Korte Kurver på Krumme Flader

Mads Ohm Larsen og Kasper Nybo Hansen
{omega, nybo}@diku.dk

10. juni 2010

Resumé

Vi finder en approksimation til den korteste diskrete kurve henover en flade defineret i $\mathbb{R}^3$. Overfladen er tilnærmet med et trekantsnet. Da fladen ikke kendes analytisk udregnes en initial vej vha. én grafalgoritme. Herefter korrigerer vi den fundne vej, til en tilnærmelse til den korteste diskrete kurve.

To forskellige grafalgoritmer er implementeret, Dijkstras korteste vej-algoritme og Fast Marching Metoden (FMM).

Eksperimentelle resultater viser at de approksimerede kurver er meget tæt på de analytiske udregnede.

Abstract

We find an approximation to the shortest discrete curve on a surface defined in $\mathbb{R}^3$. The surface is approximated by a triangulated mesh. As the surface is not known analytically we calculate the initial path with a graph algorithm. We then correct the initial path to an approximation of the shortest discrete curve.

Two different graph algorithms have been implemented, Dijkstra's shortest path-algorithm and Fast Marching Method (FMM).

Experimental results show that the approximated curves is close to the analytical calculated path.

Indhold

Kapitel 1

Indledning

Denne rapport er skrevet ifbm. et bachelorprojekt på Datalogisk Institut Københavns Universitet (DIKU). Bachelorprojektet omhandler emnet korte kurver på krumme flader. Forfatterne er Mads Ohm Larsen og Kasper Nybo Hansen. Vejleder er Knud Henriksen, lektor ved DIKU.

1.1 Baggrund

Langt de fleste flader kendes ikke ved deres analytiske repræsentation, men kan tilnærmes med et net af trekanter.

Udbredningen af 3D-skanninger, som dem foretaget ifbm. medicinske 3D-skanninger, satellitbilleder af jordens overflade m.fl., har gjort at kravet om at kunne udregne afstande på diskrete overflader repræsenteret ved et trekantsnet, er blevet meget aktuelt.

Afstande på krumme flader har f.eks. stor betydning for statistisk dataanalyse [1]. Eksempelvis vil omrids af hænder udspænde en krum flade, ligeså med billeder af lungevæv, og endeligt vil computerstyrede karakterer i et computerspil ofte skulle navigere på ikke-flade overflader.

Et af anvendelsesområderne er f.eks. udregning af afstande til statistisk dataanalyse på f.eks. skanningsbilleder af lunger. Medicinske skannere kan i dag skanne en persons lunge og danne et trekantsnet, som repræsenterer denne lunge. Ved at udregne afstande på overfladen af lungen, får forskere adgang til data, de ellers ikke ville have haft, og kan derved, muligvis, se statistiske sammenhænge de ellers ikke ville haft mulighed for.

1.2 Målsætning

Målet med denne rapport er at beskrive teorien og analysen bag en implementeret algoritme.

Givet et trekantsnet og to knuder på nettet, finder den implementerede algoritme en approksimation til den korteste kurve mellem de to givne knuder, samt afstanden denne kurve tilbagelægger. Trekantsnettets knuder tilhører, i denne rapport, det tredimensionale rum $\mathbb{R}^3$.

1.3 Afgrænsning

Vi har lagt en restriktion på vores trekantsnet, således dette skal være mangfoldigt (eng. manifold). I afsnit 3.5 er der en beskrivelse af mangfoldighed. Når vi senere bruger ordet mangfoldigt, mener vi altid 2-mangfoldigt. Endvidere kræver vi at vores figurer kan indeholde et volumen. Dette kan også ses ud fra vores definition af mangfoldighed i samme afsnit.

1.4 Resultat

Vi har udviklet et program til at visualisere 3D-objekter repræsenteret ved et trekantsnet. Programmet kan udregne og visualisere korte diskrete kurver på trekantsnettes overflade.

Programmet understøtter rotation af objektet i alle tre akser i det tredimensionale rum, zooming samt foretage valg af start og slut knuder, det er interaktivt og udregner vejen i sandtid.

Programmet er skrevet i C++ på OS X, men oversætter også på andre platforme[1]. Til at visualisere vores objekter og tegne vores kurver gør vi brug af grafikbiblioteket OpenGL.

Kildekoden er vedlagt denne rapport som en `tar.gz`-fil. Bibliotheksstrukturen i `tar`-filen er som følger

Mappe	Bestanddele
`/`	makefile m.fl.
`figures/`	test-figurer
`src/`	kildekoden
`src/classes/`	klasser
`src/headers/`	header-filer
`src/matrix/`	matrix biblioteket RasterMan
`src/ply/`	`ply`-parser

Programmet kan oversættes med

```
cmake .
make clean all
```

[1] Programmet er desuden oversat og testet på en maskine med Ubuntu 10.04 Lucid Lynx

Med programmet er vedlagt en række testfiler, heriblandt den berømte Stanford Bunny[2].

Et eksempel på en kørsel af programmet er

```
pathfinder -f figures/sphere.ply
```

[2]Kan findes her http://graphics.stanford.edu/data/3Dscanrep/

Kapitel 2

Matematisk notation

p	Knude
θ, α, β	Vinkel
$\varphi(p, q)$	Euklidisk afstand mellem punkterne p og q
$\delta(p, q)$	Geodætisk afstand fra p til q, på en analytisk kendt overfalde
Γ	Kurve
γ	Analytisk kendt kurve
Γ_i	Kurve efter i korrigeringer
$p_{j,i}$	Knude j i en kurve efter den i'te korrektion
θ_{p_j}	Den totale vinkelsum for en knude p_j
$\Lambda(\Gamma_i)$	Længden af kurven Γ_i
$\mathcal{F}_{p_j}$	Fladen, som udspændes af knude p_j og dens naboer
$\mathcal{F}^r_{p_j}$	Højreside af fladen som udspændes af knude p_j og skæres igennem af p_{j-1} og p_{j+1} og dens naboer
$\mathcal{F}^l_{p_j}$	Venstreside af fladen som udspændes af knude p_j og skæres igennem af p_{j-1} og p_{j+1} og dens naboer
θ_r	Vinkelsum af $\mathcal{F}^r_{p_j}$
θ_l	Vinkelsum af $\mathcal{F}^l_{p_j}$

Kapitel 3

Teori

I dette afsnit introduceres den teori vi vil gøre brug af i analyse afsnittet. Sammen med teorien introduceres den notation vi bruger igennem resten af rapporten.

En del af definitionerne er hentet fra [5].

3.1 Afstande i n dimensioner

Den korteste afstand $\varphi(p,q)$ mellem de to punkter, $p,q \in \mathbb{R}^n$ er den lige linie som forbinder dem. Lad $p = (p_1, p_2, \ldots, p_n)$ og $q = (q_1, q_2, \ldots, q_n)$, da er afstanden, som bekendt, givet ved

$$\varphi(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

og kaldes den euklidiske afstand. Specifikt er den korteste afstand mellem to punkter $p,q \in \mathbb{R}^3$ givet ved

$$\varphi(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \tag{3.1}$$

3.2 Afstande på overflader

Den kurve, som har den korteste længde mellem to punkter, og samtidig ligger på overfladen af et tredimensionalt objekt, kan ikke altid udregnes som en euklidisk afstand. Hvis objektets analytiske repræsentation er kendt, kan afstanden muligvis udregnes analytisk. F.eks. er afstanden mellem to punkter på overfladen af en kugle, defineret ved et udsnit af kuglens storcirkel og kan udregnes som følger.

Lad r være kuglens radius, og lad $p, q \in \mathbb{R}^3$ være punkter på kuglens overflade. Da er afstanden mellem punkterne p og q på kuglens overflade defineret som

$$\delta(p, q) = r \cdot \cos^{-1}(\widehat{p} \cdot \widehat{q}) \tag{3.2}$$

, hvor $\widehat{p} \cdot \widehat{q}$ er prikproduktet mellem de to enhedsvektorer $\widehat{p}$ og $\widehat{q}$.

Afstande, som er defineret på overflader, kaldes geodætiske afstande og kurven fra p til q kaldes en geodætisk kurve.

3.3 Definition af geodætisk kurve

Den geodætiske kurve er den korteste kurve mellem to punkter på en given overflade.

Hvor den euklidiske afstand var den rette linie mellem to punkter $p, q \in \mathbb{R}^n$, er den geodætiske afstand, den korteste afstand mellem to punkter *henover* en flade i $\mathbb{R}^n$.

I dette afsnit gennemgår vi teorien for punkter i $\mathbb{R}^3$. Først gennemgås teorien for analytisk kendte overflader kaldt glatte overflader, derefter for diskrete overflader.

Definition 1. *En geodætisk kurve γ på en flade, er en generalisering af den rette linie til kurvede rum. Lokalt er det den korteste kurve mellem to punkter på en krum overflade. [3]*

En geodætisk kurve er altså en kurve, hvor der for alle $\epsilon > 0$, og to punkter, t_1 og t_2, på kurven, gælder at for et $\tau > 0$ så $\varphi(t_1, t_2) < \tau \Rightarrow \delta(\gamma(t_1), \gamma(t_2)) < \epsilon$ hvor $\epsilon, \tau \in \mathbb{R}$.

Den geodætiske kurve mellem to punkter på en plan i $\mathbb{R}^3$ er den rette linie mellem punkterne og længden af linien er den euklidiske afstand mellem punkterne, jvnf. afsnit (3.1).

Det er sværere at finde den geodætiske kurve på en arbitrær flade i $\mathbb{R}^3$. Her bliver vi nød til at lave en generalisering af rette linier til ikke-euklidiske kurvede rum[1] [10].

Det kan vises at følgende fire egenskaber er ækvivalente for en kurve γ på en glat flade [2, 5]:

1. γ er en geodætisk kurve
2. γ er den lokal korteste kurve
3. γ'' er parallel til faldenormalen

[1] Tænk f.eks. på en trekant på overfalden af en kugle, her gælder Pythagoras' læresætning ikke

4. γ har forsvindingsgeodætiskkrumning $\kappa_g(p) = 0 \quad \forall p \in \gamma$

Her er

$$\kappa_g(p) = \frac{2\pi}{\theta}\left(\frac{\theta}{2} - \theta_r\right)$$

, hvor $\theta = \theta_l + \theta_r$ er summen af vinkler i punkter p og θ_l henholdvis θ_r er venstre- henholdsvis højresidens vinkel [5].

Ud fra punkt 4 kan det konkluderes at geodætiske kurver er så rette som de kan være.

Coblenz et. al. [3] finder analytisk, ved brug af Euler-Larange ligningen på riemannske rum, at den geodætiske kurve på en plan er

$$\frac{dy}{dx} = c$$

, hvor c er konstant, altså en lige linie. Dette betyder at geodætiske kurver i planen er en ret linie, jvnf. tidligere postulat.

De finder endvidere at den geodætiske kurve på en kugle er givet ved

$$Ay - Bx = z$$

Denne ligning betyder at de punkter der er på den geodætiske kurve mellem to punkter på en kugle, er på en plan, som indeholder origo. Den geodætiske kurve vil da være skæringen mellem denne plan og kuglen, altså en storcirkel, se også (3.2).

Det er noget sværere at finde den geodætiske kurve på en mere kompleks figur. Euler-Lagrange ligningen er kun brugbar, hvis vores flade er glat, og vi kan finde en analytisk fremstilling for den, men oftest er dette ikke tilfældet.

Den korteste kurve mellem to punkter på en *diskret* overflade, kaldes en *diskret* geodætisk kurve.

Der findes to typer af diskrete geodætiske kurver [3], glatteste og korteste.

Iflg. K. Polthier et. al. [5] er de forskellige på følgende måde.

- En diskret geodætisk kurve der ikke indeholder en knude, er både glattest og kortest
- En glatteste diskret geodætisk kurve der går igennem en sfærisk knude er ikke lokalt kortest
- Der eksisterer en familie af korteste diskrete geodætiske kurver igennem en hyperbolsk knude, kun én af dem er glattest.

Vi vil i senere afsnit definere hvad en sfærisk knude og hyperbolsk knude er.

3.4 Definition af et trekantsnet

Definition 2. *Lad der være givet tre punkter i $\mathbb{R}^3$, som ikke ligger på en ret linie. Da vil disse tre punkter danne en trekant i $\mathbb{R}^3$.*

De tre punkter vil altid kunne indeholdes i én plan, hvilket bevirker at vi kan benytte (3.1) til at udregne afstanden mellem to vilkårlige punkter på trekanten.

Definition 3. *Lad S være en overflade der er udspændt af sammensatte trekanter. Vi kalder denne overfalde for et trekantsnet.*

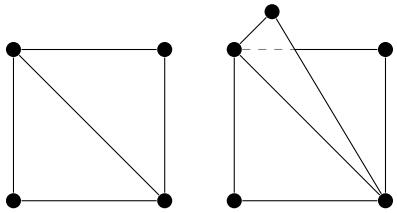
Lad trekantsnettet bestå af et endeligt antal trekanter der udgøres af mængden $\mathcal{F}$. Da gælder følgende om vores trekantsnet:

- Ethvert punkt $p \in S$ tilhører mindst en trekant $f \in \mathcal{F}$
- Ethvert punkt $p \in S$ tilhører endelig mange trekanter $f \in \mathcal{F}$
- Skæringen mellem to trekanter $f, g \in \mathcal{F}$ er tom eller består af én fælles kant og to delte knuder
- Der er isometri imellem enhver trekant $f \in \mathcal{F}$ og en trekant i $\mathbb{R}^2$

3.5 Definition af mangfoldighed

Definition 4. *Et trekantsnet er mangfoldigt hvis og kun hvis enhver kant i trekantsnettet er delt mellem præcis to trekanter.*

Et par simple figurer, som vi ikke vil klassificere som mangfoldige, kan ses på figur 3.1

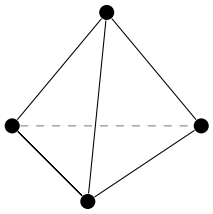


(a) De ydre kanter af hver trekant, er kun med i én trekant

(b) Kanten i midten er med i alle tre trekanter

Figur 3.1 – Disse to figurer er ikke mangfoldige

Hvis alle kanter skal være delt mellem præcis to trekanter, er den simpleste figur vi kan fremstille os, et tetraeder, altså en pyramide med fire sider.



Figur 3.2 – Et tetraeder, det simpleste trekantsnet, som er mangfoldigt

3.6 Definition af en knude

Definition 5. *En knude i* $\mathbb{R}^3$ *er et punkt* $p = (x, y, z)^T$*, som er indeholdt i et trekantsnet.*

Lad knuden p være en knude i vores trekantsnet. Da er p indeholdt i en række trekanter $\mathcal{F} = \{f_1, \cdots, f_n\}$. Lad α_{f_i} være vinklen i trekant f_i ved punktet p. K. Polthier et. al. [5] definerer da den samlede vinkelsum for p ved

$$\theta_p = \sum_{i=1}^{n} \alpha_{f_i}$$

Ud fra vinklen θ_p kan p kategoriseres som

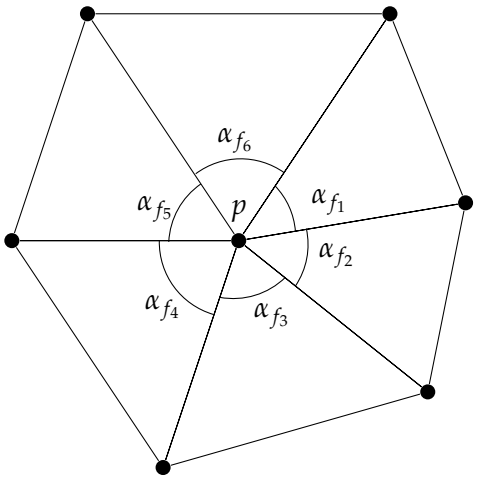
$$p = \begin{cases} \text{Sfærisk} & \text{hvis } \theta_p < 2\pi \\ \text{Euklidisk} & \text{hvis } \theta_p = 2\pi \\ \text{Hyperbolsk} & \text{hvis } \theta_p > 2\pi \end{cases}$$

Kald den flade som de tilstødende trekanter i $\mathcal{F}$ og punktet p danner for $\mathcal{F}_p$. Da kan $\mathcal{F}_p$ udfoldes isometrisk og danne en flade i en euklidisk plan i $\mathbb{R}^2$ [5].

Isometrien sikrer os at afstanden ikke ændres ved dimensionsskift.

En knude i trekantsnettet har en række naboer. Naboerne er defineret ved at være forbundet med knuden via. en direkte vej. Dvs. en knudes naboer er de andre knuder, som knuden deler kanter med.

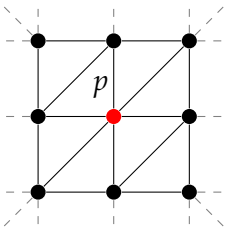
De følgende illustrationer viser hvordan knuder i de forskellige kategorier ser ud, samt deres udfoldede repræsentation i $\mathbb{R}^2$.



Figur 3.3 – Vinklerne i samtlige trekanter, som p er med i, udgør θ_p

3.6.1 Euklidisk knude

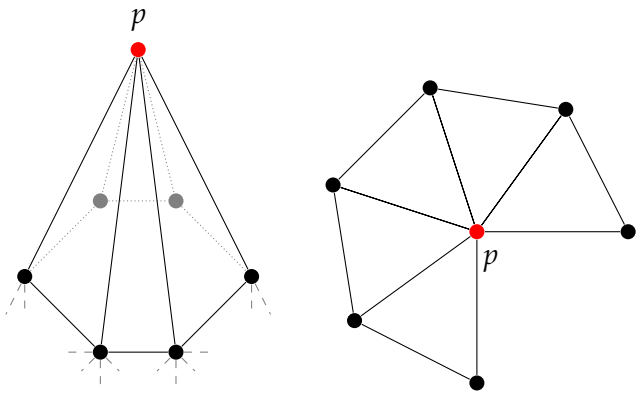
En euklidisk knude, er en knude som har en vinkelsum $\theta_p = 2\pi$. Det specielle ved disse knuder er at $\mathcal{F}_p$ udgør en flade i et tredimensionalt rum. Den udfoldede repræsentation og den oprindelige er således identisk.



Figur 3.4 – Den røde knude er euklidisk, og danner en plan i $\mathbb{R}^3$. Den udfoldede repræsentation er identisk med ovenstående, og er ikke illustreret.

3.6.2 Sfærisk knude

En sfærisk knude, er en knude med en vinkelsum $\theta_p < 2\pi$. Iflg. afsnit 3.3 vil en kurve der bevæger sig igennem en sfærisk knude aldrig være den lokalt korteste. Den sfæriske knude og den udfoldede flade, $\mathcal{F}_p$, kan ses på figur 3.5.

