com) fikk høsten 2004 ideen om å kombinere datateknikk og kunst i anledning sin store kunstutstilling i Trondheim kunstmuseum høsten 2005. En finerplate i et av vinduene på branntomta til Vækteren bistro i Trondheim fikk han på tanken. Finerplaten er ca 150*120 cm^2 og har et usedvanlig tydelig og mangfoldig mønster(se figur A.1). Zakarias har tidligere jobbet med diverse tretrykk, og finerplaten startet prosessen med å utvikle trestruktur som kunst videre. Ideen går ut på å lage et virtuelt trelandskap med basis i mønsteret i finerplaten.

1.1 Mål

Målet med oppgaven er å undersøke og utprøve metoder for å segmentere ut mønsteret en finerplate. Når mønsteret er segmenter ut, skal det behandles som høydekoter og resultere i en 3D modell med ulike former for visningspotensiale.

1.2 Begrensninger

Resultatet for oppgaven skal være en 3D modell av finerplaten. Det vil henholdsvis være tilstede et enkelt vindu hvor modellen vises. Det settes ingen krav til dette grensesnittet, da det bare vil være et testverktøy. I tillegg vil programmet som blir laget være noe tilpasset denne bestemte problemstillingen. Målet er ikke å lage en generell programvare for terrengmodellering. Kunstneren behøver en rekke materiale til sin kunstutstilling. Å produsere dette materiale vil ikke være en del av oppgaven, d.v.s. hovedinnholdet blir først og fremst å prøve metoder og teste ut løsninger, og så kan kunstneren benytte seg av disse løsningene om det skulle være ønskelig.

1.3 Utfordringer

Oppgaven skal gjøres i samarbeid med kunstneren Yngve Zakarias, noe som skiller denne oppgaven ut fra de rene datatekniske. Ved å jobbe med kunst som er noe som er i konstant bevegelse, vil det hele tiden komme nye elementer og innspill til problemstillinger og resultater. I tillegg blir dette et samarbeid mellom datateknikk og kunst, som er to grupper med helt forskjellig faglig språk og visjoner. Det som en datateknikker ser for seg av virtuelle terreng kan være totalt forskjellig fra hva en kunstner tenker. Dette vil gjøre oppgaven utfordrene, men samtidig veldig interessant.

1.4 Rapportens struktur

Rapporten starter i kapittel 2 med å beskrive terrengmodellering. Kapittel 3 gjør greie for ulike relevante metoder for terrengmodellering. Implementasjonen beskrives i kapittel 4. Kapittel 5 inneholder testing og evaluering. Kapittel 6 inneholder diskusjonen. Rapporten konkluderes i kapittel 7. Kapittel 8 beskriver videre arbeid.

Kapittel 2

Bakgrunn

2.1 Bildeanalyse

Analysering av bilder ved hjelp av dataprogrammer blir gjort innefor flere og flere fagfelt. De X-ray og ultralydsbildene som før bare ble gransket for hånd kan i dag undersøkes grundig ved å bruke kjente metoder innenfor bildesegmentering og analyse. Segmenteringsmetoder baserer seg først og fremst på gråtoner og intensitesnivå, men også på gradienter og form. Høydekoter fra et kart kan segmenteres ut på mange ulike måter, alt etter hvordan bildet ser ut, og hva man ønsker. De mest åpenlyse metodene er terskling og kantdeteksjon, men også Hough transformasjon eller aktive konturer kan finne linjer og lukkede ringer.

Det finnes mye eksisterende programvare som spesialiserer seg mer og mer på bildeanalyse, hvor segmentering er et av satsingsområdene. Noen eksempler på slik programvare er GIMP [26] og Adobe® Photoshop®.

2.2 Terrengmodellering

Detaljrike 3D modeller av jordoverflaten er noe de fleste i dag har sett. Radarsatellitter har i lang tid kunnet måle høydeverdier fra jordoverflaten, og kombinert med høyoppløselige fotografier av overflaten kan dette utvikles til naturtro modeller av terrenget på jorden. For øyeblikket er mye av dette klassifiserte data, og kan bare bli kjøpt for høye priser. Mange av teknikkene for visualiseringen av terreng er likevel offentlig kjente, og det finnes metoder for å bygge terrengmodeller både ut fra punkter og høydekurver.

Terrengmodeller blir i dag ikke bare brukt for å modellere eksisterende terreng, men også i animasjonsfilmer og dataspill. Flysimulatorer er et område hvor slike modeller er et av de viktigste elementene. Det finnes mye eksisterende programvare som kan visualisere både ekte og kunstig konstruerte terrenger. Terragen [23] er en slik programvare til bruk for fotorealistiske landskapsbilder og animasjoner. Norske SIM (Systems in Motion)[24] utvikler også programmer for terrengvisualisering, og har spesialisert seg innenfor olje og gass. De hjelper flere store oljeselskaper med visualisering av havbunn i tillegg til at de har gjort en stor visualisering av Svalbard.

Figur 2.1: Kunstig landskap laget i Terragen

Kapittel 3

Metoder

Målet med masteroppgaven er å føre en finerplate fra en treplate til et trelandskap som skal kunne vises på ulike måter. For å komme fram til disse resultatene må en rekke ulike prosesser gjennomføres. Først må mønsteret i treet skilles fra bakgrunnen og gjøres om til høydekoter, før punkter kan finnes for så å polygoniseres til å stå ut som en 3D-modell. Det finnes flere ulike metoder innenfor bildebehandling og visualisering som beskriver hver og en av disse prosessene, og noen av disse vil bli gjennomgått i dette kapittelet.

3.1 Segmentering

Å segmentere et bilde vil si å skille ut elementer som tilhører bestemte regioner. Eksempler på segmentering i dagliglivet er å segmentere ut nummerskilt på biler i en automatisk trafikkontroll, eller å segmentere ut objekter som beveger seg i et overvåkningskamera. I denne dette oppgaven vil segmentering prøve å skille ut de naturlige linjene i finerplaten. Segmentering av objekter er enklest på gråverdibilder. For det første arbeides det bare med en komponent, og for det andre kan to farger som ser ganske forskjellige ut, og skulle tilhøre ulike objekt ha ganske samsvarende fargekomponenter.

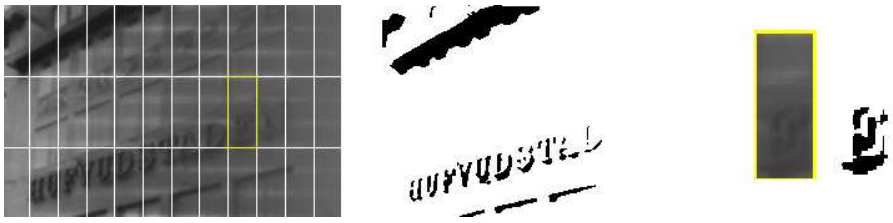
3.1.1 Terskling

Den enkleste måten å segmentere ut objekter fra bakgrunnen er å sette en terskelverdi. Terskelverdien klassifiserer bildet i to deler. En del for alle piksler med gråverdi høyere enn terskelverdien, og en for alle pikslene med lavere gråverdi enn terskelen. Alle gråverdiene til objektet vil ligge over terskelen, og alle gråverdiene til bakgrunnen ligger under terskelen(eller omvendt). Da kan alle pikslene over terskelen gis samme gråverdi, og alle pikslene under gis en annen farge. På denne måten ender vi opp med et binært bilde med et klart skille mellom bakgrunn og objekt. Dersom det finnes flere enn et objekt, og disse også skiller seg ut fra de andre i gråverdi, kan med enkle grep flere terskelverider settes og vi får en multilevel terskel.

Noen ganger kan det være vanskelig å vite på forhånd hvilke verdier som skal velges som terskelverdier, og adaptiv terskling må gjøres. Terskelverdien blir satt etter at alle gråverdiene er funnet. En typisk terskelverdi vil være gråtonegjennomsnittet.

Ofte er det ikke nok bare med adaptiv terskling. På grunn av ujevne lysforhold i et bilde kan gråverdiene til et objekt i en del av bildet tilsvare bakgrunnsverdiene

til en annen del av bildet. Da vil det være nødvendig å dele opp bildet i passelige ruter, og terskle hver rute for seg. Et eksempel er synlig i figur 3.1. Det er ønskelig at hele ordet "Hufvudstaden"(rettere sagt skyggen) skal tilhøre samme objekt. Ved enkel terskling blir de to siste bokstavene en del av bakgrunnen. Ved å dele bildet opp i ruter, og terskle det som er innenfor den gule ruten for seg kommer bokstaven E fram.

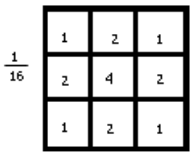


Figur 3.1: Adaptiv segmentert terskling.(Privat foto av Hufvudstaden i Göteborg)

Det viktig å sette en grense for variansen. Jo flere delbilder, desto større er sjansen for at alle pikslene i et eller flere av delbildene tilhører enten objektet eller bakgrunnen. Dersom variansen er lav må da alle piksler i et delbilde farges i objekt- eller bakgrunnsfargen.

3.1.2 Jevning

Jevning (smoothing) blir brukt til å fjerne støy i bilder. Dette kan være aktuelt for å fjerne sprekker og flekker fra finerplaten som ikke har noe med mønsteret å gjøre. Utdataen er gjennomsnittet til et lite nabolag rundt den aktuelle pikselen. Alle piksler kan vektlegges like mye, men det er også vanlig å legge størst vekt på verdien til den bestemte pikselen, og minst til de 8-connective (naboene på skrå). På denne måten fjernes irrelevante detaljer fra et bilde. Figur 3.2 viser en typisk jevningsmaske.



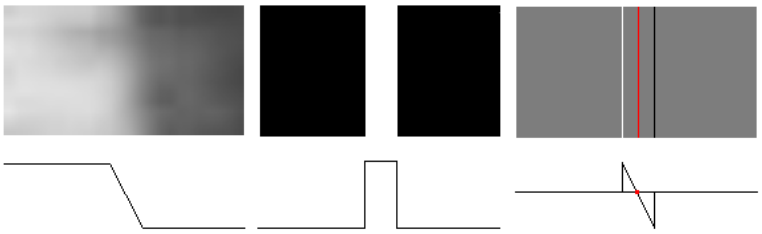
Figur 3.2: En typisk maske for jevning

Av og til kan en bedre løsning være et medianfilter. Medianfilteret velger ut medianverdien til de pikslene som dekkes av masken istedenfor gjennomsnittsverdien. Ved å bruke et medianfilter fjernes støy samtidig som bildet beholder skarpheten noe et gjennomsnittsverdifilter fjerner.

3.1.3 Kantdeteksjon

Kantdeteksjon er den vanligste metoden for å detektere diskontinuiteter i gråverdibilder. For å benytte mønsteret i treet som et høydekart, må mønsterets kanter segmenteres ut, og kantdeteksjon er den mest intuitive måten å gjøre dette på. Kantdeteksjon er mer aktuelt enn linjedetektering, som finner pikseltykke linjer,

ettersom mønsteret som regel er tykkere enn en piksel.



Figur 3.3: Kant

Kantdeteksjon baserer seg på størrelsen til den førstederiverte. Dersom absoluttverdien til en piksel er større enn en gitt terskelverdi, sier dette at pikselen høyst sannsynlig ligger på en kant. En kant vil sjelden være så så skarp at den er en piksel brei, men ofte dekke flere piksler. Den andrederiverte kan også spille en rolle for å detektere kanter. Den andrederiverte vil gi utslag for endepunktene på en kant, og henholdsvis angi om det er en kant fra mørk til lyst eller omvendt. Punktet midt mellom disse endepunktene kalles zero-crossing. Zero-crossing kan være nyttig for tykke kanter ettersom det ofte er midtpunktet på kanten som representerer den best.

Figur 3.3 viser gråverdien for en kant til venstre. Figuren i midten er den førstederiverte, og figuren til høyre viser den andrederiverte hvor den røde streken en zero-crossing.

Den førstederiverte baserer seg på gradienten til et punkt.

$$\bigtriangledown f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Størrelsen på gradientvektoren er den som avgjør om pikselen ligger på en kant eller ikke. Størrelsen finnes fra den følgende formelen:

$$\bigtriangledown f = \begin{bmatrix} G_x^2 & G_y^2 \end{bmatrix}^2$$

I bildeanalyse blir ofte gradienten approksimert ved hjelp av nummerisk differensiering. Dette gjøres ved av en maske legges over den aktuelle pikselen og de nærmeste nabopikslene. De mest vanlige maskene er Roberts kryssgradient operator, Prewitt operator, og Sobel operator[1]. Størrelsen på gradienten approksimeres ofte med absoluttverdier:

$$\bigtriangledown f = |G_x| + |G_y|$$

Som nevnt tidligere kan også den andrederiverte brukes i kantdeteksjon. Laplacian er en andrederivert definert som:

$$\bigtriangledown^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Som andrederivert produserer Laplacian doble kanter og er veldig ømfintlig for støy. Den vil derfor bare gjøre nytte for seg dersom man søker zero-crossing, eller vil vite om en piksel er på den mørke eller lyse delen av en kant.

3.1.4 Kantknytting

Kantdeteksjon finner alle klare kanter, men resultatet av deteksjonen er ikke nødvendigvis hele og kontinuerlige kanter. Av og til kan støy eller uskarpe kanter gjøre at ikke alle ønskede punkter på en kant blir funnet. Dette fører til at kantdeteksjon ofte etterfølges av kantknytting. Siden mønsteret i finerplaten skal representere stigninger og hellinger er det viktig at alle kanter som skal være kanter blir detektert for å få et riktig resultat. Siden hver kote representerer en høyde er det også viktig at kantene er kontinuerlige for å forhindre at kantstykker får egne og feil høyder.

Den enkleste måten å knytte sammen kanter på er å bruke lokal prosessering. Da analyseres egenskapene til alle pikslene innefor et lite nabolag tilhørende en kantpiksel. For å finne ut om to kantpiksler tilhører den samme kanten undersøkes to egenskaper. Den første er styrken til gradienten, og den andre er retningen på gradienten. Dersom disse egenskapene for begge piksler er noenlunde like, kan man anta at disse tilhører samme kant. De to pikslene kan da knyttes sammen med en linje.

3.1.5 Hough transformasjon

Hough transformasjon er en mye brukt global metode for å knytte sammen kanter. Den er særlig nyttig der hvor det mønsteret man søker har små diskontinuiteter, små hull, eller påvirket av støy. Idéen er å finne kurver som kan bli parameterisert som rette linjer, polynomer, sirkler, eller andre former som kan beskrives ved hjelp av matematiske uttrykk. Algoritmen kan forklares ved å ta utgangspunkt i ligningen for en rett linje:

$$y = ax + b$$

For et bestemt punkt p vil alle de punktene som har a- og b-verdi som tilfredsstiller ligningen for x- og y-verdi tilhørende p ligge på samme linje. Ligningen som da må tilfredstilles er:

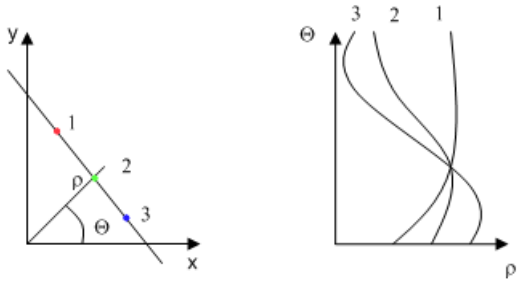
$$b = -x_p a + y_p$$

Dersom linjene tegnes opp i ab rommet, vil steder hvor linjene krysser fortelle oss at disse to punktene ligge på samme linje.

Problemet med denne ligningen er at dersom linjene er vertikale vil a og b vokse mot uendelig. Den linjerepresentasjonen som heller blir brukt er normal representasjonen av en linje.

$$\rho = x\cos\Theta + y\sin\Theta$$

Forholdet mellom de ulike variablene kan sees i figur 3.4.



Figur 3.4: Hough transform

Parameterrommet deles opp i ruter hvor det måles opp antall treff. Det er viktig å lage rutene passe store, dersom de blir for små kan to treff havne i ulike ruter, mens blir de for store kan nærliggende linjer bli regnet som en linje. De ulike stegene som må gjennomføres for å knytte sammen linjer er de følgende:
1. Bruk terskling for å få et binært bilde.
2. Angi oppdelinger i $\rho\Theta$-planet.
3. Søk etter høy konsentrasjon av piksler.
4. Sjekk forholdet mellom piksler i en bestemt celle.

Noe som kan skape problemer når det kommer til Hough transformasjonen er at den kun greier å finne bestemte former. Dersom den skal brukes for å finne rette linjer, vil den ikke finne kurvede linjer eller sirkler.

3.1.6 Aktive konturer

Et alternativ til å knytte sammen/ finne kanter er å bruke aktive konturer. Der Hough transformasjon benytter seg av kantpunktverdiene anvender aktive konturer størrelsen på gradientene i bildet. Som nevnt i forrige avsnitt fungerer Hough transformajon bare på bestemte former som sirkler eller rette linjer. Her kan aktive konturer prøve å rydde opp i problemene ettersom de ikke bryr seg om formen på kanten.

Aktive konturer er basert på en energifunksjon bestående av flere termer hvor hver av dem beskriver en kraft som virker inn på kanten. En mye brukt energiterm, blant annet beskrevet i Trucco [3] er den følgende:

$$E = \int (\alpha(s)E_{cont} + \beta(s)E_{curv} + \gamma(s)E_{image})ds$$

Denne energifunksjonen benyttes på kontrollpunkter i bildet som brukeren manuelt har plassert rundt segmenteringsobjektet. α, β og γ er vektvariabler. Disse kan modifiseres etter hvilket ledd i energifunksjonen det skal legges størst vekt på. Målet er at mengden energi som blir brukt er minst mulig. Det vil si at det er ønskelig at hvert ledd er så lite som mulig.

E_{cont} trekker konturen mot den nærmeste kanten i bildet. Dette leddet har som mål å minske avstanden mellom kontrollpunktene. Dette gjøres for å få konturen nærmere og nærmere den ønskede kanten. Dette leddet kan også ha den uønskede effekten å lage konturen mindre enn den skal. For to kontrollpunkter (x_i, y_i) og (x_{i-1}, y_{i-1}) får vi energiuttrykket:

$$E_{cont} = (x_i - x_{i-1})^2 + (y_i - yi - 1)^2$$

For å hindre at konturen krymper for mye kan en variabel $\overline{d}$ settes inn i uttrykket.$\overline{d}$ er gjennomsnittsavstanden mellom punktene i den aktive konturen. :

$$E_{cont} = (\overline{d} - (x_i - x_{i-1})^2 + (y_i - yi - 1)^2)^2$$

E_{curv} brukes for å beholde en jevn overflate. Uttrykket tar hensyn til kontrollpunkter på begge sider av valgt kontrollpunkt, og sjekker at avstanden til disse ikke er for stor.

$$E_{curv} = (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2$$

E_{image} prøver å dra konturen til den ønskede kanten. For å gjøre dette sjekkes gradienten i de ulike punktene. Man vet at en høy gradient sier oss at punktet ligger på en kant.

$$E_{image} = -|| \bigtriangledown I||$$

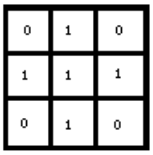
∇I er gradienten i et bestemt punkt.

Det finnes flere ulike problemer med aktive konturer. Sannsynligvis det viktigste er at det kun blir tatt hensyn til intensitetsnivået. Punkter i bildet laget av støy vil også gi utslag. Et annet problem er at det er en langt fra automatisk prosess, man må selv inn og merke av kontrollpunktene. I tillegg kan konturen legge seg på begge sider av kanter.

Det finnes flere ulike typer aktive konturer. Den typen som er beskrevet over er en kontur som krymper inn mot ønskede kant. Dette kan også gjøres motsatt ved å bruke noe som heter ballonger. En ballong starter som en liten sirkel, og blåser seg deretter opp til den treffer kanten. Aktive konturer kan også brukes for å finne ikke-lukkede objekter som f.eks en kronglete vei på et kart. For å gjøre dette må spesielle faste kontrollpunkter settes som start- og endepunkter. Hvis ikke dette gjøres kommer den aktive konturen til å krympe inn i seg selv og forsvinne helt.

3.2 Morfologiske metoder

Matematisk morfologi brukes for å analysere form og utseende på objekter. Dette gjøres ved at bildet blir testet opp mot et såkalt strukturelement. De fleste operasjoner, inkludert formen på strukturelementet, bestemmes etter at man har informasjon om formen og gråverdiene på objektene i bildet. Vanligvis består dette strukturelementet av et vanlig nabolag av piksler. Figur 3.5 viser et typisk strukturelement.



Figur 3.5: Strukturelement

3.2.1 Erosjon og Dilasjon

Erosjonen ε med strukturelementet B på et sett X (objekter i et bilde), er definert som den delen av X som inkluderer B dersom sentrum av B er i et objekt x i X.

$$\varepsilon_B = \{x | B_x \subseteq X\}$$

Erosjon kan også forklares ved at man kjører strukturelementet rundt hvert objekt i settet hvor sentrum av B hele tiden ligger på ytterkanten av det enkelte objekt. De punktene i objektet som B dekker når det blir kjørt rundt objektet fjernes. Et strukturelement vil typisk være en 3×3 matrise, og vil da fjerne det ytterste piksellaget til et objekt. I dette tilfellet vil objekter mindre enn 3×3 piksler forsvinne helt. Dette er en metode som kan fjerne støy i et bilde, men man må være forsiktig med at de ønskede objektene ikke er for små.

