Speciale

3-D REPRÆSENTATION OG VISUALISERING AF EKSISTERENDE 2-D GIS DATA

Eksamensprojekt (civil)

Danmarks Tekniske Universitet
Informatik og Matematisk Modellering
I samarbejde med Geograf A/S

ECTS Point: 30
Afslutningsdato: 01-07-2007

Vejleder Allan Aasbjerg Nielsen (aa@imm.dtu.dk)

Morten Kvistgaard
(s002022)

Abstrakt

Dette dokument omhandler metoder og analyser, vedrørende konverterting af 2D-GIS-data til 3D, med efterfølgende visualisering. Dette indbefatter konvertering af 2D-vektor-GIS til TIN-flader, samt generering af bymodel-objekter. Projektet undersøger derudover ovenstående processers mulighed for praktisk implementation og anvendelighed.
Projektets terræn-konvertering viser sig, at være både opnåeligt datamæssigt og i praktisk brug, om end emner som Gouraud-shading, viser sig at kræve mere optimerede algoritmer.
Projektets generering af bymodel-objekter, viser sig at være opnåeligt i praktisk brug, men lider dog af datamæssige mangler, grundet manglende dansk standardisering og digitalisering, indenfor området.
Et afsluttende interview konkluderer projektets resulterende programmel, for praktisk anvendeligt, erhvervsmæssigt set.

Abstract (English)

This paper contains methods and analysis regarding conversion of 2D-GIS-data to 3D, with afterwards visualizing thereof. This includes conversion of 2D-vector-GIS to TIN-surfaces and generation of city model objects. The project also investigates implications and usability of practical implementation of the above processes.

The terrain conversion of the project turns out to be both data like and practical achievable, although subjects like Gouraud shading turns out to require better optimized algorithms.

The generating of the city model objects, turns out to be achievable considering practical application but suffers however from insufficient standardization and digitalizing throughout the available Danish data.

A conclusive interview ends the paper by deeming the resulting program of the project, practical useable regarding further commercial use.

Indholdsfortegnelse

Bilag findes i 2. sektion af rapporten.

Af hensyn til papirforbruget, så forefindes dele af 2. sektion (der fylder ca. 480 sider) kun på den vedlagte CD.

1 Forord

Denne rapport indeholder beskrivelse og analyse af de teknikker der skal til for at vise og behandle 2D-GIS-data i 3D. Rapporten bygger desuden på et konkret udviklingsprojekt, lavet i samarbejde med GIS-virksomheden Geograf A/S. Projektet er grundet virksomhedens interessegruppe, primært henvendt til MapInfo-miljøet.

2 Læsevejledning

Undervejs i rapporten vil der blive brugt en række fag- og engelske termer i teksterne. De termer der bedømmes for specifikke eller for uigennemskuelige vil blive skrevet i *kursiv* og vil optræde i ordlisten, sektion 12.
Igennem rapporten vil navnet ”GIS3D” forekomme flere gange. Dette er det foreløbige navn på projektets resulterende applikation.
Når der i rapporten refereres til ”MapInfo”, henvises der til programmet ”MapInfo Professionel”. Ordet ”MapInfo” er, hvis det skal være helt korrekt, navnet på det firma der udgiver programmet.

3 Problemformulering

Projektets problemstilling drejer sig om 3D-visualisering af eksisterende 2D-GIS-data og har været under udarbejdelse og er et resultat af flere års erfaring indenfor GIS-konsultation, primært i de kommunale arbejdsmiljøer. GIS er i dag meget udbredt i de danske kommuner og er flere steder det primære værktøj i de tekniske afdelinger. Den fornyelige større milepæl i kommunal-historie, kommunesammenlægningen har ydermere forstærket behovet, for mere organiserede arbejdsmetoder og værktøjer. I forsøget på at højne abstraktionsniveauet og produktiviteten i disse, vil dette projekt derfor undersøge muligheden for, at på en naturlig og effektiv måde, at indføre den tredje dimension i generel GIS-brug. 3D-dimensionel GIS vil modsat den gængse 2D give mulighed for:

- Hurtigere/bedre orientering eller abstraktion
- Korrekt positionering af kort-elementer
- Udpegning og navigering i den vertikale retning
- 3D-analyser og tematisering

Lidt groft sagt, så vil 3D-GIS give 50% mere GIS, end det vi har nu.
Projektet vil derfor søge at tage det første skridt imod dette.
Dette vil blive søgt løst via følgende 2-trins-model:

- Konvertering af 2D-data
- Visualisering af 3D-data

Flere 3D-GIS programmer og løsninger findes allerede, i flere af de danske kommuner. Fælles for disse er dog, at ingen af dem bruges i sagsbehandlingen og generelt GIS-brug. Den almindelige GIS-bruger ser kun disse 3D-løsninger til medarbejdermøder og show cases. Ydermere indbefatter disse løsninger også ofte, at store resurser skal afsættes til digitalisering af kommunen i 3D. Resurserne der ofte befinder sig i millionklassen, bruges bl.a. til fotografering af byernes husfacader, så disse

kan indsættes i programmerne og på den måde skabe livagtige *fly throughs*. Efter at 3D-digitaliseringen er overstået, står kommunen tilbage med et 3D-program, indeholdende en elektronisk modelby.
Dette projektet vil derfor søge at undgå dette scenario, ved at tage udgangspunkt i de eksisterende værktøjer og data og derefter belyse de nødvendige tiltag, der skal til for at tilføje den tredje dimension i arbejdsgangene. Dette gøres ikke ved, blot at lave et 3D-GIS-program, men derimod ved, at lave et program der kan vise eksisterende 2D-kort i 3D og som der tillader forsat viderebearbejdning i 2D-programmer.
Da GIS-verdenen, eller i hvert fald Geograf og dens kundegruppe, altid har været meget 2D-fikseret, skal projektet også udgøre indgangsportalen for indtræden i 3D. Ment på den måde, at projektet skal belyse og udvikle teknikker til at behandle 3D-grafik og GIS, således at der i fremtiden kan forbedres og viderebygges på disse.
På længere sigt håber vi, helt at kunne afskaffe 2D-tankegangen i GIS-verdenen og på den måde, uanset det givne syns- eller print-perspektiv, altid datamæssigt set, arbejde i 3D.

4 Analyse

Det følgende afsnit vil forsøge at belyse det ønskede funktionalitet og områder indenfor GIS, som projektet vil blive baseret på.