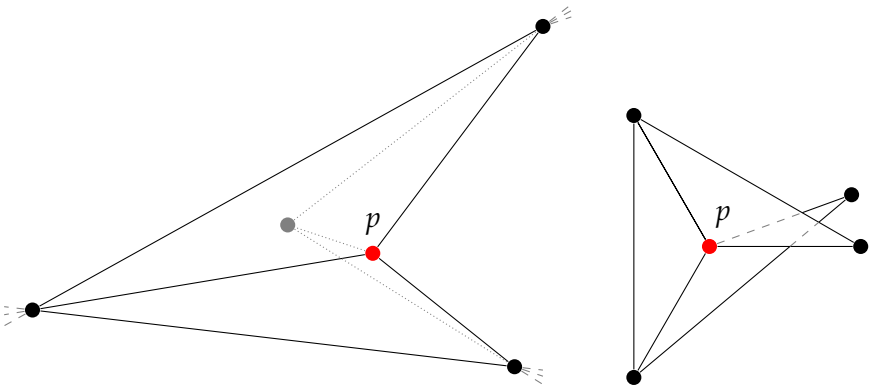


(a) p er en sfærisk knude (b) Figur (a) projekteret ned i $\mathbb{R}^2$

Figur 3.5 – Den røde knude er sfærisk og har en vinkel $\theta_p < 2\pi$.

3.6.3 Hyperbolsk knude

En hyperbolsk knude, er en knude med vinkelsum $\theta_p > 2\pi$. Den udfoldede flade vil overlappe når denne projekteres ned i $\mathbb{R}^2$, som det ses på figur 3.6(b).



(a) p er en hyperbolsk knude (b) Figur (a) projekteret ned i $\mathbb{R}^2$

Figur 3.6 – Den røde knude er hyperbolsk og har $\theta_p > 2\pi$. Her sker et overlap når fladen bliver projekteret ned i $\mathbb{R}^2$. Dette skyldes at vinklen er større end 2π.

3.7 Definition af vej og kurve

Vi skelner mellem veje og kurver på følgende måde

Definition 6. *En vej er afstanden fra p til q gennem trekantsnettets knuder.*

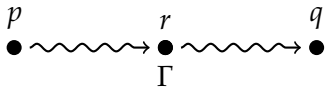
Definition 7. *En kurve er en korrigeret vej, som ikke behøver være bundet til trekantsnettets knuder.*

For både veje og kurver gælder at de betegnes med Γ og består af n knuder p_i for $i = 1 \dots n$. Knuderne p_i, hvor $1 < i < n$, kaldes de interne knuder af Γ.

3.7.1 Optimal delstruktur

Både veje og kurver udviser optimal delstruktur.

Lad $\Gamma = \{p, \dots, r, \dots, q\}$ være den korteste kurve mellem knuderne p og q, som tilhører vores trekantsnet og lad r være en knude på kurven Γ. Den del af Γ, som ligger mellem p og q er den korteste kurve mellem p og q. Ligeledes må den kurve, der går fra r til q være den korteste fra r til q. Hvis dette ikke var tilfældet, ville vi kunne lave en kortere kurve fra r til q, men denne delkurve ville da også kunne gøre kurven fra p til q kortere. Dette strider mod antagelsen om at Γ var den korteste kurve.



Figur 3.7 – Den korteste kurve fra p til q, er den korteste kurve fra p til r efterfulgt af den korteste kurve fra r til q

3.8 Længde af diskret kurve

Lad Γ være en diskret kurve. Denne vil da bestå af en række knuder

$$\Gamma = \{p_1, \dots, p_n\}$$

Længden af kurven Γ betegnes Λ og defineres som

$$\Lambda(\Gamma) = \sum_{j=1}^{n-1} \varphi(p_j, p_{j+1})$$

hvor $\varphi(p, q)$ er den euklidiske afstand mellem de to knuder p og q som defineret i (3.1).

3.9 Grafalgoritmer

Da vi regner på diskrete figurer i $\mathbb{R}^3$, repræsenteret ved et trekantsnet, vil det være nærliggende at bruge en grafalgoritme til at finde afstanden mellem to vilkårlige punkter.

Vi har valgt at kigge på to korteste vej-algoritmer Fast Marching Metoden (FMM) og Dijkstras korteste vej algoritme. I det følgende afsnit introduceres en del af teorien bag FMM.

3.10 Fast Marching Metoden

Fast Marching Metoden (FMM) er en algoritme til at beregne hvordan fronte bevæger sig fremad. Denne blev først brugt af Sethian [7] til at regne niveaukurver på monotone fronte. Den kan også bruges som en utrolig hurtig måde at løse Eikonal[2] lignigen (3.3) på. Dette er en god måde at finde en approksimation til en diskret geodætiske kurve på, som vist af Sethian og Kimmel [8].

3.10.1 Introduktion til FMM

Har vi en afgrænsning af et område i $\mathbb{R}^2$, f.eks. en cirkel, som bevæger sig udad fra dens centrum, med en given hastighedsfunktion F (denne kan være 1), vil vi gerne vide på hvilke tidspunkter, T, at cirklen vil have udvidet sig til et punkt p. Vi skal altså have lavet en funktion, $T(s)$, som, givet p, vil give os en tidsparameter, for hvor lang tid det har taget cirklen at udvide sig til punktet p. Hvis vi har en kurve $\Gamma : [0 : n] \to X$, så $\Gamma(0) = p_0$ og $\Gamma(n) = p_n$, da vil

$$\Gamma'(t) = \frac{d}{dt}\Gamma(t) = -\nabla T(\Gamma(t)) \quad 0 \leq t \leq n$$

, og hvis Γ er en geodætisk kurve, vil $|\Gamma'| = 1$ iflg. A. Bronstein et. al. [9]. Vi kan heraf konkludere at afstandsfunktionen T må overholde

$$\begin{aligned} |\nabla T| &= \frac{1}{F} \\ |\nabla T| F &= 1 \end{aligned} \tag{3.3}$$

Denne partielledifferentialligning kaldes også Eikonal ligningen. ∇ kaldes gradienten, altså den partielafledet på alle ledene

$$\nabla T(x, y) = \left(\frac{\partial T}{\partial x}, \frac{\partial T}{\partial y} \right)$$

[2]Kommer af græsk $\epsilon\iota\kappa\omega\upsilon$, som betyder billede

I $\mathbb{R}^2$ kender vi allerede gradienten, vi kalder den normalt hældningen. I $\mathbb{R}^3$ svarer gradienten til hældningen af en plan. Gradienten bliver i $\mathbb{R}^3$ et vektorfelt, som går væk fra det punkt vi har udregnet den i (hvis gradienten vel og mærke er positiv).

Hvis vi vil finde den korteste vej henover vores graf, skal vi vælge den vej, hvor summen af alle T værdier er minimeret. På denne måde bruger fronten kortest tid på at bevæge sig fra start til slut.

Vi 'maler' knuderne i vores graf med tre forskellige farver. Kimmel, Bronstein og Bronstein [9] forklarer FMM ved at visualisere en skovbrand. Grønne knuder er træer som endnu ikke er berørt af flammerne. Røde knuder er de træer som i øjeblikket står i flammer, mens vi maler knuder, som allerede er brændt helt til aske, sorte. Flammerne vil aldrig nogensinde begynde at bevæge sig tilbage til allerede udbrændte træer.

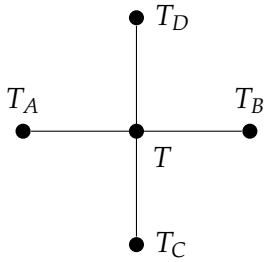
3.10.2 Afstandsfunktionen til FMM

Ligesom ved Dijkstras korteste vej-algoritme, skal FMM bruge en afstandsfunktion. Den måde som Sethian og Kimmel [8] forklarer det på, er ved først at tænke på denne afstandsfunktion i et plant kvadratisk gitter og så gå over til et arbitrært trekantsgitter herefter.

Antag at vi gerne vil udregne afstanden T ved punktet (i, j). Står vi i punktet (i, j) har vi fire naboer (se figur 3.8), nemlig $T_A = (i-1, j)$, $T_B = (i+1, j)$, $T_C = (i, j-1)$ og $T_D = (i, j+1)$. Nogle af disse kan have værdien ∞, hvilket betyder at de er grønne i algoritmen. Kigger vi på de muligheder vi har, kan vores knude i (i, j) f.eks. blive opdateret fra enten T_A eller T_C. Vi skal altså løse følgende andengradsligning, hvor vi antager at $T_A \leq T_C$

$$(T - T_A)^2 + (T - T_C)^2 = h^2 F^2 \tag{3.4}$$

, hvor h er afstanden mellem punkterne i gitteret.

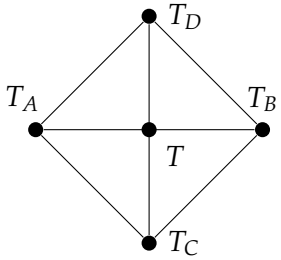


Figur 3.8 – Naboer der kan påvirke T

Der er to muligheder for denne andengradsligning. Hvis $T_C < T$ vil der være en reel løsning til (3.4). Hvis $T_C \geq T$ vil der være en reel løsning til den degenererede andengradsligning $(T - T_A)^2 = h^2F^2$. Vi bliver nød til at tjekke alle løsningsmuligheder for alle par og tage den mindste værdi af T vi kan finde.

Afstandsfunktionen på et pænt trekantsgitter

Vi kan nu udvide vores afstandsfunktion fra afsnit 3.10.2 til at gælde på pæne trekantsgitre. Med pæne menes her en triangulering af vores kvadratgitter fra forrige afsnit, hvor alle trekanter således er ens.



Figur 3.9 – Naboer, i pænt trekantsgitter, der kan påvirke T

Kigger vi igen på de gyldige steder opdateringen kan komme fra f.eks. T_A og T_C, kan vi nemt opskrive en formel for den plan de to, og T, ligger i

$$\left(\frac{T - T_A}{h}\right) x + \left(\frac{T - T_C}{h}\right) y + T = z$$

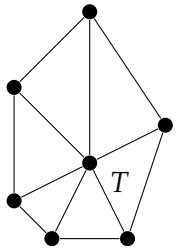
Tager vi gradienten af denne plan, vælger vi T så

$$\left(\frac{T - T_A}{h}\right)^2 + \left(\frac{T - T_C}{h}\right)^2 = F^2$$

Afstandsfunktionen på et arbitrært trekantsgitter

Vi vil nu gå videre til et arbitrært trekantsgitter og vise hvordan opdateringen af afstanden vil fungere her.

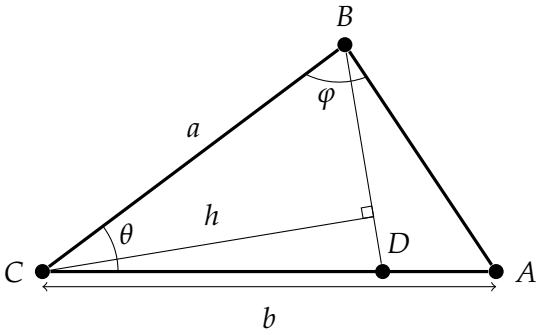
Kigger vi på en knude kan denne have uendeligt mange naboknuder. Vi leder nu efter de to knuder, som sammen med vores midtpunkt, danner en trekant, som kan bruges til at finde den mindste værdi for T. Hvis vi ligger en restriktion på vores trekantsgitter, at ingen trekanter må have stumme vinkler, kan vi opdatere afstanden til midtpunktet som beskrevet i dette afsnit. Andre har udviklet procedurer til at komme uden om dette, men



Figur 3.10 – Arbitrært trekantsgitter

disse går ud på at lave virtuelle trekanter, som igen ingen stumme vinkler har. Hvis vi selv styrer vores trekanter, kan vi blot lade vær med at danne trekanter med stumme vinkler.

Vi laver nu en trekant ΔABC, hvor den knude vi gerne vil opdatere er C.



Figur 3.11 – Trigonometrien på selve trekanten

Vi søger at finde $T(C) = T(A) + t$, hvor $t = EC$, således at $\frac{t-u}{h} = F$. Her er $u = HB = T(B) - T(A)$ og

$$h = a \sin \varphi = a \frac{CD}{BD} \sin \theta = \frac{aCD \sin \theta}{\sqrt{a^2 + CD^2 - 2aCD \cos \theta}}$$

Vi ender altså op med en andengradsligning

$$(a^2 + b^2 - 2ab \cos \theta)t^2 + 2bu(a \cos \theta - b)t + b^2(u^2 - F^2 a^2 \sin^2 \theta) = 0$$

For at løsningen til t kommer inden fra trekanten, og på den måde er en gyldig opdatering til T, skal $u < t$ og $a \cos \theta < \frac{b(t-u)}{t} < \frac{a}{\cos \theta}$. Hvis de to kriterier er opfyldt, kan vi opdatere $T(C)$ ved $\min\{T(C), t + T(A)\}$. Ellers

gælder $T(C) = \min\{T(C), bF + T(A), aF + T(B)\}$. Husk på at her stadig gælder $T(A) \leq T(C)$, som fra tidligere afsnit. Dette var afstandfunktionen Sethian og Kimmel [7] første gang foreslog til FMM.

Alternativt kunne man udregne afstandfunktionen, som Kimmel, Bronstein og Bronstein gør i [9]. De finder afstandsfunktionen T på en anden måde end Sethian og Kimmel først gjorde. Kigger vi på trekanten (p_i, p_j, p_k), hvor afstanden $d_i = T(p_i)$ er den korteste, p_j er en knude hvor vi har en afstand $d_j = T(p_j)$, det vil sige at knuden er enten rød eller sort, og p_k er en rød eller grøn knude, hvor afstanden $d_k = T(p_k)$ er den vi forsøger at gøre bedre. Det antages, uden at miste generalitet, at p_k ligger i origo og at trekanten ligger i xy-planen.

Vi ved at fronten ankommer til p_i med tiden d_i, til p_j med tiden d_j og vi forsøger nu at regne tiden d_k. Vi antager at fronten er plan, det vil sige at opdateringerne d_i og d_j kommer fra en plan kilde, som kan skrives på formen $n^T x + w = 0$, hvor n er normalvektoren, som bestemmer frontens retning og skalaen w angiver hvor planen kommer fra. Det ses tydeligt at p_i henholdsvis p_j må ligge d_i henholdsvis d_j fra planen, altså

$$d_i = n^T p_i + w$$
$$d_j = n^T p_j + w$$

I matrixnotation kan dette system skrives som

$$V^T n + w \cdot 1_{2\times 1} = d$$

hvor V er en 2×2 matrix med kolonerne p_i og p_k, $1_{2\times 1} = (1,1)^T$ og $d = (d_i, d_j)^T$. Vores mål er at udregne dette system med henblik på n og w, og derved udregne

$$d_3 = n^T p_k + w = w$$

Vi kan udregne n som

$$n = (V^T)^{-1}(d - w \cdot 1_{2\times 1}) = V^{-T}(d - w \cdot 1_{2\times 1})$$

Det lader altså til at vi har to ligninger med tre ubekendte, men vi skal huske at n er normalvektor, så vi har at $|n| = 1$, hvilket giver os

$$\begin{aligned} 1 = n^T n &= (d - w \cdot 1_{2\times 1})^T V^{-1} V^{-T}(d - w \cdot 1_{2\times 1}) \\ &= (d - w \cdot 1_{2\times 1})^T (V^T V)^{-1}(d - w \cdot 1_{2\times 1}) \\ &= w^2 \cdot 1_{2\times 1}^T Q 1_{2\times 1} - 2w \cdot 1_{2\times 1}^T Q d + d^T Q d \end{aligned}$$

hvor $Q = (V^T V)^{-1}$. Da $d_k = w$ kan vi regne d_k ved følgende andengradsligning

$$d_k^2 \cdot 1_{2\times1}^T Q 1_{2\times1} - 2d_k \cdot 1_{2\times1}^T Q d + d^T Q d - 1 = 0 \tag{3.5}$$

Ligning (3.5) vil have to løsninger, da både n og $-n$ har $|n| = 1$. Den mindste løsning er det n hvor vinklen mellem n, p_i og p_j er spidse. Det betyder at fronten ankommer til p_k før den kommer til p_i og p_j. Dette modsiger vores antagelse om hvordan fronten propagerer fremad. Denne løsning er derfor ugyldig og bliver smidt væk.

En gyldig løsning skal danne stumme vinkler med (p_k, p_i) og (p_k, p_j), figur 3.10.2, hvilket kan udtrykkes som $V^T n < 0$.

Kimmel et. al. skriver endvidere at det også er påkrævet at en stigning i d_i eller d_j også medfører en stigning i d_k. Denne monotone betingelse kan udtrykkes ved

$$\nabla_d d_k = \left(\frac{\partial d_k}{\partial d_i}, \frac{\partial d_k}{\partial d_i} \right)^T > 0$$

hvor uligheden skal ses koordinatvis. For at finde $\nabla_d d_k$ kan vi differentiere (3.5) med hensyn til $(d_i, d_j)^T$

$$0 = d_k \cdot \nabla_d d_k \cdot 1_{2\times1}^T Q 1_{2\times1} - \nabla_d d_k \cdot 1_{2\times1}^T Q d - d_k \cdot Q 1_{2\times1} + Q 1_{2\times1}$$

hvorfra

$$\nabla_d d_k = \frac{Q(d - w \cdot 1_{2\times1})}{1_{2\times1}^T Q (d - w \cdot 1_{2\times1})}$$

Substituere vi $n = V^{-T}(d - w \cdot 1_{2\times1})$ kan vi skrive

$$\nabla_d d_k = \frac{Q V^T n}{1_{2\times1} Q V^T n}$$

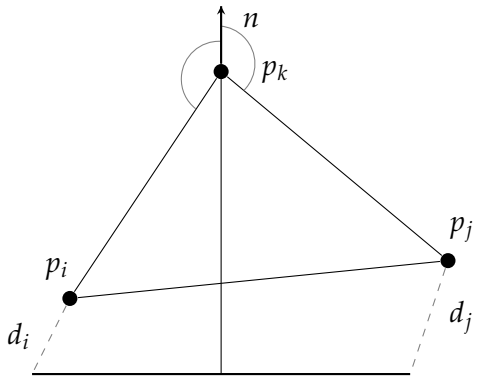
For at denne altid er under 0 skal begge fortegn af $QV^T n$ være ens, men da $V^T n$ altid er negativ (for at løsningen er gyldig), kan $QV^T n$ ikke have begge koordinater positive, så derfor $QV^T n < 0$.

Dette kan tolkes geometrisk, som at n skal ligge inden i trekanten.

Hvis dette ikke er tilfælde, må n ligge på en af kanterne i trekanten, og vi kan altså regne d_k på samme måde som ved Dijkstra's algoritme

$$d_k = \min \left\{ d_i + |p_i|, d_j + |p_j| \right\}$$

Lige meget hvad, opdatere vi selvfølgelig kun d_k hvis den udregnede d_k er mindre end den nuværende d_k.