Dilasjon δ er den duale operasjonen til erosjon. Med det menes det at en dilasjon av et sett er det samme som en erosjon av det inverterte settet. Den stiller spørsmålet: Treffer strukturelementet settet? Dilasjon med strukturelementet B på

settet X er definert som X i tillegg de punktene som ligger i B dersom B treffer et objekt x i X.

$$\delta_B = \{x|B_x \cap X \neq \emptyset\}$$

Det som skjer er at når du kjører B rundt objektene i X med sentrum av B på innerkanten av objektene, blir de punktene som B treffer rundt objektene lagt til i settet.

3.2.2 Åpning og lukking

Erosjon fjerner ikke bare alle objekter som er for små for strukturelementet, men i tillegg tynner operasjonen også ut de andre objektene. To viktige morfologiske operasjoner er åpning γ og lukking δ. Begge metodene jevner ut en kote, men der hvor åpning fjerner det som den ser som uvesentlige kanter, tetter lukking igjen hull mellom kotene. Siden en del av kotene i finerplaten ikke er fullstendig kontinuerlige, er dette metoder det er verdt å teste. Åpning er definert som erosjonen med strukturelementet B av et bilde f, etterfulgt av dilasjonen av det reflekterte strukturelementet $\check{B}$. Det reflekterte strukturelementet er definert som $\check{B} = w|w = -b$ for $b \epsilon B$. Det vil si, elementet er flippet over både x- og y-aksen.

$$\gamma_B(f) = \delta_{\check{B}}[\varepsilon_B(f)]$$

Åpning er definert som unionen av alle strukturelementer som passer i settet.

$$\gamma_B(X) = \bigcup_x \{x|B_x|B_X \subseteq X\}$$

Åpning fungerer slik at for hver piksel vil den laveste verdien innenfor et bestemt område gi ny verdi til pikselen. For lukking er det helt motsatt. Hver piksel tildeles den høyeste pikselverdien som finnes innenfor et bestemt område. Lukking er definert som punkter i struktureringselementet som hører til komplementet til lukkingen av settet. Lukkingen med B på bildet f er definert som dilasjonen med B på f, etterfulgt av erosjonen med det reflekterte strukturelementet $\check{B}$.

$$\phi_B(f) = \varepsilon_{\check{B}}[\delta_B(f)]$$

$$\phi_B(X) = \bigcap_x \{B_x^c|X \subseteq B_x^c\}$$

3.2.3 Tynning

Tynning går ut på å fjerne de forgrunnspikslene som ikke er nødvendige for at objektet skal holdes sammen og at det konvekse hullet rund objektet blir værende i samme størrelse. En vanlig måte å utføre tynning på er å fjerne alle forgrunnspiksler som ikke er ytterpunkt. Dersom dette gjøres får man skjelettet til objektet.

3.3 Terrengmodellering

Den vanligste teknikken for å representer en digital terrengmodell er å bruke punkter i 2,5D format som er bundet sammen med polygoner. Dette vil si at overflaten representeres ved hjelp av funksjonen

$$z = z(x, y)$$

En slik funksjon vil bare kunne gi en høydeverdi for hvert punkt. Dersom man er ute etter å modellere fjellhyller som prekestolen, eller f.eks bruer i terrenget, behøver

man uavhengige punkter og en modell som er 3D.

Den tradisjonelle måten for å representere et terreng er ut fra høydekoter. Som kjent fra tradisjonelle kart er stigninger og grad av bratthet i terrenget beskrevet ved hjelp av antall koter på tilsvarende del av kartet. Ved så å digitalisere kartet kan nødvendige x,y,z verdier lett finnes. Det finnes også muligheter for å få punktene direkte dersom radar blir brukt. Etter at punkter er funnet må modellen polygoniseres. De to mest vanlige teknikkene er et regulært gitter bestående av kvadratiske flater eller rettvinklede trekanter, eller et irregulær gitter bestående av trekanter med ulik form og størrelse.

For et regulært gitter legges polygoner over terrenget uavhengig om et hjørne er på et punkt eller ikke. Hjørneverdien blir satt som en funksjon av de nærliggende punktene. Slik kan antall polygoner og størrelsen på polygonene lett varieres. For et irregulært gitter legges vanligvis hjørnene mellom de eksisterende punktene. Siden det naturlig framstilles flere punkter i områder med mye høydeforandringer, vil det også her komme flere triangler. Dette er med på å gjøre en irregulær triangelmodell mer troverdig enn den regulære gittermodellen. Noe som kan oppstå som et problem for begge gittermulighetene er at polygonene er flate. Gold og Dakowicz [4] presenterer en metode for å bruke skjelettet mellom kotepunktene i tillegg til kotepunktene. Dette kan gi en jevnere overflate, uten at det behøver å koste mye mer.

Det største problemet med terrengmodeller er kompleksiteten. For store områder kan det være snakk om millioner av polygoner i tillegg til all tekstur som må lagres i teksturminnet. For å kunne kjøre en applikasjon i sanntid, må antall polygoner og oppløsningen på teksturen avgrenses for å få et bra resultat. Det som ofte gjøres er å ta hensyn til kameraets posisjon (hvor terrenget blir sett fra), og la oppløsning og antall polygoner avhenge av dette. Jo nærmere kamera desto høyere blir oppløsningen. For et regulært gitter endres bare størrelsen på polygonet, mens for et irregulært gitter må spesielle regler lages.

Terrengmodellering krever ofte my CPU kraft, da det innegår mange matematiske utregninger. For å lette trykket på CPUen kan mye av det grafiske håndteres av maskinens GPU(graphic processing unit), og CPU kraften kan brukes til andre ting. GPU er en enkelt-chip prosessor som brukes til transformasjoner og lyseffekter for hver nye frame av en 3D scene. Hoppe [21] beskriver en metode for terrengmodellering hvor en stor del av arbeidet er basert på GPU. Dette gjøres ved å lagre terreng geometrien som et sett av bilder istedenfor. Utførelsen de viser til er fra en interaktiv flyging over en terrengmodell av USA. Modellen har hele 20 milliarder(!) punkter, som lagres i bare 355 MB, og renderer 90 frames i sekundet.

Når terrenget er fremstillt i 3D må noen tilleggsparametere justeres for å få det til å se realistisk ut. Å legge på tekstur vil gjøre litt for å oppnå 3D effekt, men terrenget vil ikke se naturlig ut før belysning og skyggelegging blir brukt.

Som beskrevet over skiller man mellom 2,5D og 3D. Dette er noe som det vil bli sett bort fra. Siden modellen bygges opp på grunnlag av et kotekart er det tydelig at den bare vil være 2,5D. De fleste kilder og referanser ser bort fra dette, og skiller ikke mellom disse to begrepene. På grunn av dette vil det også her bli brukt uttrykket 3D modellering.

3.3.1 Finne punkter

For å bygge opp en 3D-modell behøver man punkter i rommet som et grunnlag for den endelige modellen. Det finnes flere måter å få tak i disse punktene på. Dersom man bygger en kunstig modell kan punktene være selvvalgte verdier. For en terrengmodell basert på data fra jordoverflaten (eller månen eller mars), kan modeller bygges opp fra punkter funnet av satellitter, og undervannsmodeller (f.eks for oljeboring) kan punktene finnes ved hjelp av radar. Når radar brukes blir to bilder av samme objekt tatt fra litt like vinkler. Høyden måles så ved å måle ulikhetene i refleksjon mellom de to bildene. Dersom man har et kotekart, som i dette tilfelle, kan punktene være fordelte stikkprøver fra kotene. Slike koter kan blant annet skapes ved at folk personlig bruker en høydemåler i fjellet, og merker av posisjon og høyde med jevne mellomrom.

Stikkpunkter fra kotene kan enkelt brukes i en irregulær modell, men hvordan skal kotene brukes til å finne punkter til en regulær modell? Det er implisitt at dette blir et interpolasjonsproblem. For hvert punkt i det regulære gitteret som ikke treffer direkte på en kote, må denne verdien regnes ut ved hjelp av interpolasjon av de nærliggende kotene. Også for irregulære modeller kan interpolasjon være nødvendig dersom men ønsker punkter mellom kotene for å gjøre modellen glattere. Gousie [19] beskriver to metoder for å approksimere høydedata fra koter til et gitter. Den første metoden går ut på å legge til ekstra koter mellom de allerede eksisterende kotene, og den andre baserer seg på gradientlinjer. Gradienter blir funnet i hver kote, og det legges så en Catmull-Rom spline over. Punkter kan da velges ut fra stikkpunkter av splinen.

3.3.2 Polygonisering

De fleste 3D scener er oppbygd av trianglestrips. En trianglestrip er er en rekke med sammenbundne trekanter. Siden trekantene henger sammen behøver ikke alle de tre kantene til hver trekant spesifiseres da en har kontroll på hvilke kanter som er felles for hvilke trekanter. Denne besparelsen for gjentakelse av kanter fører til effektiv bruk av minne og god kjøretid.

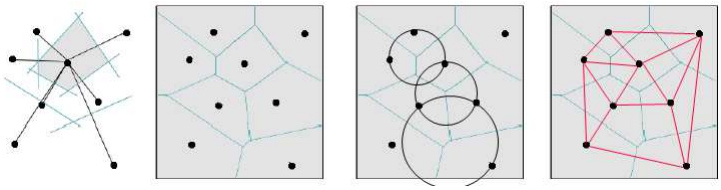
Det er store forskjeller på polygoniseringen av regulære punkter og irregulære punkter. For et regulært gitter vil alle trekantene være rettvinklet, og en vil til en hver tid vite hvilke tre punkter som danner en trekant. Dvs hvert punkt vil f.eks danne en trekant med punktet ovenfor og punktet til høyre. For et irregulært gitter trengs spesielle algoritmer for å finne hvilken triangulering som er den beste.

Delauney triangulering

En populær trianguleringsmetode er Delauney triangulering. Denne er mye brukt fordi den har veldig mange positive egenskaper. Den garanterer mest mulig equilaterale trekanter, en unik triangulering(med unntak av trivielle tilfeller som fire hjørnepunkter i et kvadrat), og den kan håndtere punktskyer både med varierende tetthet fordi den ikke krever et regulært gitter. Den kan garantere for equilaterale trekanter fordi den baserer seg på sirkelraduser, d.v.s. den minste sirkelen som får inn tre punkt. Dette vil hindre lange, tynne trekanter(så langt det er mulig), da disse resulterer i store sirkler. En annen måte å si dette på er at maksimum vinkel over hele trianguleringen blir minimert.

Algoritmen forklares enklest ved å ta utgangspunkt i Voronoy diagrammet. Voronoy diagrammet er dualen til Delauney trianguleringen, og begge kan lett omformes til den andre. Gitt en punktsky. For hvert punkt trekkes en linje til hver av

de andre punktene (innefor en forhåndsbestemt avstand). Når linjene er tegnet opp, merkes midtpunktet på hver av linjene. Gjennom midtpunktene til hver av linjene trekkes så en ny linje som er ortogonal med den første linjen. (se fig 3.6).



Figur 3.6: Fra punkter til trekanter

Rundt det bestemte punktet vil det da danne seg et polygon, et slikt polygon kalles en voronoycelle. Etter at det samme blir gjort med alle punktene sitter en igjen med et Voronoy diagram. En kjent analogi for Voronoy diagrammet er et område med mobiltelefonsendere. Se for deg at hvert punkt er en sender, og hver sender sender ut stråling med lik styrke, da vil hver voronoycelle være sendeområdet til den bestemte senderen.

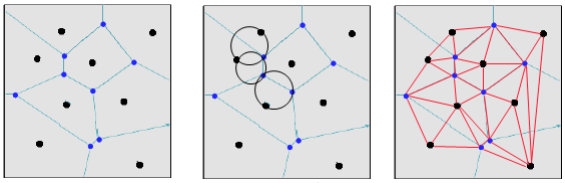
For å få trekanter ut av et voronoydiagram tas det utgangspunkt i hjørnene til cellene. Rundt hvert av hjørnene trekkes en sirkel med hjørnet som sentrum. Denne sirkelen gjøres større helt til den treffer 3 punkter. De 3 første punktene som er truffet danner en trekant. Når dette er gjort for alle cellehjørnene er resultatet en Delauney triangulering.

En variant av Delauney trianguleringen er en såkalt constrained Delauney triangulering. For en slik triangulering er prosessen den samme som for vanlig Delauney, bortsett fra at noen kanter er forhåndsbestemt. Dette er typisk ytterkanter, eller kanter i spesielle områder hvor tynne trekanter er ønskelig.

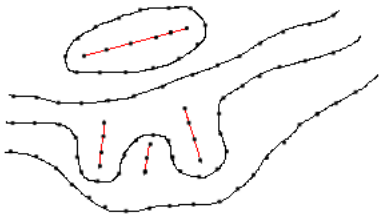
Når det er bare kotene som brukes som utgangspunkt for punktene vil områder hvor trianguleringer skjer mellom punkter på samme kote f.eks topper, bunner, eller kronglete områder, bli helt flate. For et fjell som ellers er kurvet, kan det være uønsket at toppen er helt flat, da en avrundet topp kan gi et bedre visuelt resultat. Gold og Dakowicz [4] beskriver en del måter dette kan gjøres på. Ideen ligger i å utnytte forholdet mellom Voronoi diagrammet og Delauney trianguleringen. En algoritme går ut på å sette inn alle hjørnene til voronoycellene inn i diagrammet (se fig 3.7). På denne måten vil det danne seg midtlinjer mellom kotene, også i kroker mellom samme kote. Å gjøre dette blir det samme som å legge til skjelettet til kotene. Etter dette er gjort kommer spørsmålet om hvilken høyde disse skjelettpunktene skal få. Et forslag er å sette høyden som en funksjon av avstanden til kotekurven rundt. Dersom skjelettet ligger mellom koter av ulik høyde må alle høyder tas hensyn til. Det finnes en annen metode som sier at alle bakker skal være like bratte. Dette er en framgangsmåte som krever mer programmering, da det hele tiden må regnes ut forholdet for største avstand mellom kotene og de andre avstandene for å finne hvert enkelt stigningstall. For å gjøre bakkene enda mykere kan det være aktuelt å bruke noe jevning helt til slutt.

Hvis en bare er ute etter å hindre flate polygoner, er det nok å ta med hjørnene til voronoycellene/skjelettet som ligger mellom kotene av samme høyde (se figur 3.8).

Det er ikke bare det å unngå flate trekanter på fjelltopper og i dalbunner som



Figur 3.7: Voronoy punkter er også tatt med

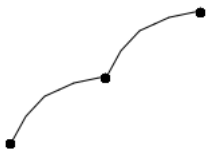


Figur 3.8: Koter med utvalgte voronoypunkter.

er tema, det er også viktig å se for seg at et landskap vanligvis vil ha runde former, og det vil se unaturlig og terrassert ut dersom flate trekanter trekkes kun mellom kotene.

Det fins en betegnelse på hvor glatt en overgang mellom to flater eller linjer er. Det vi har nå, både med punkter kun på kotene, men også til en viss grad dersom punkter kun blir lagt på medialaksen er en overflate som kun er C^0 kontinuerlig. C^0 kontinuerlig betyr at to segmenter treffer hverandre i et punkt, men setter ingen krav til hvordan dette treffpunktet skal se ut. For å få en glattere overflate er det viktig at modellen er C^1 kontinuerlig. Da vil tangentvektorene til de to segmentene som kyttes sammen ha samme størrelse og retning. Vi kan også gå ut fra at landskapet hvor kotene er hentet fra er C^1 kontinuerlige.

Hormann [18] beskriver en metode for å interpolere koter som tar hensyn til topper og bunner. Denne metoden hindrer at uønskede oscillasjoner oppstår. Dette var et problem i alle metodene beskrevet i [4]. Oscillasjonene oppstår fordi det for alle punkter som blir valgt på medialaksen blir det kun tatt hensyn til høyden til de nærliggende kotene og ikke hellingen(se figur 3.9).



Figur 3.9: Uønskede oscillasjoner.

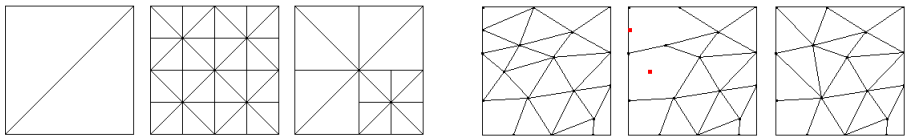
Metoden i [18] går ut på at punkter mellom koter blir interpolert med de nærliggende ved hjelp av hermite interpolasjon. I tillegg blir både hellingen på kotene og i områdene mellom kotene tilpasset for å unngå oscillasjoner.

3.3.3 Level of Detail

En stor terrengmodell kan lett komme opp i millioner av trekanter. For å kunne bevege seg over eller i modellen må dette tallet begrenses for å få en akseptabel frame rate. Det er to observasjoner som kan utnyttes. Den første er at dersom man er i modellen er det for det meste av tiden noe som er utenfor synsvinkelen. Dette kan være terreng som er bak deg eller terreng som er langt borte, da særlig ut til sidene. Den andre observasjonen er at detaljer ikke er synlige på avstand. Disse observasjonene åpner for muligheter til å redusere antall trekanter, og å ha en dynamisk triangulering som endres automatisk etter øyeposisjonen til tilskueren. Ettersom ønsket om sanntid i spill og landskapsmodeller er stor, samtidig som dette ikke skal gå på bekostning av oppløsning og kvalitet, er Level of Detail et konsept det er skrevet mye om i senere tid.

For å starte utviklingen mot en dynamisk triangulering må en først avgjøre om gitteret er regulært eller irregulært. I utgangspunktet vil et irregulært gitter gi best resultat, og færre trekanter, ettersom det kun blir lagt polygoner mellom viktige punkter. Problemene kommer først når man har behov for å variere nøyaktigheten. For et regulært nettverk vil utgangpunktet være dårligere, men variasjon i gitteroppløsning er mye enklere.

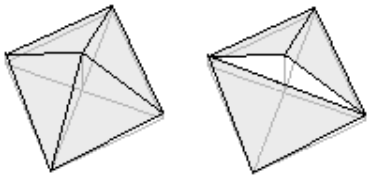
For et regulært gitter er det enkelt å modifisere antall trekanter. Venstre del av figur 3.10 viser hvordan. For maksimalt med detaljer velges punkter med minste avstand mellom. For færre detaljer dobles eller mangedobles avstanden mellom punktene. For et irregulært gitter er operasjonen ikke så enkel. For å triangulere brukes Delauney triangulering, og det fins ingen regel på at man bare kan flytte punkter. Nettverket kan bygges opp som en trestruktur slik som det regulære nettet, men dette vil gi veldig tynne og ugunstige trekanter. Berg og Dobrindt [4] prensenterer en alternativ metode. Startpunktet er et gitter med høy nøyaktighet. For å minske nøyaktigheten i enkelte områder fjernes punkter, og henholdsvis de tilhørende kantene (se høyre del av figur 3.10). Etter at punktet er fjernet benyttes Delauney triangulering for å trianguleret det området på nytt. Ettersom Delauney trianguleringen garanterer for mest mulig equiliterale trekanter og minst mulig areal, vil det ikke være nødvendig å retriangulere resten av området. For å hindre at viktige punkt fjernes, får hvert punkt en viktighetsgrad, dette hindrer popping, noe som diskuteres senere.



Figur 3.10: Level of Detail

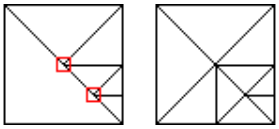
Å bruke LOD på et regulært gitter er enkelt nok i seg selv, men kan skape enkelte problemer i gitteret. Det kan oppstå hull, eller artifakter kan plutselig forsvinne eller dukke opp, såkalt popping. Hullene kan oppstå i overgangene mellom ulike deler av nøyaktighet (se figur 3.11).

Hullene oppstår ved at det punktet som fjernes fra en oppløsning skiller mye i høyde i forhold til nabopunktene. Lindstom [13] benytter seg her av en deltaverdi. Deltaverdien til et punkt er høydeforskjellen til dette punktet i forhold til de



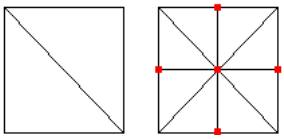
Figur 3.11: Hull

nabopunktene som ligger på samme linje. Et punkt kan kun fjernes fra en eller flere trekanter dersom deltaverdien er mindre enn en gitt verdi θ. En annen måte å hindre hull på er beskrevet av Röttger [15]. Ingen hull vil oppstå dersom ingen T-kryss er tillatt, og nabotrekanter maksimalt kan skille en grad av oppløsning. Til venstre på Figur 3.12 er det merket av hvilke steder hull kan oppstå. Figuren til venstre er modifisert etter at alle T-kryss er fjernet. For å løse dette problemet, må en rekursiv algoritme sjekke alle trekanter rundt den som modifiseres for å sjekke om noen av disse også må modifiseres.