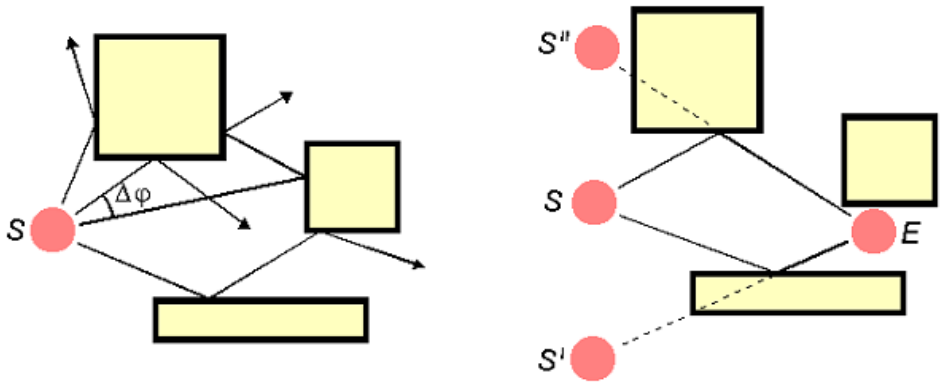
4.1 Kilde- og målgruppe

Det er vigtigt først at gøre klart, hvilke målgrupper vi prøver at nå ud til. Grupper som eksempelvis professionelle digitaliseringsfolk, har dedikeret deres tid og deres uddannelse til 2D og 3D konstruktion af GIS-kort. At nå ud til folk som disse vil derfor kræve fleksible/præcise/tids-optimerede funktioner, hvor brugervenlighed og orientering ikke har høj prioritet. CAD-værktøjer giver ofte den ønskede funktionalitet for denne gruppe.
Grupper som superbrugere og ”kort-knusere” har ligeledes dedikeret meget af deres tid på forståelse af alle koncepter i deres fag og søger derfor også værktøjer, der indeholder alle tænkelige muligheder. Igen er brugervenlig og orientering ikke top-prioritet. Produkter fra eksempelvis Arc, er et godt bud til disse.
Gruppen vi søger at nå ud til, er derfor alle dem der ikke hører under de ovenstående. Dette er klart den største gruppe og er ofte meget overset i GIS-verdenen. Denne gruppe indeholder alle dem, der bruger GIS i deres arbejde, men som hverken har studeret GIS, dedikeret sig i det, eller i det hele taget interesserer sig for det. For denne gruppe betyder brugervenlighed og orientering, ofte forskellen på hvorvidt deres opgave i det hele taget kan udføres og på hvorvidt det er det korrekte resultat der bliver brugt i det videre sagsforløb.
Traditionelt set handler GIS-stillinger, om at fremstille, analysere og producere resultater, til netop denne sidste gruppe, således at disse på en effektiv måde kan høste frugten af GIS, uden selv at skulle bruge årevis på uddannelse og indblik. Indenfor det sidste årti, har flere og flere af de traditionelle GIS-opgaver, dog rykket sig fra ”kort-knuser”-gruppen og fra digitaliseringsgruppen og ud til de respektive sagsbehandlere. Omallokering

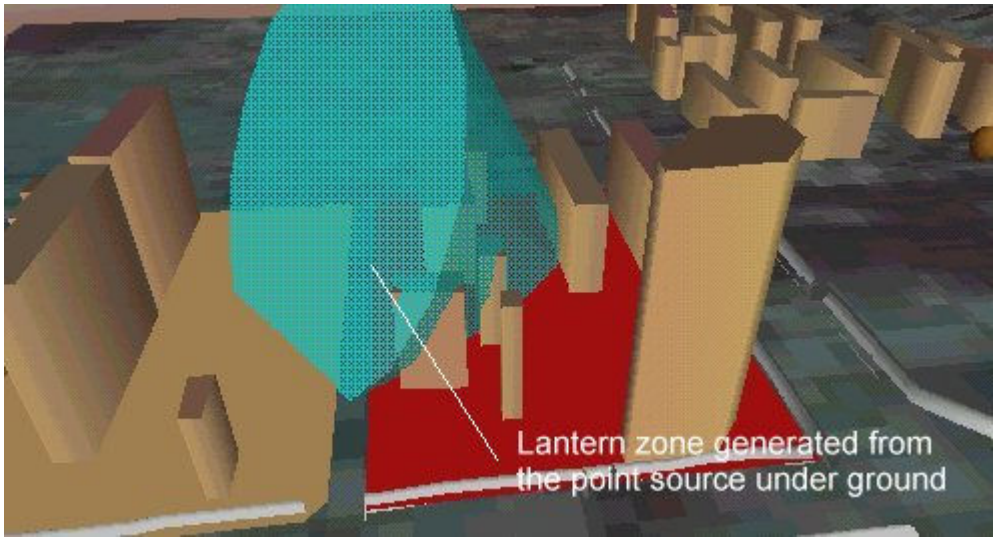
af opgaverne til de respektive konsumere, giver forøget effektivitet i en hverdag, hvor der ikke er tid til "bestilling af GIS-resultater eller digitalisering". GIS er dermed ikke længere et fagteknisk område, kun beregnet på special-uddannede folk og projektet forsøger dermed at nå ud til denne gruppe af brugere.
De ovenstående overvejelser er dog ikke kun baseret på ideologisk tankegang, hvori favorisering af den lille mand i GIS, er målet. Projektet er lavet af og i samarbejde med Geograf A/S, hvor jeg har været ansat som GIS-konsulent, i de sidste 4 år. Dog selvom ingen andre Geograf-ansatte, har dog bidraget implementationsmæssigt til projektet, så er dette ikke uden indflydelse fra erhvervsmæssige aspekter. Målet om at nå ud til den almindelige GIS-bruger, giver erhvervsmæssige fordele i form af målgruppestørrelse og optimeringsaspekter og målet om at rendere 2D-data i 3D og fokuseringen på selve kode-delen, giver mulighed for praktisk implementation af 3D i erhvervslivet. Det akademiske aspekt i projektet, interesserer sig derimod for projektets potentielle konvertering af 2D-, kombineret med implicit 3D-data og dets potentielle mulighed for komplet fjernelse af 2D-GIS-aspektet.
Igennem rapporten vil der derfor forekomme analyser og beslutninger, der er baseret på erhvervsmæssige fordele.

4.2 3D vs. 2D

Projektets umiddelbare formål, er at skabe mulighed og basis for 3D-analyser og videre udbygning, via konvertering og præsentation af data. Eksempelvis så er der et stort potentielt marked, for bedre strålings/bølge-analyser, end hvad der er opnåeligt med 2D-GIS. Dette indfatter bl.a. støj-analyser til brug i kommunal byplanlægning, elektromagnetiske dækningsanalyser til brug for teleselskaber og refleksions-analyser til brug for GPS-bestemmelse. Den mest almindelige måde, at lave lydudbredelses-analyser, i nuværende kommunal GIS-brug, er blot at tegne en buffer omkring det givne lyd-center. Dette vil dog ikke tage højde for eventuelle bygninger, træer eller andre ting, der måtte stå i vejen og dermed påvirke resultatet. Figur 1 viser en 2D-illustration af raytracing, hvilket kan give et godt værktøj til lyd-analyser, da både kilder og modtagere ofte befinder sig i gadeplan. Til analyser for radio-udbredelse og -skygger, kræver beregningerne dog, en 3D-bymodel, grundet kildens højde i forhold til modtageren og i forbindelse med GPS-beregninger, er det primært den vertikal retning der er interessant. Figur 2 viser en kegle-analyse, der modsat den tilsvarende 2D-analyse, kan vise, ikke blot hvilke bygninger der berøres, men også hvor store dele, der bliver og med efterfølgende mulighed for præcise skygge-analyser.



Figur 1: 2D-stråle-analyse (billede fra [KOPPERS])



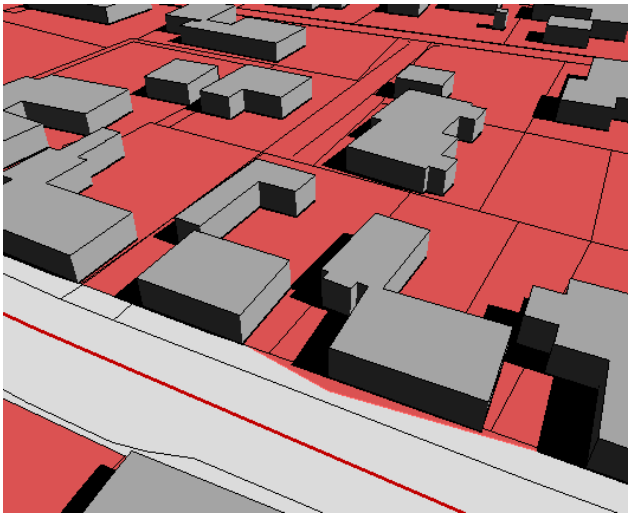
Figur 2: 3D-GIS-kegle-analyse fra [KIM & LEE & LEE]

Derudover så resulterer projektet dog også i nogle konkrete fordele, frem for normal 2D-GIS:

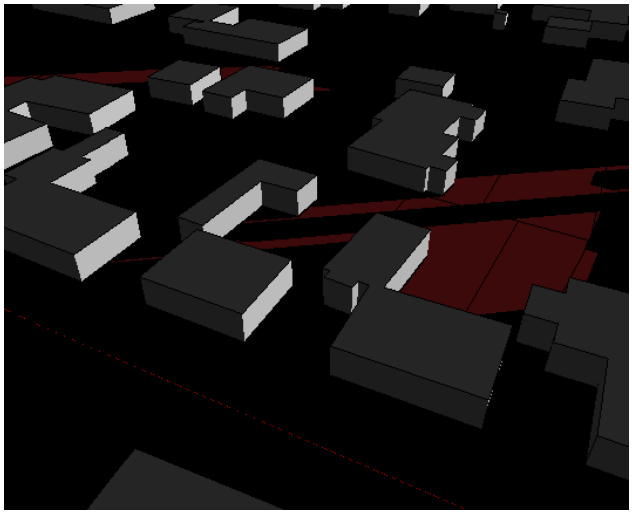
4.2.1 Slagskygger

En umiddelbar fordel i 3D, er muligheden for at rendere slagskygger til kortelementerne. På Figur 3 - Figur 5 ses nogle eksempler på dette. Bygningsskygger er bl.a. interessante for kommunale byggesagsbehandlere. Uden slagskygger fra bygningerne, er det svært og i bedste fald unøjagtigt, at medregne forhold, som påvirkning af naboejendomme, som eksempelvis skygger for lys og vind.
Projektionsskygger [GAFFGA], som dem der er implementeret på figurerne, er mulige at lave i 2D, via samme beregninger, som dem der præsenteres i afsnit 6.3.2, hvis man vel og mærke kender bygningshøjderne og eventuelle tagrygge. Dog er det ikke nogen let sag, i de fleste 2D-GIS-løsninger og udregningerne til det, er per definition også nødt til at være 3D-beregninger.
En mere korrekt form for skygge er volume shadows [BEAM], der medtager informationer som terræn og forhindringer. Disse skygger er,

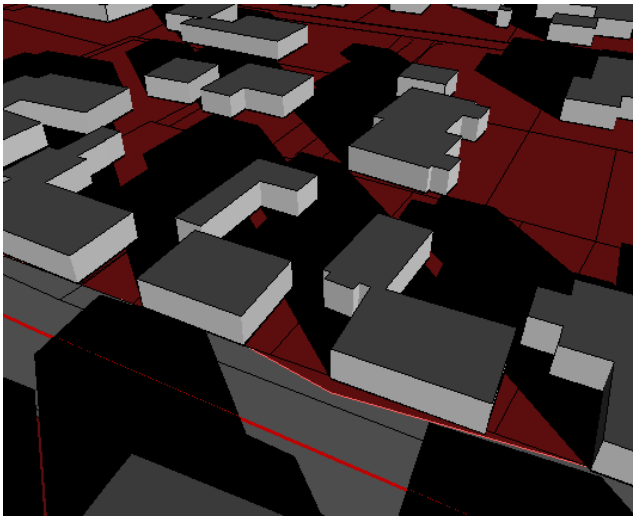
grundet den medtagede 3D-information, umulige at lave via 2D. Selv med en bagvedliggende DTM, ville de beregningsmæssigt (uden brug af 3D-grafikkort) være noget nær umulige at lave i 2D. Og hvis man vil have endnu mere korrekte skygger, som eksempelvis soft shadows [ASSARSSON & AKENINE], så sætter det yderligere krav til 2D-GIS-systemerne, der pt. kan være svære at opfylde.
3D-accelererede programmer, har derimod et væld af indbyggede muligheder og teknikker til skygge-rendering, hvilket gør slagskygger til et oplagt emne at implementere i 3D-GIS. (Dog vil skygger som soft shadows, heller ikke være uproblematiske i hardware-rendering.)
Modsat generisk skygge-rendering eller skygger i spil-applikationer, så betyder retningen og størrelsen på skygger meget i GIS. Skyggerne er defineret via solens placering på himlen, som igen afhænger af tidspunkt på dagen, årstid og breddegrad. Disse ekstra beregninger, skal derfor også medtages, når der behandles skygger.



Figur 3: Aprilsol ved 13-tiden (billede fra GIS3D)



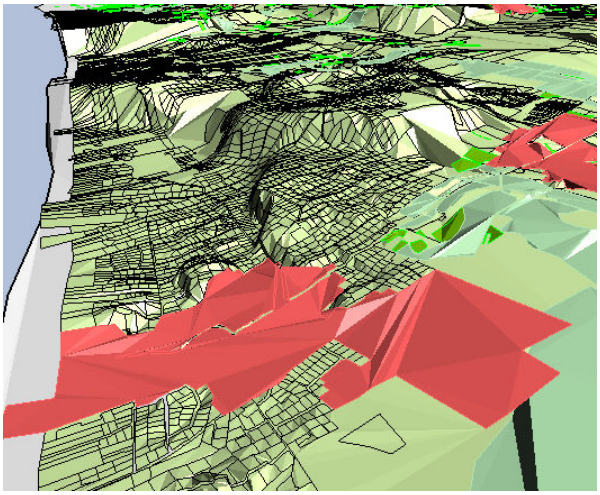
Figur 4: De første solstråler i januar (billede fra GIS3D)



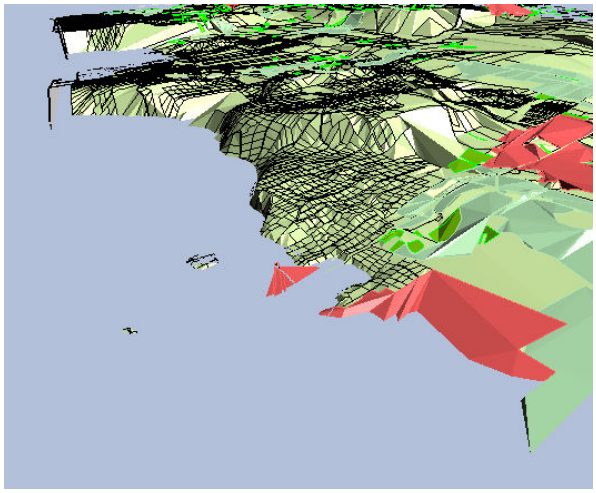
Figur 5: Aftensol i juli (billede fra GIS3D)

4.2.2 Oversvømmelsesanalyser

En analyse der optager mange danske kommuner, er udarbejdelse af oversvømmelses-scenarier. Hvis vandstanden stiger x antal meter, så vil kommunerne have styr på, hvor vandlinien da går. Dette kan bl.a. bruges udarbejdelse af nødsituationsplaner, placering af depoter, bevarelse af infrastruktur mm. Resultatet af en oversvømmelses-analyse (den nye vandlinie), kan naturligvis illustreres på et 2D-kort. Selve beregningerne skal dog foretages i 3D, da vandlinien jo netop afhænger terrænets højdeinformation og en bagvedliggende DTM er derfor nødvendig. 3D-accelerede programmer har indbyggede Z-buffere, til netop at tage sig af den slags problemer og selve beregningen vil derfor blot foregår ved, at man i programmet, hæver den region der repræsenterer vandstanden. Et eksempel på dette kan ses på Figur 6 og Figur 7.