Figur 3.12 – Geometrisk fortolkning af fronten og hvordan den påvirker afstanden d_k

Gør vi dette for alle punkter i fladen vil vi få den mindste tid det tager, at komme fra p, til alle andre punkter i fladen.

Kapitel 4

Analyse

Givet to knuder i et trekantsnet, ønsker vi at finde en approksimation til den diskrete geodætiske kurve mellem disse to knuder, samt længden af denne kurve.

I dette afsnit præsenteres læseren for en række løsningsforslag. Afsnittet sluttes af med en analytisk beskrivelse af den implementerede algoritme.

4.1 Grafalgoritmer

Da vores objekt er repræsenteret ved et trekantsnet, er det nærliggende at bruge en grafalgoritme, som f.eks. Dijkstras korteste vej-algoritme, til at finde afstanden mellem to knuder. En anden grafalgoritme, som kan bruges, er Sethians Fast Marching Metode [7].

4.1.1 Dijkstras algoritme

Dijkstras korteste vej-algoritme tager en retningsorienteret graf som input, samt en startknude, som er en knude i grafen. Algoritmen finder den korteste afstand fra startknuden til alle andre knuder i grafen, forudsat at alle andre knuder er tilgængelig via en vej fra startknuden.

Lad vores trekantsnet være en graf med dens knuder og kanter. Vi lader vægten af hver kant være længden af kanten, dvs. den euklidiske afstand mellem de to knuder kanten forbinder.

Som skrevet virker Dijkstras korteste vej-algoritme kun på retningsorienterede grafer. Vores trekantsnet er ikke retningsorienteret. Vi kan løse dette problem ved at erstatte hver kant i vores trekantsnet med to nye kanter, én kant i hver retning.

Dijkstras algoritme er skitseret i algoritme 1. Algoritmen finder den korteste vej fra startknuden p_0 til slutknuden p_n

Algoritme 1 Dijkstras algoritme

```
{d(p_i) er afstanden fra p_0 til p_i}
{f(p_i) er peger til foregående knude}
for all knuder p_i do
  d(p_i) ← ∞
  f(p_i) ← null
end for

{p_0 = startkunde}
d(p_0) = 0

{S = liste af alle knuder i graf}
while S ≠ ∅ do
  p_j = knude med mindst d(p_j)
  for all kanter til andre knuder p_i do
    if d(p_i) > d(p_j) + w(p_j, p_i) then
      {w(p_j, p_i) er euklidisk afstand fra p_j til p_i}
      d(p_i) ← d(p_j) + w(p_j, p_i)
      f(p_i) ← p_j
    end if
  end for
  if p_j = p_n then
    break
  end if
  S = S\{p_j}
end while

while f(p_j) ≠ null do
  Γ ← f(p_j)
  p_j = f(p_j)
end while
```

Køretid for Dijkstras algoritme

Køretiden for Dijkstra er $O(E \log V)$, hvor E er antallet af knuder og V er antallet af kanter.

4.1.2 Fast Marching

En anden grafalgoritme til at finde afstanden mellem de valgte knudepunkter er Fast Marching metoden, udviklet af Sethian [7].
FMM er en algoritme som forsøger at finde den korteste afstand fra én knu-

de, p_0, til alle andre knuder i grafen, på samme vis som Dijkstras korteste vej-algoritme. Forskellen ligger i udregningen af kanternes vægt. Algoritmen er som vist i algoritme 2.

Algoritme 2 Fast Marching Method

{S er vores trekantsnet}
for all $s \in S$ **do**
  $T(s) \leftarrow \infty$
end for

{p_0 er vores start knude}
$p_0 \leftarrow$ sort
for all naboer n til p_0 **do**
  $n \leftarrow$ rød
end for

resten af knuderne $\leftarrow$ grøn

while der er flere ikke-sorte knuder **do**
  {Find den røde knude med mindst $T(s)$}
  $p_i \leftarrow \min(T)$

  {Opdater alle trekanter, indeholdende p_i}
  for all trekanter (p_i, p_j, p_k), hvor p_j er sort **do**
    $p_k \leftarrow$ rød
    Opdater afstand T for p_k ved brug af trekant (p_i, p_j, p_k)
  end for

  $p_i \leftarrow$ sort
end while

Afstandsfunktionen vi bruger til at opdatere afstanden, kan ses i afsnit 3.10.2.

Effektiviteten i denne algoritme ligger i propagationen af T over overfladen S, og det at vi hele tiden holder styr på vores naboer, som er tættest fronten, ved at male dem røde. Når en knude p_i er blevet taget ud og afstanden er blevet beregnet, er vi sikre på at det er den korteste afstand til p_i. Dette skyldes at vi hele tiden tager den knude ud, som har den mindste T-værdi, og knuder med større værdier kan ikke påvirke den knude vi har. At genberegne T for p_i's naboer kan ikke frembringe en mindre værdi af T, da denne ellers allerede ville have været dukket frem.

Køretid for FMM

Algoritme 2 har en køretid på $O(V \log V)$, hvor V er antal knuder i grafen.

Dette skyldes følgende betragtning. Ved at bruge en min-hob som datastruktur, kan vi udtage det mindste element i konstant tid. Det tager $O(\log V)$ at sortere en min-hob, og dette skal gøres efter hver iteration. Køretiden er derfor $O(V \log V)$.

Korteste vej ved FMM

Vi har indtil videre kun udregnet afstandsfunktionen T for knuden p_0 og mangler nu blot at udregne den initielle vej fra p_0 til p_n. Kimmel og Sethian foreslår [8] at udregne denne ved at integrere

$$\frac{d\Gamma}{ds} = -\nabla T$$

Dette kan gøres ved brug af Heuns integrationsmetode.

En lettere tilgang, blev foreslået af D. Martínez et. al. [2]. De tilføjer blot knuderne til den initielle vej én efter én, som beskrevet i algoritme 3.

Algoritme 3 Dan den initielle vej

```
{p_0 og p_n er startknude henholdsvis slutknude}
{Γ er den initielle vej}
p ← p_n
while p ≠ p_0 do
  Γ ← p
  p ← nabo til p med mindst T
end while
```

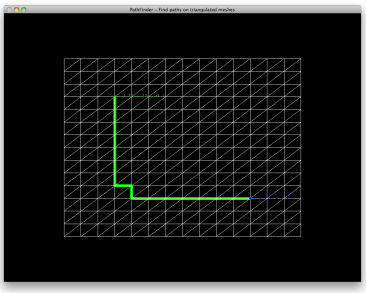
Dette er samme fremgangsmåde som ved Dijkstras korteste vej-algoritme og vi finder på denne måde vores korteste vej, hvor afstandene er udregnet med Fast Marching Metoden.

4.2 Ulemper ved grafalgoritmer

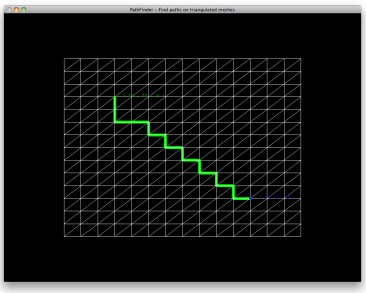
Det er ikke optimalt at bruge grafalgoritmer til at finde diskrete geodætisk kurver. Grunden er at grafalgoritmer har en række svagheder.

Et af problemerne ses illustreret på figur 4.1(a). Figuren viser et net af trekanter som udgør en plan. Her er Dijkstra korteste-vej algoritme meget langt fra den direkte vej mellem de to punkter.

En løsning til dette problem er at bruge en grafalgoritme der søger at finde den korteste og mest direkte vej. Iflg. D. Martínez et. al. [2] tager FMM en mere direkte vej end Dijkstra. Dette ses også på figur 4.1(b).

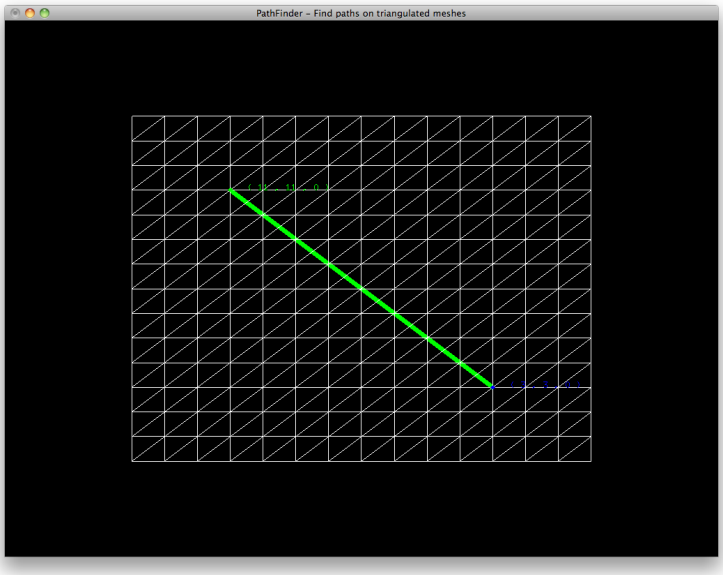


(a) Initial approksimation med Dijkstra



(b) Initial approksimation med FMM

Figur 4.1 – Sammenligning af Dijkstra og FMM



Figur 4.2 – Korteste kurve mellem de 2 valgte punkter.

Grafalgoritmer har yderligere den restriktion på sig. De er nødt til at følge kanterne på grafen og kan derfor ikke 'skyde genvej', hvilket vil få de fundne veje til at være en del længere end en kurve som kan bevæge sig frit på overfladen. Dette ses også på de to illustrationer vist i figur 4.1. Her er begge algoritmer meget kantede, fremfor den korteste kurve vist på figur 4.2

4.3 Optimal løsning af problemet

I det foregående afsnit beskrev vi problemerne ved grafalgoritmer som løsningsforslag til at finde en approksimeret tilnærmelse til den diskrete geodætiske kurve kurve mellem to punkter. Ved at imødegå disse problemer, kan vi optimere den vej, som grafalgoritmerne kommer frem til. En måde at imødegå problemerne som grafalgoritmerne har, er at korrigere den fundne vej, således denne bliver glattere og derved kortere.

Vi ved at den initielle vej, produceret af grafalgoritmen, er én af, muligvis flere, korteste veje, som har den restriktion at den kun bevæger sig på trekantsnettets kanter. Hvis vi skal korrigere denne vej, skal vi således finde en metode til at korrigere vejen, så denne nærmer sig en diskret geodætisk kurve.

Idéen er at flytte de interne knuder langs grafens kanter og derved opnå en kortere kurve.

Ved flytning af de interne knuder, gælder en række restriktioner. Knuderne kan ikke bare flyttes vilkårligt, trekantsnettets struktur bliver nødt til at blive taget i betragtning. Lad der derfor gælde følgende restriktioner ved flytning af en intern knude:

- Hvis en intern knude befinder sig på en af trekantsnettets kanter, kan denne knude kun flyttes i retningen mod en af kantens endepunkter
- Hvis en intern knude befinder sig på en af trekantsnettets knuder, kan den flyttes i retning af en trekantsknudens naboer

Ovenstående to restriktioner betyder at en knude kun kan flyttes langs kanterne i trekantsnettet.

I det følgende beskrives en korrigeringsmetode der korrigerer knuderne ved at flytte dem langs kanterne. Metoderne tager udgangspunkt i metoderne foreslået af D. Martínez et. al. [2]

4.4 Korrigering

Efter vi har fundet den initielle vej, skal vi korrigere denne vej til en kortere kurve. Vi gør dette ved at korrigere de interne knuder i den initielle vej, og derved opnå en kurve der har en kortere længde end den initielle vej. Korrigeringen sker i flere iterationer. Lad $\Gamma_i = \{p_{1,i}, \cdots, p_{n,i}\}$ betegne vores kurve efter i iterationer. I denne forbindelse betyder $p_{n,i}$ den n'te knude i vores kurve efter i iterationer. Lad da Γ_0 være vejen fremkommet vha. en grafalgoritme som f.eks. Dijkstras korteste vej algoritme.

Længden af Γ_0 er $\Lambda(\Gamma_0)$. Vores mål med korrigeringen er at korrigere de interne knuder i Γ_0 således den samlede kurve bliver mindre for hver iteration. Generelt ønsker vi af vores algoritme at

$$\Lambda(\Gamma_i) \geq \Lambda(\Gamma_{i+1}) \quad i \in \mathbb{N}_0 \tag{4.1}$$

og at Γ nærmer sig den korteste diskrete geodætiske kurve, dvs.

$$\lim_{i \to \infty} \Lambda(\Gamma_i) = \min(\Gamma) \tag{4.2}$$

hvor $\min(\Gamma)$ er den diskrete geodætiske kurve mellem punkterne p_1 og p_n.

Fordi vores kurve udviser optimal delstruktur jvnf. afsnit 3.7.1, kan vi korrigere de interne knuder i kurven, og derved gøre hele kurven kortere.

Lad p_j være en intern knude i Γ_i. Da kan vi korrigere p_j ud fra p_{j-1} og p_{j+1}. Vi korrigerer altså hver knude p_j vha. dens naboer således at delstykker af Γ_i bliver kortere.

I nogle tilfælde er det nok blot at flytte p_j til en ny position, eller helt slette knuden. I andre tilfælde bliver vi nødt til at erstatte p_j med m nye knuder $s_1 \dots s_m$. I hver iteration kan der komme flere eller færre knuder i Γ_i. Ved indsættelse af nye knuder, bruger vi de indsatte knuder, som næste korrigeringsknude.

Der er to muligheder for hvorledes p_j kan være placeret på vores trekantsnet. Enten er p_j placeret på en kant, eller også er den placeret på en af trekantsnettets knuder. Der gælder forskellige korrektionsmetoder alt efter hvordan p_j er placeret.

Ligeledes er det ikke alle knuder der kan korrigeres, som vi vil se senere. De knuder der kan korrigeres gør brug af udfoldning af en del af $\mathcal{F}_{p_j}$.

4.4.1 Udfoldning

Lad p_j være en intern knude i vores kurve Γ. Knuden p_j udspænder, sammen med dens naboer, fladen $\mathcal{F}_{p_j}$. Lad kurven fra p_{j-1} til p_j og kurven fra p_j til p_{j+1} dele $\mathcal{F}_{p_j}$ i to stykker. Kald disse to stykker $\mathcal{F}^l_{p_j}$ og $\mathcal{F}^r_{p_j}$, for henholdsvis venstre- og højredel.

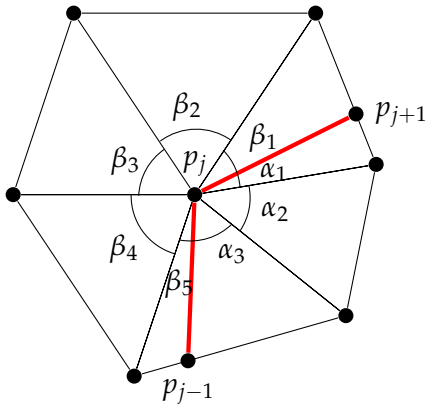
Da kan vi udregne den totale vinkelsum for $\mathcal{F}^r_{p_j}$ som

$$\theta_r = \sum_i \alpha_i$$

og den totale vinkelsum for $\mathcal{F}^l_{p_j}$ som

$$\theta_l = \sum_i \beta_i$$

Se figur 4.3 for grafisk illustration.



Figur 4.3 – $\mathcal{F}_{p_j}$ skåret igennem af p_{j-1} og p_{j+1}

De to endepunkter, p_{j-1} og p_{j+1} behøver ikke være knuder i trekantsnettet, men man godt befinde sig på en kant.

Antag nu at $\theta_r \leq \theta_l$ og $\theta_r \leq \pi$, da kan vi lave en 'vifte' af fladen $\mathcal{F}^r_{p_j}$, og udregne skæringspunkter med de kanter som befinder sig på delfladen $\mathcal{F}^r_{p_j}$. Udregningen af skæringspunkterne beskriver vi i det kommende afsnit.

Grunden til at vi vælger den side med den mindste vinkel er cosinusrelationen

$$c^2 = a^2 + b^2 - 2ab\cos\theta \tag{4.3}$$

Vi ønsker at minimere c. Da $\cos\theta$ er en faldende funktion for $\theta \in [0, \pi]$, skal vi for at minimere c, vælge den mindste vinkel af θ_l og θ_r.

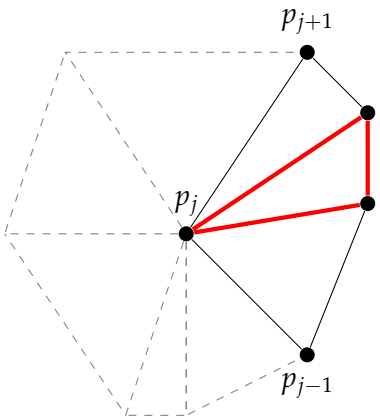
Ovenstående antager at $\theta_r \leq \theta_l$, men for god ordens skyld skal det nævnes at hvis $\theta_l \leq \theta_r$, gælder udfoldningen også og denne foretages analogt med det gennemgåede.

Hvis $\theta_r = \theta_l$ er det ligegyldigt hvilken side vi vælger, og vi kan således blot vælge én af dem.

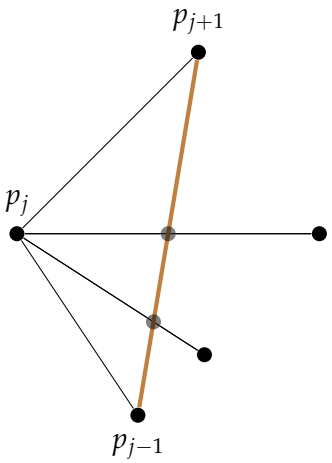
4.4.2 Skæringspunkter

Lad fladen fra forrige afsnit, $\mathcal{F}_{p_j}$, være splittet op i to delflader, $\mathcal{F}^l_{p_j}$ og $\mathcal{F}^r_{p_j}$,. Lad $\mathcal{F}'_{p_j}$ betegne den mindste af de to sider, $\mathcal{F}^l_{p_j}$ og $\mathcal{F}^r_{p_j}$, og lad $\mathcal{F}'_{p_j}$ have en vinkelsum mindre end π.

Da kan vi beregne en kortere kurve, fra p_{j-1} til p_{j+1} ved at folde $\mathcal{F}'_{p_j}$ ud i en vifte og projektere denne ned i $\mathbb{R}^2$. I $\mathbb{R}^2$ kan vi beregne skæringspunkter



Figur 4.4 – Vifte indesluttet af kanterne fra p_j til p_{j+1} og p_j til p_{j-1}. De røde kanter er indeholdt i viften. De stiplede linier er den anden halvdel af $\mathcal{F}_{p_j}$.