Figur 3.12: T-kryss

Noe annet som kan skje når man beveger seg fra høy- til lav oppløsning og omvendt, er at små detaljer plutselig popper frem. Dette vil typisk være tynne, høye ting, som spisse fjell eller høyt gress. Dersom utbredelsen av artefaktet i x-y planet er så liten at den ikke kommer frem i trianguleringer med lav oppløsning, vil den plutselig komme fram når gitteret blir fint nok. Dette vil se svært merkelig og unaturlig ut. For å unngå popping kan geomorfing brukes. Geomorfing kan forklares som en jevn synlig overgang mellom to geometriske gitter av ulik oppløsning. Hver gang det er endring fra en oppløsning til en annen endres høydene til de nye punktene gradvis istedenfor å poppe fram med en gang. En slik gradvis overgang kalles morfing. Det kan være mulig å se morfingen foregå ved at hjørner beveger seg, men det vil ikke oppleves like forstyrrende som popping. Metoden blir forklart av Röttger [15]. Et holdepunkt man har er at dersom ny popping skjer, vil det kun skje i løvnodene til den aktuelle trianguleringen, og det vil for hver løvnode være snakk om maksimalt 5 nye hjørner. Fig 3.13 viser hvilke nye hjørner som kan oppstå i endring av oppløsning. Men hvordan fungerer morfingen? Vi tar utgangspunkt i



Figur 3.13: Geomorfing

en blandingsfaktor. Dersom blandingsfaktoren ligger innenfor intervallet 0-1 vil 0 si at høyden kun er basert på interpolasjon mellom andre høyder, mens 1 sier at høyden er den virkelige, d.v.s. den som er funnet i samplingen. Blandingsfaktoren

beskrives som:

$$blandingsfaktor = 2 * (1 - f)$$

f er en bestemmelsesvaribel som angir om en blokk kan bli subdividert. En blokk(en trianglestrip) kan bli subdividert dersom f<1.

$$f = \frac{l}{d * C * max(c * d2, 1)}$$

l= avstand til øyet
d= kantlengden av blokken
C= en konstant
c= ønsket global oppløsning
d2= øvre grense for error som kommer dersom en går ned en level av detail

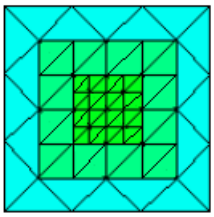
Culling

Det er ikke bare detaljnivået som kan være med å øke frameraten. Som nevnt i begynnelsen av seksjonen vil det ofte være deler av terrenget som ligger utenfor synsvinkelen. Dette impliserer at det er deler av terrenget som ikke behøves å bli renderet. Hovedrutinen for å gjøre dette er frustrumculling. Frustrumculling fjerner de flatene som ligger utenfor synsfrustrumet. Også flater som peker port fra tilskueren kan fjernes. Denne cullingvarianten kalles bakside-culling. Denne varianten virker bare på konvekse flater. Anta en kube hvor en flate er fjernet slik at man kan se inn i kuben. Da ville culling av bortpekende flater fjernet baksiden av kuben, og vi ville sett rett gjennom.

Wagner [20] beskriver en optimaliserings metode som fjerner ikke-synlige flater som ikke blir fjernet bare med culling. Dette er typisk flater som vender mot øyeposisjon, men som er skjult bak andre flater, f.eks et fjell. Metoden kalles PVS (Potentially Visible Sets). PVS bestemmer i run-time hvilke deler av terrenget som er synlige, og hvilke som er skjult. Utregningene blir gjort offline når terrenget er designet, så kostnadene i sanntid er så og si null. Metoden fungerer som følger. Terrenget blir først delt opp i ruter f.eks 16*16. For hver ruter blir det sjekket for flere ulike høyder hvilke av de andre rutene som er synlige. For 16*16 ruter og 32 høyder for hver rute får vi 8192 PVS celler. Hver celle må inneholde hvilke av rutene som er synlige. Hvilke ruter som er synlige blir funnet med en LOS(line of sight) algoritme. En linje blir dratt mellom øyepunkt og en annen rute. Dersom ruten blir truffet uten at linjen har truffet noe annet på veien vil ruten være synlig. Dersom noe er truffet vil den være ikke-synlig. Når programmet kjører vil det for hver frame bli slått opp i PVS listen, og stoppe renderingen av de rutene som er satt til ikke-synlig.

GPU og LOD

De LOD metodene som er nevnt til nå benytter seg alle av polygongitter som modifiseres under kjøring. Losasso [22] beskriver en alternativ fremgangsmåte som baserer seg på geometri clipmap. En geometri clipmap lagrer en terreng pyramide(se figur 3.14) til hurtigbufferen. Pyramiden har m nivå, hvor hvert nivå n har oppløsning $(n-1)^2$. Kun på det fineste nivået blir renderingen gjort som en komplett gitter firkant. For andre nivå er høyere oppløsning lagt inn i rammer av lavere oppløsning. Ettersom tilskueren flytter på seg, flytter også clipmapene på seg, og oppdateres med ny data. Geometri og teksturer blir morfet for å interpolere ulike nivåer av oppløsning. En videreføring av metoden beskrives i [21]. Som nevnt i begynnelsen av seksjonen kan også GPUen utnyttes, og det er det som gjøres her. I [22] blir



Figur 3.14: Mipmap pyramide

polygonhjørnene lagret i en buffer, men GPUen kan ikke håndtere slike. Hjørnene blir her håndtert på en litt annen måte. GPU kan arbeide med teksturer, og med utgangspunkt i dette lagres z koordinatene i høydekart, og (x,y) koordinatene som konstant hjørnedata. Høydedata som bilder tillater direkte prosessering ved bruk av GPUen sin rasterisering pipeline. Milliarder av hjørner kanrepresenteres, og interaktiv flygning kan gjøres i sanntid. Plutselig er ikke antall polygoner et hinder lenger.

3.3.4 Teksturmapping

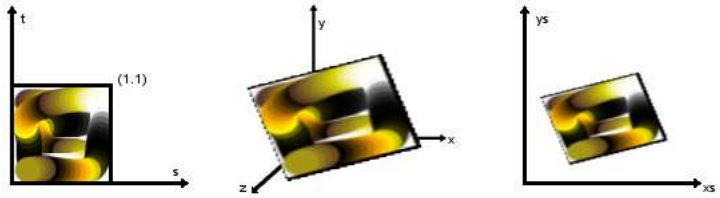
Bruk av mapping gir realisme til datagenererte objekter. En kan skimte formene i terrenget etter trianguleringen, men det vil fortsatt ikke se ut som et landskap, eller woodscape i dette tilfellet. Ved å mappe tekstur på et objekt, kan objektet få høy synlig kompleksitet uten at det nødvendigvis er et så veldig høyt antall polygoner.

Når mapping brukes for å legge på tekstur(i motsetning til å farge hver eneste piksel) trengs bare teksturen, og start og slutt posisjonene til der teksturen skal mappes. Her spares det arbeid da en ikke trenger å dele teksturen opp i riktige polygoner, men tar hensyn til de nødvendige polygonene med hensyn til trianguleringen.

En tekstur kan både være 2D og 3D. En 2D tekstur en vanligvis et bilde man har fra før, og kan være alt fra et fotografi av et menneske til et bilde av en finerplate. En 3D tekstur kan ses på som en kloss med innhold, f.eks en murstein eller en svamp. Forskjellen mellom de to typene kan forklares ved at 2D teksturen må males på objektet, mens objektet må hugges ut av 3D teksturen.

Det er tre ulike koordinatsystem som det må tas hensyn til for å mappe en tekstur på et objekt. Selve teksturen blir lagret i et eget koordinatsystem i området [0,1) for begge(tre akser for 3D tekstur) akser uavhengig av størrelsen på teksturen. I tillegg har objektet et eget 3D koordinatsystem, ofte kalt verdenskoordinater, bestående av aksene x, y og z, og skjermen har et 2D koordinatsystem bestående av aksene x_s og y_s. Det er viktig at teksturen blir lagt på objektkoordinatene, og ikke på skjermkoordinatene. Dersom teksturen blir koblet mot skjermkoordinatene vil teksturen være konstant selv om objektet snurres, d.v.s. polygonene i objektet vil få ny tekstur for hver bevegelse. De ulike koordinatsystemene vises i figur 3.15.

For en sylinder eller et flatt objekt er det enkelt å angi start- og stopposisjoner for teksturen på objektet, og mapping vil være en enkel sak. Verre blir det for lukkede objekter som en kube eller kule. Et kjent problem er det å konstruere kart fra jordoverflaten. Alle kart fra store områder av jordoverflaten vil ha feilkilder ved at enten blir områder på midten forminsket eller områder i kantene forstørret. Mapping fra en flate til en kule, blir det motsatte, men kartproblemet illustrerer tydelig at det er umulig med perfekt mapping mellom en firkant og en kule.



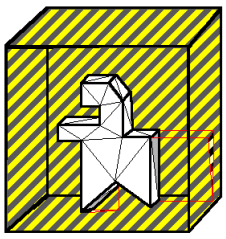
Figur 3.15: Tekstur mapping

For å få en så jevn og sammenhengende mapping som mulig kan denne mappingen gjøres parametrisk med koordinatene u og v. En mulig mapping fra parameterrommet til en kule er [1]:

$$x = r\cos(2\pi u), y = r\sin(2\pi u)\cos(2\pi v), z = r\sin(2\pi u)\sin(2\pi v)$$

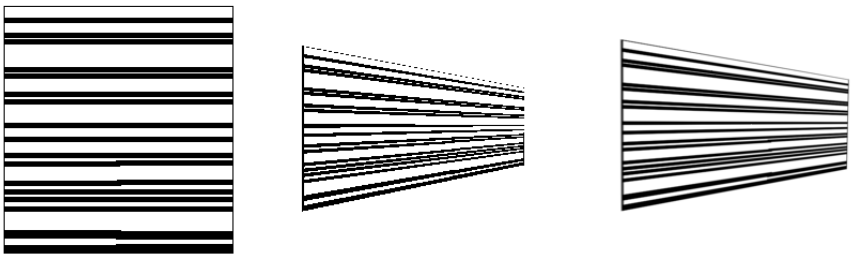
Hvor r er radiusen til kulen. Da teksturen allerede er definert innenfor rommet [0,1), kan s og t brukes i steden for u og v.

Dersom teksturen skal mappes på et komplekst 3D objekt, er det ikke nok å mappe parametrisk, men det kan også være behov for en mellomstasjon. Denne mellomstasjonen vil være å først angi teksturparameterne som funksjon av en kjent geometrisk figur, f.eks en kule eller sylinder. Dette behøves dersom objektet som skal tekstureres har en vanskelig form, f.eks en tekanne eller et glass. Det finnes flere ulike måter for så å mappe tekstur fra en omsluttende geometrisk figur til objektet inni. En mulighet er å bruke normalene fra mellomstasjonen. De steder hvor normalene treffer på objektet gis med normalens tilhørende tekstur. Alternativt kan normalene til objektet brukes, og her vil den pikselen normalen treffer brukes. En tredje metode er å bruke midtpunktet i objektet, og skyte ut normaler herfra. Vanligvis velges normalen i hjørnene til hver polygon. Når hvert hjørne i et polygon har funnet sin piksel, mappes området i mellom disse pikslene i teksturen på trekanten. Figur 3.16 viser et eksempel hvor normalene skytes ut fra mellomstasjonen.



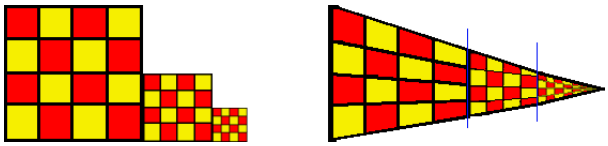
Figur 3.16: Mapping fra mellomstasjon

En fare med å bruke en tekstur som ikke har akkurat samme form og størrelse som objektet, er at noe som heter aliasing oppstår. Aliasing kommer av at teksturen er for stor for et gitt område av objektet, og de pikslene som blir valgt fra teksturen for dette området, ikke nødvendigvis er de som gir best resultat. Antialiasing kan løse dette problemet. Metoden velger gjennomsnittsfargen til de aktuelle pikslene istedenfor fargen til en bestemt piksel. Figur 3.17 viser en tekstur og to ulike mappinger av den. Trekanten helt til høyre er gjort med antialiasing. Den gir et finere helhetsinntrykk, men en av ulempene med antialiasing er at teksturen blir noe utjevnet (blurret).



Figur 3.17: Antialiasing

Å finne gjennomsnittsfargen for alle pikslene i sanntid kan være unødvendig arbeidskrevende. En annen metode er å bruke MIP-mapping. MIP er forkortelsen for "multum in parvo", som betyr "mye på et lite sted"[17]. MIP refererer til lagring av multiple versjoner med ulik oppløsning av samme tekstur Det som er idéen er at flere varianter av samme tekstur i ulik oppløsning lagres, slik at hver av disse kan brukes når det passer på modellen. I tilfeller hvor objektet er mindre enn teksturen, kan en benytte en tekstur med lavere oppløsning, og dermed unngå aliasing. Dette blir gyldig da en husker at aliasing bare skjer dersom en stor tekstur mappes på et lite objekt.



Figur 3.18: Mipmapping av en trekant

Et spørsmål som stiller seg er hvordan man kan vite hvilken tekstur som skal brukes i hvilket tilfelle. Noe grafikk hardware begrenser teksturer til å være kvadrater med 2^n piksler på hver side. På grunn av dette blir det naturlig å tenke seg at alle teksturer må være i dette formatet. Blow [16] setter opp en formel for å finne hvor stor en tekstur må være for å kunne brukes på et gitt polygon. Det blir her gått ut fra at polygonet er rettvinklet. Den teksturen som behøves har størrelse $2^n \times 2^n$ hvor $n = \lceil \frac{1}{2} log_2(r) \rceil$.
r=km.
k=antall piksler som polygonet okkuperer.
1/m= arealet av teksturen som polygonet dekker(m=1 sier at polygonet og teksturen er samme størrelse).

3.3.5 Lyssetting

Tekstur kan i seg selv få et objekt til å se veldig virkelighetsnært ut. Men for å øke troverdigheten enda mer, er det nødvendig med lyskilder som lyser opp utstående områder, og gjør andre områder skyggebelagt. Ray tracing er en måte å gjøre dette på. Du starter i øyepunktet, og skyter ut en stråle gjennom hver piksel i projeksjonsvinduet. Dersom et objekt treffes før solen, vil den delen av objektet være skyggebelagt. Sider som ikke blir truffet vil enten være totalt skyggebelagt eller ligge i høylys. Dette kan lett avgjøres ved å se om normalvektoren ligger fra

eller mot solen. Wagner[20] beskriver en måte å finne normalvektorene på. Utgangspunktet er to 3D-vektorer, hvor hver av dem peker fra den aktuelle høyden til høydeposisjonene til naboene. Linjen som binder de tre punktene sammen må være parallell med solstrålene. Normalen til det aktuelle punktet er kryssproduktet av de to 3D-vektorene. Det fins også en fysikkbasert modell kalt Radiositets ligningen. Her deles objektene opp i flatelapper, og hvor mye lys som blir tiltrukket, sendt ut og reflektert mellom dem blir målt.

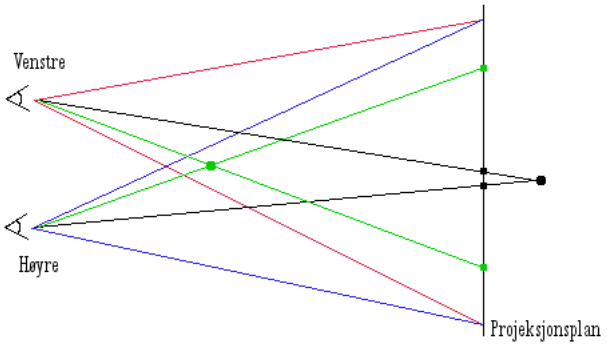
En mye billigere metode er å lage skyggekart/lyskart. Dette er en måte å lage teksturen med skygge og lys klar før modellering. På denne måten slipper man å sjekke for nye skygger for hver nye Level of Detail. Dette er aktuelt fordi for hvert tap av hjørner, vil det være et tilsvarende tap av normaler. Tap av normaler kan føre til at de gjenværende normalene får endret lysverdi. Slike endringer i lysforhold kan lett merkes av tilskueren. Ulempen med slike forhåndsutregnede lyskart er at lysposisjonen er forhåndsbestemt, og kan ikke endres interaktivt i modellen. Marghidanu [14] beskriver en algoritme for å lage lyskart. Utgangspunktet er et høydekart.Et høydekart er et kontinuerlig bitmap hvor hver farge/gråtone representerer en høydeposisjon. Fra solhøyden trekkes en linje til hvert punkt i høydekartet. For det aktuelle punktet sjekkes alle høyder mellom solen og punktet som ligger under linjen. Dersom noen av høydene er høyere enn høyden på linjen, vet vi at det aktuelle punktet vil være skyggelagt. For å få en jevn overgang mellom der det er skygge og hvor det er lyst, kan en ta hensyn til høydeforskjellen mellom den høyeste høyden og linjen. Dersom denne er liten, vil det være svak skygge, men dersom den er stor vil skyggen være mørk.

3.4 Stereoskopisk projeksjon

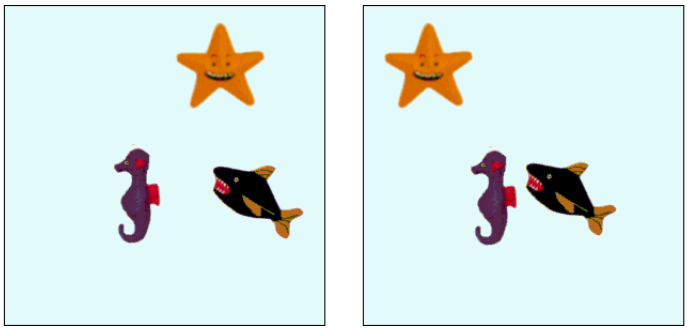
Mennesker oppfatter et objekt til å ha dybde ved å sammenligne hva det høyre og venstre øyet ser. Man kan også oppfatte dybde ved hjelp av ett øye, men da spiller bevegelsen av objektene en stor rolle. Dersom en bil kjører foran en bygning, vil man selv med ett øye oppfatte bilen som nærmere enn bygningen. For å se dybde i omgivelser uten å bevege hodet eller at objektene beveger ser, er det nødvendig med stereosyn. Definisjonen på stereoskopi er å det å lage kunstige bilder med dybde ved hjelp av to flate bilder.

Hjernen bruker forskjellene mellom bildet til hvert øye for å avgjøre hvor langt borte et objekt er. Jo større forskjell mellom et objekt sett av venstre øye, og det samme objektet sett av høyre øye, desto nærmere er objektet. Hjernen benytter disse to bildene, og avstanden mellom dem til å konvergerer dem til et 3D bilde. Figur 3.19 illustrerer den fysiske modellen. Arealet innenfor den røde trekanten er hva det venstre øyet ser, og arealet innenfor den blå trekanten er hva det høyre øyet ser. Projeksjonsplanet tegnes opp i en gitt avstand fra øyet eller kameraet. Denne avstanden kalles fokallengde, eller avstanden til null-parallaksen. Betydningen av null-parallakse er at objektet som skal avbildes ligger på projeksjonsplanet. I dette punktet vil bildet sett med venstre, og bildet sett med høyre øye være det samme. Det grønne punktet i figuren ligger i den negative parallaksen, mens det svarte punktet ligger i positiv parallakse. For et punkt på negativ parallakse vil punktet sett fra venstre øyet ligge til høyre for punktet sett fra det høyre øyet.

Figur 3.20 viser et stereopar. Alle objektene ligger foran projeksjonsplanet. Her er forskjellen mellom posisjonene til sjøstjernen størst, så denne er nærmest. Mens havhesten ligger på samme posisjon i begge bilder, og vil være helt i bakgrunnen. To objekter må ligge veldig nærme øyet for å få så store forskjeller som man ser på



Figur 3.19: Fysisk modell



Figur 3.20: Stereopar (figurer funnet på http://www.flibberkids.com)

sjøstjernen. Eksemplet er først og fremst for å demonstrere ekstremtilfeller.

Det finnes mange ulike måter å presentere stereoparene på. Den enkleste måten er å ha et bildepar foran seg. Ved å stille seg så nærme bildene at bildene konvergerer, vil et 3D bilde oppstå. Hvor god den enkelte person er til å styre øyefokuseringen er avgjørende for resultatet.

Det finnes måter som ikke krever øyeanstrengelser i form av ulike fokuseringer. Alle disse tar i bruk ulike varianter av briller. Den mest kjente og enkleste måten er å bruke fargede brilleglass og et anaglyfbilde. Et anaglyfbilde er et bilde hvor bildene i et stereopar gis hver sin farge før de limes oppå hverandre. Vanligvis er det ene brilleglasset rødt, og det ande cyan. Ved å trekke ut den røde komponenten i det ene bildet og den cyan komponenten i det andre bildet, vil hvert øye se ulike bilder. Sett gjennom det røde brilleglasset vil den røde komponenten bli fjernet, og sett gjennom det cyan-fargede brilleglasset vil den cyan-fargede komponenten bli fjernet(Se figur 3.21).