Figur 6: Blokhus (forreste røde zone) vest for Pandrup (billede fra GIS3D)

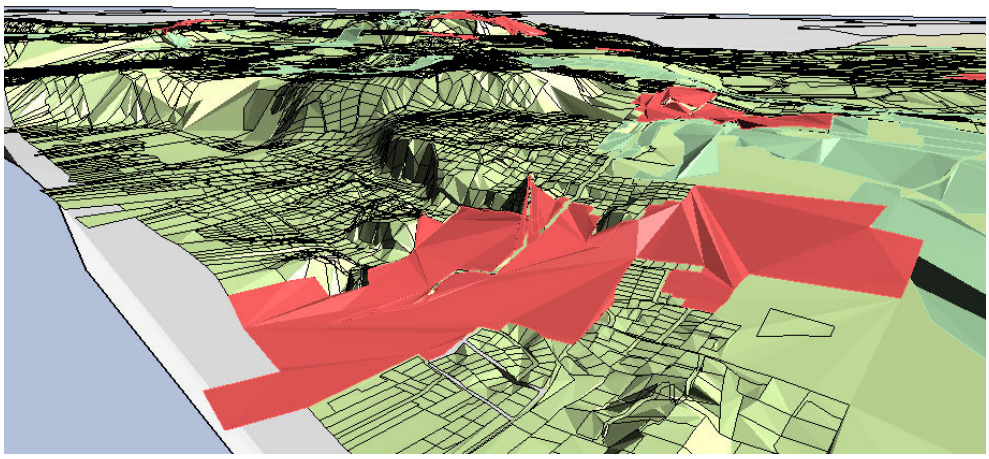


Figur 7: Vandet er nu steget 10 m og har oversvømmet det meste af Blokhus (billede fra GIS3D)

Et andet interessant emne i denne forbindelse, er projektets kombination af terræn- og bymodel-data. Langt de fleste oversvømmelses-analyser og forudsigelser, bygger kun på terræn-data, ligesom scenerne på Figur 6 og Figur 7. Et eksempel på et sådant, kan bl.a. ses i [SINNAKAUDAN & GHANI & KIAT]. Disse scenarier illustrer dog kun en situation, hvor vandstanden er steget relativt langsomt og efterfølgende er stabil i et stykke tid. Et mere realistisk scenario, vil dog indbefatte beregninger og simulering af hurtigt stigende vandmasser, som eksempelvis ved Tsunami'er, dæmningssammenbrud eller orkaner. I disse tilfælde vil terrænet ligeledes udgøre en stor del af datagrundlaget, men bymodel-objekter som eks. bygninger og træer, vil også have stor betydning for resultatet, hvilket ofte er en mangelvare i disse analyser. Dette projekt kan derimod skabe betingelserne, for disse forbedrede oversvømmelsesanalyser.

4.2.3 Terræn

En information man altid har søgt illustreret på kort, er eksempelvis højdeinformation. En hyppig måde at illustrere dette på i 2D-GIS, er via højdekurve-kort eller tematisering. Disse metoder giver dog, i bedste fald stilistisk terrænhøjde-information og i værste fald, ulæselige kort grundet sammenfald af konturlinier og andre kortelementer. 3D-løsninger har derimod fordelen af, at have højdeinformationen indbygget, på linie med x- og y-data. Man får derfor information, som terrænhøjder givet, uden brug af stilistiske hjælpemidler som konturlinier, eller andet der potentielt kan forstyrre overblik og man får derudover et langt mere detaljeret og præcist højdekort, end de tilsvarende konturkort.
Et eksempel på illustrering af terrænhøjder, kombineret med by-, skov- og matrikel-zone-information, kan ses på Figur 8.

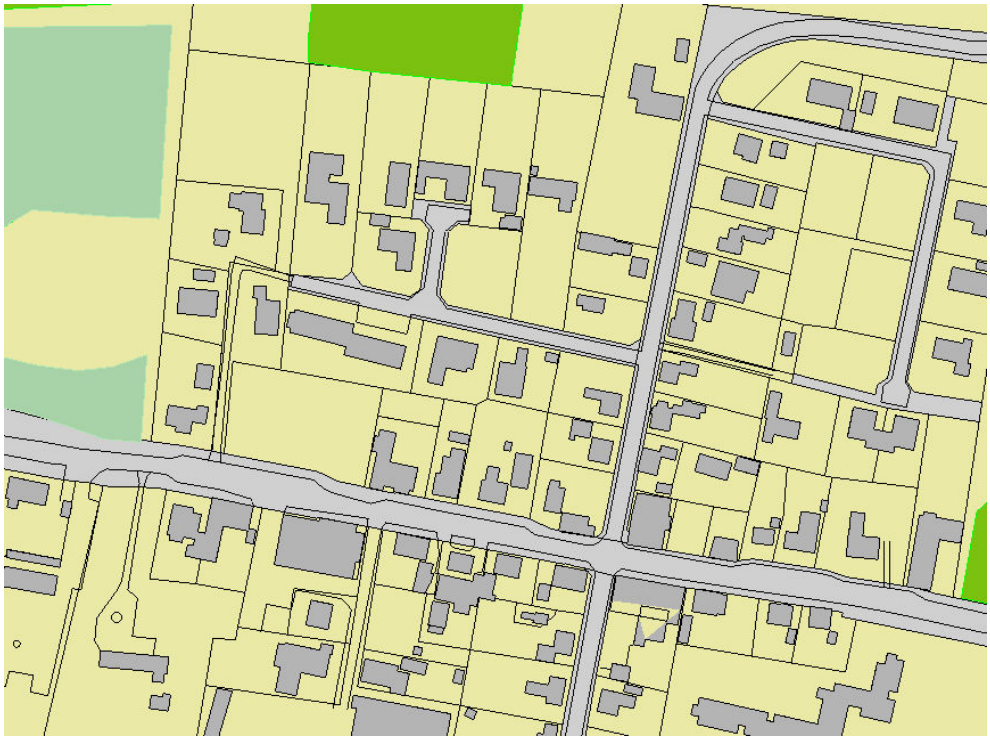


Figur 8: DTM (forhøjet), med farvede zoner, over Blokhus (billede fra GIS3D)

4.2.4 Præsentation

En yderligere fordel ved 3D-GIS, er dets mulighed for mere virkelighedstro/naturlige kortudsnit. Dette giver bl.a. fordele i præsentations-sammenhæng, da man her kan præsentere kortudsnit til både GIS- og ikke-GIS-folk, der giver et bedre situations-indtryk, end de tilsvarende 2D-kort. Figur 9 og Figur 10 giver et eksempel på et kortudsnit set henholdsvis i 3D og 2D. Billederne viser hovedgaden i Hune. Et potentielt byggeprojekt og dets påvirkning af Hune, vil langt bedre kunne bedømmes på Figur 9 end på Figur 10. Især for den almene befolkning.

Figur 9: Hune by i 3D (billede fra GIS3D)



Figur 10: Hune by i 2D (billede fra GIS3D)

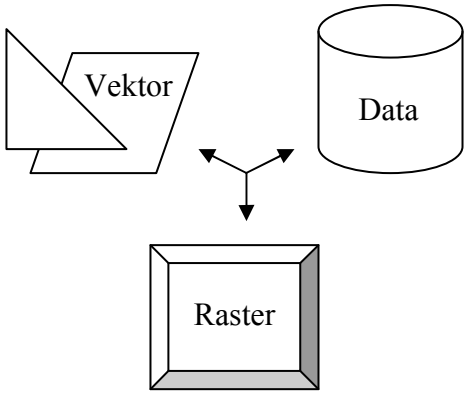
4.3 Kort-repræsentation

Det vigtigste element i "visualisering af GIS-data" er naturligvis selve kortet. Andre former for visualiseringstyper som eksempelvis Laplace/frekvens-diagrammer e.l. er urelevante, set i forhold til målgruppen. Målet er derfor at skabe et interaktivt kort, der kan forsyne brugeren med de fornødne værktøjer og forbedringer. Igennem dette kort vil vores 3D-behandling ligeledes blive visualiseret og gjort tilgængelig.