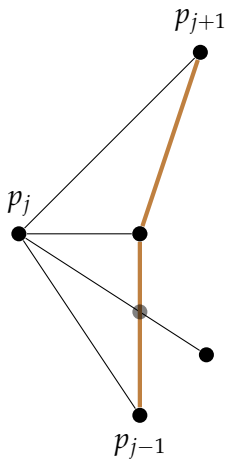


Figur 4.5 – Udregning af skæringspunkter mellem kanterne indeholdt i viften. De grå knuder er skæringspunkter mellem den rette linie fra p_{j-1} og p_{j+1} og de kanter der er indeholdt i viften.

mellem den rette linie fra p_{j-1} og p_{j+1} og kanterne indeholdt i viften. Se figur 4.4 for illustration.

På figur 4.5 er den rette linie mellem p_{j-1} og p_{j+1} indtegnet. Skæringspunkter mellem denne rette linie og de andre kanter der befinder sig i $\mathcal{F}'_{p_j}$ er illustreret med grå knuder.

I nogle tilfælde findes der ikke en skæring mellem de interne kanter i viften og linien fra p_{j-1} til p_{j+1}. Denne situation opstår, når en kant indeholdt i viften ikke er lang nok. I disse tilfælde vælger vi den knude,



Figur 4.6 – Vifte, hvor der ikke findes en skæring mellem den ene kant indeholdt i viften, og den rettet linie mellem p_{j-1} og p_{j+1}.

som er tættest på den rette linie mellem p_{j-1} til p_{j+1}, dvs. endeknuden af kanten. Figur 4.6 illustrerer dette scenario.

Efter vi har fundet skæringspunkterne, kan vi projektere viften tilbage til $\mathbb{R}^3$. Isometrien sikrer at vi ikke mister afstandsinformation.

4.4.3 Position på kant

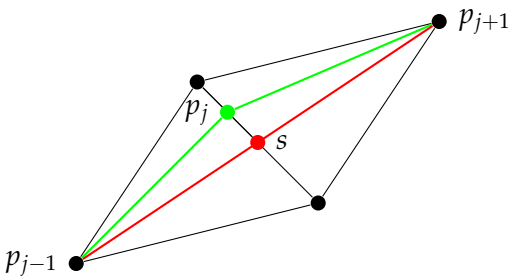
Som nævnt i afsnit 4.4 er der forskellig korrigeringsmetoder, alt efter hvordan den interne knude p_j er placeret på trekantsnettet.

Lad p_j være placeret på en af trekantsnettes kanter. Da er der to retninger p_j kan flyttes, mod en af dens to naboer. Eftersom p_j ligger på en kant, må der være to trekanter der støder op til denne kant, jvnf. vores antagelse om at trekantsnettet er mangfoldigt (se afsnit 1.3). Vi kan korrigere p_j ved at udfolde de tilstødende trekanter og finde skæringspunktet mellem linien der går fra p_{j-1} til p_{j+1} og linien, som p_j ligger på.

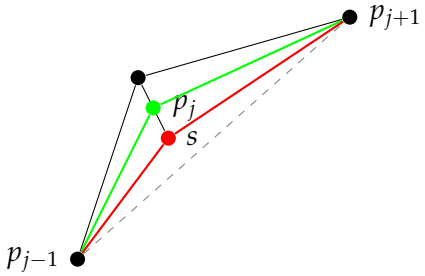
I de tilfælde hvor der findes en skæring er det intet problem at flytte knuden. Knudens nye position er skæringspunktet mellem linien fra p_{j-1} til p_{j+1} og den linie, som p_j ligger på.

I nogle tilfælde vil p_j være placeret således at der ikke er en skæring mellem linien p_{j-1} til p_{j+1} og linien som p_j ligger på. I disse tilfælde vælger vi at flytte p_j så tæt på skæringspunktet som muligt, jvnf. figur 4.8.

Den nye korrigerede position af p_j, er således skæringen mellem de to linier eller, hvis der ingen skæring finder sted, endepunktet for linien hvor p_j befinder sig på.



Figur 4.7 – s er den nye position for p_j. Den røde kurve, er den nye korrigeret kurve fra p_{j-1} til p_{j+1}.



Figur 4.8 – s er den nye position for p_j. Den røde kurve, er en nye korrigeret kurve. Den stiplede linie, er den rette linie fra p_{j-1} til p_{j+1}. Der findes ingen skæring mellem linien, som p_j ligger på, og den rette linie fra p_{j-1} til p_{j+1}

Vi kan garantere at afstanden fra p_{j-1} til p_{j+1} er minimeret. Dette skyldes den betragtning at når vi udfolder de tilstødende trekanter udgør de en todimensional plan i det tredimensionale rum. Jvnf. afsnit 3.1 er den korteste afstand på dette plan den rette linie.

Det er netop skæringen mellem den rette linie mellem p_{j-1} og p_{j+1} og linien, som p_j ligger på, vi udregner, hvis denne finder sted. Hvis der ingen skæring findes, flytter vi p_j så tæt på endepunktet som muligt. I begge tilfælde kommer p_j til at have en position så tæt på den rette linie fra p_{j-1} til p_{j+1} som muligt.

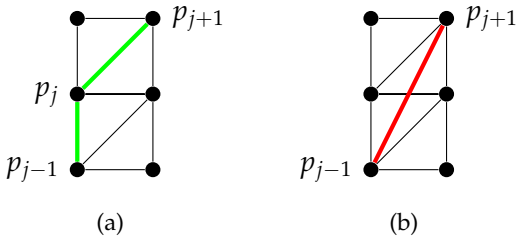
Isometrien mellem trekanterne i tre dimensioner, og trekanterne i to dimensioner, sikrer at afstanden ikke ændres når vi projekterer tilbage til tre dimensioner.

4.4.4 Position på knude

Lad p_j være positioneret på en af trekantsnettets knuder. p_j kan da flyttes i retning mod en af sine naboer, men vi bliver nødt til at tage typen af p_j i betragtning, hvis vi ønsker at gøre kurven mellem p_{j-1} og p_{j+1} kortere. I det følgende gennemgås korrigeringsmetoderne, baseret på typen af p_j.

Knuden er euklidisk

Hvis p_j er euklidisk, kan vi blot fjerne den fra vores kurve. Dette skyldes den betragtning af $\mathcal{F}_{p_j}$ er en plan i $\mathbb{R}^3$, hvilket vil sige at den korteste kurve mellem p_{j-1} og p_{j+1} er den rette linie som går hen over planen $\mathcal{F}_{p_j}$ jvnf. afsnit 3.1.



Figur 4.9 – $\mathcal{F}_{p_j}$ danner en plan i $\mathbb{R}^3$. p_j kan fjernes, og den korteste vej mellem p_{j-1} og p_{j+1} er den rette linie.

Knuden er sfærisk

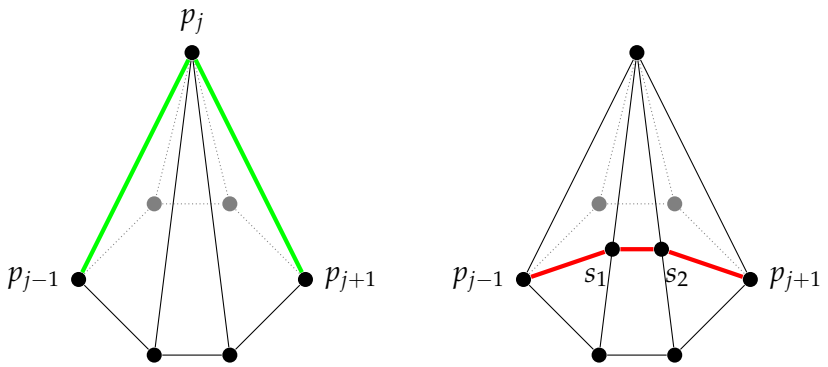
Hvis p_j er sfærisk, gælder der, jvnf. afsnit 3.6.2, at der ikke kan være en kortest kurve, som passerer igennem den. Knuden kan korrigeres ved at vælge den delflade, $\mathcal{F}^l_{p_j}$ og $\mathcal{F}^r_{p_j}$, med mindste vinkelsum, i fladen $\mathcal{F}_{p_j}$, som beskrevet i afsnit 4.4.1. Da knuden er sfærisk ved vi at $\theta_{p_j} < 2\pi$. Derved kan vi også konkludere at én af de to vinkler θ_l og θ_r er mindre end π.

Vi udfolder den valgte delfalde i en vifte, som forklaret i afsnit 4.4.2. Selve korrigeringen sker ved at p_j bliver erstattet af skæringspunkterne, $s_1 \dots s_m$.

Knuden er hyperbolsk

Igen udregner vi $\mathcal{F}^l_{p_j}$ og $\mathcal{F}^r_{p_j}$. Hvis både $\theta_l > \pi$ og $\theta_r > \pi$, da er den korteste kurve mellem p_{j-1} og p_{j+1} den der går igennem p_j [5]. Vi kan således ikke korrigere knuden. Figurene 4.11(a) og 4.11(b) illustrerer denne situation.

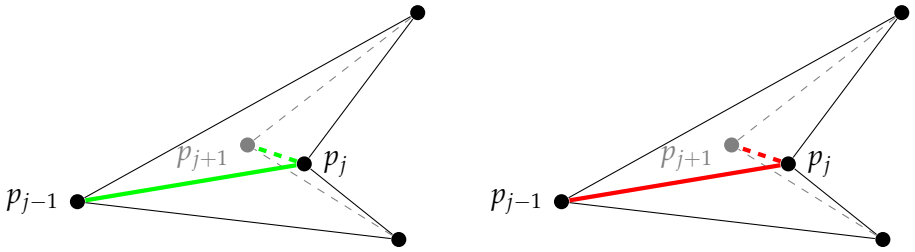
Hvis en af vinklerne er mindre end π, dvs. hvis $\theta_l \leq \pi$ eller $\theta_r \leq \pi$, da kan vi udfolde denne side, og korrigere den jvnf. afsnit 4.4.1. Figur 4.12



(a) p_j er sfærisk

(b) Skæringspunkterne s_1 og s_2 illustreret

Figur 4.10 – p_j ertstattes af skæringspunkterne s_1 og s_2 som vist på figur (b)

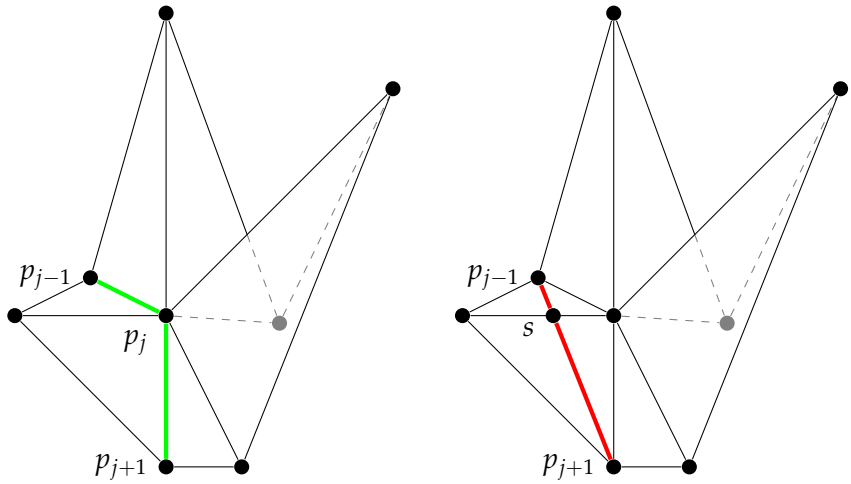


(a) p_j er hyperbolsk og $\theta_l > \pi$ og $\theta_r > \pi$. Den grønne vej er den oprindelige kurve fra p_{j-1} til p_{j+1}

(b) p_j er hyperbolsk, hvor $\theta_l > \pi$ og $\theta_r > \pi$. Vejen fra p_{j-1} til p_{j+1} kan ikke korrigeres og er den samme som (a)

Figur 4.11 – Korteste vej gennem en hyperbolsk knude, p_j hvor begge sider har en vinkel skarpt større end π.

illustrerer dette. Den oprindelige knude i vores kurve, p_j, bliver da erstattet af skæringspunkterne $s_1 \dots s_m$ mellem den rette linie p_{j-1} til p_{j+1}, og de kanter der er indeholdt i viften.



(a) p_j er hyperboolsk, men én af siderne har en vinkelsum mindre end π. Dvs. enten er $\theta_l > \pi$ eller $\theta_r > \pi$.

(b) p_j er hyperboolsk, men én af siderne har en vinkelsum mindre end π. Siden der har en vinkelsum mindre end π udfoldes og skæringspunkterne udregnes. p_j erstattes af skæringspunkerne

Figur 4.12 – Illustration hvor p_j er hyperbolsk og én af siderne, $\mathcal{F}^l_{p_j}$ eller $\mathcal{F}^r_{p_j}$, har en vinkelsum mindre end π.

4.5 Optimal algoritme

Algoritme 4, sammenfatter ovenstående korrigeringsmetoder. Algoritmen har en køretid på $O(N)$, hvor N er antallet af knuder i vores kurve.

Algoritmen korrigerer hver knude én gang for hver iteration.

Algoritme 4 Korrigerings algoritme

for all $p_i \in \Gamma$ **do**
  if p_i er euklidisk **then**
    $\Gamma = \Gamma \backslash \{p_i\}$

  else if p_i er sfærisk **then**
    $\Gamma = \Gamma \backslash \{p_i\}$
    Vælg $\mathcal{F}^l_{p_i}$ eller $\mathcal{F}^r_{p_i}$
    Regn skæringspunkter, som beskrevet i afsnit 4.4.2
    $\Gamma \leftarrow$ skæringspunkter

  else if p_i er hyperbolsk **then**
    if θ_l eller θ_r er mindre end π **then**
      $\Gamma = \Gamma \backslash \{p_i\}$
      Regn skæringspunkter, som beskrevet i afsnit 4.4.2
      $\Gamma \leftarrow$ skæringspunkter
    else
      Vi har den korteste kurven gennem p_i
    end if
  end if
end for

4.6 Stopkriterie

Vi kan stoppe vores iteration når følgende er opfyldt

$$\Gamma_i = \Gamma_{i+1} \tag{4.4}$$

Dvs. at to iterationer indeholder nøjagtig de samme knuder, og disse knuder ikke har ændret position. Dette skyldes den betragtning at hvis (4.4) er opfyldt, da er

$$\Lambda(\Gamma_i) = \Lambda(\Gamma_{i+1})$$

Vi kunne sagtens iterere videre, men dette ville ikke have en effekt på den samlede længde af kurven.

Dette stopkriterie har dog den ulempe, at vores algoritme kan have en køretid på $O(\infty)$. Betragt f.eks. tilfældet hvor den korteste diskrete geodætiske afstand mellem to punkter er $\sqrt{2}$. Da kan vi blive ved med at korrigere vores kurve vilkårlig tæt på $\sqrt{2}$ uden at (4.4) vil være opfyldt.

Rent praktisk arbejder computere med endelig precision, hvilket bevirker at stopkriteriet vil blive opfyldt på et tidspunkt pga. manglende præcision, men vi anser det stadig som værende et problem.

Et alternativt stopkriterie, som er foreslået af D. Martínez et. al. [2], er at give hver intern knude i kurven en fejlværdi bestemt ud fra dens position og type.

Hvis knuden befinder sig på en kant er fejlværdien defineret som forskellen mellem θ_l og θ_r.

Hvis knuden befinder sig på en trekantsknude, som er sfærisk, er fejlværdien defineret som ekstrem stor.

Hvis knuden befinder sig på en trekantsknude som er hyperbolsk og begge sider er større end π da er fejlværdien defineret til 0, ellers meget stor.

Kurvens samlede fejlværdi er defineret som den største af de interne knuders fejlværdi. Dette bevirker at jo mere lige kurven er, jo mindre fejlværdi får den.

Vores korrigeringsalgoritme kan da blive ved med at korrigere indtil fejlværdien er mindre end ét $\epsilon \in \mathbb{R}$.

Kapitel 5

Implementation

5.1 Overordnet overblik

Det udviklede program har titlen `pathfinder`. Programmet er udviklet i C++ , og en række hjælpeprogrammer er brugt. F.eks. har vi, til at visualisere vores resultater, valgt at bruge OpenGL biblioteket GLUT[1].

`pathfinder` kan indlæse en fil i filformatet `ply`[2], som er et filformat, udviklet på Stanford University, indholdende en liste af knuder og en liste af kanter. Vi har valgt `ply`, fordi det er et læsbart format og der findes mange eksempelfiler frit tilgængelige på internettet. Til at parse `ply`-filer har vi brugt en parser udviklet af João Oliveria[3].

Vi gør kraftig brug af vektor/matrix biblioteket RasterMan udviklet af vores vejleder, Knud Henriksen. Biblioteket gør os i stand til, på en nem måde, at foretage vektorberegninger som prikprodukt, krydsprodukt m.fl.

`pathfinder` er et interaktivt program. Brugeren har f.eks. selv mulighed for at zoome, vælge start- og slutknuder og vælge hvordan figuren skal repræsenteres osv. Når brugeren har valgt en start- og en slutknude, udregner programmet den initielle vej og viser den til brugeren. Herefter kan brugeren selv vælge at køre én eller alle iterationer af korrektionsalgoritmen.

Vi vil i det følgende kapitel beskrive en del af implementation. For detaljer henvises læseren til det vedlagte program.

[1] GLUT - The OpenGL Utility Toolkit `http://www.opengl.org/resources/libraries/glut/`, besøgt 31. maj

[2] The Stanford 3D Scanning Repository, `http://graphics.stanford.edu/data/3Dscanrep/`, besøgt 31. maj

[3] `http://www.cs.ucl.ac.uk/staff/joao.oliveira/ply.html`, besøgt 31. maj

5.2 Mangfoldighed

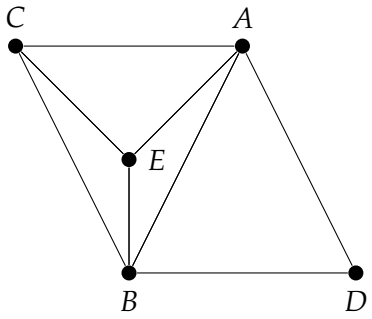
Vi har udviklet algoritmen vist i algoritme 5 til at kontrollere om en figur er mangfoldig.

Algoritme 5 Kontrol af mangfoldighed

```
{S er vores trekantsnet}
for all knuder p ∈ S do
  while p har ubesøgte naboer do
    Gem sidst besøgte nabo n
    for all flader F_p do
      Gem sidst besøgte flade F'_p
      Lad u og v være to knuder i fladen F_p, som ikke er med i F'_p
      Hvis der findes flere end én u eller v, som er lig med n er figuren
      ikke mangfoldig
    end for
  end while
end for
```

Vores oprindelige afledede definition af mangfoldighed viste sig at være forkert. Vi havde fra definition 4, udledt at fællesmængden mellem de to knuders naboer indeholdt præcis to elementer.