En annen spennende metode baserer seg på Pulfrich effekten. Metoden blir beskrevet i [11]. Dette er en metode som bare fungerer på objekter som beveger seg. For å skape 3D effekt må det ene av to brilleglass dekkes med et mørkt men gjennomsiktig filter. Teorien dette grunner ut i er at ved lavere lysintensitet bruker øyet lengre tid på å oppfatte objektene. Øyet som ser gjennom den mørke linsen vil derfor oppfatte objektene noen frames senere enn det andre øyet. Det er viktig at riktig øye dekkes av den mørke filmen i forhold til hvordan objektene beveger seg.

Figur 3.21: Rød-cyan stereoskopi

Den metoden som er mest aktuell for dette prosjektet er en metode med polarisert lys. Lys kan bli beskrevet som en bølge som vibrerer både i vertikalt og horisontalt plan. Polariserte lysbølger er lysbølger som kun vibrerer i et eneste plan. For å få polariserte lysbølger trengs to filter, et i vertikal og et i horisontal retning. Det vertikale stenger ute de horisontale lysbølgene og omvendt. Dersom både et horisontalt og vertikalt filter blir brukt, stenges alt lys ute. For å skape 3D effekt av dette fenomenet trengs to prosjektører med hvert sitt filter. Dersom to versjoner en film lages, med ørlite forskyvning, og vises med disse prosjektørene, vil briller med tilsvarende filter skape 3D effekt.

Av avanserte metoder finnes det elektroniske briller med LCD-glass. På en monitor vises et bilde eller film som bytter mellom å vise bilder sett fra høyre øye og bilder sett fra venstre øye. Linsene i brillene mørklegges annenhver gang alt etter hvilket bilde som vises på monitoren.

Stereoskopi kan brukes til å fenge og imponere et publikum. Blant annet kan man finne filmen 'Honey, I shrunk the Audience' i Euro Disney, Paris. Hver enkelt publikum får et par polariserte briller, og blir tatt med inn i filmen. Elementer som slanger som kommer mot deg, og et romskip som flyr rundt i salen, ser svært ekte ut ved hjelp av polarisert lys. For at publikum også skal kunne føle 'Woodscape' på kroppen, vil stereoskopisk projeksjon stå fram som en viktig del av kunstutstillingen.

Kapittel 4

Implementasjon

Til implementasjonen brukes Microsoft Visual Studio. Programmeringsspråket er C++, og OpenGL blir brukt til grafikken. Biblioteket SIM Scenery ble brukt i deler av oppgaven. Det programmet som er implementert ved hjelp av SIM Scenery blir referert til som det SIM baserte programmet. Det egenproduserte programmet kalles 3Dtre.

4.1 Segmentering

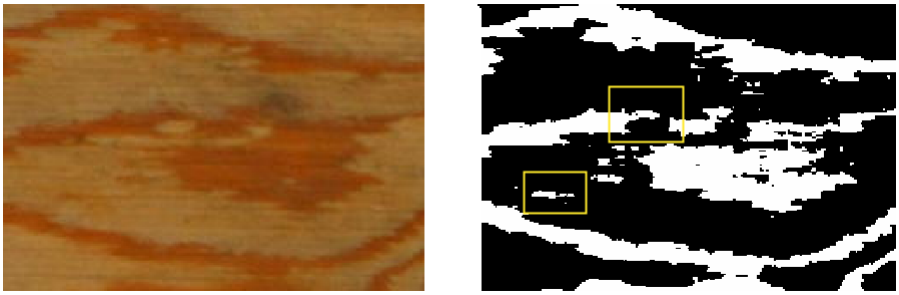
4.1.1 Terskling

Noe av det som gjør denne finerplaten interessant, og spennende nok til at den kan brukes som kunst er det tydelige mønsteret som preger mesteparten av platen. Dette åpner for gode muligheter for terskling. Ulike varianter ble utprøvd. Selv om mønsteret er tydelig overalt, er det mange ulike grader av lysintensitet i ulike områder på platen. Som et eksempel er midten av platen mye lysere enn kantene. Med bakgrunn i dette sier det seg selv at det vil være nødvendig med flere enn en terskelverdi for å segmentere ut linjene. Etter nøye gransking av fargeverdiene på ulike steder av platen, viste det seg at den røde komponenten var så og si uendret over hele platen, mens de grønne og blå komponentene endret seg med mønsteret. For å dra utbytte av dette i tersklingen, ble kun de blå og grønne komponentene lagt vekt på når platen ble omgjort til gråtoner.

Platen blir tersklet med adaptiv terskling med et rutenett på 25×25 ruter. Terskelen ble satt til gjennomsnittlig gråverdi + 10. Dette ble valgt etter mye testing. Resultatet ble for de fleste områder ganske bra, og det meste av mønsteret som synlig skiller seg fra bakgrunnen ble segmentert ut (se A.2. Siden det er bare gråverdien som bestemmer om et piksel skal være forgrunn eller bakgrunn blir det likevel en del støy rundt i bildet. Som det er synlig i figur 4.1 er noen kanter for lyse og har blitt fjernet, mens noe støy er for mørk og har blitt tatt med.

4.1.2 Kantdeteksjon

Mønsteret i treet er for det meste horisontalt og sirkulært. For å få med alle kanter fra begge sider, blir det da aktuelt med enten en horisontal og en vertikal kantmaske, eller 2 skrå kantmasker (nord_vest-sør_øst og nord_øst-sør_vest). Overlappingen mellom disse to forslagene er såpass stor at det vil ikke være noe hensikt i å bruke flere enn til sammen to kantmasker. Det ble først prøvd med vertikale og horisontale kantmasker. Resultatet ble ikke så veldig bra. Grunnen til dette er at de fleste kantene som ser horisontale ut ikke er helt horisontale, og det som faktisk var helt



Figur 4.1: Problemer med terskling

horisontalt var linjer i treet som ikke skulle regnes med i kotekartet. Figur 4.2 viser disse horisontale linjene.

Figur 4.2: Horisontale linjer til problem

Løsningen ble da å bruke skrå kantmasker.

Et spørsmål som stilte seg var hvilken bildeoppløsning som skulle brukes for kantdeteksjonen. Som utgangspunkt var det to oppløsinger å velge mellom. Det ene var et bilde på 3150×2450 piksler, mens det andre var et bilde sammensatt av fire tilsvarende store bilder. For den beste oppløsningen var det lettere å trekke ut små detaljer, blant annet ringer rundt kvistene, men det ble tilsvarende trukket ut like mye som ikke skulle være med. Det ble prøvd med noe jevning i vertikal retning for å fjerne de horisontale linjene, men da ble også korrekte kanter jevnet ut. For den dårligste oppløsningen ble helhetsinntrykket bedre, samtidig som at prosesseringstiden var mye kortere. Selv om ekstra detaljer kom med for den beste oppløsningen, var det såpass mye som ikke ble registrert her heller, f.eks de innerste ringene rundt enkelte kvister, slik at poenget med å bruke den forsvant.

Konklusjonen ble å bruke dårligst oppløsning og skrå kantmasker. For å få tynnere kanter, og forhindre at ikke-tilhørende kanter knyttet seg sammen, ble kantmaskene kjørt to ganger hver før resultatet fra hver enkel maske ble lagt sammen.

Etter at kantmaskene er gjort må bildene terskles slik at bare høye verdier blir valgt. Resultatene fra terskling- kantdeteksjon, og kantdeteksjon-terskling ble ganske like. Den metoden som ble valgt for de endelige kotene var kantdeteksjon først og så terskling.

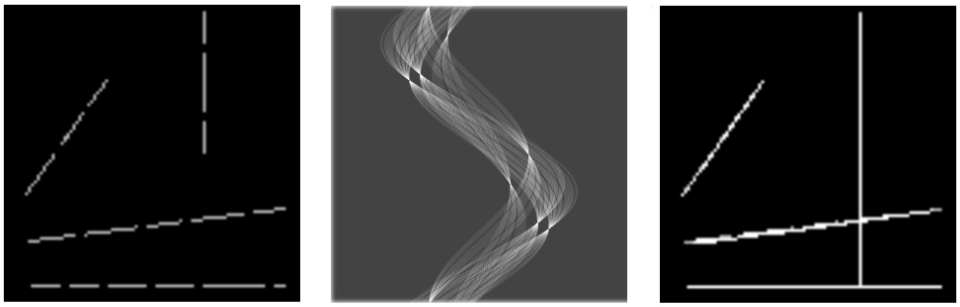
Årsaken til at kantdeteksjon måtte bli gjort i tillegg til terskling er for å få ut begge sider av et bånd. I tillegg ble et medianfilter brukt for å fjerne pikselvis

støy. Medianfilteret ble tilpasset til å bare fjerne frittstående punkter. For å kunne bruke kotene som et høydekart må tilhørende koter henge sammen. Resultatet fra kantdeteksjonen har flere manglende kanterdeler og hull.

4.1.3 Hough transformasjon

For å kunne benytte kantene i treverket som høydekoter må de være kontinuerlige, noe den segmenteringen som har blitt gjort ikke har greid fullstendig. Problemet ligger særlig rundt kvistene, hvor årringene ligger veldig tett, og store deler av dem forsvinner i ett med bakgrunnen. For dette problemet er det lite Hough transformasjonen kan bidra med. Den kan bare finne koter beskrevet matematisk, og årringene rund kvistene er langt fra perfekte sirkler. Det syntes likevel mulig å bruke Hough transformasjonen i andre problemområder. Til høyre på finerplaten finnes det en hel del skrå linjer som er noenlunde rette, men har mange hull. Det synes å være verdt et forsøk.

Det blir først prøvd med kunstige data(se figur 4.3). Metoden gir riktig resultat med unntak av en forlengelse av den vertikale linjen. Denne forlengelsen er ikke en feil i henhold til algoritmen, da det nederste punktet faktisk er på linje med resten av de vertikale punktstykkene, men det er noe annet enn hva som kanskje kunne se naturlig ut.

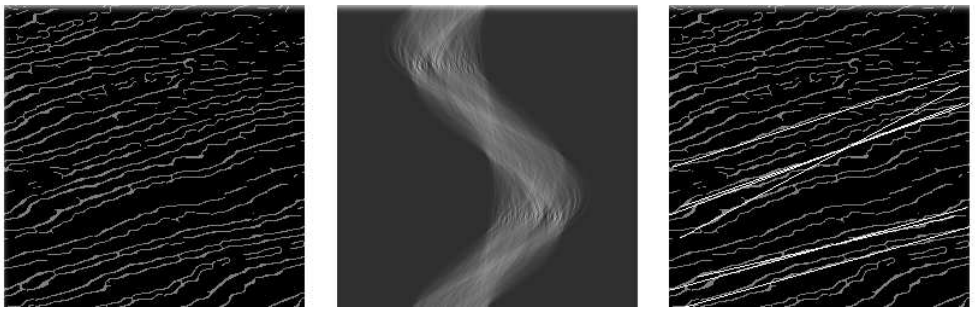


Figur 4.3: Hough transformasjon utført på testbilde

Metoden har så blitt testet på en del av det stykket som ble nevnt tidligere. Resultatet kan sees i figur 4.4. Det er her blitt brukt 250×250 akkumulatorceller, og de 10 prosent beste cellene har blitt valgt ut. Innenfor hver av de utvalgte cellene testes så forholdet mellom de tilhørende treffpunktene.

Det høye antallet akkumulatorceller er valgt på grunnlag av det høye antall punkter og linjer. Men selv med så mange celler, og høy nøyaktighet, er ingen av de linjene som er funnet helt korrekte. Sinuskurven viser også svært godt hvor problematisk dette området er, da det finnes veldig mange punkter med mange treffpunkter.

Selv om en linje ser rett ut for øyet, er det ikke nok til å få en Hough transformasjon til å fungere som ønsket. Enkelte områder i treplaten vil muligens få enkelte treff, men ikke nok til at det er hensiktsmessig å utføre Hough transformasjon på flere steder.



Figur 4.4: Hough transformasjon utført på koter

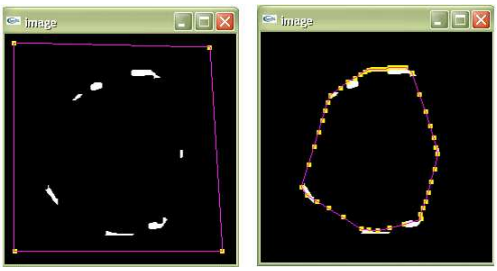
4.1.4 Aktive konturer

Aktive konturer har andre egenskaper enn Hough transformasjon, og er også verdt å prøve ut for problemområdene fra segmenteringsresultatet. Aktive konturer vil trolig uten problem greie å binde sammen de rette linjene, men ettersom det kreves manuelt utsatte kontrollpunkter, vil dette praktisk talt ikke føre til noe annet enn det å manuelt dra en linje mellom to separerte punkter. Det hele blir litt mer interessant når vi kommer til områdene rundt kvistene. Dette er områder hvor deler av årringer viser, men de færrest plasser hele. Her hvor Hough transformasjon ikke kunne benyttes ettersom ringene ikke kan beskrives matematisk, åpnes det opp for bruk av aktive konturer. Siden det er ringer som søkes, arbeides det med sluttede konturer.

Metoden starter med at brukeren plasserer ut kontrollpunkter. Det kan være så få eller mange som ønskelig, men konturen de danner må ligge på yttersiden av objektet. Etter at kontrollpunktene er plassert ut vil et høyreklikk sette i gang prosessen. For å få en jevn og rund kontur, blir flere punkter satt inn etter behov. Det hadde også vært mulig å bruke splines for å gjøre kantene runde, eventuelt også andregrads ligninger. Det å gjøre kantene runde ved å bruke mange korte rette kanter var likevel den metoden som var lettest å implementere, og resultatet ble vurdert som bra nok. Et nytt punkt settes inn dersom avstanden mellom punktene det plasseres mellom er større enn et gitt tall. Etter at nye punkter er plassert ut modifiseres konturen etter energiligningen fra seksjon 3.1.6. Pseudokoden vises under.

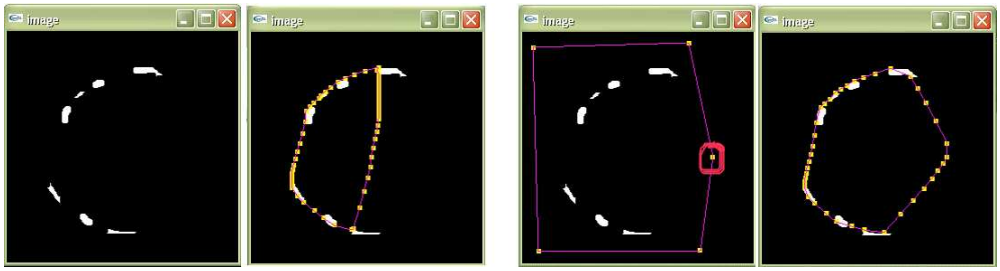
procedure Aktive konturer(Avstand)
1: Plasser ut kontrollpunkter
2: **while** Høyreklikk **do**
3: **if** $abs(p_n - p_{n-1}) > Avstand$ **then**
4: Sett inn nye punkt
5: **end if**
6: RunSnake()
7: **end while**

Metoden ble utført på ulik enkel testdata. Figur 4.5 viser et eksempel hvor fire kontrollpunkter har blitt plassert ut. Etter noen iterasjoner med algoritmen har mange nye punkter blitt lagt til og konturen har lagt seg fint rundt kanten. Det ble valgt å teste med svart hvitt bilder da det er slike resultater som kommer ut av kombinert kantdeteksjon og terskling.



Figur 4.5: Aktive konturer rundt en hullete sirkel

Et problem kan oppstå dersom for mye av en kant er fjernet. Figur 4.6 er et eksempel hvor mindre enn halvparten av det som skal være en sirkel står igjen. For å hindre at konturen blir for liten settes det inn et kontrollpunkt som er fast (andre bilde fra høyre), og resultatet går fra å være halvmåne til sirkel.



Figur 4.6: Aktive konturer med et fast kontrollpunkt

Selv om aktive konturer kan løse problemet med hullete ringer, er det lite som tyder på at det er det ideelle for kvistproblemet. Prosedyren ble utført på en kvist etter terskling. Det viser seg at de områdene hvor det er vanskelig å tyde linjer i treverket, er det også problemer for den aktive konturen. I disse områdene ligger linjene tett, og konturen greier ikke å finne riktige punkter til riktige ringer. I figur 4.7 er 9 kontrollpunkter satt ut manuelt. Dette er en jobb i seg selv, ettersom alle punktene skal ligge rundt ønsket ring, og ringen bukter seg fram og tilbake. Resultatet ble, selv etter så mange kontrollpunkter, ikke helt som ønsket. Enkelte hvite verdier er valgt feil.

Figur 4.7: Aktive konturer

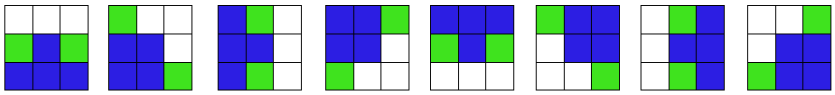
Konklusjonen blir at aktive konturer skaper flere problemer enn løsninger i dette tilfellet. Dersom aktive konturer skal kjøres rundt alle kvister, vil arbeidet med kontrollpunkter, i tillegg til kontrollen om at rett punkter er valgt, bli mye større enn om kantene skal tettes for hånd.

4.2 Morfologiske metoder

4.2.1 Tynning

Etter at kotene har blitt segmentert ut ved hjelp av kantdeteksjon og terskling vil alle linjene ha en tykkelse på minimum 2 piksel. Dette er fordi dersom en 3×3 kantmaske blir brukt vil både pikselraden før maskens sentrum og pikselraden på maskens sentrum gi utslag. Det vil være en fordel for de videre operasjonene at kotene er så tynne som mulig. Dette er en fordel for fordeling av punkter på kotene og for de videre morfologiske metodene.

Algoritmen ble implementert slik den er beskrevet i [2]. Det kjøres følgende 3×3 masker over hver piksel i bildet, se figur 4.8. Blå piksler er objektpiksler og hvite



Figur 4.8: Masker for tynning

er bakgrunnspiksler. De grønne pikslene er såkalte 'don't care' piksler. Dette er piksler som det ikke spiller noen rolle for om de tilhører objektene eller bakgrunnen. Dersom en objektpiksel har tilsvarende nabopiksler som i en av disse maskene, kan pikselen bli regnet som uviktig og blir fjernet fra objektet. Algoritmen må vanligvis kjøres flere enn en gang. Dette er fordi det er bare de ferdigprosesserte naboene til en piksel som kan garantere for riktig resultat. En piksel kan bli stående selv om den ikke burde fordi man ennå ikke vet om alle pikslene rundt skal bli fjernet eller bli stående.

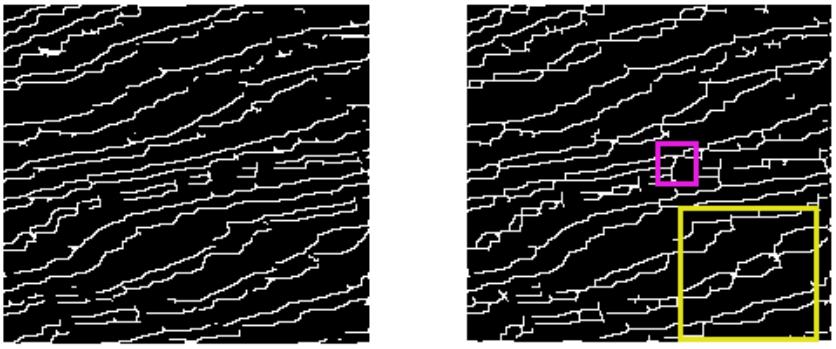
Figur 4.9: Tynning utført på kotene

Figur 4.9 viser resultatet etter tynning. Resultatet er pikselbreie kanter, men noen steder har forgrunnspiksler blitt værende der de egentlig burde ha blitt segmentert bort. Disse taggene kommer på grunn av at enkelte piksler har blitt stående fordi de har blitt basert på ikke-prosesserte piksler. Når en piksel først har blitt stående i en gjennomgang, er det lett at denne danner en spiss, og derfor blir sett på som et viktig piksel videre. Resultatet ble likevel såpass bra at det ble brukt videre i prosessen.

4.2.2 Åpning og Lukking

Som nevnt over blir kotene tynnet ned til en piksel i bredden. Dette var svært fordelaktig her dersom erosjon eller dilasjon skulle gjøres for å lukke igjen små hull

i kotene. Siden kotene ligger så tett som de gjør enkelte steder kunne de videre operasjoner ført til at store områder mellom kotene ble fylt opp med objektfarge. To omganger med lukking og åpning resulterer i resultatet avbildet i figur 4.10 Det venstre bildet er det opprinnelige, og det høyre er det modifiserte.