4.3.1 De 3 datatyper

For at opfylde definitionen på et GIS-system, skal projektet kunne gemme, hente, analysere og sammenkoble de 3 datatyper:

- Vektor
- Data
- Raster



Figur 11: De 3 datatyper skal kobles sammen for at opfylde definitionen

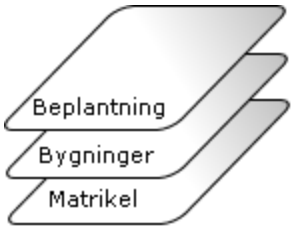
Vektor-data er det primære data, som dette projekt vil focusere på at visualisere. Vektor-data er spatial information der er vektoriseret og også ofte simplificeret. Eksempelvis kan bygninger ofte vektoriseres til rektangler med koordinater opnået via opmåling eller aflæsning af orthofotos.

Det som jeg blot har kaldt for data, er et udtryk for alt det, der ikke har noget direkte spatial information, men som der implicit kan stedfæstes

geografisk. Eksempelvis kan det være informationer om bygningshøjder, tilknyttet til en bygning, eller fradragsoplysninger, tilknyttet en person, der igen er tilknyttet til en adresse.
Den sidste type, raster, er ofte billeddata. Denne er den mest besværlige at foretage beregninger og analyser på og bliver af målgruppen kun brugt til orthofotos. Og dette kun sjældent i kommunalt sagsbehandling. Mængden af information i forbindelse med disse, er derfor ofte kun forbedret/alternativ visualisering. Orthofotos i 2D-løsninger lider dog af de samme perspektiv-problemer som 2D-GIS generelt. En stor forbedring af disse orthofotos, kunne dog være dette projekts 3D-rendering, således at orthofotosene ville blive projekteret ud på et 3D-terræn, i stedet for 2D. Dette er dog ikke målet for projektet og jeg vil derfor lade dette emne ligge, i første omgang.
Til sidst skal de 3 ovenstående datatyper sammenkobles. Dette kan ske ved selve kort-visualiseringen, der kan placere og rendere vektor- og raster-data i samme rendering og på den måde skabe koordinatmæssig sammenhæng. Det ikke-spatiale data, kan kobles til de andre typer, ved at tilknytte dette til vektor- og raster-objekterne, således at brugeren, ved at undersøge et af disse, også kan fremfinde ikke-spatial data.

4.3.2 Lag-opdeling

For at gøre GIS-systemet til et brugbart værktøj, er muligheden for lag-opdeling af kortet en nødvendighed. Lag-opdeling muliggøre modul-opbyggede kort, hvor hver objekttype kan samles i hvert sit lag. Denne form for opdeling, hjælper også til, med arbejdet med kortene, da dette derfor kan specialopbygges af netop de kort, der skal til, at løse opgaven.



Figur 12: Lag-opdeling af kort, således at temaer (eks. bygninger) kan slås til og fra efter behov og er med til at forbedre overblik.

4.3.3 Navigation

For at være brugbart, skal GIS-systemet indeholde en række værktøjer til navigering i kortet. Dette er blot de mest basale funktioner såsom:

- Panorering: Flytte kortet fra side til side eller op og ned.
- Zoom: Forstørre eller formindske udvalgte dele af kortet
- Orbit: I dette projekt forefindes en yderligere form for navigation, der ellers ikke findes i normal GIS. Orbit-navigation bruges i 3D-applikationer, til at bevæge sig rundt om et givent center punkt i 3D. Øjepunktets afstand til centerpunktet, vil altid være det samme, således at øjets potentielle placering forefindes på kuglens overflade, hvis center er i centerpunktet og hvor radius, er øjets afstand til dette. Denne form kaldes også ofte for trackball-navigation.

4.3.4 Projektion

En stor del af energien i GIS-systemer, går med at holde styr på kortenes projektioner. Projektioner er nødvendige grundet 2D-visualiseringen i

traditionel GIS-fremstilling. Altså grundet at Jordens 3D-overflade-koordinater skal visualiseres i 2D. Metoderne til at projektere Jorden ned på et 2D-kort er mangfoldige og mange af dem har tillige samfundsmæssige og politisk rødder.

Den ideelle måde at behandle disse projektioner på, ville i dette projekt være, at transformere de lokale projektioner, tilbage til reelle 3D-koordinater (Geographic Coordinate System, [LO & YEUNG1] eller tilsvarende). På den måde, ville et almindeligt kort blive let kurvet. Dog afhængigt at kortudsnittets størrelse, ville denne krumning næppe være synlig og behovet for 2D-projektion på skærmen, vil derfor ligeledes være begrænset.

Figur 13: Verdenskort vist i orthogonal projektion (billede fra Paint Shop Pro)

Fordelen ved denne form for (manglende) projektion, er at præcision i afstandsberegninger og andre former for opmålingsberegninger, nu vil være åbenlys. De fleste projektioner er lavet med henblik på, at favorisere en eller anden form for måling, i et givent område. Ofte stiler man efter at gøre Euclid-afstands-målinger så præcise som muligt. Denne præcision vil dog altid være tilnærmet. I projektionen der er vist på Figur 13, er denne approksimation ikke til stede. Hvis man udregner den Euclidiske afstand fra Danmark til Kina, så får man logisk nok en afstand, der gennemkrydser kloden. Hvis det man ønsker i virkeligheden er terrænafstanden, så er det let at se, at der skal nogle andre udregninger til dette. Den viste projektion indeholder heller ikke områder hvor målinger vil blive mere upræcise eller præcise end andre steder. Hvis man som bruger har et problem med, at ens kort er rundt, så skal man måske overveje, hvad det er man rent faktisk har lavet et kort over.

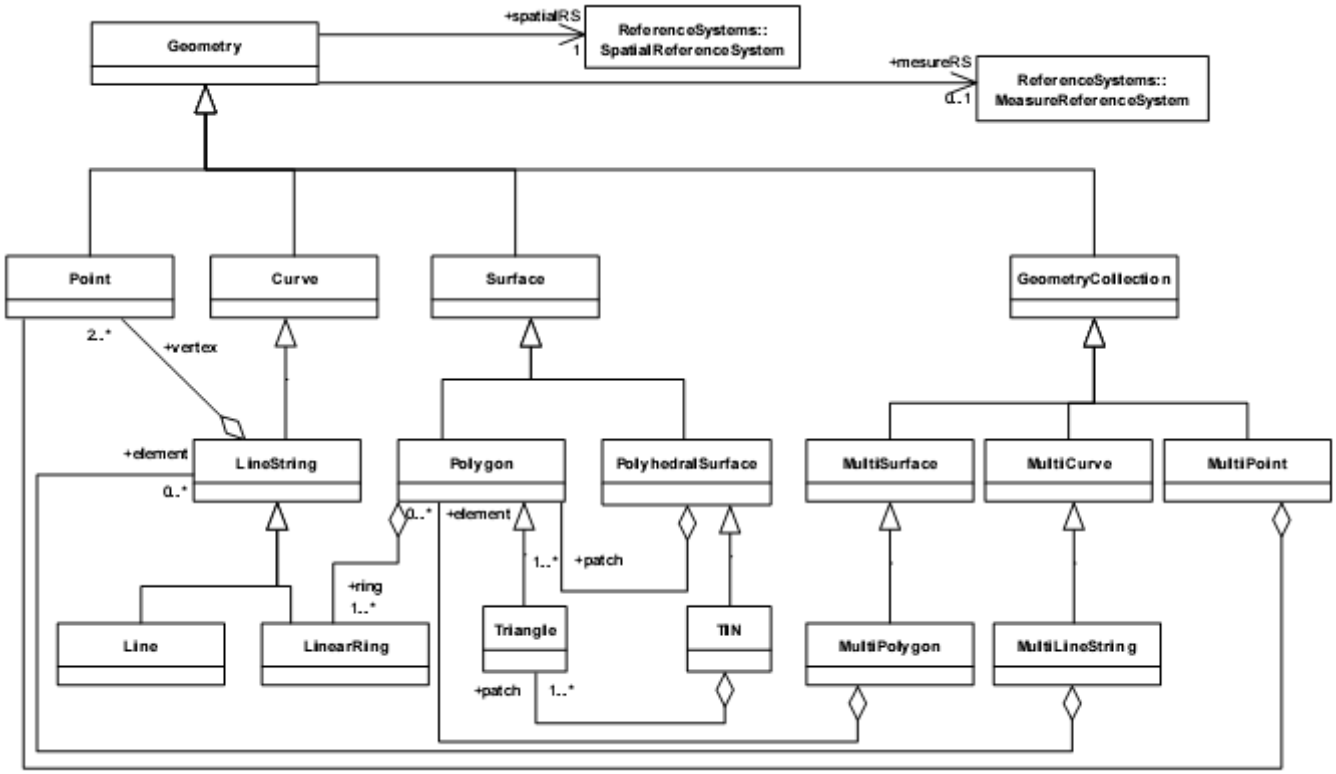
På længere sigt kan projekter som dette, være med til fjerne mange af projektions-overvejelserne fra GIS-systemer. Dette kunne ske hvis man allerede på digitaliserings-niveau, fastholder 3D-strukturen. Projektionen vil dermed kun blive relevant i udprint-situationer. Derudover vil man i disse udprint-tilfælde, endda kunne specialtilpasse den givne projektion, til at give optimal præcision for det givne område.