Dette viste sig at være *forkert*, som figur 5.1 viser.

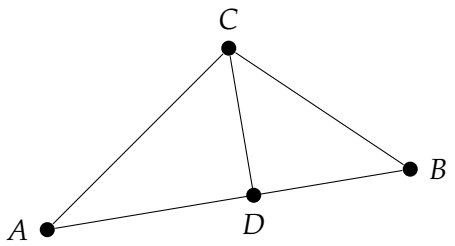


Figur 5.1 – Trekantsnet, som modbeviser vores første antagelse om mangfoldighed

Figur 5.1 er iflg. definition 4 mangfoldig, men A og B's fællesmængde af naboer indeholder $\{C, E, D\}$, altså mere end to elementer, og vores afledede definition sagde derfor at den var ikke mangfoldig. Dette er en modstrid og vores afledede definition var derfor forkert.

5.3 Sortering af en knude

Når vi første gang indlæser vores figur, ud fra den givne `ply`-fil, oprettes alle knuder og kanter i vores datastrukturer. Vi sørger for at ingen af de trekanter vi opretter har stumme vinkler, da vores implementation af FMM ikke kan håndtere dette, jvnf. afsnit 3.10.2. Vi kan garantere at ingen trekanter indeholder stumme vinkler, ved at kontrollere samtlige vinkler på de indlæste trekanter. Har en trekant en vinkel, som er stum, deler vi trekanten i to, ved at tage den vinkelrette linie fra punktet til den modstående side, også kaldet højden. Det punkt, hvor højden går ind på siden, lader vi være et nyt punkt i de to trekanter vi opretter som i stedet for den første. Da højden er vinkelret på den modsatte side, er vi garanteret at de to andre vinkler er under $\pi/2$ og vi er således fri for stumme vinkler.



Figur 5.2 – Trekant ΔABC har en stum vinkel i C. Vi splitter trekanten i to og opretter trekanterne ΔACD og ΔBCD

Herefter opretter vi alle trekanterne og lægger de punkter trekanterne indeholder i en liste. Vi ved nu at i denne liste, ligger alle vores punkter sorteret således at de tre på hinanden følgende punkter n_1, n_2 og n_3 er i samme trekant. Dette udnytter vi når vi skal finde frem til alle naboerne for en given knude.

Da der ingen ordningsrelation er i $\mathbb{R}^3$, bliver vi nødt til at have mere information end position for at kunne sortere vores naboliste. Med ovenstående kan vi dog altid garantere at vores naboliste for en given knude er sorteret, således at naboerne enten ligger med eller mod uret startende fra en af dens naboer. Vi er nødt til at kende rækkefølgen af vores naboer når vi skal korrigere en knude. Naboer til de enkelte knuder dannes samtidig med at vi kontrollere om figuren er mangfoldig, se evt. algoritme 5.

5.4 Initial vej

Til at finde den initielle vej gennem grafen har vi implementeret to forskellige korteste vej-algoritmer, som beskrevet i afsnit 4.1. Disse to algoritmer

er Dijkstras og Fast Marching Metoden.

Dijkstras algoritme er implementeret som pseudokoden i algoritme 1.

5.4.1 FMM

Vi har implementeret FMM som den angivne pseudokode i algoritme 2. En af de forskelle der dog er, ligger i at vi tjekker om vi faktisk har opdateret vores afstand, således at denne ikke længere er ∞, før vi fjerner den fra de røde knuder. Til at holde styr på de røde knuder, bruger vi en min-hob.

Når vi er færdige med at regne afstanden fra vores startknude til alle andre, bruger vi algoritme 3 til at finde den initielle vej.

FMM vil, ligesom Dijkstra, give Manhatten-afstanden, men den vil dog give en bedre, mere direkte, Manhatten afstand, som det kan ses i afsnit 6.1.

5.5 Korrektion

Når brugeren trykker på `i` vil vores program forsøge at korrigere vejen fra A til B, således at denne bliver kortere, og derved en opnå en bedre approksimation til den diskrete geodætiske kurve.

Korrektion kan, i vores implementation, foretages sålænge en knude i vores kurve kan ændre position.

Vi løber således alle knuderne i kurven igennem og ser om vi kan korrigere dem, til en bedre position, som gør kurven kortere.

Hvis knuden vi prøver at korrigere ligger på en kant, korrigerer vi den som beskrevet i afsnit 4.4.3. Hvis knuden ligger på en trekantsnet knude korrigerer vi knuden som beskrevet i afsnit 4.4.4.

5.5.1 Knude på kant

En knude vil ligge på en kant, når denne har præcis to naboer. Dette skyldes vores antagelse om at figuren er mangfoldig, altså at alle kanter er delt af præcis to trekanter.

Vi udregner nu vinklerne rundt om knuden, for at finde ud af hvilken af de to naboer, som der skal korrigere over imod. Selve punktet vi skal flytte knuden over til bliver udregnet af en funktion vi har valgt at kalde `fan`. Denne er beskrevet i afsnit 5.6.

5.5.2 Knude på knude

Har knuden mere end to naboer, betyder det at den må befinde sig på en af trekantsnettes knuder.

Vi kan ud fra knudens vinkelsum finde knudens type euklidisk, sfærisk eller hyperbolsk.

Er knuden euklidisk kan vi blot fjerne den, da den korteste vej vil gå fra knudens forgænger til dens efterkommer.

Er knuden derimod sfærisk eller hyperbolsk knude, må vi finde ud af hvor på kanterne vi skal oprette nye knuder, og forbinde disse ind i vores kurve. Dette gøre ved at finde skæringspunkter med `fan` funktionen.

Nyindsatte knuder, bliver straks korrigeret som de næste i rækken. Vi venter således ikke en iteration.

5.6 Fan

`fan` er en funktion, som finder skæringer mellem liniestykker. Funktionen er en implementation af algoritmen beskrevet i analyse afsnittet 4.4.2.

Skæringspunkterne den finder bliver brugt til at korrigere kurven gennem punkterne.

Måden hvorpå den finder skæringspunkterne er simpel. Vi laver en projektion af vores vektor i $\mathbb{R}^3$ ned i $\mathbb{R}^2$, og regner skæringspunkter ved at sætte de to linier $l_1 = l_2$, hvor

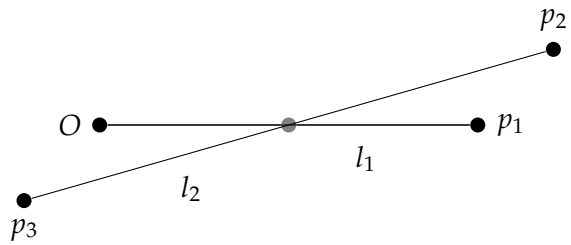
$$l_1 : \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + s \begin{pmatrix} p_{1_x} \\ p_{1_y} \end{pmatrix}$$
$$l_2 : \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} p_{2_x} \\ p_{2_y} \end{pmatrix} + t \begin{pmatrix} p_{3_x} \\ p_{3_y} \end{pmatrix}$$

På figur 5.3 kan dette ses geometrisk. For at finde skæring i $\mathbb{R}^3$ ud fra dette, søger vi at finde s når $l_1 = l_2$, da dette er afstanden vi skal bevæge os ud af $\vec{Op_1}$ for at komme frem til skæringspunktet. Udfra determinantmetoden kan vi regne s ved

$$s = \frac{p_{2_x} p_{3_y} - p_{2_y} p_{3_x}}{p_{1_x} p_{3_y} - p_{1_y} p_{3_x}}$$

Vi ved nu at der findes et skæringspunkt, s ude af vektoren $\vec{Op_1}$. Dette skæringspunkt kan bruges i stedet for den knude vi kigger på, og vil gøre kurven kortere.

Vi kan projekterer viften ned i $\mathbb{R}^2$ fra $\mathbb{R}^3$ ved at beregne længden af kanterne og deres indbyrdes vinkler. Når vi skal projektere de nye skæringspunkter op fra $\mathbb{R}^2$ til $\mathbb{R}^3$ placerer vi skæringspunkterne som en brøkdel af den oprindelige kants længde.



Figur 5.3 – Den grå knude er her skæringen mellem l_1 og l_2

5.7 Køretid af vores program

Når man første gang kører programmet, med en figur, skal denne figur dannes og vises på skærmen. Da vi lader GLUT overtage visning, er det eneste vi skal bekymre os om at danne trekanter og knudepunkter. Dette sker ved at vi lineært løber `ply`-filen igennem og danner knuderne. Herefter danner og splitter vi trekanterne i $O(T)$-tid, hvor T er antallet af trekanter.

Alle knuderne får de rigtige naboer efter at de blevet dannet. Dette sker ved at vi, for hver knude, løber alle trekanter, den er en del af, igennem og tilføjer de to andre knuder, som naboer. En hurtig analyse fortæller os at med T trekanter vil der være $\frac{3T}{2}$ kanter, da hver kant er delt af to trekanter. Vi vil i værste tilfælde skulle tilføje alle, bortset fra én, af knuderne i figuren, som naboer til den knude vi kigger på. At det er alle bortset fra én kommer af at figuren skal være mangfoldig. Hvis vi skal tilføje alle bortset fra én, vil vi skulle bruge halvdelen af kanterne til at gøre dette.

Efter at vi har oprettet knuderne, med deres naboer, skal vi finde den initielle vej. Dette gøres for Dijkstra i $O(E \log V)$, hvor E er antal kanter og V er antal knuder, mens det for FMM sker i $O(V \log V)$, hvor V er antallet af knuder.

Herefter skal vi korrigere vores vej, og finde en approksimation til den geodætiske kurve. Dette gøres som allerede beskrevet i afsnit 4.5 i $O(N)$-tid pr. korrigering, hvor N er antal knuder i den aktuelle approksimation. Det kan ikke udregnes hvad køretidskompleksiteten for den samelede korrigering er, da vi ikke på forhånd kan vide hvor mange korrigeringer vi skal lave, for at vi ikke kan foretage flere korrigeringer.

Kapitel 6

Resultater

Vi vil i dette afsnit lægge vægt på at overbevise læseren om at vores implementation er korrekt, og at de kurver `pathfinder` finder nærmer sig de diskrete geodætiske kurver. Vi har i et enkelt tilfælde kunnet sammenligne vores implementation med en analytisk udregnet geodætisk afstand, se evt. afsnittet omkring afstande på overfladen af en kugle.

Det er ikke lykkedes os at finde sammenlignelige testdata på mere komplekse figurer.

I de fleste tests, bruger vi Dijkstra som den initielle algoritme. Dette skyldes at der er en fejl i vores implementation af FMM, som gør at den er utrolig langsom på figurer, som består af mange knuder.

Alle testfigurer er vedlagt som `ply`-filer sammen med implementationen, og kan findes i det bibliotek som hedder `figures`.

På nogle illustrationer, ser det ud som om at den korrigerede kurve, ligger under overfladen, i stedet for ovenpå figuren. Dette er ikke tilfældet. Kurven ligger på overfladen, men da den har samme dybde som de andre punkter på overfladen, ser det i nogle tilfælde ud til at den tegnes under. Dette er udelukkende et visualiseringsproblem.

Alle billeder er at finde i større versioner som bilag.

6.1 Test af vej på flatgrid

Figuren vi tester på i denne test kalder vi flatgrid og figuren kan findes i filen `flatgrid.ply`.

Flatgrid har formen som en pyramide. Bunden af pyramiden er en plan sammensat af trekanter.

Vi tester ved at vælge to punkter på planen i bunden. Punkterne er placeret således at grafalgoritmerne er tvunget til at benytte de horisontale

og vertikale kanter.

Vi forventer at FMM tager en mere direkte vej, end Dijksta. Dette skulle gerne bevirke at antallet af korrektioner inden kurven ikke kan korrigeres mere bliver reduceret drastisk.

Vi forventer at længden af den initielle FMM vej er identisk med længden af den initielle Dijkstra vej. Vores forventning skyldes grafalgoritmernes restriktioner om kun at bevæge sig på kanterne og at de begge garanterer at finde én, af muligvis flere, kortest(e) vej(e).

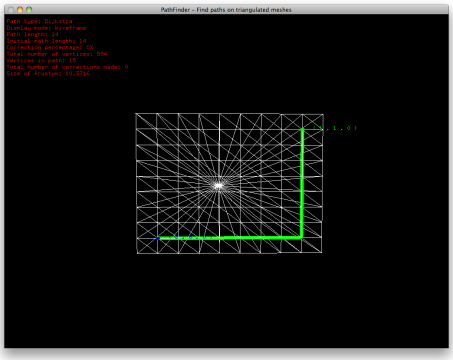
Vi forventer også at længden af de korrigerede kurver er identiske, uanset om den initielle algoritme er Dijkstra eller FMM. Vi forventer derimod ikke at antallet af nødvendige korrektioner, er identisk. Vi forventer at antallet af korrektioner af den vej dannet af FMM er mindre end antallet af korrektioner af den vej dannet af Dijkstra.

Figur 6.1(a) viser den initielle vej dannet af Dijkstra. Vejen har en længde på 14 og består af 14 knuder. Figur 6.1(b) er den initielle vej dannet af FMM. Denne vej har også en længde på 14 og består af 14 knuder.

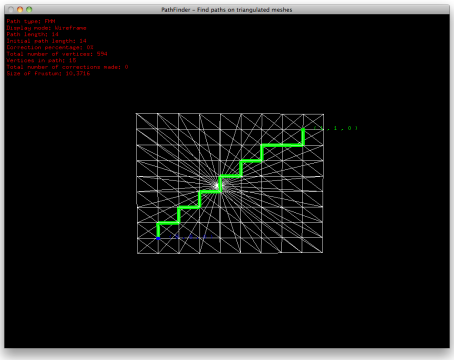
Som det ses på figur 6.1 reducerer FMM algoritmen antallet af nødvendige korrektioner inden kurven ikke kan korrigeres mere.

Ifht. Dijkstra reducerer FMM antallet af nødvendige korrektioner med ca. 50%, hvilket skyldes at FMM danner en mere direkte initial vej. Dette stemmer overens med vores forventninger om at antallet af korrektioner er forskellige og at den initielle vej dannet af FMM skal bruge færre korrektioner.

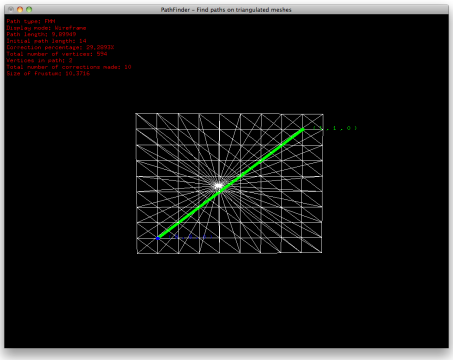
Som forventet er længden af de initielle veje identiske, ligeledes er længden af de korrigerede veje identiske.



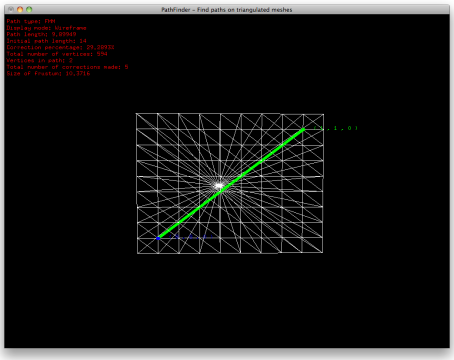
(a) Initial vej dannet af Dijkstra algoritmen. Vejen har en længde på 14, og indeholder 14 knuder.



(b) Initial vej dannet af FMM algoritmen. Vejen har en længde på 14, og indeholder 14 knuder.



(c) Kurve fremkommet af den initielle vej som er vist i (a). Kurven er fremkommet efter 10 iterationer af vores korrigeringsalgoritme. Kurven har en længde på ca. 9,89 og er blevet forbedret med ca. 29,28%.



(d) Kurve fremkommet af den initielle vej som er vist i (b). Kurven er fremkommet efter 5 iterationer af vores korrigeringsalgoritme. Kurven har en længde på ca. 9,89 og er blevet forbedret med ca. 29,28%.

Figur 6.1 – Sammenligning af Dijkstra og FMM på en euklidisk overflade.

6.2 Forhindringer

I afsnit 6.1 så vi hvordan Dijkstra dannede en initial vej der var meget langt fra den optimale. Derfor har vi i denne test lavet en modificering af flatgrid, der gerne skulle illustrere problemet med at bruge grafalgoritmer, til at danne den initielle vej.

Vi har modificeret flatgrid ved at tilføje to bakker, således den diskrete geodætiske kurve går igennem bakkedalen mellem de to. Se figur 6.2 for illustration. Den modificerede udgave af flatgrid findes i filen `gridWithHills.ply`.

I denne test forventer vi at den initielle vej dannet af FMM kan korrigeres til en kurve tættere på den diskrete geodætiske kurve, end den initielle vej, som Dijkstra danner. Dette skyldes en forventning om at FMM danner en mere direkte rute end Dijkstra.

Vi forventer ikke at Dijkstra vil danne en vej igennem bakkedalen, men i stedet danne en initial vej der ligner den der blev dannet i afsnit 6.1.

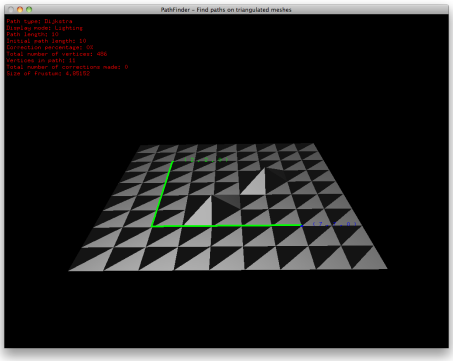
Igen forventer vi at begge grafalgoritmer har identisk længde initielle veje.

Som det ses på figur 6.2(a) danner Dijkstra en initial vej som ikke går igennem bakkedalen. Dette bevirker at når vi prøver at korrigerer den initielle vej ligger den meget langt fra den diskrete geodætiske kurve, se figur 6.2(c). Problemet er at Dijkstras initielle vej, ligger sådan placeret, at den skal blive længere før den kan blive kortere. Kort sagt, den korrigerede kurve skal over forhindringen før den kan blive kortere. Vores algoritme tillader ikke at kurven bliver længere i en iteration, jvnf. (4.1).

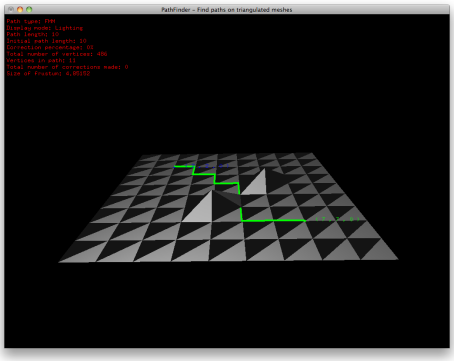
FMM klarer sig væsentlig bedre i denne test. På figur 6.2(b) ses det at FMM finder en initial vej igennem bakkedalen. Dette bevirker at vi kan korrigere den initielle vej, til en diskret geodætisk kurve, se figur 6.2(d).