Figur 4.10: Åpning og lukking utført på kotene

Operasjonene har gitt både positive og negative resultater. Som avmerket innenfor den gule firkanten, har en hel del linjer knyttet seg sammen. Ulempen er at også enkelte punkter som ikke hører sammen også har knyttet seg sammen. Dette kommer fram inni den lilla firkanten. I dette lille eksempelet ga åpning og lukking forholdsvis gode resultater, og de er blitt brukt i en omgang for å tette hull på en piksel. Større hull enn dette blir vanskelig i kotekartet ettersom linjene ligger så tett og det av og til bare skiller et par piksler. Dette er likevel den mest vellykkede metoden til nå når det kommer til a binde sammen tilhørende koter.

Som nevnt er det så og si en nødvendighet for den videre utviklingen av modellen at kotene henger sammen. Løsningen ble å gå inn og gjøre en del manuelt.

4.3 Alternative metoder

Et alternativ til å programmere segmenteringsmetodene selv er å bruke eksisterende tegne/grafikk-verktøy. En mulighet er Adobe® Photoshop®. Slike programmer har ofte flere varianter av terskling og kantdeteksjon innebygd.

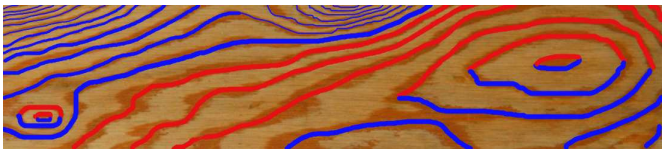
Et annet alternativ i Adobe® Photoshop® er vektorgrafikk. Slik grafikk kan legge vektorer langs kantene og bruke disse som koter. Jo større forstørrelse, desto mer detaljer kommer fram. Dette kan være er en fordel da kotene blir lagt nøyaktig rundt hver minste ruglete detalj av mønsteret. Denne vektorutgaven vil ha fordeler og ulemper i forhold til pikselkoter. Fordelene er som nevnt at kotene ikke er pikselbasert, men tilpasses til oppløsningen. Det fører til at det samme kotekartet kan brukes uansett oppløsning. For et pikselbaser kotekart kan det være vanskelig å bruke et høyoppløselige kotekart på lavoppløselig tekstur. En ulempe er at vektorer krever større lagringsplass enn piksler. Det er i tillegg vanskeligere å håndtere vektorer når punkter for trianguleringen skal finnes.

De nyeste bildene av finerplaten er 64 nærbilder som viser platen nøye detaljert. Dersom kantdeteksjon blir gjort for disse bildene, vil vi få en oppløsning på samme nivå som teksturen. Den minste detalj på på bildet er selvsagt en piksel. Når kotene også har en tykkelse på en piksel, vil disse kantene bli så nøyaktige som mulig.

Zakarias har gitt uttrykk for at han bare er interessert i høyoppløselige bilder, og da forsvinner poenget med et skalerbart kotekart. Når vektorer trenger mer lagringsplass og ikke gir bedre resultater vil det ikke være noen i hensikt å bruke dem.

4.4 Terrengmodellering

Allerede tidlig i prosjektfasen ble det gjort klart at det var ønskelig at en 3D modell av liten størrelse skulle bli klar så fort som mulig. Dette var ønskelig fordi Zakarias ville promotere tanken bak prosjektet. For å få gjort dette rask og enkelt, ble det tenkt ut en metode hvor det var aktuelt med minst mulig preprosessering. Tanken bak ble å angi på treplaten hvor kotene skulle gå opp og ned. Mønsteret på en liten og enkel del av finerplaten ble påtegnet med røde og blå streker. Med utgangspunkt i nedre venstre hjørne betyr blå at det er stigning, og rød at det er helling. Denne



Figur 4.11: Rød/blå teknikk

metoden fungerte fint på prøvemodellen, men viste seg å være vanskelig i bruk for større modeller. Når kotene var funnet, ble det for innfløkt å lete etter oppover- og nedoverbakker, og en lettere og mer automatisk løsning ble søkt.

Det man er klar over ved et kotekart er at alle sammenhengende koter vil ha samme høyde i terrenget. Dette ble grunnlaget for resten av modelleringen. Ved å gå ut fra at samlet sum av koter ikke oversteg (256*256*256)-2(svart og hvitt)=16777214 ble et program laget som farget hver sammenhengende linje i en bestemt farge. Programmet er rekursivt og leter seg fram til sammenhengende hvite punkter. Siden programmet er rekursivt er det også her viktig at de hvite linjene er så tynne som mulig for å hindre stakk-overflyt. Se figurA.3 for kotekartet.

Etter at kotene er farget er det mulig å finne høydeverdier for alle punkter på kotene ettersom man vet at alle punkter med samme farge har samme høyde.
Algoritmen for å finne punkter på kotene:

procedure Finne punkter(Høydeforskjell)

```
1:  for i<SizeX do
2:    for j<SizeY do
3:      if Punkt! = svart og Punkt! = hvitt then
4:        if Fargen er tildelt en høyde then
5:          Sett inn nytt punkt(i,j,høyde)
6:        else
7:          if Forrige punkt var hvitt then
8:            høyde-=Høydeforskjell
9:            Fargen tildeles denne høyden
10:           Sett inn nytt punkt(i,j,høyde)
11:         else
12:           høyde+=Høydeforskjell
13:           Fargen tildeles denne høyden
```

```
14:              Sett inn nytt punkt(i,j,høyde)
15:            end if
16:          end if
17:        end if
18:      end for
19:  end for
```

Algoritmen går ut på at for hver nye fargeverdi så vil det punktet få en angitt høyere høyde enn forrige punkt. Dersom dette gjøres automatisk vil både de delene som ser ut som fjell, og de som ser ut som daler bli fjell. For å kunne velge å lage hellinger i terrenget, legges det til hvite punkt like nedenfor koten, og algoritmen registrerer at neste punkt skal synke og ikke stige. For å begrense antall punkter blir bredden på bildet diskretisert med 20 pikslers mellomrom. For de fleste delene av modellen vil dette fungere bra. For nærbilder av egenskaper i kvister kan det være aktuelt å minske denne avstanden.

For et irregulær gitter blir det lett å se hvordan punktene skal samples fra kotekartet. Punktene skal på et eller annet vis plukkes på kotene. For å lage et regulært gitter er situasjonen en ganske annen, her kan vi ikke lenger forvente at punktene skal treffe akkurat der hvor det er koter. For å få riktige høydeverdier for samplede punkt, lages først et gråverdibilde basert på den opprinnelige algoritmen for å finne høyder. Her lagres ikke bare punkter for de stedene hvor det er koter, men for alle piksler i inndatabildet.

Algoritmen for interpoleringen fungerer som følger. For hvert punkt mellom kotene søkes de nærmeste kotene i retningene nord, sør, øst og vest. Interpolasjonsverdien blir satt som forholdet mellom høydene til de nærliggende kotene og avstanden fra den enkelte kote og punktet. For at dette skal fungere må hele kotekartet gjennomkjøres to ganger. Først må høydeverdiene på kotene bestemmes, og så må punktene i mellom sjekkes i en ny gjennomkjøring. Se figur A.4 for høydekartet.

Når dette bildet er laget blir det enkelt å diskretisere punkter. Samme hvilken avstand som velges i gitterstrukturen vil høydeverdien som plukkes ut være riktig. I tilfeller med en stor modell med mye høydeforskjell vil det ikke være noe problem å lagre en fil med høyere gråverdi enn 256. Det eneste som skjer da er at resultatet ikke blir et bilde i rett forstand, men kun en punktfil. Et alternativ her for å kunne åpne det som bilde kunne være å bruke farger, noe som vil åpne for 3*256 verdier.

I kapittel 2 ble det nevnt en hel del interpolasjonsmetoder som blir brukt i virkelige landskapsmodeller. De var ikke aktuelle for resultatet her. Årsaken til dette er at kunstneren ikke ønsker en jevn modell, men at det skal være klare skiller mellom kotene. Den interpolasjonsmetoden som er brukt ovenfor gjør nettopp dette, da den bare benytter seg av de (maks)fire omringende høydene for hvert punkt.

Noe som skapte litt forvirring var hvordan bilder blir lest til fil i forhold til hvordan de blir skrevet ut til skjerm. For at punkter ikke skal bli vist speilvendt på skjermen måtte de gjennom en transformasjon fra hvordan de ble lest til vektoren der de blir lagret. Formelen for å overføre fra fil til skjerm er den følgende:

$$n * m - (m * (i/m + 1) - i * mod(m))$$

i=pikselnummer fra fil
n=høyde
m=bredde

4.4.1 Triangulering

Trianguleringen er basert på trianguleringsalgoritmen til Paul Bourke [6]. Her var det også implementeringer av algoritmen i Java og C. Koden ble så modifisert til å passe i Visual C++ i dette prosjektet. Koden ligger ute til fri bruk på hans hjemmesider.

Algoritmen starter med å ta utgangspunkt i et supertriangel som dekker alle punktene. Hvert av punktene blir så satt inn et etter et. For hvert punkt blir trekanten som omslutter det omgjort til nye trekanter som inkluderer det nye punktet. Trekantene blir bygget opp etter Voronoy/Delauney prinsippet beskrevet i seksjon 3.3.2. For hvert nytt punkt som settes inn dannes sirkler som inkluderer dette punktet og nøyaktig to andre. Den minste sirkelen som inneholder tre slike punkt bestemmer at disse tre punktene skal danne en trekant. De trekantene som allerede eksisterer og påvirkes av dette modifiseres til å passe inn i den nye polygonmodellen. Når alle punktene er triangulert fjernes de trekantene som har kantene i supertrianglet som sine egne. Kjøretiden for algoritmen er $O(N^{1.5})$, men er noe avhengig på minnestørrelsen på arbeidsstasjonen. Se figur A.5 for trianguleringen av et lite stykke av finerplaten.

Det finnes allerede flere gode systemer som kan hjelpe til med triangulering og betraktning av store modeller med høy oppløsning. Et av disse er laget av Systems in Motion(SIM). Biblioteket SIM Scenery inneholder metoder som kan visualisere store terrengdatabaser i sanntid. Det er svært aktuelt for dette prosjektet. Særlig når det skal kunne vises i kunstmuseumet, og om det skal være mulig å navigere i terrenget, og ikke bare se en video av det. SIM Scenery bruker en database som inndata. Databasen blir først laget i deres eget program Creator. I denne databasen lagres all informasjon om teksturer og høydeverdier. På denne måten kan man få rask tilgang til all denne dataen for hvilket som helst område i modellen, og for hvilken som helst tekstur. Resultatet anno 2003 er en interaktiv framerate med database størrelse på 12GB, men 200×200km høydedata, og oppløsning fra 25 til 0.5 meter.

4.4.2 Culling

Culling kan aktiveres ved å skrive kommandoen glEnable(GL_CULL_FACE) i koden. Det går an å velge om en vil ha bakside eller forside culling. For å sjekke at det virker kan vi se hvilke flater som blir stående i forside culling. De flatene som blir stående er de som ville blitt cullet ved bakside culling. Resultatet finnes i A.6.

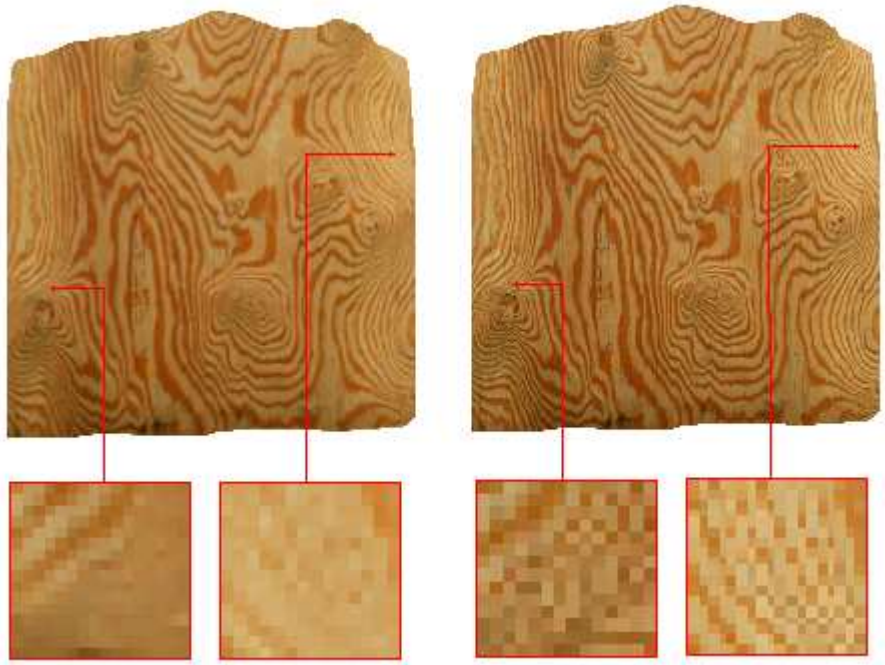
4.4.3 Tekstur

For teksturmapping ble standard metoder i OpenGL brukt. Noe som måtte behandles var størrelsen på teksturene. Maksimumsgrense for teksturen er 1024×1024, og for høyoppløselig tekstur måtte denne deles opp. Dette skapte ikke store problemer i skjøter dersom teksturene hadde noe overlapp.

Ulike teksturer ble utprøvd. Et forslag utenom det naturlige alternativet finerplaten var en modifisert tekstur laget av Zakarias. Teksturen fikk navnet Porcelainscape, som tar utgangspunkt i det tradisjonelle blå og hvite porselen servise(se figur A.7).

For å øke 3D og perspektiveffekten ble også tåke lagt til modellen. Dette gjøres ved å kalle metoden glEnable(GL_FOG). Figur A.9 viser et stykke av terrenget med tåke.

Teksturmappingen ble implementert både med og uten mip-mapping. Figur 4.12 viser forskjellene. Teksturen til venstre er mappet med mip-mapping, mens stykket



Figur 4.12: Varianter av teksturmapping på et lite stykke av finerplaten

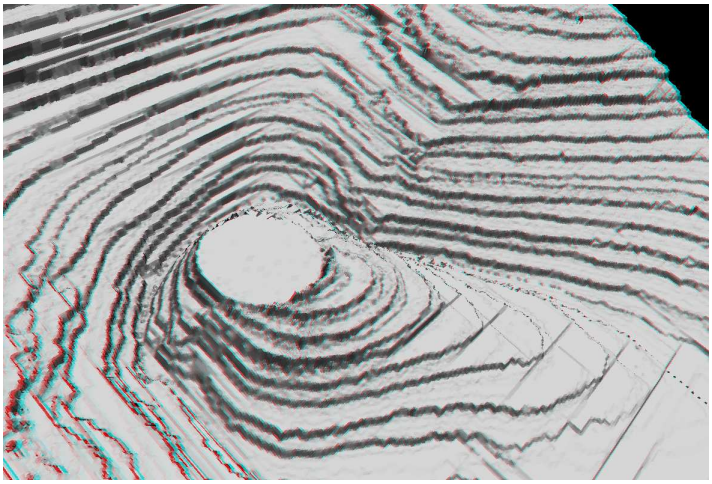
til høyre er mappet med nærmeste pixel. Forskjellene er tydelige. Som beskrevet i seksjon 3.3.4 lager mipmapping teksturen noe utjevnet og detaljer kan forsvinne, men helhetsinnstrykket blir bedre. For nærmeste-piksel mappingen er det mye aliasing det hvor linjene ligger tett. Valget ble satt til mipmapping.

4.4.4 Lyssetting

Metoden for lyskart er den som ble valgt. Det er en enkel metode, og i tillegg brukes den fordi LOD er så aktuelt. For å få best mulig resultat ble eksisterende software for modifisering av terrengdata brukt. Wilbur [25] er en enkel applikasjon som kan lage lyskart. Inndata må være et høydebilde. Det kan f.eks være et som er laget med metoden som ble brukt for diskretiseringen til regulært gitter. Etter at lyskartet er laget forsterkes kontrastene, og det adderes eller subtraheres med teksturen(alt ettersom hva som skal være lyst og hva som skal være i skyggen), og vi får ut en tekstur med tydelige 3D effekter. Se A.8 for eksempelbilde. Hva som er gjort er bare å subtrahere lyskartet fra den opprinnelige teksturen. Grunnen til at resultatet er ganske rødt er at rødverdien i teksturen generelt er høyere enn den blå og grønne komponenten. Dersom gråverdien i lyskartet er høy vil subtraksjonen føre til at den røde komponenten er høyere i forhold til den blå og den grønne. Hele platen med lystekstur modellert i 3Dtre vises i figur A.10

4.5 Steroskopisk projeksjon

I et standard brukergrensesnitt fra SIM Scenery er det innebygde muligheter for stereoview. Figur 4.13 viser anaglyfisk stereo. Dette er alternativet Red-Cyan fra



Figur 4.13: Rød-cyan

menyen. Det andre mulige alternativet er Quadbuffer, og fungerer med polarisert lys.

4.6 Andre Ting

En siste ting er å legge mulighetene åpne for store trykk av skjermbilder. Dette er viktig da Zakarias har behov for slike i kunstutstillingen. Løsningen ble å bruke OffScreenRendering i Sim Scenery. Vanligvis når en renderer en scene gjøres dette til skjermstørrelsen, og en screen shot vi gi en oppløsning som tilsvarer skjermstørrelsen. Nå vi bruker en off screen renderer, settes renderområdet til en fri størrelse. Det vil si, den enkelte arbeidsstasjon har en maks grense for et slikt renderområde. Aktuell arbeidsstasjon oppgir maksimumstall på 32000×32000 piksler.

Kapittel 5

Testing og evaluering

Datamaskinen som ble brukt er en Intel(R)prosessor Pentium(R) 4 CPU 2.40GHz 512 MB RAM. Skjermkortet er RADEON 9200 SE Family (Microsoft Corporation)

5.1 Testing

5.1.1 Finne punkter

For å teste ytelsen på punktalgoritmene ble det gjort tester på to kotekart. Det ene er FargeSpekterSpeil.ppm på 1024×1024 piksler, og det andre er av hele finerplaten FullKont.ppm på 4725×3675 piksler. For hver piksel i kotekartet blir det sjekket farge, og sammenlignet med verdiene i en fargetabell. Kjøretiden på algoritmen vil være $O(n)$.

punkter	*tid(sekund)*
53232	0.30
763304	3.92

Ved å se på resultatene får vi den generelle formelen :

$$tid = n/(1.95 * 10^5)$$

En million punkter kan utregnes på 5.13 sekunder som er en absolutt akseptabel tid da en husker på at dette bare må gjøres første gang programmet kjøres.

For diskrete punkter blir det funnet verdier for alle pikslene i kotekartet. Antall punkter blir her mye større. Kjøretiden er fortsatt $O(n)$ da det også her kun blir gjort uavhengige operasjoner for hver piksel.

punkter	*tid(sekund)*
1048576	13.02
17364375	240.49

Ved å se på resultatene får vi den generelle formelen:

$$tid = n/(7.22 * 10^4)$$

Det er naturlig at det tar lengre tid per diskrete punkt enn per irregulære punkt, da alle piksler må gjennomkjøres to ganger i tillegg til at en interpolasjonsverdi må regnes ut. 4 minutter(240.49 sek) er litt ventetid, men for store bilder med millioner av punkter må man nesten forvente slikt.

5.1.2 Delauney triangulering

For trianguleringer er det veldig mange matematiske beregninger som skal til, og det er viktig at de blir gjort så effektivt som mulig. Delauney trianguleringen som er brukt i 3Dtre er som nevnt tidligere hentet fra [6]. Bourke nevner i teksten at algoritmen kan effektiviseres ved å sortere punktene etter den koordinaten som har flest samplinger. Da vil kjøretiden gå fra O(n^2) til å bli O($n^{1.5}$)

Det ble gjort to forsøk på kotekartet FargeSpekter.ppm. Dette er 1024*1024 piksler. 4 ulike diskretiseringer i x-retning ble gjort. Det første forsøket ble gjort med usorterte punkter.

Avstandx	*punkter*	*trekanter*	*tid(sekund)*
64	682	1274	1.5
16	2790	5476	24.3
8	5598	11082	92.3
4	11197	22270	394.9

Den generelle formelen for kjøretid i forhold til triangler blir da:

$$(antTrekanter)^2 * 8.1 * 10^{-7}$$

I følge Bourke vil sortering av inndata gi kjøretid $O(n^{1.5})$ Den generelle formelen for kjøretid i forhold til triangler blir da:

$$(antTrekanter)^{1.5} * 8.1 * 10^{-7}$$

For 100000 trekanter vil det uten sortert inndata ta 135 minutter, og for en million trekanter vil det ta hele 225 timer! For $O(n^{1.5})$ ville tilsvarende tid for en million trekanter bare vært 13,5 sekunder.

Inndataen ble sortert, og nye resultater ble som følgende.

Avstandx	*punkter*	*trekanter*	*tid(sekund)*
64	682	1274	0.47
16	2790	5476	2.27
8	5598	11082	7.11
4	11197	22270	25.2

Bourke sier at kjøretiden algoritmen vil bli $O(n^{1.5})$, men at minnestørelsen på arbeidsstasjonen kan være avgjørende. Kjøretiden på denne arbeidsstasjonen er på $O(n^{1.73})$, noe som er en god del høyere. En million trekanter med denne kjøretiden vil bli ca 5.4 timer.