Når dette er sagt, så lad mig også tilføje, at jeg i dette projekt ikke har planer om at ændre på traditioner, der går flere hundrede år tilbage. Jeg vælger derfor at rendere mine datasæt, som digitaliseret. Det vil sige at jeg antager at forholdet mellem x og y er 1 alle steder og at koordinaterne er placeret på et plan (Cartesian Coordinate System, [LO & YEUNG1]). System34-kort vil eksempelvis derfor blive renderet som flade kort, med standard x og y skala.

Et projekt der dog har taget det omtalte skidt og har fjernet projektionerne er f.eks. Google Earth [GOOGLE]. Denne visualiserer faktisk kortet, som set på Figur 13.

4.4 Objekttyper

Inden GIS-systemet kan opbygges, skal det fastlægges hvilke objekttyper, der som minimum skal understøttes. MapInfo, som projektet lægger sig op af, understøtter en vifte af typer, som eksempelvis linier, punkter, polylinier, regioner, polyregioner, polypunkter, symboler, tekster, arcs mm. Denne objektmodel, vil jeg af hensyn til implementationstid og fremtidig understøttelse af ikke-MapInfo-baserede platforme, ikke bruge. I stedet vil jeg basere projektet på OpenGIS® Simple Features specifications [OGC], der allerede nu bruges af flere af databaser. Denne definerer geometri som beskrevet på Figur 14:



Figur 14: Geometri-hiraki defineret af [OGC] (billede fra [OGC])

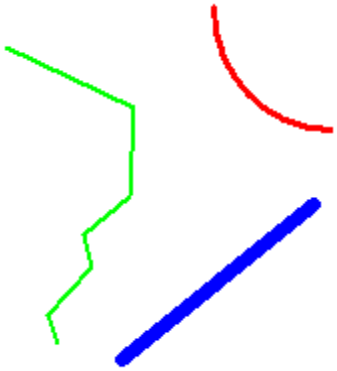
Teknisk set er kollektionerne, der defineres i diagrammet, uinteressante da disse netop er samlinger af de andre objekttyper. Det forudsættes derfor at disse understøttes, når de andre typer er tilgængelige. Jeg vil derfor i dette projekt arbejde på understøttelsen af punkter, kurver (linier) og flader.
Læg derudover mærke til diagrammets ”Surface”-klasse, der nedarver fra ”Polygon” (flad/2D) og ”PolyhedralSurface” (kantet/3D). Det er i disse at projektets hovedgeometri vil komme til at findes.

4.4.1 Punkter

Figur 15: Punkter lavet i MapInfo

Punktregistreringer er meget brugte i GIS, da man via et punkt kan tilføje arbitrært data til kortet eller til eksisterende objekter, uden at skulle tage stilling til evt. geometri, men kun geografisk position. Punkter bruges derudover også til oversigtgivende symbolik (som eksempelvis lufthavs-symboler ved lufthavne og brandstations-iconer ved brandstationer).

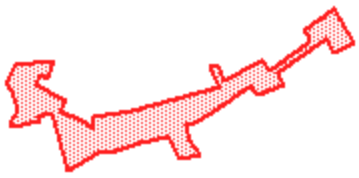
4.4.2 Linier



Figur 16: Linier lavet i MapInfo

Linier er muligvis den mindst brugte type i GIS. Dette skyldes til dels, at en linie ofte har svært ved at repræsentere et fysisk objekt, da disse nødvendigvis altid har et rumfang/areal. Linierne bliver derfor brugt til lidt mere abstrakte ting, som eksempelvis ledninger, kloakker, veje i lav LOD og vejmidter, hvor retning og position er de mest interessante informationer. (Informationer som eksempelvis skel, er som oftest repræsenteret via flader.)

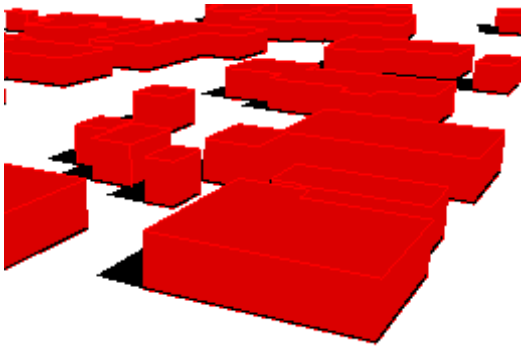
4.4.3 Flader



Figur 17: Region lavet i MapInfo

Flader er meget brugte i GIS, til alt der har et areal eller definerer et område. Det er oftest fladerne, der udgør selve kortet. (I hvert fald visuelt set.) Selv ledningskort har gavn af flade-lag, for at give synlige referencer. Informationer som eksempelvis bygninger, veje, matrikler, skove, søer, lokalplaner mm. bliver som regel repræsenteret via flader.

4.4.4 Strukturer



Figur 18: Regioner lavet i MapInfo og renderet som bygninger i GIS3D

Den sidste type som projektet vil behandle, findes ikke i almindelige GIS-applikationer. Det er den type, som der i OGC's geometri-definition omtales som *PolyhedralSurface*. Altså en geometri bestående af polygoner. Dette kommer i projektet mest til udtryk i form af eksempelvis bygninger eller *TIN*-flader.

4.5 MapInfo Professionel

Som tidligere nævnt, så afhænger projektets erhvervsmæssige succes af, at brugerne kan bruge deres eksisterende data og deres eksisterende værktøjer og for øvrigt kan forsætte med begge dele, hvis de ønsker det. Da en stor del af Geograf's kunder er MapInfo-brugere, skal projektet derfor som minimum, kunne læse MapInfo-data, uden at dette først skal konverteres. Efter læsningen (og den interne konvertering) skal data også forblive med at være MapInfo-data. Eventuelle dataændringer skal derfor også gemmes i MapInfo-formatet. Data-informationer som MapInfo ikke understøtter, kan evt. gemmes som meta-data til kortene, hvis det da ikke er muligt at genskabe disse via konvertering e.l. Selve konverteringen fra MapInfo-2D-data til 3D-GIS, skal for så vidt muligt, ske internt, uden interaktion fra brugeren og uden at det tager for lang tid.
Derudover vil det også være at foretrække, hvis brugeren kan få vist et MapInfo-kort, åbnet i MapInfo Professionel, direkte i 3D, uden at skulle tage stilling til, at kortet bliver åbnet i et eksternt program. Dette gøres ved udvikling af et plugin (MBX) til MapInfo Professionel. Funktionaliteten vil på den måde fremstå på linie med MapInfo's egen 3D-visning. (3D-visningen i MapInfo Professionel er forholdsvis ubrugelig og de færreste brugere kender derfor til den.)