Der er ingen forskel på længden af de initielle veje. Men der er stor forskel på længden af de korrigerede kurver. Den initielle vej dannet af Dijkstra kan korrigeres med $14,33\%$ og længden er $8,56$. Den initielle vej dannet af FMM kan korrigeres med $28,51\%$ og længden er $6,43$. Den initielle vej dannet af FMM kan derfor korrigeres til en væsentlig kortere kurve end den initielle vej dannet af Dijkstra.

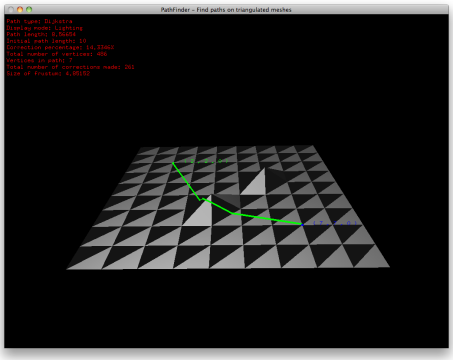
Kigger vi på antallet af iterationer, ses det også tydeligt at FMM klarer sig bedre. Hvor den initielle vej dannet af Dijkstra skal korrigeres 261 gange, før den ikke kan korrigeres mere, skal den initielle vej dannet af FMM kun korrigeres 28 gange, før den ikke kan korrigeres mere.



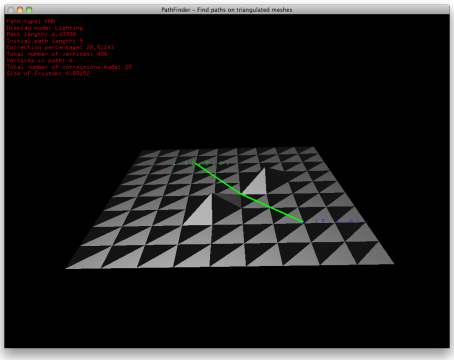
(a) Initial kurve dannet af Dijkstra algoritmen. Vejen har en længde på 10, og indeholder 11 knuder.



(b) Initial kurve dannet af FMM algoritmen. Vejen har en længde på 10, og indeholder 11 knuder.



(c) (a) korrigeret. Vejen er fremkommet efter 261 iterationer af vores korrigeringsalgoritme. Vejen har en afstand på ca. 8,56 og er blevet forbedret med ca. 14,33%.



(d) (b) korrigeret. Vejen er fremkommet efter 28 iterationer af vores korrigeringsalgoritme. Vejen har en afstand på ca. 6,43 og er blevet forbedret med ca. 28,51%.

Figur 6.2 – Sammenligning af Dijkstra og FMM på en overflade med forhindringer.

6.3 Analytisk kendt figur

Formålet med denne test, er at sammenligne længden af de kurver vores implementation laver, med en analytisk udregnet afstand. Vi kan lave denne sammenligning ved at teste på en figur som vi kender den analytiske repræsentation af. I denne test har vi valgt kuglen som analytisk kendt figur.

Den analytiske geodætiske afstand på en kugle kan udregnes som vi gjorde i (3.2).

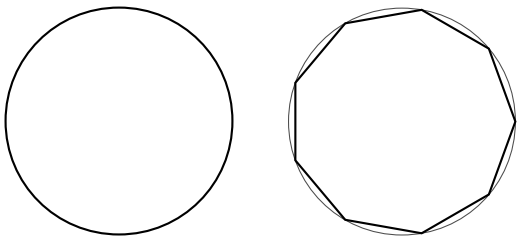
Lad radius på vores kugle være $r = 127$. Lad punkterne $p = (70, 97, 39)^T$ og $q = (109, -12, -64)^T$ være punkter på kuglens overflade, da er $\widehat{p} \approx (0,5563; 0,7709; 0,3099)^T$ og $\widehat{q} \approx (0,8584; -0,0945; -0,5040)^T$. Vi kan nu udregne den analytiske geodætiske afstand som

$$\delta(p,q) \approx 127 \cdot \cos^{-1}\left(\begin{pmatrix}0,5563\\0,7709\\0,3099\end{pmatrix} \cdot \begin{pmatrix}0,8584\\-0,0945\\-0,5040\end{pmatrix}\right) \approx 167,59 \tag{6.1}$$

Figur 6.4 viser en tilnærmelse til en kugle, dannet med et trekantsnet. Radius på den tilnærmede kugle er den samme som den analytiske kugle, dvs. 127.

Figuren kan findes i filen `sphere.ply`.

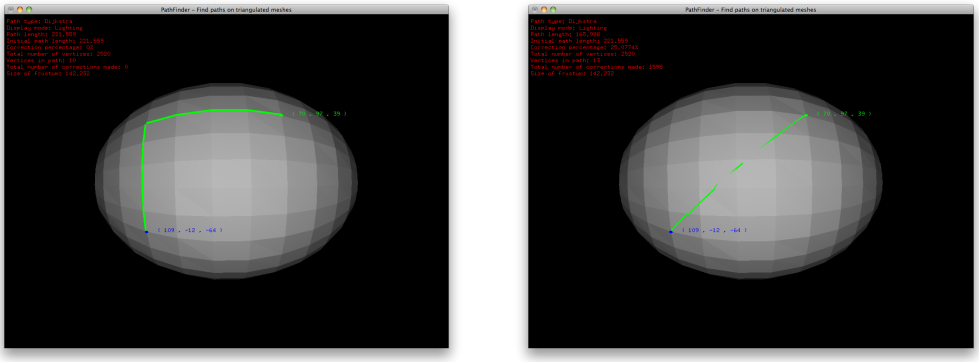
Vi forventer vores implementation finder en kortere kurve end den analytiske. Dette skyldes det faktum at vores implementation udregner vejen på baggrund af diskrete data, og derved skærer storcirklen af i nogle tilfælde.



(a) Cirkel (b) Segmenteret cirkel

Figur 6.3 – Illustration af segmentering af cirkel.

Som det ses på figur 6.4(b) har vores implementation fundet en kortest kurve med længde 165,98. Dette tal skal sammenlignes med den vi udregnede analytisk i (6.1). Længden af den kurve vores implementation udregner er altså ca. 1,52 kortere. Dette var også forventet. Testen viser at



(a) Initial vej dannet af Dijkstra algoritmen. Vejen har en længde på 221,55 og indeholder 10 knuder.

(b) (a) korrigeret med 1.598 iterationer. Den korrigerede kurve har en længde på 165,99 og indeholder 13 knuder.

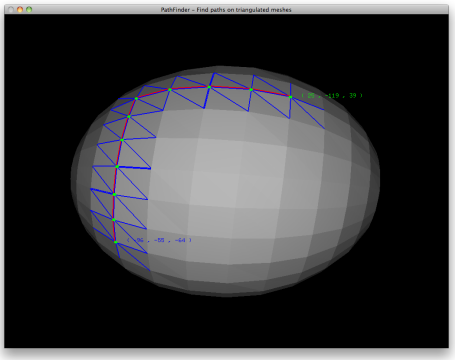
Figur 6.4 – Sammenligning af vores implementation med analytisk udregning

vores implementation laver en kurve, hvis længde ligger meget tæt op af den analytiske udregnede geodætisk afstand.

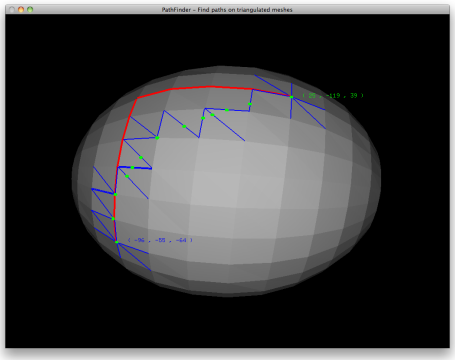
6.4 Udvikling af korrigering

Denne test viser hvorledes den oprindelige vej, tilnærmer sig en diskret geodætisk kurve. Figuren er igen *sphere* og kan findes i filen `sphere.ply`.

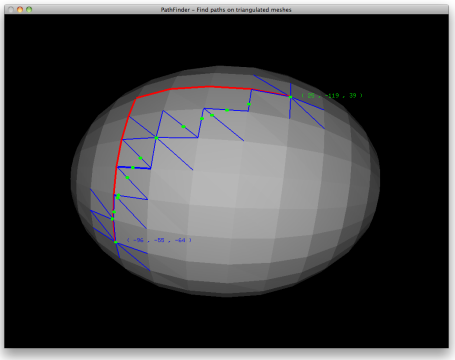
Figur 6.5 viser detaljerne i vores korrektionsalgoritme. Figur 6.5(a) viser den initielle vej dannet af Dijkstra. De grønne punkter er knuder på vejen, og de blå linier er de retninger, som de interne knuder kan bevæge sig i. Den røde vej, er den initielle vej. Jo mere vi korrigerer jo mere flytter de interne knuder sig mod den diskrete geodætiske kurve.



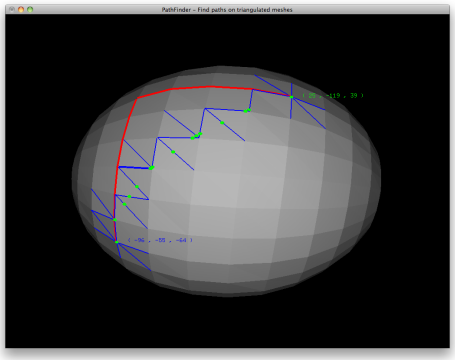
(a) Initial vej dannet af Dijkstra



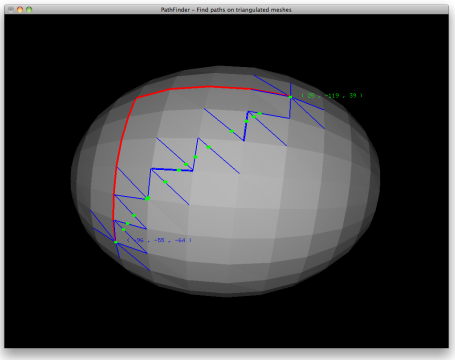
(b) Korrigeret 144 gange



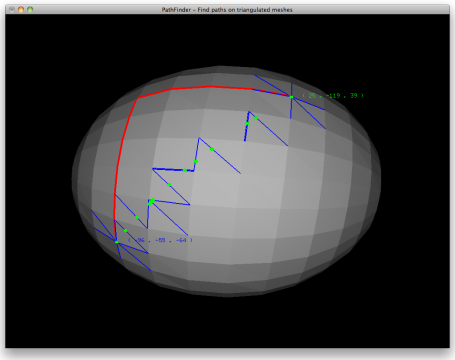
(c) Korrigeret 521 gange



(d) Korrigeret 883 gange



(e) Korrigeret 1.522 gange



(f) Korrigeret 3.202 gange

Figur 6.5 – Illustration af selve korrigeringen. De blå kanter er de kanter som knuderne i vores kurve kan bevæge sig på.

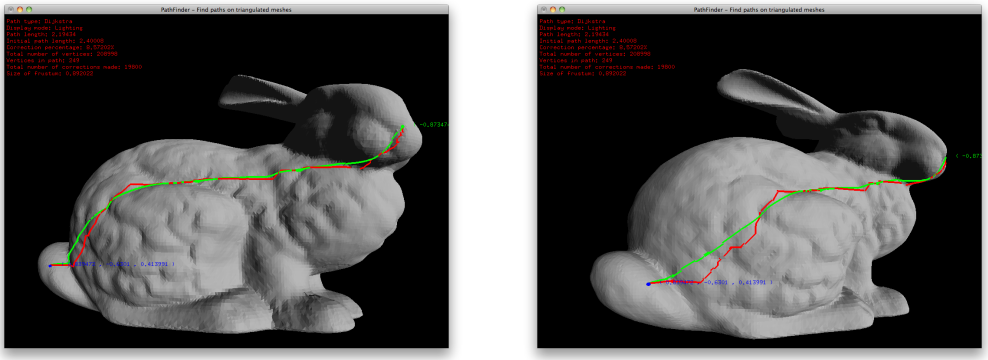
6.5 Stanford Bunny

I denne test har vi valgt at teste vores implementation på en kompleks figur. Figuren vi har valgt er den berømte Stanford Bunny.

Figuren er hentet fra Stanford universitetets hjemmeside[1]. Figuren består af 34.835 knuder og 69.666 trekanter. Figuren kan findes i filen `Bunny_fixed.ply`.

Figur 6.6 viser den initielle vej fremkommet ved Dijkstra, samt den korrigerede kurve. Den initielle vej er på figuren farvet rød og den korrigerede kurve er farvet grøn.

Den initielle vej har en længde på 2,40. Efter vi har korrigeret den initielle vej 19.800 gange har den korrigerede kurve en længde på 2,19 og er derfor blevet 8,57% kortere. Den korrigerede kurve består af 249 knuder.



(a) Initial Vej og korrigeret kurve på stanford Stanford bunny

(b) Figur 6.6(a) roteret

Figur 6.6 – Den initielle approksimerede vej rød og den korrigerede kurve er grøn. Den korrigerede kurve, er fremkommet ved at 19.800 iterationer. Den totale længde af vejen er blevet 8,57% kortere. Det ses tydeligt at den initielle vej er blevet korrigeret således dens kanter er blevet glattet ud.

På figur 6.6(b) ses det tydeligt at korrektionen har gjort den initielle vej mere glat og direkte.

[1]The Stanford 3D Scanning Repository, http://graphics.stanford.edu/data/3Dscanrep/

Kapitel 7

Konklusion

Vi har implementeret et program, som kan finde korte kurver på krumme flader, hvor de krumme flader er defineret vha. et trekantsnet. Programmet er implementeret som beskrevet i analyse og implementationsafsnittet.

Det implementerede program udregner og viser korte kurver mellem to punkter på en diskret overflade spændt ud af et trekantsnet. Programmet udregner en initial vej vha. én af to grafalgoritmer. Den initielle vej korrigeres vha. metoder beskrevet i analyse afsnittet.

Vi har implementeret to grafalgoritmer til at udregne den initielle vej. Dijkstras korteste vej-algoritme og Fast Marching Method (FMM). Af disse er det FMM der er mest anvendelig, fordi den danner en initial vej, som ligger tættere på den diskrete geodætiske kurve. Antallet af nødvendige korrektioner bliver derfor minimeret.

Målet med opgaven var at finde korte kurver på krumme flader – dette mål er opfyldt. Vi kan dog ikke garantere at de fundne korte kurver er de korteste. Dvs. vi kan ikke garantere at de konvergerer mod de diskrete geodætiske kurver. Dette skyldes vores brug af grafalgoritmer til at finde den initielle vej. En initial vej, kan være én der er så langt væk fra den diskrete geodætiske kurve, at det ikke kan lade sig gøre at korrigere vejen, således denne konvergerer mod den diskrete geodætiske kurve. Dette problem blev bla. illustreret i afsnit 6.2.

En mulig løsning på dette problem er at korrigere alle korteste veje fundet af grafalgoritmen. Dette ville dog være beregningsmæssigt meget tungt.

Programmet har en fejl i implementationen af FMM, gør at udregningerne er meget langsomme, og til tider forkerte. Dette skyldes til dels den datastruktur vi har valgt at repræsentere vores trekantsnet i og er grunden til at vi har valgt at lave vores test med Dijkstras korteste vej-algoritme.

Sammenligner vi vores implementation med en naiv løsning, som blot består af en grafalgoritme uden korrektion, klarer vores implementation

sig klart bedst. De korrigerede kurver er kortere og glattere end dem produceret af grafalgoritmerne. I nogle tilfælde kan vi korrigere den initielle vej således denne forkortes med 28%. Dette må bestemt siges at være tilfredsstillende.

En fremtidig udvidelse af programmet, kunne omhandle at implementere stopkriteriet, med fejlværdier, som foreslået i afsnit 4.6.

Derudover kunne en udvidelse af korrigeringen til også at omfatte ikke-mangfoldige trekantsnet være interessant. Samtidig kunne det også være interessant at udvide teori og implementation til flerdimensionale rum.

Litteratur

[1] Jon Sporring, *Projektoplæg*, Jon_Bachelor_project_proposals_JSP.pdf, forefindes på Absalon, sidst besøgt d. 4.juni.

[2] D. Martínez et. al., *Computing geodesics on triangular meshes*, Computers & Graphics 29 (2005) 667–675.

[3] Coblenz et. al., *Geodesics: Analytical and Numerical Solutions*, 2007.

[4] M.G. Calkin, *Lagrangian and Hamiltonian Mechanics*, World Scientific Publishing, River Edge, NJ, 1996.

[5] K. Polthier et. al., *Straightest Geodesics on Polyhedral Surfaces*, Mathematical Visualization, Springer, 1998.

[6] C.I. Grima et. al., *Computational Geometry on Surfaces*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.

[7] J. A. Sethian, *A fast marching level set method for monotonically advancing fronts*, Proc. Natl. Acad. Sci. USA Vol. 93, pp. 1591-1595, February 1996.

[8] R. Kimmel et. al., *Computing geodesics paths on manifolds*, Proc. Natl. Acad. Sci. USA Vol. 95, pp. 8431–8435, July 1998.

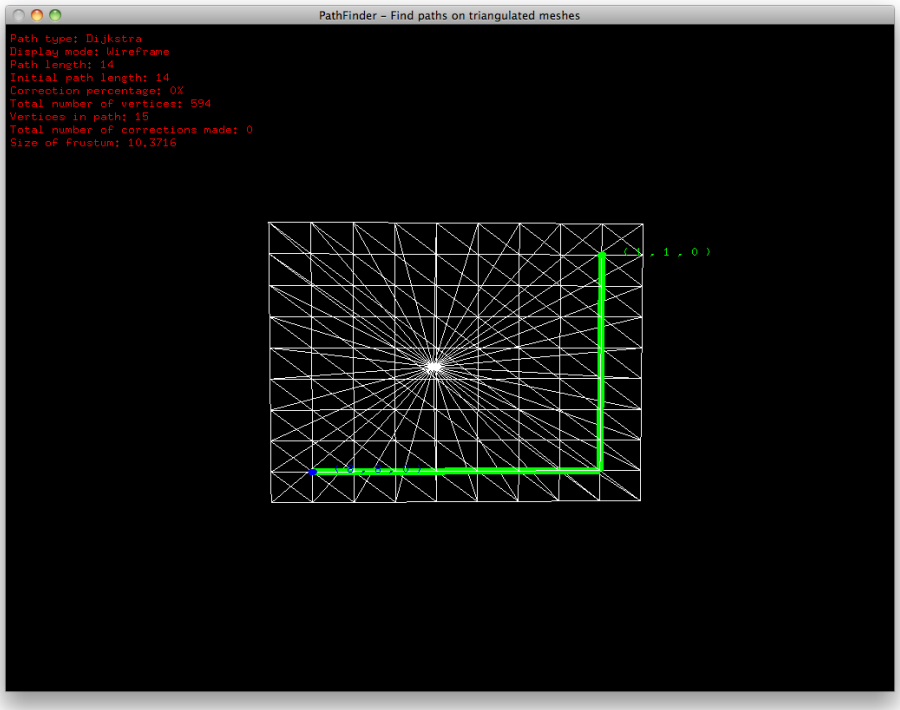
[9] A. Bronstein et. al., *Numerical geometry of non-rigid shapes*, Springer 2008.