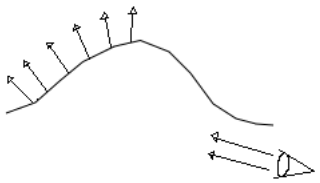
5.1.3 Modellen i sanntid

Det er mulig å flytte både øyeposisjon og synspunkt i modellen. Hver gang dette blir gjort må alle polygoner tegnes opp på nytt. For at bevegelsene skal se naturlige ut bør ikke denne opptegningen være synlig. Her er det store forskjeller på den SIM baserte modellen og den i 3Dtre. 3Dtre ble implementert uten noe form for LOD. Det ble ikke brukt tid på det siden SIM Scenery har veldig gode LOD algoritmer innebygd, og dette biblioteket kunne brukes. Dette fører til at man i 3Dtre tydelig kan se trekantene som blir tegnet opp, mens dette ikke skjer i den SIM baserte. Resultatet er likevel ikke helt synlig perfekt i det SIM baserte, da man tydelig kan

se endringer i detaljering i teksturen. Man ser et kort øyeblikk teksturen(eventuelt polygonstrukturen dersom det ikke er brukt tekstur) som udetaljert, helt til den detaljeres mer og mer. Dette skjer særlig ved store oversiktsbilder eller dersom raske bevegelser med modellen blir gjort. Det er likevel ikke noe stort problem da dette ikke skjer så veldig ofte. Dette skyldes sannsynligvis geomorphing. Som nevnt i kapittel 3.3.10 blir polygonstrukturen gradvis endret for å hindre popping.

5.1.4 Bakside culling

Det ble også gjort en test med og uten bakside culling i 3Dtre. Resultatet for begge ble ca 2.5 frames per sekund dersom fullscreen ble kjørt og det meste av skjermen var dekket av modellen. Dette skyldes nok strukturen på terrenget. Det er mange flate og jevne overganger, og det er ofte mye som ikke kan culles bort, særlig dersom en er ute etter et visst overblikk. Frustrum culling blir uansett gjort for begge varianter. Det vil si at kun de trekantene som vises på skjermen krever cpu-kraft. Det ble også enkelte problemer med å ha culling. Dersom en beveget seg litt under modellen kunne en se enkelte hull foran og til sidene. Dette skjedde fordi alle normalvektorene peker opp fra overflaten, og dersom man beveger seg under, vil mange av disse vende bort fra synspunktet, og derfor bli cullet (se figur 5.1). Resultatet ble å ikke bruke bakside culling i 3Dtre.



Figur 5.1: Problemer med bakside culling

5.2 Evaluering

5.2.1 Segmentering

Segmenteringen av platen viste seg å være vanskeligere enn først antatt. Ved første øyekast så oppgaven ganske grei ut, da mønsteret tydelig skiller seg fra bakgrunnen. Forestilingen ble en ganske annen de generelle metodene ble forsøkt anvendt på platen. Store områder hadde mønster som var lite synlig, eller ikke eksisterende bindeledd. Dette gjaldt særlig rundt kvistene. Ulike oppløsninger ble forsøkt, men nye problemer dukket opp. Horisontale linjer og rusk kom fram, og gjorde ikke situasjonen noe lettere. Resultatet ble å kombinere adaptiv terskling og kantdeteksjon, og å bruke noe morfologisk tynning og tetting. De tynneste ringene ble tegnet inn manuelt.

Slik segmenteringen har foregått har ulike metoder blitt tilpasset og anvendt etter behov, og brukt på ulike måter på ulike steder av platen. Det var ikke innenfor rammen av dette prosjektet å lage et tilpasset program som kan ta inn en hvilken som helt treplate og få ut et kotekart. Zakarias har selv prøvd seg på segmentering på veldig høyoppløselige bilder. Han har brukt Adobe® Photoshop®, og utført ulike operasjoner for å segmentere ut mønsteret. Resultatet har vært vellykket og

brukbart som inndata i 3Dtre. Dette er tidkrevende manuelt arbeid, og ble ikke forsøkt i denne oppgaven. Det viser likevel at kan selv benytte seg av programmet ved å lage egne kotekart, og at programmet på denne måten kan bli tilpasset mer generelle problem.

5.2.2 3Dtre

Arbeidet med modelleringen gikk først å fremst ut på å finne en god måte å beskrive kotene i terrenget på. En enkel opp-ned (blå-rød metode) metode ble forsøkt, men forkastet da den viste seg å være vanskelig i bruk. Løsningen ble å farge kotene og implementere en metode som økte høydeverdien for hver nye farge som ble funnet. Algoritmen ga gode resultater, og brukte for de dataene som ble testet (4725×3675 piksler) aldri mer enn 4 sekunder. Etter at høydedataene var fastsatt måtte punktene velges. Det ble i utgangspunktet naturlig å legge punktene på kotene for så å polygonisere disse til et irregulært gitter. Punktene ble polygonisert med Delauney triangulering. Denne trianguleringen viste seg å bli flaskehalsen for utførelsen av 3D modelleringen. Noe særlig mer enn 100000 (tidsbruk i overkant av 4 minutter) er tidsmessig uakseptabelt. Dersom 100000 er et tall man er fornøyd med, er resultatet bra, og gir trekanter som ligger mellom kotene og gir et naturlig og pent utseende.

Modellen ble teksturert med blant annet selve finerplaten. Et problem som oppsto var teksturbegrensningene. Hver tekstur kunne være maksimum 1024×1024 piksler, og for høyoppløselig tekstur på en stor modell var ikke dette godt nok. Løsningen på dette ble å legge til flere teksturer, og legge disse utover modellen. Det ble gjort et forsøk på generalisering av tekstureringsfunksjonen slik at den som kjører programmet selv kan velge antall teksturer. Dersom teksturene klippes til riktig og legges til i rett rekkefølge blir resultatet bra med unntak av enkelte steder på skjøtene mellom teksturene. Dette problemet bør kunnes løses, men ble ikke prioritet da dette ville løse seg selv i SIM modellen, som er i det hele mer tilpasset en generell bruker. For å få riktige lysforhold ble lysteksturer laget, men det er ikke laget et program for å lage disse. Som nevnt tidligere fins det programvare som gjør dette for deg.

Noen helt simple navigeringsfunksjoner ble lagt til. D.v.s. tastetrykk kan bevege synspunkt og øyepunkt. Styrelsesmekanismen kan virke noe kronglete for noen som ikke har prøvd det mye, men som sagt er grensesnittet først og fremst laget for personlig testing. Se A.11 for eksempel av høyoppløselig kvist modellert i 3Dtre.

5.2.3 SIM Scenery

De manglene fra 3Dtre ble fjernet da SIM Scenery ble tatt i bruk. Det må nevnes at ettersom det lenge var klart at SIM Scenery kunne brukes, ble det ikke prioritert framerate og brukbarhet i 3Dtre da dette var aspekter som allerede eksisterte i SIM Scenery. Se A.12 for eksempel av høyoppløselig kvist modellert i det SIM baserte programmet.

For å kunne benytte SIM Scenery måtte punktalgoritmen i 3Dtre tilpasses til også å lage punkter for regulære gitter. Dette ble gjort med en enkel interpolasjonsmetode som for hvert punkt sjekker høydeverdien til nærmeste kote i retning nord, sør, øst og vest, for så å beregne høydeverdi basert på avstanden til de enkelte kotene. Resultatet ble bra i henhold til Zakarias' ønsker. Siden det kun er snakk om lineær interpolasjon vil resultatet bli terrassert, men dette er også det som var

ønskelig.

Punktverdier og tekstur legges til i SIM Creator. Den store fordelen med SIM Scenery er at teksturstørrelsen ikke er begrenset. I tillegg kan teksturen være .jpg, noe som krever mye mindre lagringsplass enn .ppm filer som ble brukt i den egne modellen. Det eneste det står på da er hvor mye arbeidsminne en har da teksturen lages. Brukergrensesnittet er også ganske enkelt å bruke. Skalering, translering og rotering blir gjort på intuitive måter. I tillegg er frameraten god, og man merker svært lite til level of detail funksjonen. Et annet positivt aspekt er muligheten for stereosyn. Det er innebygd både funksjon for rød-cyan anaglyf visning, og visning med polarisert lys. Dette førte til at en egen stereo implementasjon ikke var nødvendig.

Det er nevnt i introduksjonen at det vil være aktuelt med ulike former for animasjoner. Dersom en ser for seg interaktivitet som animasjon vil dette være gjeldende ved å kjøre programmet. Zakarias har selv sagt at videoproduksjon ikke er noe problem, og at han selv kan gjøre dette kjapt. Siden kommunikasjonen mellom Trondheim og Berlin(hvor Zakarias befinner seg) har vært noe vanskelig har det også vært usikkert hva som skulle bli animert, og det ble derfor ikke prioritert. For å nevne noe så finnes det utallig programvare som lager stillbilder om til video. Slike stillbilder kan enkelt trekkes ut fra den interaktive modellen.

Et problem som er foreløpig uløst med det SIM baserte programmet er en feilmelding som kommer når programmet lukkes. Denne feilmeldingen kan fjernes ved å kommentere ut en linje i koden som sletter et objekt. Å gjøre det resulterer i at programmet blir ustabilt på en annen måte. Det har en tendens til å lukke seg selv. Her er det bare å åpne det på nytt til det blir værende åpent. Problemet er nevnt for SIM, men de har foreløpig ikke funnet en løsning på det.

Som konklusjon ble ulike metoder studert og noen av disse implementert, mens prosjektet ble realisert ved hjelp av SIM Scenery. Mer og mer programvare for terreng visualisering er tilgjengelig, og det er ikke noe hensikt i å ikke benytte seg av dem. Ettersom metoder ble studert er målet like fullverdig nådd med SIM Scenery som om alt skulle være implementert på egenhånd.

Kapittel 6

Diskusjon

Terrengmodellering er en visningsform som mennesker blir mer og mer vant til. Mange har sett værkart, fjellområder og oldtidsområder fremstilt i 3D. Slike modeller kan være nyttige for å tilegne seg kunnskap og å skape interesse. Ettersom slike modeller blir mer og mer vanlige, og både dataspill og animasjonsfilmer ofte har så god grafikk at det kan være vanskelig å skille mellom konstruerte og ekte figurer, blir høyere og høyere krav satt til grafikk og oppløsning. Dette er med på å føre utviklingen videre med stormskritt, og det blir stadig mer mulig med utrolig detaljerte bilder og høydedata fra jordoverflaten hentet fra satellitter.

Som framstilt i rapporten kan et terreng vises i 3D ved hjelp av ikke nødvendigvis så alt for mange kodelinjer. Selv om det er mulig å programmere mye selv, er dette nok et aspekt av terrengmodellering som kommer til å forsvinne mer og mer. Dette er en naturlig konsekvens av at det blir laget mer og mer ferdig programvare som kan håndtere enorme punktmengder og høyoppløselig tekstur. Slike programmer har ofte også innebygde metoder for lyssetting, skyggelegging og stereoskopisk visning. Muligheter for animasjoner og videoproduksjon er også i flere tilfeller tilrettelagt.

Kapittel 7

Konklusjon

Hovedmålet med denne masteroppgaven var å sette seg inn i Yngve Zakariaz' problemstilling om å visualisere en finerplate som en terrengmodell i 3D. Gjennom å undersøke og sammenligne metoder for å gjennomføre dette har ulike resultater blitt oppnådd. Som første del av oppgaven har mønsteret i finerplaten blitt segmentert ut og omgjort til et kotekart. Andre del av oppgaven var selve terrengmodelleringen. Dette har blitt gjennomført i to faser. Først ble et eget program, 3Dtre, utviklet, som produserer en 3D modell basert på et irregulært polygongitter. I den andre fasen ble SIM scenery brukt som basis, og data konstruert i 3Dtre ble brukt her. Resultatet ble en 3D modell som oppfyller alle krav til interaktivitet, teksturmapping, stereosyn og nøyaktighet.

Som nevnt i introduksjonen var en av de største utfordringene med denne masteroppgave å håndtere samarbeidet mellom ulike fagområder henholdsvis kunst og datateknikk. Forvirringene kom allerede på første møte da begge parter mente at sin oppgave var visualiseringen av terrenget. Dette viser at ulik bakgrunn og ulikt fagspråk kan føre til kommunikasjonsproblemer. Selv om små konflikter oppsto på grunn av kommunikasjonsproblemer og ulike forventninger, førte samarbeidet til en rekke nyttige erfaringer og innsyn i andre tankeganger for begge parter.

Resultatet med de to programmene er et resultat begge parter kan være fornøyd med. Selv om kvaliteten er langt fra kommersiell, gir programmene foreløpige resultater og kan utvikles videre. Det er allerede koblet inn folk til å ta prosjektet videre, og forhåpentligvis vil utstillingen i september bli en suksess.

Kapittel 8

Videre arbeid

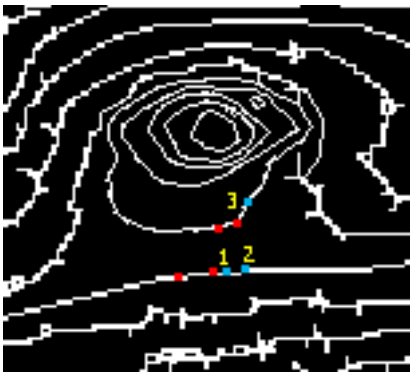
Yngve Zakarias har helt siden slutten av 1980-tallet jobbet med kunst som baserer seg på treets naturlige mønster. Den ideen det har blitt jobbet med her er hans siste innenfor dette området, og er en prosess som han selv sier så vidt har begynt. Han har planer om å jobbe med virtual woodscape minst et par år fremover, eller så lenge han mener finerplaten har noe mer å by på. Sånn i nærmeste framtid blir det å gjøre bilder og animasjoner klare til kunstutstillingen i september 2005.

Det første punktet i det videre arbeidet blir å lage et nytt kotekart. Det kotekartet som er laget nå er basert på et bilde med ca 1/64 av oppløsningen til de nyeste bildene. Den kantdeteksjonen som er blitt gjort på det minste bildet tar for seg begge sider av hvert mørke bånd, men i områder hvor båndene er svært tynne og tette , f.eks rundt kvister, vil kantene som er detektert på hver av sidene smelte sammen til en linje. Da dette skjer blir det umulig å ha to ulike høydeverdier for hvert mørke bånd. Resultatet blir at en midtlinje blir valgt som basis, og båndet blir liggende i en bakke, med unntak av hvor linjene skiller seg igjen. Der vil flate områder oppstå. Selv om kotekartet passer til en mer høyoppløselig tekstur ettersom forholdet er det samme, vil båndene her bli tykkere slik at kotene ikke ville blitt samlet. For å få en modell med egen høydeverdi for hver side av båndene og ikke en til sammen, må et nytt kotekart lages. Dette kan gjøres på samme måte som beskrevet tidligere i rapporten, men ekstra hensyn må tas til rusk og horisontale linjer.

Ved å ta utgangspunkt i SIM Scenery behøves et regulært gitter. For å oppnå absolutt optimal triangulering hvor trekantene blir liggende langs kotene vil ikke dette fungere. Problemet med den irregulære polygonmodellen er ytelsen. Jo høyere nøyaktighet desto flere trekanter, og tiden for Delauney trianguleringen øker med $O(trekanter^{1,73})$. En mulighet er å bruke level of detail metoden som er nevnt tidligere, men det vil uansett være nødvendig å starte med punkter som er diskretisert tett. Slik algoritmen fungerer nå tar den alle punkter som ligger på koter for x-verdier diskretisert med bestemte avstander. Avstandene mellom x-verdiene kan selvsagt settes lik 0, og en optimal punktvektor er garantert, men dette vil skape unødvendig mange trekanter i flate nivå. En fremgangsmåte er å velge flere punkter desto brattere og mer kronglete kotene er. I tillegg vil topper kreve høyest diskretisering fordi her ligger ulike artifakter veldig tett. Et forslag på en algoritme er å følge hver kote for seg. Dersom overflaten mellom et punkt og de to forrige punktene ikke er jevn, må nye punkter plasseres. Dette vil gi mange punkter på kurver og få på rette linjer.

$$Kurve = g|(2x_i - x_{i-1} - x_{i-2})| + h|(2y_i - y_{i-1} - y_{i-2})|$$

Denne formelen garanterer også at avstanden mellom punkter på rette linjer heller ikke blir for stor. Leddet h er en vektlegging på y-leddet, og g en vektlegging på x-leddet. Ved å ha kotekartet liggende, vil det være mest horisontale linjer. Da vil det være der det er mye endring i y- leddet at vi vil legge inn flest punkter. Det vil da være naturlig å gi h større verdi enn g. Eksempelvisk kan h settes 2 og g til 1.



Figur 8.1: Alternativ punktalgoritme

Resultatet blir da:
1=1*23+2*2=27
2=1*37+2*6=49
3=1*12+2*15=42

Ved å se på resultatet, ville ved en terskelverdi på 40 satt ut punkter i posisjon 2 og 3, men ikke i posisjon1.

For å få følelsen av at du riktig befinner deg i et landskap er navigeringen viktig. Slik programmet fungerer nå er det fullt mulig å se landskapet fra ulike sider og vinkler, men det er ingenting som hindrer deg fra å gå gjennom landskapet. Dersom en sammenligner med spill som Super Mario 64, eller World of Warcraft, ser en at ting kunne vært annerledes. Det som hadde vært interessant var å lage en navigasjon som baserer seg på at det er en person som går på landskapet. Personen vil da verken kunne bevege seg opp fra landskapet(fly) eller under landskapet, men skal hele tiden være plantet på overflaten. Dette kan gjøres ved å se til at øyeposisjonen samsvarer med høydeverdien(eventuelt noe høyere for en person med høyde) for tilsvarende x- og y-verdi. I tillegg til dette må også posisjonen til referansepunktet ligge innen rekkevidde, altså ikke bak et fjell o.l.

Det kunne også være interessant å vise landskapet i RAVE eller lignende. I første omgang ville det være naturlig å kjøre en film, men det kunne også være aktuelt med joystick til å navigere med, eller utstyr som følger øyeposisjonen til tilskueren.

Slik 3Dtre er implementert er alt basert på kommandolinje. Et grafisk brukergrensesnitt ble ikke prioritert, da det ville tatt mye tid, og programmet ble sett på først og fremst som et personlig testverktøy. For å gjøre 3Dtre mer brukervennlig kan det i fremtiden være aktuell med et GUI. Dette kan gjøre programmet brukbart for flere, og kan mulig hindre eventuelle forvirringer. En ting jeg ser vil være nyttig i et GUI er særlig tidsbrukperspektivet. Delauney triangulering tar relativt lang tid på store punktmengder, og for noen som ikke vet dette kan de tro at programmet har låst seg. En progress-bar vil hindre at disse forvirringene oppstår.

En mulig løsning for prosjektet videre er å kombinere de to programmene. Selv om det vil være aktuelt å bare bruke regulært gitter må en likevel innom det selvproduserte programmet for å lage gitterpunktene. Det hadde vært mer praktisk om alt kunne gjøres samme plass. På en annen side må uansett punktene og teksturene legges inn i en database ved hjelp av SIM Creator, så det er vanskelig å få et program som kan gå hele veien fra kotekart til regulær gittermodell.

Bibliografi

[1] Angel,E(2003).*Interactive computer graphics*(3. utgave. Kapittel 7,8,13).AddisonWesley. ISBN: 0201773430.

[2] Gonzalez,R og Woods, R (2002) *Digital Image Processing*(2. utgave. Kapittel 9,10) PrenticeHall inc. ISBN: 0201180758.

[3] Emanuele Trucco and Alessandro Verri*Introductory Techniques for 3-D Computer Vision, Prentice Hall, 1998*(Kapittel 5)

[4] Gold, Dakowicz *Terrain Modelling from Contours*

[5] Pierre.Soille,2003*Morphological Image Analysis*(2. utgave. Kapittel 3,4)Springer, ISBN: 3-540-42988-3

[6] Bourke,P (1989) *Triangulate*Lastet ned 15. mars, 2005, fra http://astronomy.swin.edu.au/ pbourke/terrain/triangulate

[7] Bourke,P (1993) *Computer Based Terrain Visualisation Techniques*Lastet ned 15.mars, 2005, fra http://astronomy.swin.edu.au/ pbourke/terrain/terrainvis

[8] Bourke,P (1999) *3D Stereo Rendering Using OpenGl and GLUT*Lastet ned 01.06.2005, fra http://astronomy.swin.edu.au/ pbourke/opengl/stereogl/

[9] Patterns in Nature *Polarized Light*Lastet ned 25.05.2005 fra http://acept.la.asu.edu/PiN/rdg/polarize/polarize.shtml

[10] Blackburn, T *Stereographic in Theory and Imaging Practice*Lastet ned 25.05.2005, fra http://www.terryblackburn.us/WildIdeas/Stereography_theory.htm

[11] *Pulfrich Effect*Lastet ned 25.05.2005, fra http://www.stereoscopy.com/faq/pulfricheffect.html

[12] Berg, Dobrindt (1995). On Levels of Detail in Terrains *Symposium on Computational Geometry 1995.* Lastet ned fra http://doi.acm.org/10.1145/220279.220334

[13] Lindstrom, Koller, Ribarsky, Hodges, Faust (96). Real-Time, Continuous Level of Detail Rendering of Height Fields *SIGGRAPH '96.* Sider pages 109–118. Lastet ned fra http://www.cc.gatech.edu/ lindstro/papers/siggraph96/siggraph96.pdf.