4.6 3D-rendering af 2D

En umiddelbar og nem metode til at lave 2D-kort om til 3D, er blot at tilføje orbit-navigation (afsnit 4.3.3) til GIS-systemet. Dette bevirker at brugeren nu kan se kortet fra en ikke-ortogonal vinkel og vil dermed føle at kortet rent faktisk er 3D. Teknisk set vil dette også være 3D, hvilket er grunden til at almindelige GIS-applikationer ikke understøtter orbit, da forskellen på at rendere 2D kontra 3D er ganske stor. På Figur 19 og Figur 20 ses et udsnit af Nordjylland i de 2 nævnte vinkler. Sammenligner man de to billeder, vil man kunne se at 2D-billededet, er det der giver det hurtigste overblik, hvorimod 3D-billedet er det mest livagtige.

Figur 19: Nordjylland i 2D-perspektiv (billede fra GIS3D)

Figur 20: Nordjylland i 3D-perspektiv (billede fra GIS3D)

Som sidenotits kan det bemærkes, at billedet i 3D er et ganske godt bud på hvordan det rigtige 3D-billede også ville se ud. (Med ”rigtig” menes der et billede taget af et kort med terrænforskelle, bygningshøjder osv.) Dette skyldes bl.a. kameraets højde over jordoverfladen, da man i store højder forsøger at reducere detaljegraden [CITYGML], såvel i GIS som i 3D-rendering. Man vil derfor ikke kunne se 3D-elementer som eksempelvis bygninger. Derudover er det afbillede terræn også relativt fladt i virkeligheden.

4.6.1 Konvertering af 2D til 3D

For at der overhovedet skal være en ide i 3D-GIS-rendering, så skal kortene der vises, også være i 3D. Mange sagsbehandlere vil muligvis finde orbit-navigationen (vist udført på 2D-kort, på Figur 20) interessant nok i sig selv, men hvis der skal være mulighed for brug af 3D-analyser og værktøjer, så skal 3D-kortet opfylde nogle krav:

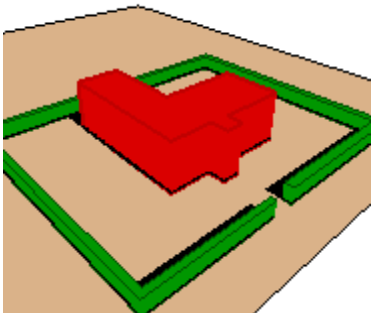
- Terrænets højdeforskelle skal være synlige, uden brug af tematisering og skal være gældende for alle kortets lag.
- Kortets rummelige emner, som eksempelvis bygninger, skove, hække, træer, lygtepæle mm, skal som minimum repræsenteres med deres gennemsnitlige højde, uden brug af tematisering.
- Emner hvis z-værdi ikke er placeret i terrænhøjde skal naturligvis også repræsenteres som sådan.

Da kravet er at bruge eksisterende 2D-kort, så skal de ovenstående punkter opfyldes, uden ekstra brug af digitalisering eller opmåling. Heldigvis så indeholder de fleste 2D-kort en del implicit 3D-information. Terræn-informationen eller z-koordinater, er ikke noget der decideret findes eller understøttes af MapInfo. Derimod findes dette dog (muligvis) i kortene, som arbitrært data tilknyttet til den egentlige grafik. Det er derfor muligt at omdanne 2D-kortene til kort der opfylder punkterne, via få udpegninger fra brugerens side:

- Hvilke kort (lag) der allerede indeholder terrænkote-data.
- Hvilke kort der indeholder rummelige emner eller emner der har en højde.

- Hvilke kort der har z-koordinater generelt. (Altså ikke nødvendigvis placeret i terrænniveau.)

Eksempler på terrænkote-kort kan eksempelvis være DAS-brønde (kloak-kort), da disse indeholder information om top- og bund-koter. Top-koterne eller dæksel-koterne må formodes at ligge på terrænniveau, eller i hvert fald ret tæt på. Af andre eksempler på potentielle kort med terræninformation, kan nævnes borings-kort, bygningskort med kælderdata, deciderede højdekurve-kort eller i det hele taget blot tilfældige GPS-genererede kort. (Professionelle GPS'er måler som regel også z-koordinater med, om end de er en smule mere upræcise end deres x og y.)
Ved at brugeren udpeger disse implicitte terrænkort, vil man få en række fixpunkter, der forhåbentligt er spredt ud over hele kort-området. Disse kan derefter bruges til, at interpolere terræninformation ud på alle kortets andre lag og på den måde skabe den ønskede DTM.
Indrømmet, denne form for DTM, er ikke den mest præcise. Især ikke hvis der eksempelvis kun er kloak-kort til rådighed. Det er f.eks. ikke så almindeligt at placere brønde, på en kommunes højeste punkt. Og der er generelt også flest brønde i byområderne. Til gengæld forsøger projektet heller ikke at skabe kort, til brug i autonome gravemaskiner, men blot at forbedre det almindelige 2D-kort. Hvis man er seriøs omkring terrænet, så har de fleste kommuner, deciderede højdekurve-kort til rådighed og det resulterende interpolerede kort vil via disse, da blive acceptabelt mht. præcision.
Af eksempler på kort, der indeholder rummelige emner eller emner der har en højde, kan naturligvis nævnes bygningskort. Højderne til disse, findes ikke altid i kommunale bygningskort, men kan dog skaffes fra datakoblinger, som eks. Ejendoms- og Miljø-registrene. Derudover er eksemplerne dog få. Hvis man kigger på digitaliserede bymodeller, vil man ofte se både biler og mennesker på kortene, der ligeledes er rummelige emner, men for GIS i administrativ sagsbehandling, er ekstra-udstyr som disse unødvendige, hvilket begrænser antallet af rummelige emner, man ønsker at se på et kort.



Figur 21: Bygning og hæk (billede fra GIS3D)

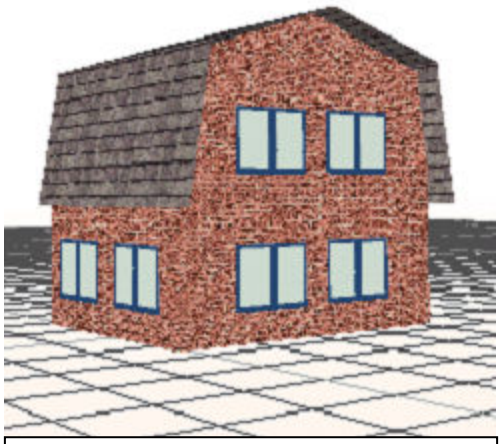
Derudover kommer kravet om "ingen yderligere digitalisering" også lidt i konflikt med dette punkt. Tag eksempelvis billedet på Figur 21. Her er vist en 3 m høj, rød bygning og en 1 m høj, grøn hæk. Dog er hække ofte blot digitaliserede som linier. Hvis man derfor vil have en fyldig hæk, som vist på billedet og ikke blot en papirs-tynd væg, så skal hækken omdigitaliseres til en region i stedet, hvilket muligvis ikke er umagen værd. Derudover så er bygningen vist med fladt tag. Dette skyldes at programmet blot har fået at vide, at den røde region er x antal meter høj. Langt de fleste 1-etagers huse har dog ikke flade tage. En korrekt gengivelse af dette, vil dog ligeledes kræve ekstra digitalisering. En bedre løsning er muligvis derfor, at lade programmet give et gæt på, hvorvidt en polygon skal have en tagryg eller ej

og hvordan denne skal placeres. Uden at have undersøgt den statistiske sammensætning af Danmarks tagrygge, så kan et bud på hvordan programmet skal basere dets evaluering være noget i retning af:

- Gavlen på et hus placeres på den korteste side
- Hvis huset har højden til et 1-etagers hus har det 90% chance for et ikke-fladt tag.
- Hvis huset har højden til et 2-etagers hus har det 80% chance for et ikke-fladt tag.
- Hvis huset har højden til mere end 2 etager har det 10% chance for et ikke-fladt tag.