[10] A. Prékopa et. al., *Non-euclidean Geometries*, Springer 2006

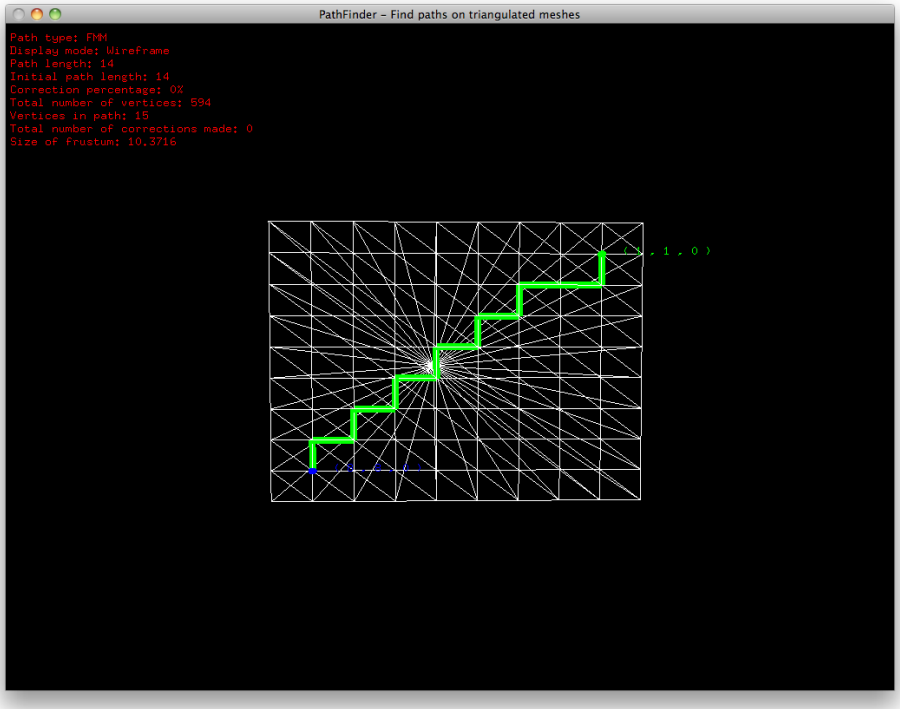
Bilag A

Figurer

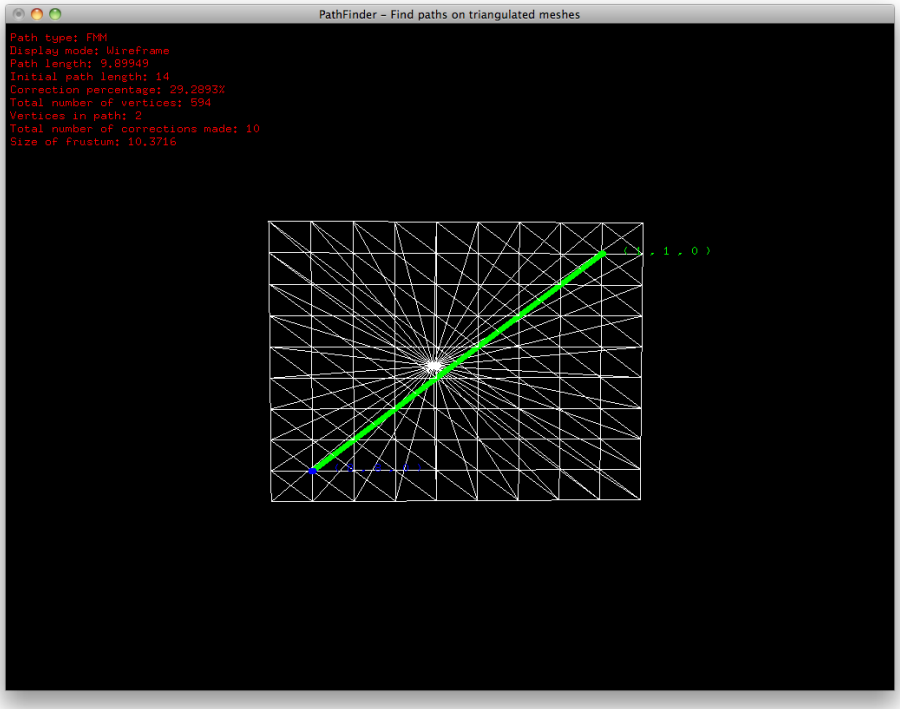
A.1 Test af vej på flatgrid



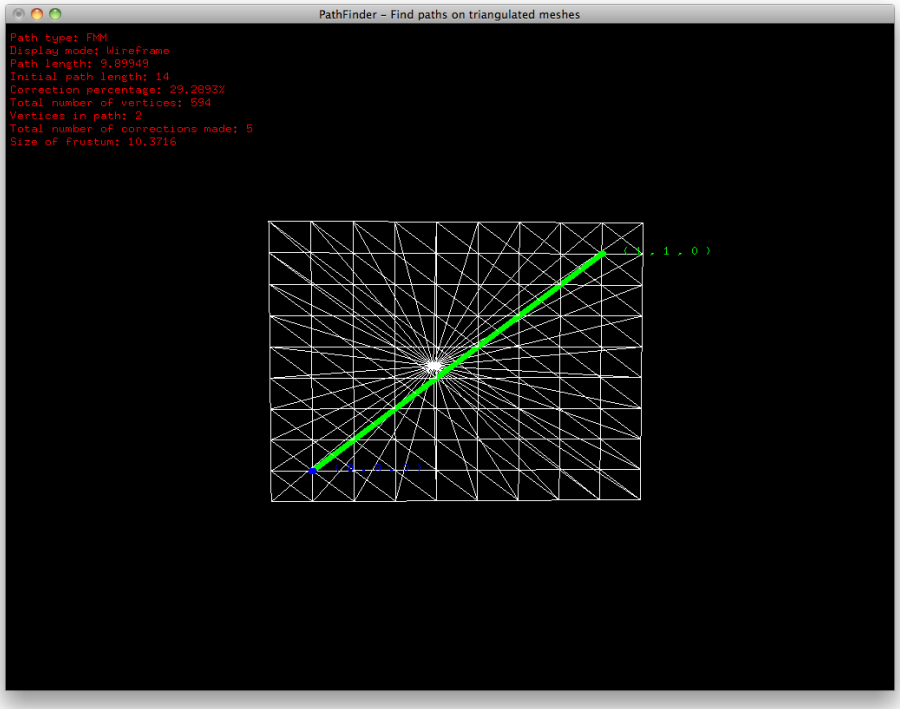
Figur A.1 – Figur 6.1(a) i stor størrelse



Figur A.2 – Figur 6.1(b) i stor størrelse

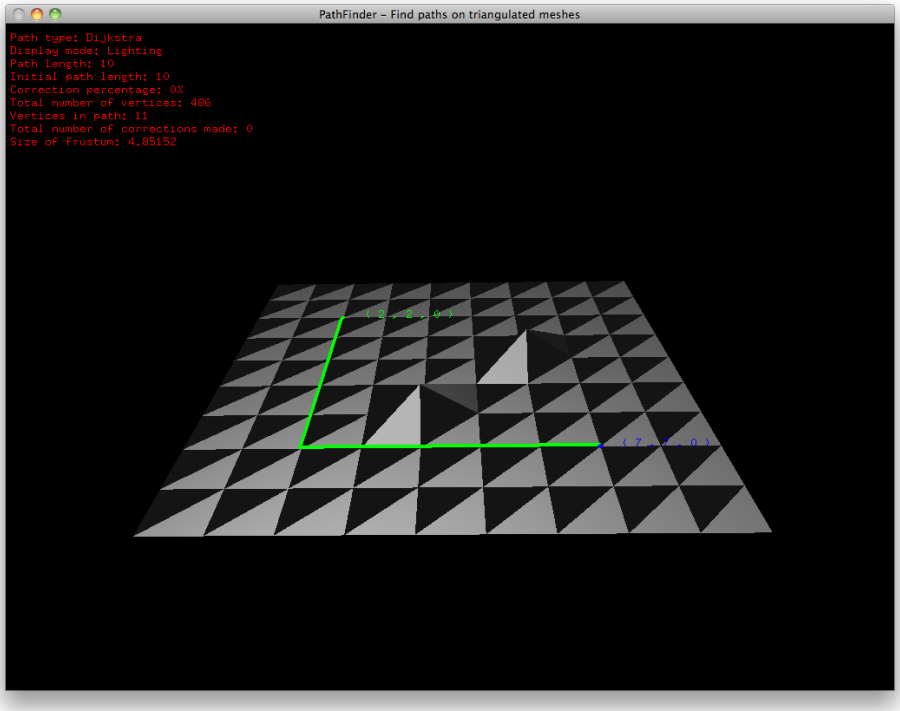


Figur A.3 – Figur 6.1(c) i stor størrelse

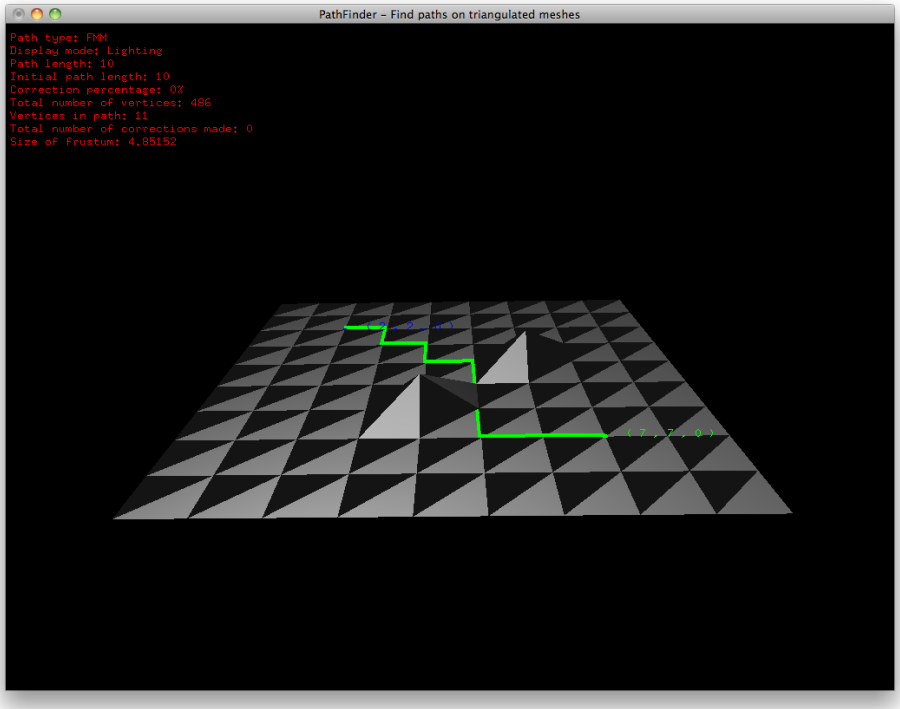


Figur A.4 – Figur 6.1(d) i stor størrelse

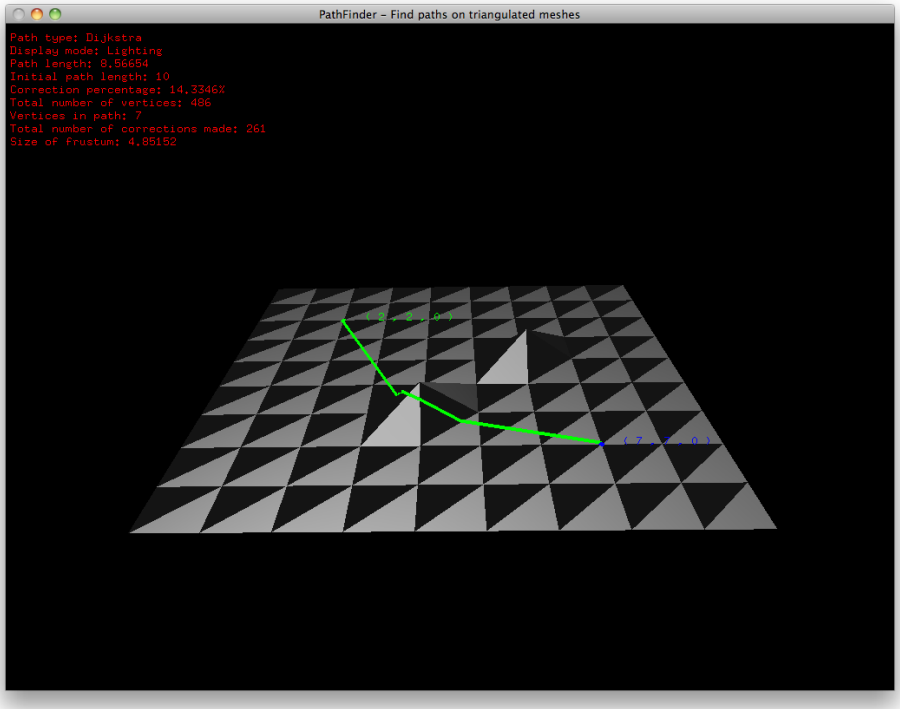
A.2 Forhindringer



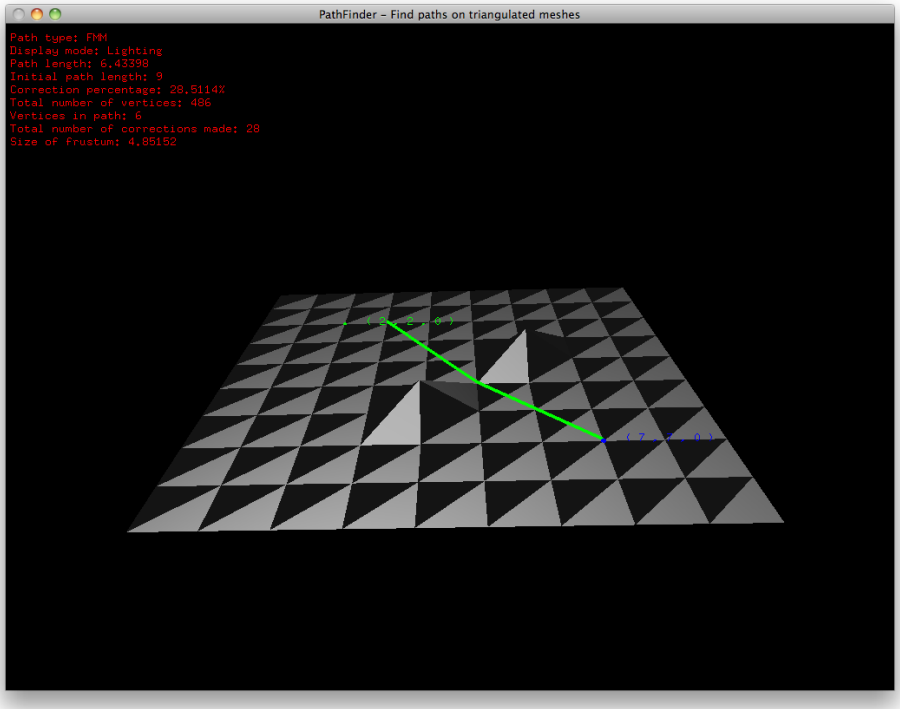
Figur A.5 – Figur 6.2(a) i stor størrelse