[14] Mircea Marghidanu *Fast Computation of Terrain Shadow Maps.* Lastet ned 29.04.2005, fra http://www.gamedev.net/reference/articles/article1817.asp.

[15] Röttger, Heidrich, Slusallek, Seidel (1998) RealTime Generation of Continuous Levels of Detail for Height Fields*V. Skala, editor, Proc. WSCG '98.* Sider 315-322, 1998. Lastet ned fra http://www9.informatik.uni-erlangen.de/Persons/Roettger/papers/TERRAIN.PDF

[16] Blow, J(2000). Terrain Rendering at High Levels of Detail *Game Developers' Conference*(2000). Lastet ned fra http://number-none.com/blow/papers/terrain_rendering.pdf

[17] Ewins, Waller, White, Lister (1998) MIP-Map Level Selection for Texture Mapping*IEEE Transactions on Visualization and Computer Graphics.* Vol 4. No 4. Oktober-desember 1998. Lastet ned fra http://www.sussex.ac.uk/Units/vlsi/mipmap.pdf

[18] Hormann, Spinello, Schröder (2003). C1-continuous Terrain Reconstruction from Sparse Contours *Thomas Ertl (Ed.): Proceedings of the Vision, Modeling, and Visualization Conference 2003*Thomas Ertl (Ed.): Proceedings of the Vision, Modeling, and Visualization Conference 2003 (VMV 2003), München, Germany, November 19-21, 2003. Aka GmbH 2003, ISBN 3-89838-048-3.

[19] Gousie,M, Franklin R (1998). Converting Elevation Contours to a Grid *Proceedings, Eighth International Symposium on Spatial Data Handling* (1998), T. Poiker and N. Chrisman, Eds., pp. 647-656 Lastet ned fra http://cs.wheatonma.edu/mgousie/contour3.pdf

[20] Wagner,D (2003) Terrain Geomorphing in the Vertex Shader*ShaderX*^{2}*-Shader Programming Tips and Tricks*Wordware, 2003. Lastet ned fra http://www.gamedev.net/reference/articles/article1936.asp

[21] Asirvataham, A og Hoppe, H (2005) Terrain Rendering Using GPU-Based Geometry Clipmaps*GPU Gems 2*Pharr,M og Fernando,R (2005). Addison-Welsey. ISBN: ISBN: 0321335597. Lastet ned fra http://research.microsoft.com/ hoppe/

[22] Losasso, F og Hoppe, H (2004) Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids *ACM SIGGRAPH 2004, 769-776* Lastet ned fra http://research.microsoft.com/ hoppe/

[23] http://www.planetside.co.uk/terragen/

[24] http://www.sim.no

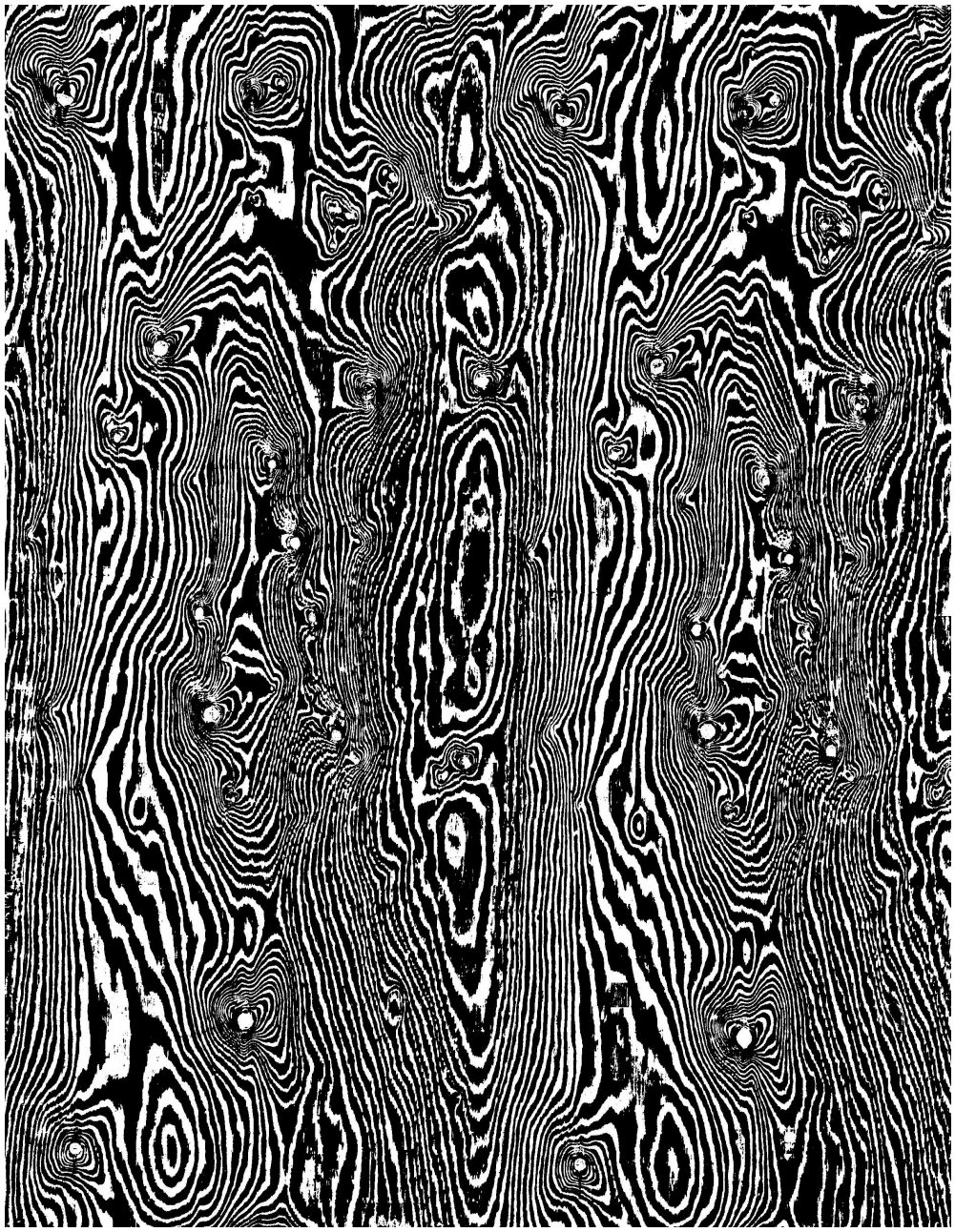
[25] http://www.ridgenet.net/ jslayton/software.html

[26] http://www.gimp.org

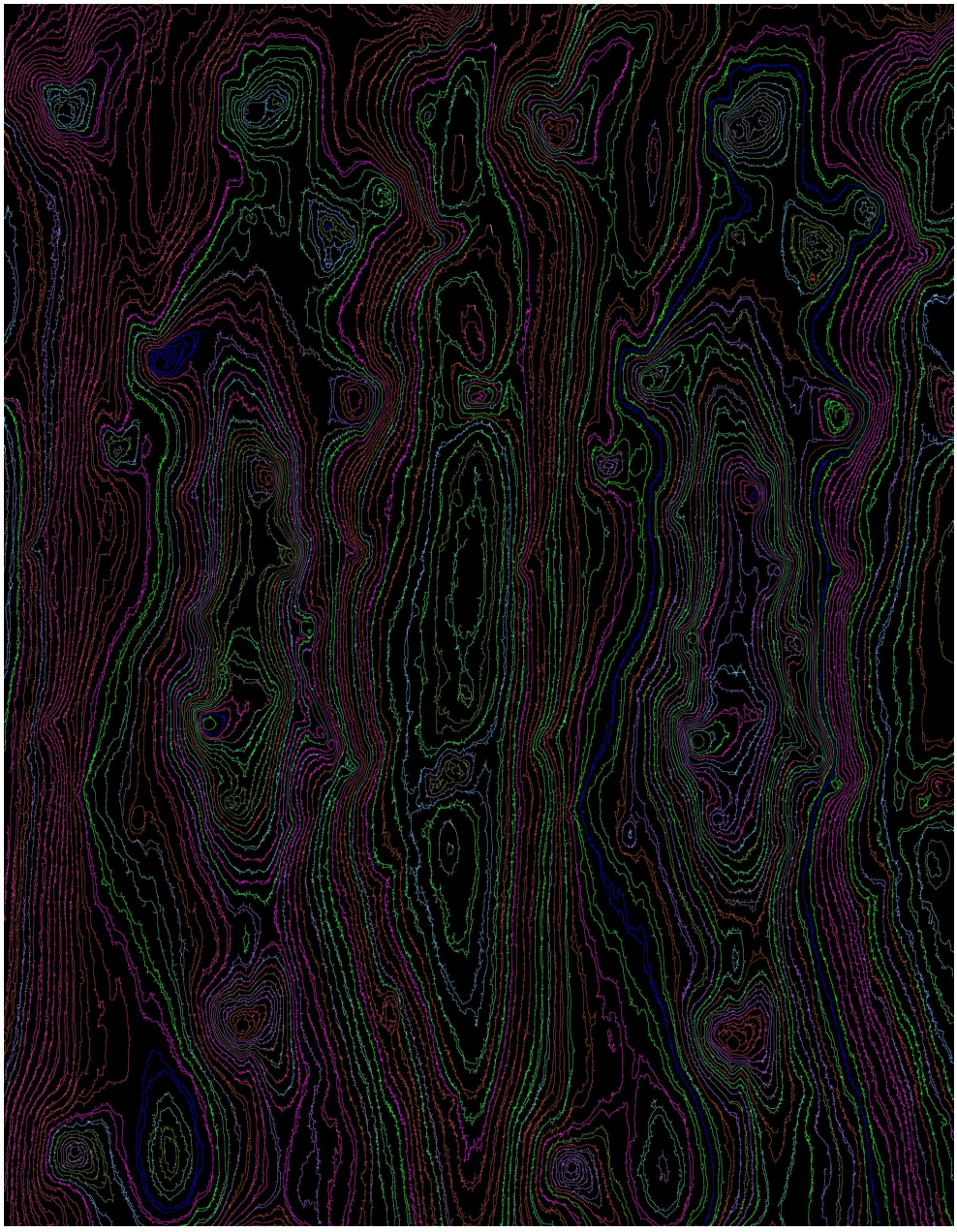
Tillegg A

Bilder

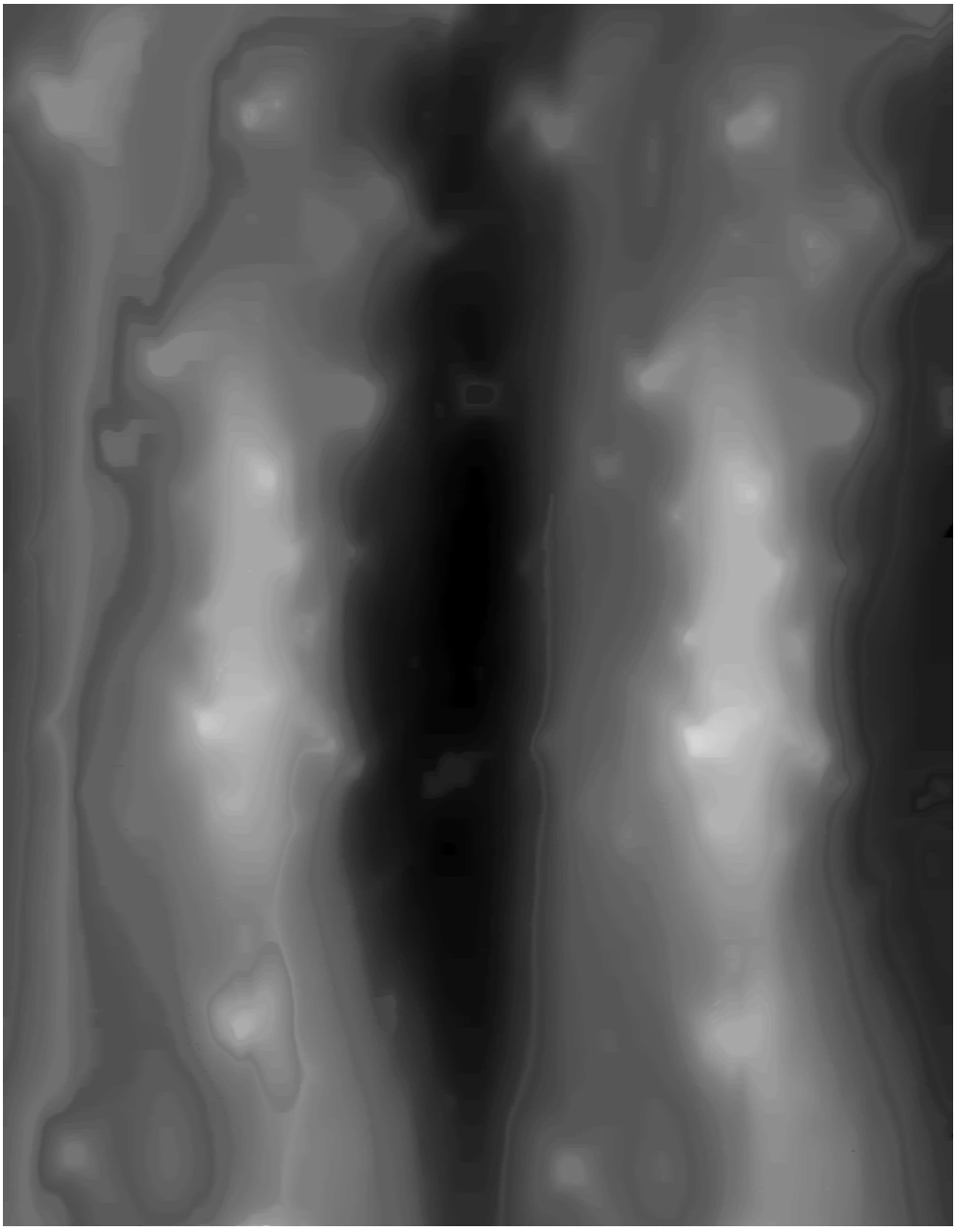
Figur A.1: Finerplaten



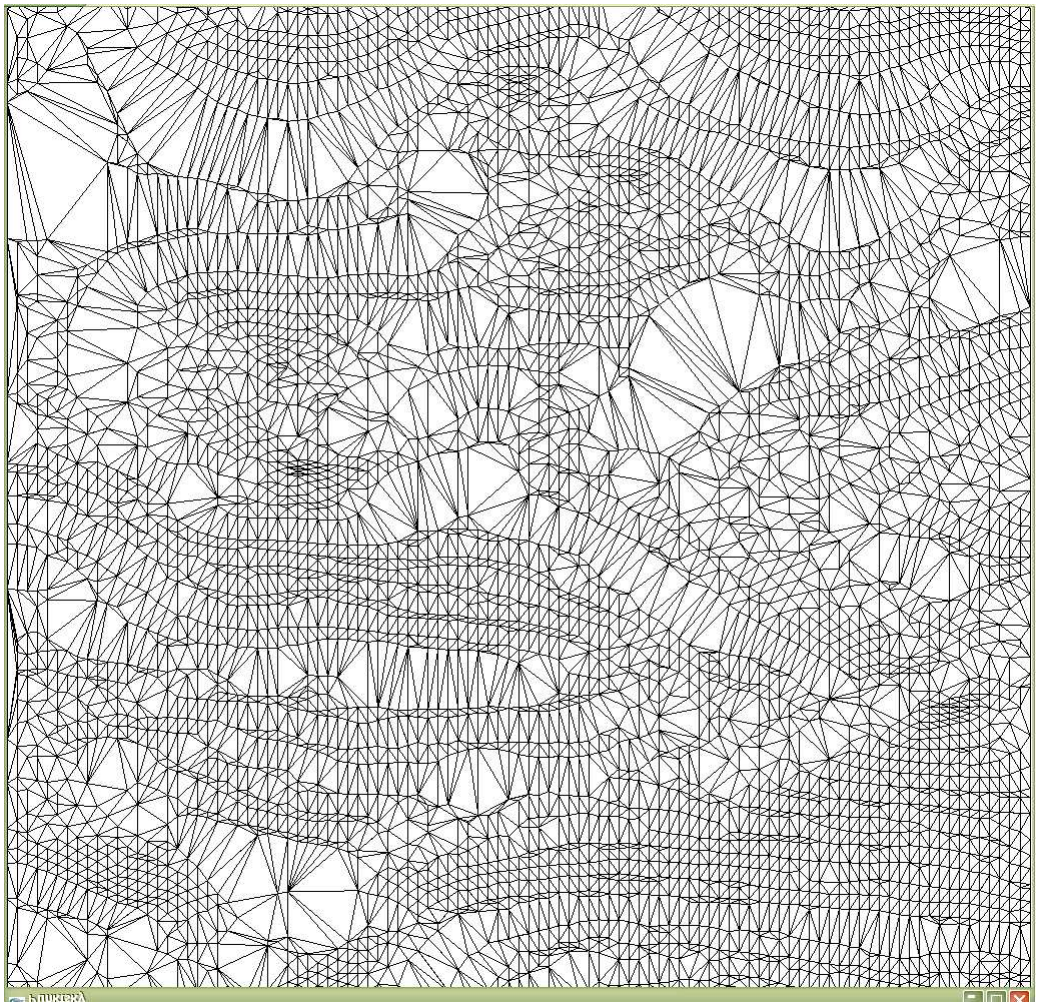
Figur A.2: Resultatet fra adaptiv terskling