Figur A.6 – Figur 6.2(b) i stor størrelse

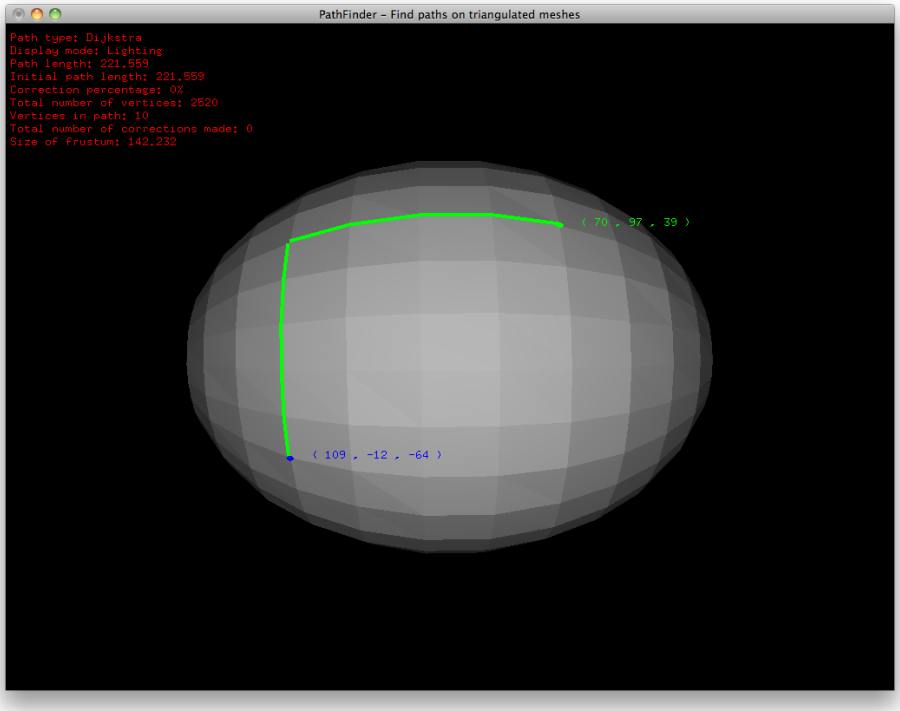


Figur A.7 – Figur 6.2(c) i stor størrelse

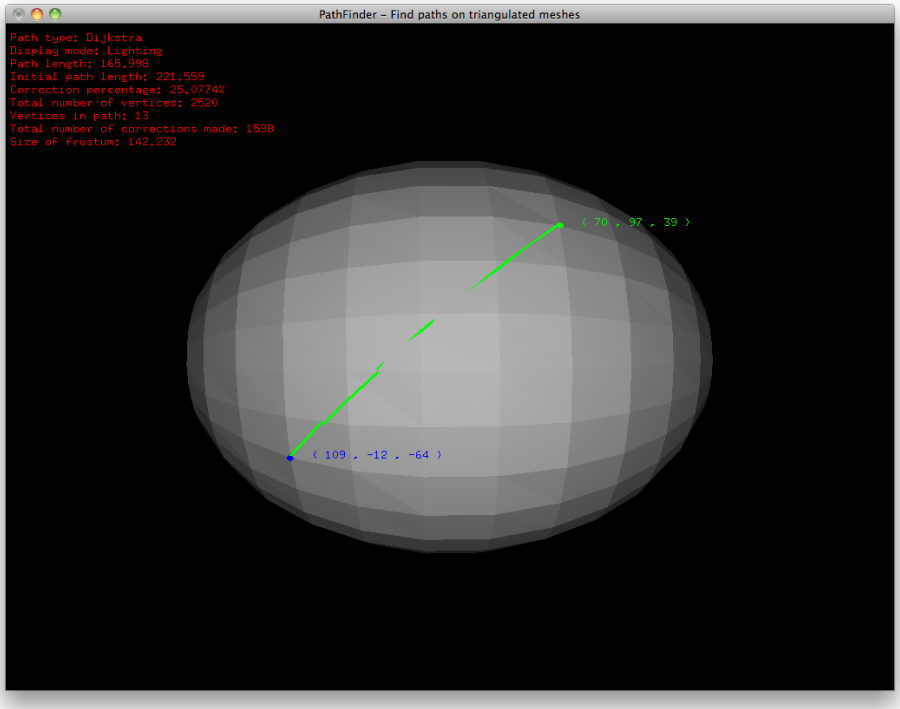


Figur A.8 – Figur 6.2(d) i stor størrelse

A.3 Analytisk kendt figur

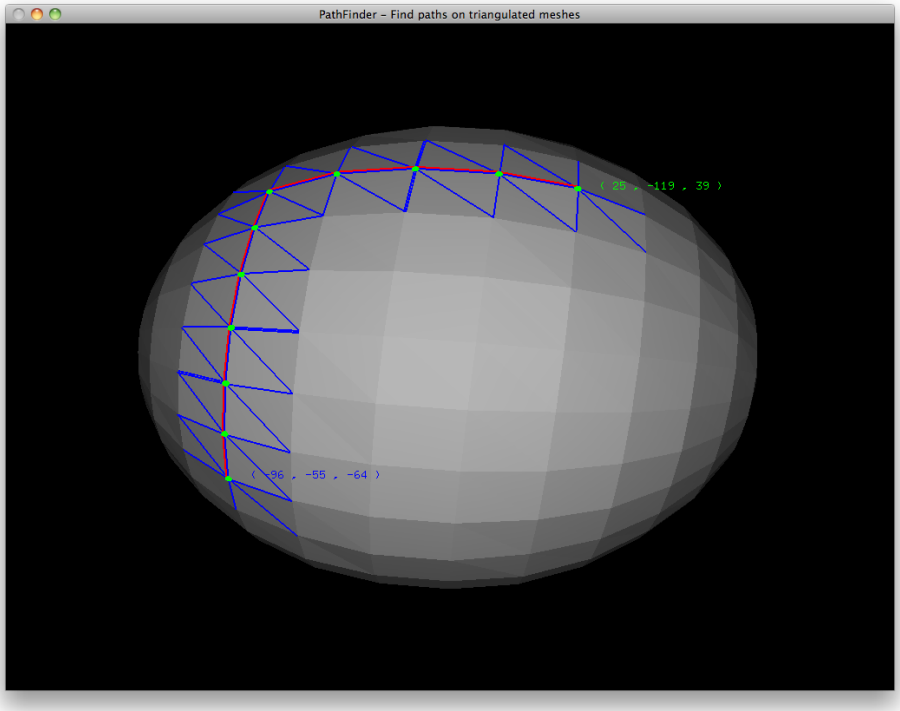


Figur A.9 – Figur 6.4(a) i stor størrelse

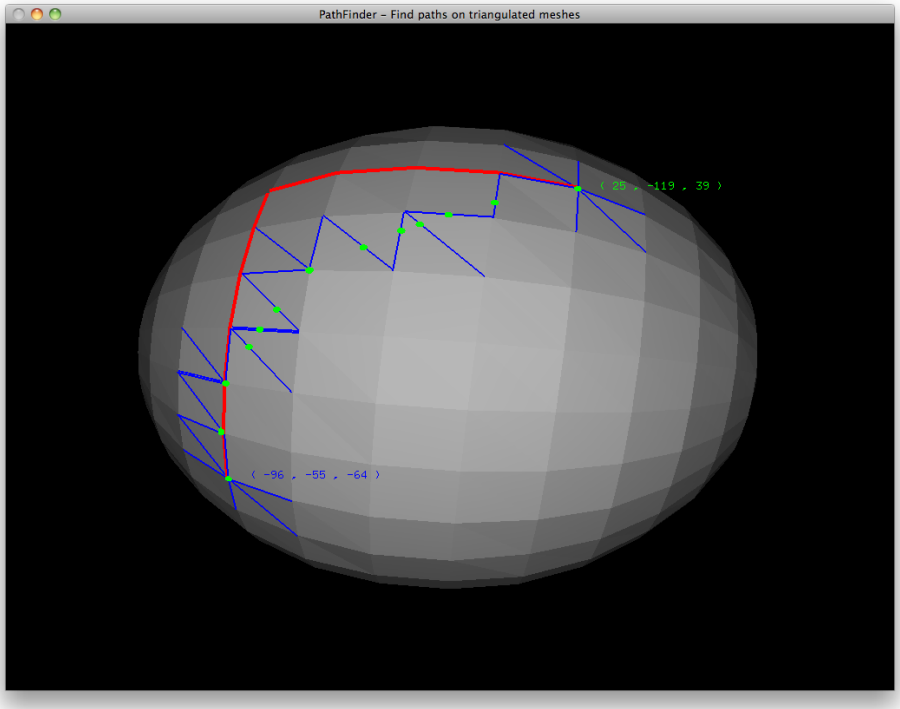


Figur A.10 – Figur 6.4(b) i stor størrelse

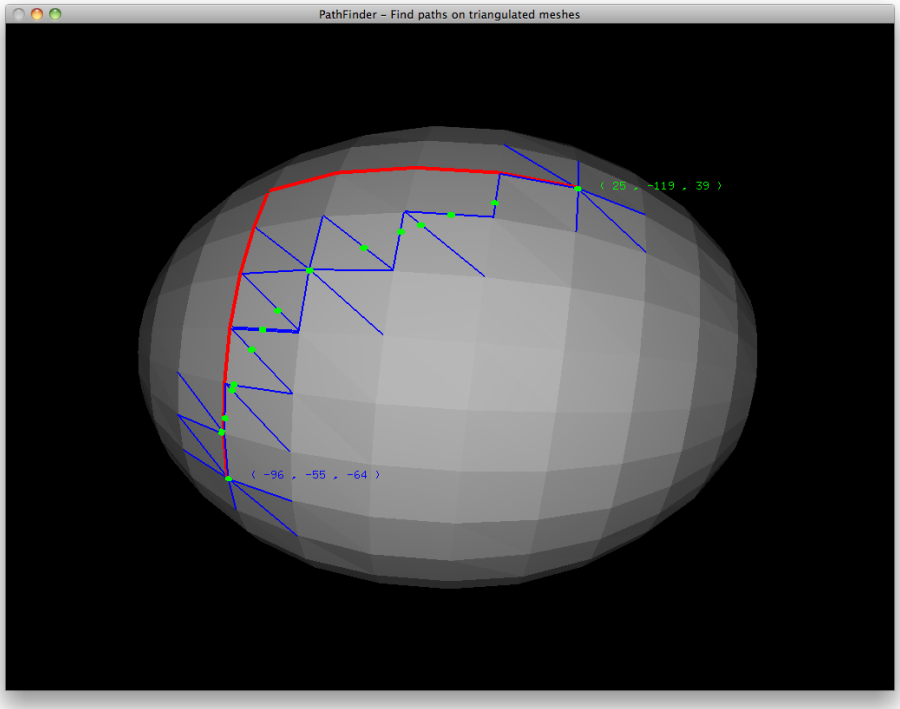
A.4 Udvikling af korrigering



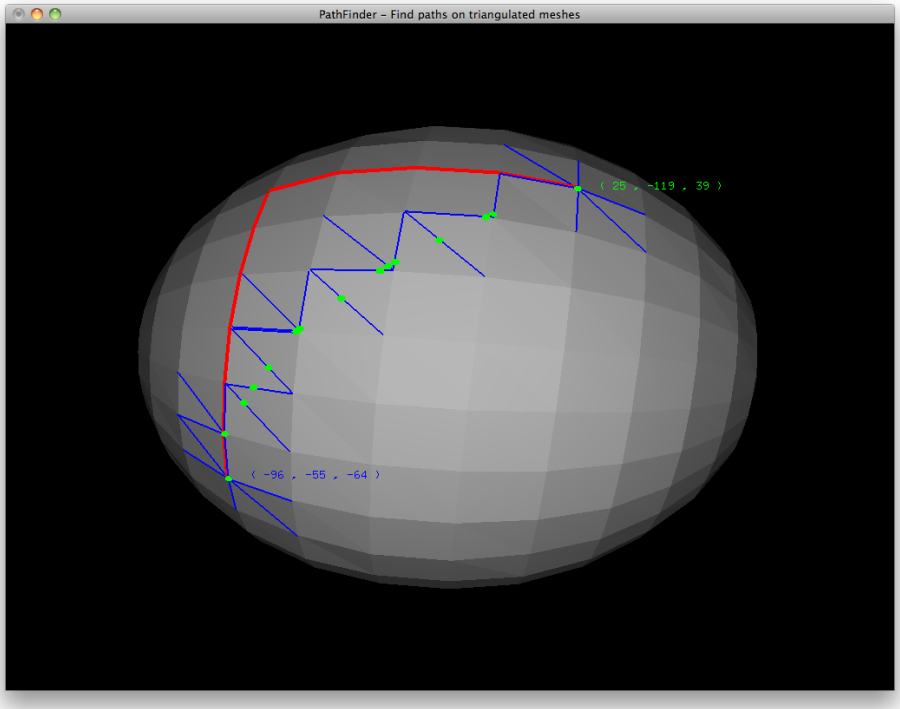
Figur A.11 – Figur 6.5(a) i stor størrelse



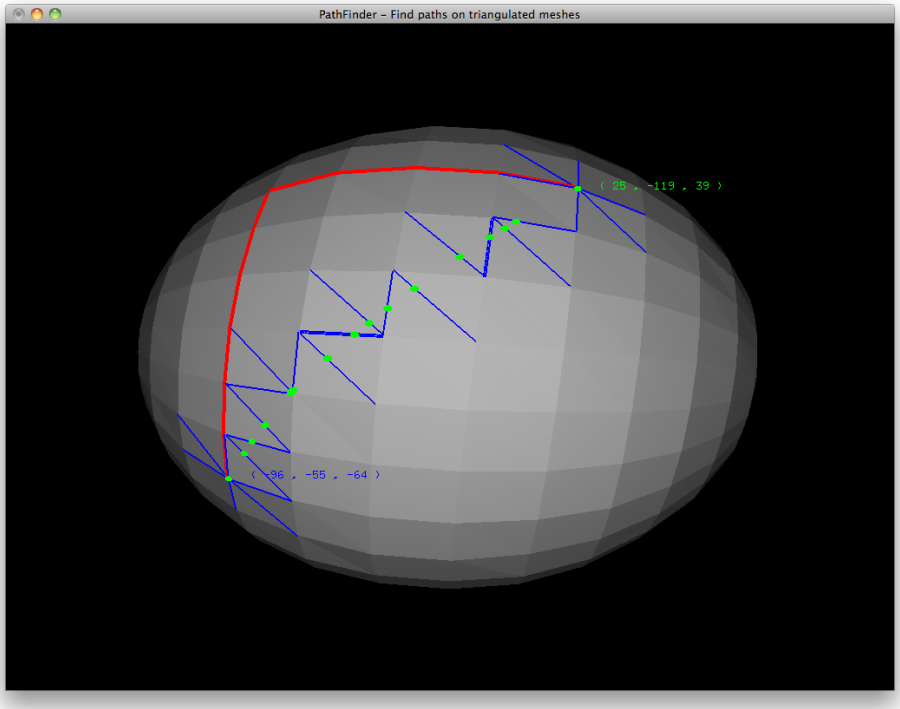
Figur A.12 – Figur 6.5(b) i stor størrelse



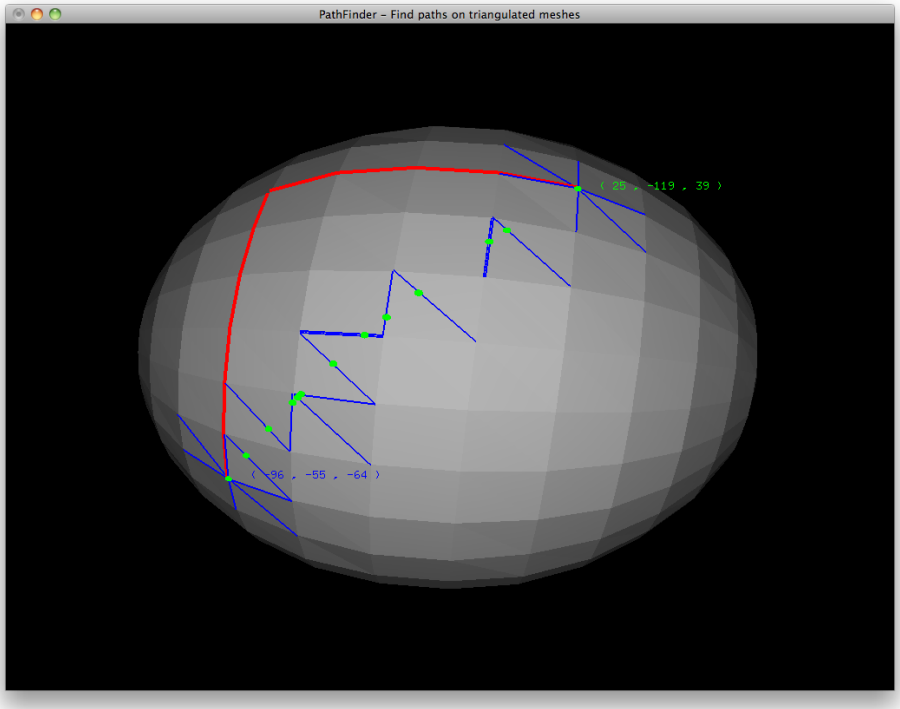
Figur A.13 – Figur 6.5(c) i stor størrelse



Figur A.14 – Figur 6.5(d) i stor størrelse

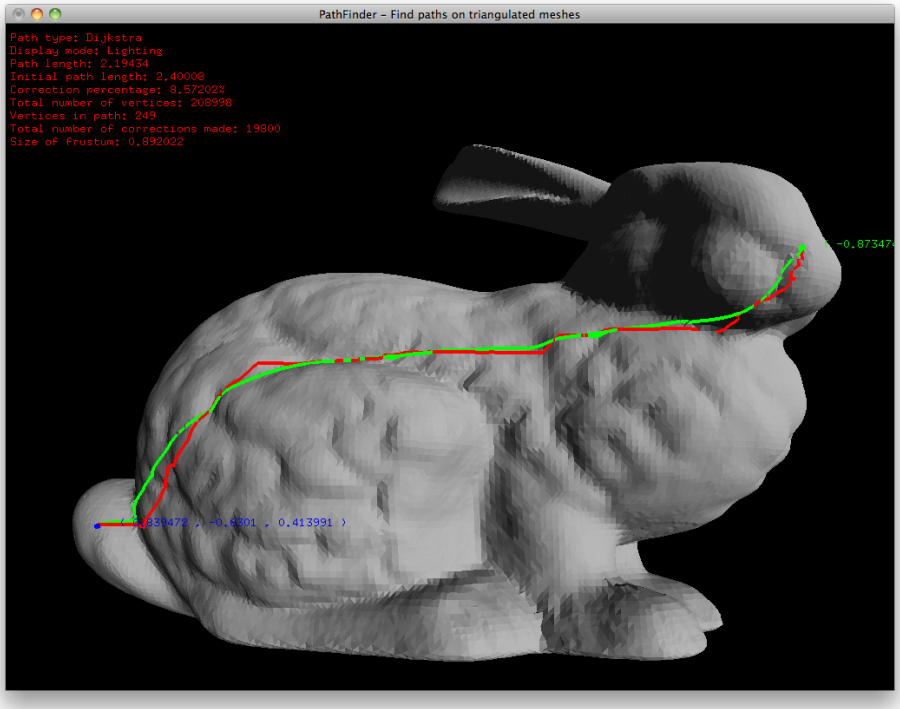


Figur A.15 – Figur 6.5(e) i stor størrelse

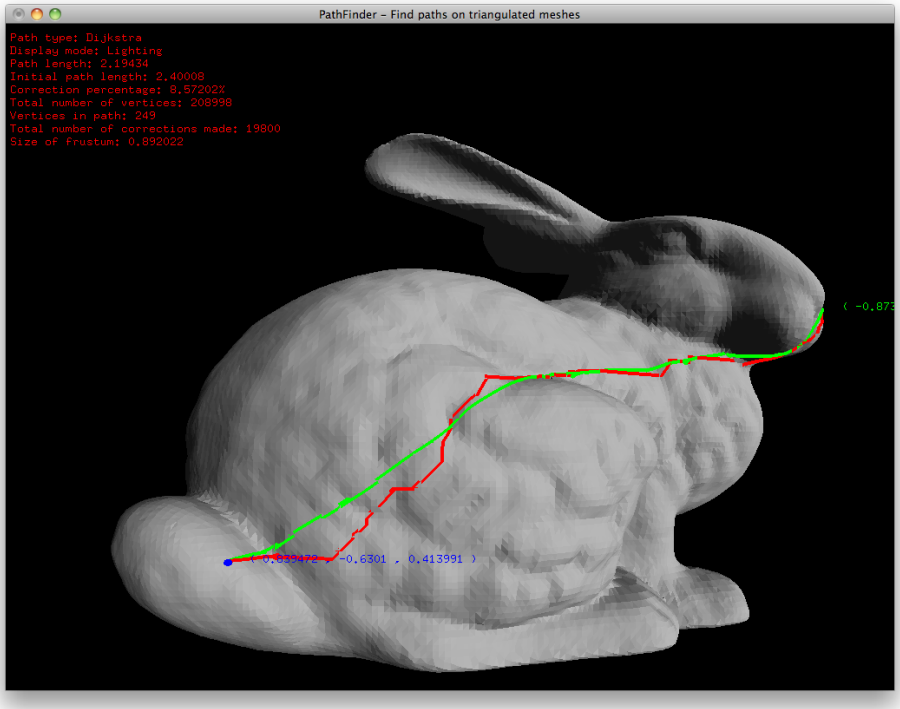


Figur A.16 – Figur 6.5(f) i stor størrelse

A.5 Stanford Bunny



Figur A.17 – Figur 6.6(a) i stor størrelse



Figur A.18 – Figur 6.6(b) i stor størrelse