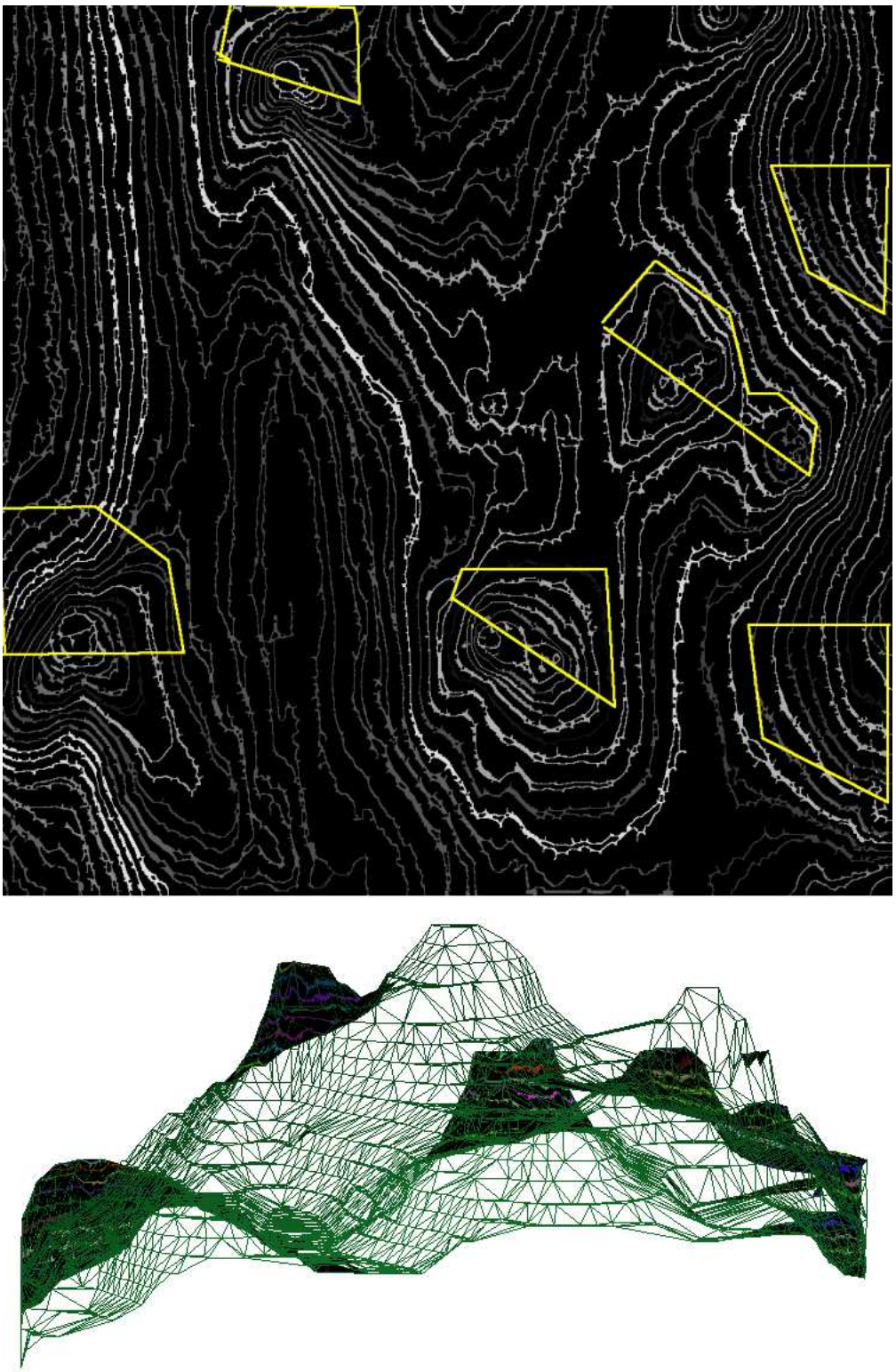
Figur A.3: Kotekartet



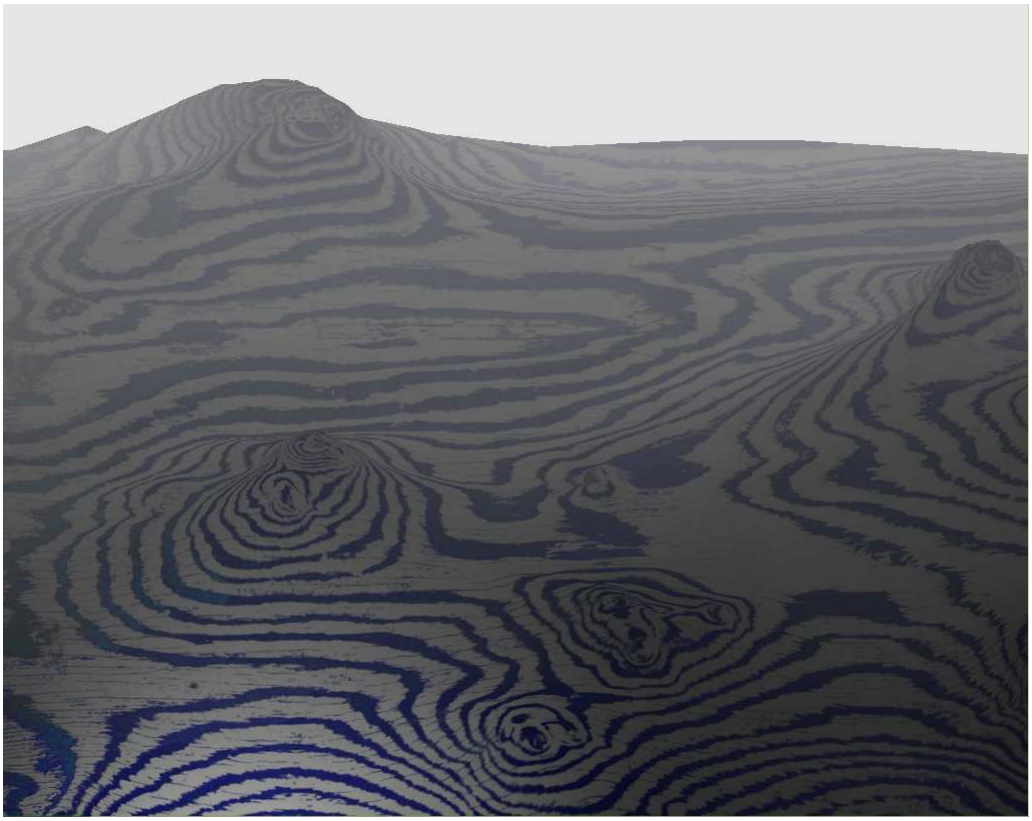
Figur A.4: Høydebilde



Figur A.5: Delauney triangulering



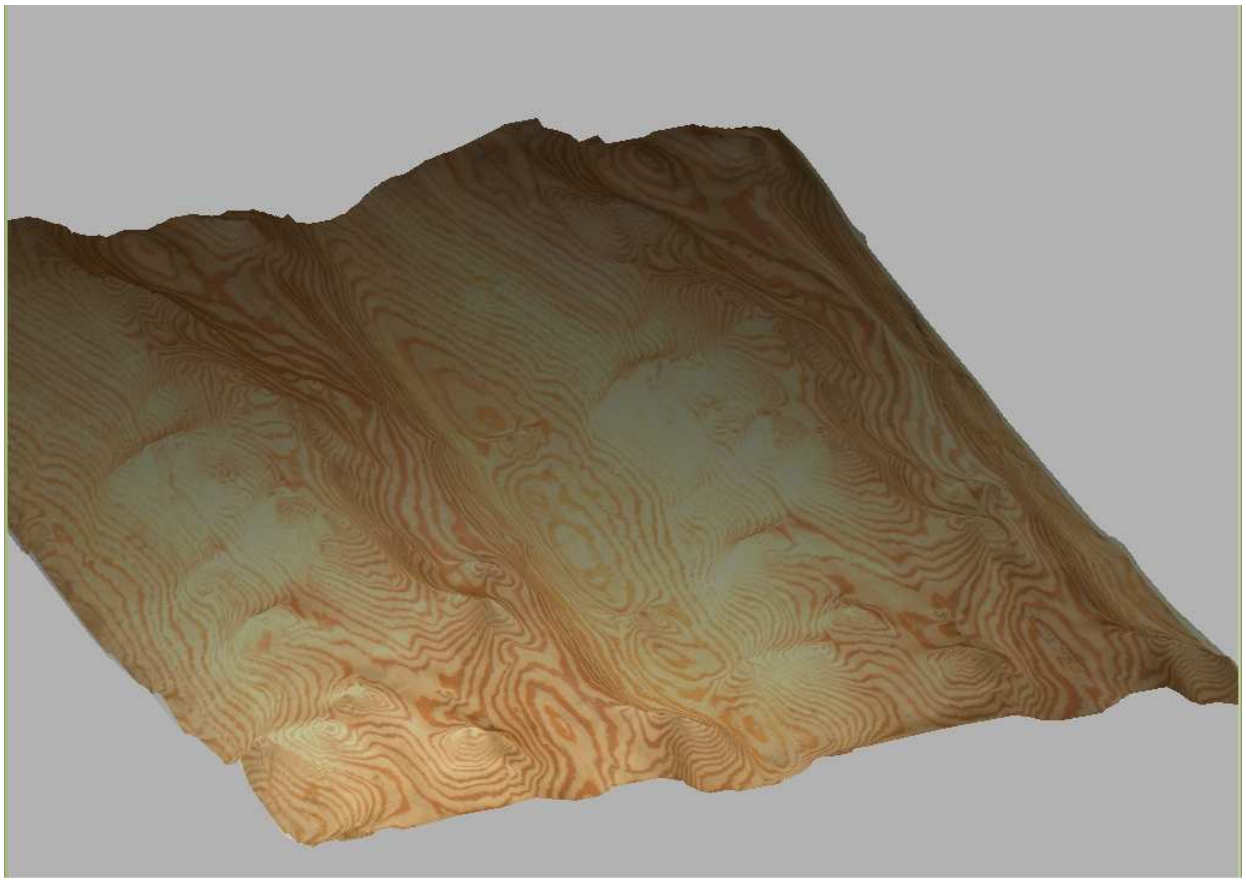
Figur A.6: Forside culling



Figur A.7: Porcelainscape

Figur A.8: Lystekstur

Figur A.9: Modell med tåke



Figur A.10: Oversikt over hele terrenget med lystekstur

Figur A.11: Nærbilde av et lite utsnitt med høy oppløsning fra 3Dtre

Figur A.12: Nærbilde av et lite utsnitt i SIM Scenery

Tillegg B

Bruksanvisning 3Dtre

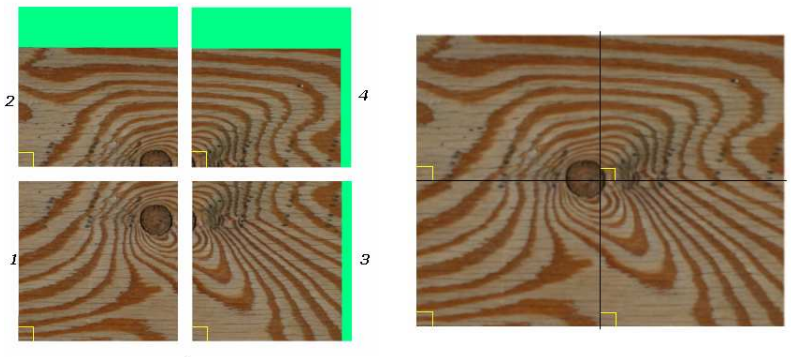
B.1 Hva som må være klart

B.1.1 Kotekart

Et farget kotekart må være klargjort. Kotekartet må være i .ppm format, og hver sammenhengende kote må ha en og bare en farge.

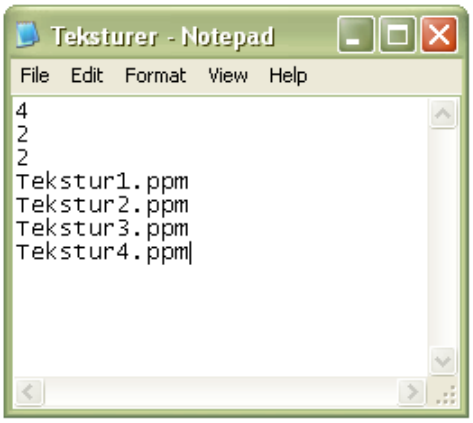
B.1.2 Teksturer

For å få riktig teksturmapping må alle teksturer klippes riktig til på forhånd. For å få minst mulig unaturlige skjøter må teksturene lages med litt overlapp. Det må nøyaktig være 6.25 prosent overlapp. I praksis vil dette si at for hver tekstur på 1024×1024 må 960 piksler være egen tekstur, og 64 piksler være overlapp. Først må antall teksturer bestemmes. Anta at det kotekartet som skal brukes er 2000×1800 piksler. For å få omtrent samme oppløsning på teksturen som på kotekartet, behøves det nå 4 teksturer, hvor hver av disse skal dekke 1000×900 piksler på kotekartet. Ettersom hver kant skal ha 64 piksles overlapp må hver av disse teksturene skaleres ned til 960×864 piksler. Etter at teksturene er blitt skalert, må overlappet legges til. Størrelsen på kanvas må skaleres til 1024×1024, uten at teksturen skaleres med. Når dette er gjort må teksturene legges nede i venstre hjørne som vist på figuren. Den overflødige delen av kanvas må nå fylles med overlappingstekstur de stedene hvor de overlapper med andre teksturer. NB: Dette må gjøres selv om det bare er en tekstur.



Figur B.1: Teksturer som er tilpasset

For å forenkle det senere arbeidet kan teksturene lagres i en tekst fil. Først



Figur B.2: Tekstfil med teksturer

linje inneholder antall teksturer. Andre linje er hvor mange det er horisontalt, og tredje linje er antall vertikalt. De neste linjene inneholder filnavnene. Rekkefølgen er ned-opp, venstre-høyre.

B.1.3 Eventuelt

Andre startdokumenter kan være polygonliste eller høydekart. Disse kan lages i programmet.

B.2 Kjøring av programmet

Programmet starter i Windows Xp ved å dobbelklikke på .exe filen. Første kommando er :

>Velg type inndata: 1=Kurvekart, 2=Hoydebilde, 3=Polygonliste

Alternativ 1 er for inndata av kotekartet. Dette er typisk første gangen programmet blir kjørt. Når programmet kjøres første gang lagres polygonene i en tekstfil slik at ved senere kjøringer slipper kotekartet å bli lest. Mulige utdata for dette alternativet er 3D modell, høydekart, og polygonliste.

Alternativ 2 er et høydebilde. Dette er typisk et bilde du allerede har laget i programmet. Ved å velge dette alternativet blir utdata en tekstfil med regulære gitterpunkter som kan importeres i SIM Creator.

Alternativ 3 er en tekstfil med polygoner som typisk er laget i dette programmet. Utdata er en 3D modell.

B.2.1 Fra kurvekart til høydebilde

>Velg type inndata: 1=Kurvekart, 2=Hoydebilde, 3=Polygonliste

Velger alternativ 1:

>Skriv inn navnet paa filen:f.eks minFil.txt eller kurvekart.ppm

Nå må inndatafil skrives inn. Dette er kotekartet som er i .ppm format.

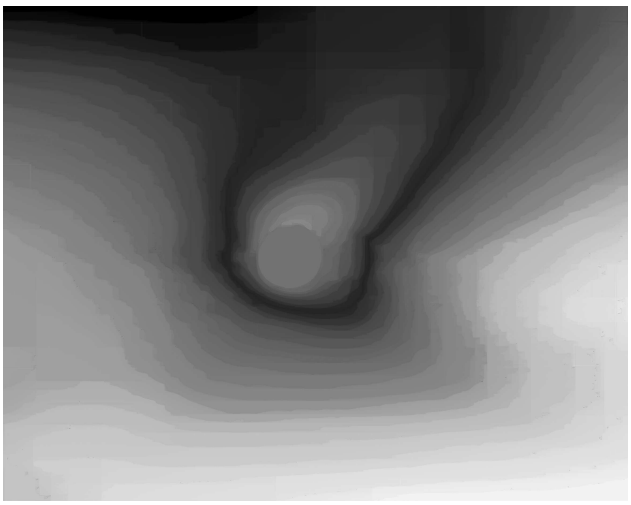
>Velg resultat: 1= Høydebilde, 2=Polygonliste, 3=Irregulær polygonmodell.

Alternativ 1 lagrer et .pgm bilde hvor alle kotehøydene er interpolert.
Alternativ 2 lagrer en polygonliste.
Alternativ 3 lagrer polygonliste i tillegg til å vise 3D modellen.

Velger alternativ 1:

>Skriv inn navnet på resultatbildet: f.eks myHeights.pgm. NB: *.pgm

Her må navnet på filen som høydebildet skal lagres som skrives. Det behøver ikke være en eksisterende fil. Filen må være av .pgm format. Når filnavn er valgt blir det litt ventetid. Ventetiden kommer an på hvor stort kotekartet er. Når bildet har blitt lagret avsluttes programmet og høydebildet er lagret.



Figur B.3: Høydebilde

Høydebildet blir speilvendt i forhold til kotekartet. Dette er fordi det blir lest fra fil og skrevet til skjerm speilvendt. Når høydebildet er speilvendt blir punktene som diskretiseres senere vist riktig på skjermen.

B.2.2 Fra høydebilde til regulære punkter

>Velg type inndata: 1=Kurvekart, 2=Hoydebilde, 3=Polygonliste

Velger alternativ 2:

>Skriv inn navnet paa filen:f.eks minFil.txt eller kurvekart.ppm

Nå må inndatafil skrives inn. Dette er høydebildet som er i pgm format.

>Skriv inn navnet på resultatbildet: f.eks myPoints.txt. NB: *.txt

Skriver inn hva jeg vil lagre punktlisten som.

>Hvor store steg?

Skriver inn hvor stor avstand det skal være mellom hvert punkt som blir samplet. 10 vil typisk være passe, mens 1 lagrer punkter fra hver eneste piksel.

B.2.3 Fra kurvekart til irregulær polygonmodell

>Velg type inndata: 1=Kurvekart, 2=Hoydebilde, 3=Polygonliste

Velger alternativ 1:

>Skriv inn navnet paa filen:f.eks minFil.txt eller kurvekart.ppm

Nå må inndatafil skrives inn. Dette er kotekartet som er i .ppm format.

>Velg resultat: 1= Høydebilde, 2=Polygonliste, 3=Irregulær polygonmodell.

Velger alternativ 3:

>Hva vil du lagre polygonlisten som? f.eks polygoner.txt.

Polygonlisten må lagres til en tekstfil. Når polygonene ligger på en tekstfil blir mye tid spart om programmet skal kjøres på nytt. Dette er fordi det er triangulerin-gen som er flaskehalsen.

>Hvor store skritt i x-retning(int)

Her må du oppgi hvor stor avstanden mellom punktene skal være. For et stort kotekart bør avstanden være ganske stor, mellom 20 og 50. Metodene er begrenset til ca 100 000 trekanter, så dette må tilpasses. For et mindre kotekart, kan avstanden være ca 10. Nå er det noe ventetid sterkt avhengig av antall punkter.

>Ortho=1, Perspektiv=2.

Alternativ 1 resulterer i en 3D modell med parallell perspektiv. Denne kan man se fra alle sider, men kan vanskelig bevege seg i den. Alternativ 2 resulterer til en 3D modell med perspektivisk perspektiv. Her havner man inni modellen.

>Hvordan lages teksturene til? 1:tekstfil, 2: Skriv inn her

Dersom alternativ 1 velges må tekstfilen med teksturene skrives inn. Alternativ 2

>Hvor mange teksturer er det til sammen?(nb: en tekstur kan være max 1024× 1024 piksler)

Her skrives antall teksturer inn. fra eksempelet ovenfor vil det være 4.

>Hvor mange teksturer er det loddrett?(nb: en tekstur kan være max 1024× 1024 piksler)

Her skrives antall teksturer i vertikal retning. 2 fra eksempelet over.

>Hvor mange teksturer er det vannrett?(nb: en tekstur kan være max 1024× 1024 piksler)

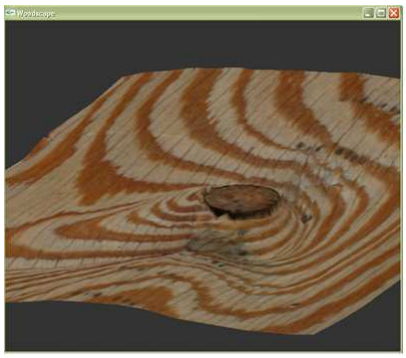
Her skrives antall teksturer i horisontal retning. 2 fra eksempelet over.

>Skriv de inn topp-bunn, venstre-hoyre.

Skriv inn en og en tekstur, trykk 'enter', vent og skriv inn neste.

>Vil du ha tåke? 1=ja, 2=nei

Om ønskelig kan modellen tåkelegges. 1 er ja, og 2 er nei. Etterpå kommer modellen opp. den kan nå styres med tastene vist nedenfor. Dersom modellen ikke er synlig med det først kan det være nødvendig å forandre synspunktet litt.



Figur B.4: Tekstur

B.3 Navigering

Det går an å navigere i modellen i tillegg til å flytte tåken nærmere og fjernere(bare om den er valgt).

Tast	Operasjon
x	Flytter hvor du står og hvor du ser i -x retning
X	Flytter hvor du står og hvor du ser i +x retning
y	Flytter hvor du står og hvor du ser i -y retning
Y	Flytter hvor du står og hvor du ser i +y retning
z	Flytter hvor du står og hvor du ser i -z retning
Z	Flytter hvor du står og hvor du ser i +z retning
i	Flytter hvor du ser i -x retning. (Vinkel blir endret)
I	Flytter hvor du ser i +x retning. (Vinkel blir endret)
j	Flytter hvor du ser i -y retning. (Vinkel blir endret)
J	Flytter hvor du ser i +y retning. (Vinkel blir endret)
k	Flytter hvor du ser i -z retning. (Vinkel blir endret)
K	Flytter hvor du ser i +z retning. (Vinkel blir endret)
g	lyser tåke.

G	mørkere tåke.
d	lettere tåke.
D	tykkere tåke.
s	tåken starter nærmere-
S	tåken starter fjernere.
e	tåken slutter nærmere-
E	tåken slutter fjernere.

Tillegg C

Bruksanvisning SIM basert program

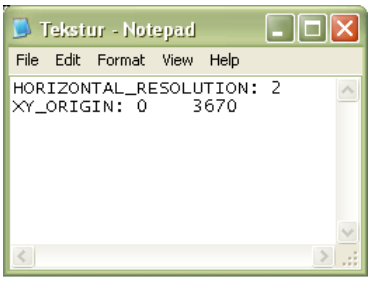
C.1 Hva som må være klart

C.1.1 Punkter

SIM Scenery bruker regulært gitter, og inndata er henholdvis regulært diskretiserte punkter. Disse punktene kan bli laget i 3Dtre. Inndata er først et kotekart. Dette blir laget til et høydekart. Til sist er inndataen høydekartet, og en punktliste er resultatet.

C.1.2 Tekstur

SIM Scenery kan bruke .jpg teksturer av vilkårlig størrelse. I tillegg til teksturen behøves en .hdr fil med samme navn som teksturen.



Figur C.1: hdr fil

Den første linjen skal inneholde teksturoppløsningen. Tallet skal være antall kvadratmeter som blir dekket av en teksturpiksel. Eksempelvis vil teksturen her ha halvparten så høy oppløsning som terrenget. Den andre linjen bestemmer origo. I forhold til de produserte punktene må det første tallet være 0, og det andre maks y-verdi.

C.1.3 Database

Før dataene kan brukes i programmet må de legges til i en database. Dette gjøres i SIM Creator. Først må en ny database opprettes. Etterpå velges 'import elevation data' som er punktlisten, og 'import texture data' som er teksturen. Når dette er

blitt gjort er databasen klar til bruk. Creator har innebygd hjelpefunksjon for mer detajert forklaring.

C.2 Kjøring av programmet

Når databasen er blitt laget kan programmet startes. Inndata vil være navnet på databasen. Modellen kan transleres, skaleres og roteres. I tillegg finnes det ulike renderingtyper ved å klikke på høyre museknapp. Pilen på hovedmenyen gjør det mulig å lagre skjermbilder. Dersom pilalternativet velges og det klikkes på modellen vil et skjermbilde lagres. Spørsmål kommer da opp på kommandolinje om bredde, høyde og filnavn på bildet. Lagring av store bilder vil ta noe tid.

Etter bildet har blitt klikket på må hånden på hovedmenyen velges for å komme tilbake i normalmodus.

Figur C.2: SIMScenery