Brugeren skal derefter blot tage stilling til, hvorvidt programmet skal give et tagryg-bud på et givent lag.
Af andre digitaliserings-dilemmaer, kan nævnes eksempler som træer og lygtepæle. Disse findes ofte i kort, som punktregistreringer og har dermed ikke en fylde. Selvom man gav disse deres korrekte højde, ville de i bedste fald derfor blot blive til tynde streger i luften. En mulig løsning er at lade brugeren angive, om et specifikt kort er et decideret træ- eller lygte-kort, på samme måde, som i det forrige eksempel med bygningerne. Programmet vil derefter kunne repræsentere punkt-registreringerne, via et stilistisk træ eller lygtepæl. En anden løsning der ikke kræver ekstra stillingtagen fra brugerens side, er blot at rendere punkt-registreringernes symboler som *billboards,* med den angivne højde. Om end dette sandsynligvis ikke altid bliver lige kønt, da man derved enten via *camera faced billboards,* vil få træer der altid er placeret ortogonalt på kamera-vinklen og derved vil se ud som om de ligger ned, eller via *axis faced billboards* vil få træer, der er flade og dermed ikke kan ses oppefra.
Konverteringen i projektet vil, afhængigt af hvad brugeren har tilgængeligt af data, allerhøjest resultere i en vektor-baseret DTM, med et antal kort-lag indeholdende rummelige arbitrære objekter, svarende til [CITYGML]'s LOD1-niveau. Yderligere detaljering og klassificering af data er ikke muligt med det tilgængelige data. En grov klassificering ville kunne foretages, ved at lade brugeren udpege hvilke lag, der er bygnings-, vej-, vegetations-, vejudstyr-, kloak-, lednings,- matrikel-lag, o.s.v. Dette ville trods alt kun skulle foretages én gang. Dog vil en sådan klassificering ikke umiddelbart tilføje muligheder for yderligere detaljering af modelleringen, da det kommunale kort-data, ikke er standardiseret. Man vil dog i tilfælde som med træ- og lygte-lag, kunne give et gæt på en mulig rummelig model. Der findes områder, som eksempelvis matrikel-kortene fra KMS, hvor data i teorien er ens for hele landet (Frederiksberg undtaget). Men selv i disse områder, møder man ofte kommuner, der benytter selv-redigerede udgaver, i stedet for de originale. I tilfælde hvor man kan tilknytte et kort-lag til eks. E&M-registrene, kan man få en vis standardisering. Det er dog langt fra alle kommuner, der har adgang til E&M. Alternativer som S035 og OIS benyttes også og har ikke nødvendigvis samme formater eller data. Et nyligt projekt [PLANDK2] er godt i gang med at standardisere kommunal plan-data, både med hensyn til geometri og data. Lignende projekter indenfor områder som bygninger, veje og vejudstyr ville være sand gave, for projekter som dette.

Lothar Koppers har lavet et tilsvarende projekt [KOPPERS], hvor han ligeledes beskæftiger sig med automatisk generering af 3D-objekter. Koppers fokuserer mest på modellering af huse og træer, i stedet for konvertering af kortet som en helhed. Til gengæld er han i stand til, at generere bygninger i højere præcisionsgrad (ca. svarende til LOD2 i [CITYGML]), end hvad dette projekt formår. Dette skyldes at Koppers netop kan udnytte det tyske BBR-data, som i hvert fald med hensyn til bygninger, er mere detaljeret, end det tilsvarende danske.



Figur 22: Koppers kan via tysk BBR-data, modellere bygninger som denne (billede fra [KOPPERS])

Hvis man skal sammenligne med andre projekter, så skal man først klarlægge, hvad det er som dette projektet beskæftiger sig med. Projektet beskæftiger sig med konvertering af 2D-data, til generering af terræn- og bymodel-data. De fleste andre projekter, som eks. [KOPPERS] beskæftiger sig primært kun med en af delene, hvor dette projekt interesserer sig for kortet som helhed. [LO & YEUNG2] lister bl.a. en rækker programmer, der ligesom [NORTHWOOD] beskæftiger sig med DTM'er. Resultatet fra dette projekt, indeholder dog kun en DTM, i kraft af at brugeren har nok terræn-regioner, til at skabe DTM'en. Projektet interesserer sig i virkeligheden mere for, at en brugers given region (eks. kommunalplan) bliver præsenteret som *TIN*, med indeholdende højdeinformation. Terræn der ikke er omfattet den givne region, er for dette projekts vedkommende, uinteressant, hvis dette da ikke er indbefattet andre af de konverterede regioner. Dette betyder at hvis brugeren har et tomrum i kortene, så får den resulterende DTM det også. Ligeledes så giver [LO & YEUNG3], nogle eksempler på *fly-through* scener, af by-modeller, hvis oprindelse stammer fra special-digitalisering og hvis efterfølgende funktion, som regel begrænser sig til de omtalte *fly-throughs*. Som omtalt tidligere i rapporten, så har disse derfor heller ingen decideret interesse.

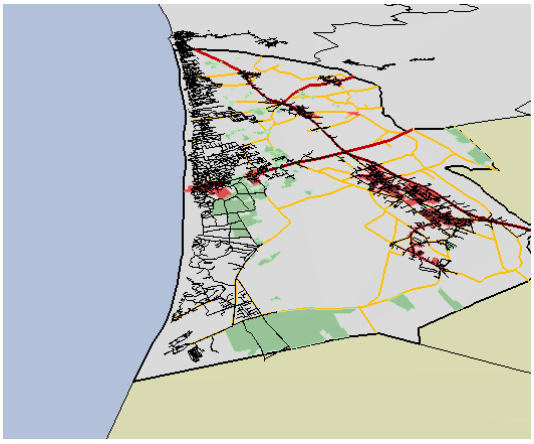
4.6.2 Skalering af Z-akse

Efter tidlige tests og konsultation af Geograf Hasse Hauch[1], har jeg konstateret at Danmark er relativt fladt geografisk set. Et vigtigt værktøj i forbindelse med 3D-GIS, er derfor muligheden for at kunne opskalere Z-aksen (forhøje), således at man kan fremhæve de eventuelle terrænforskelle, der måtte være.

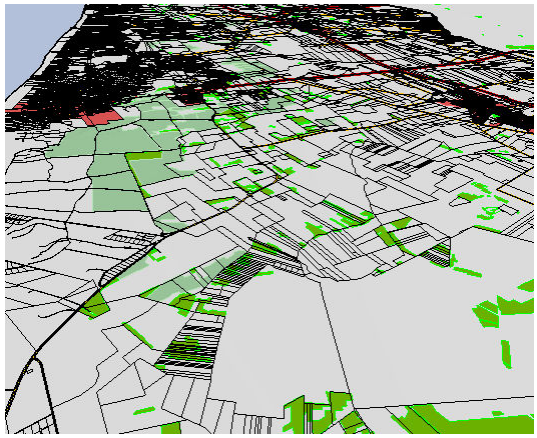
[1] Geograf ansat ved Geograf A/S

4.7 LOD

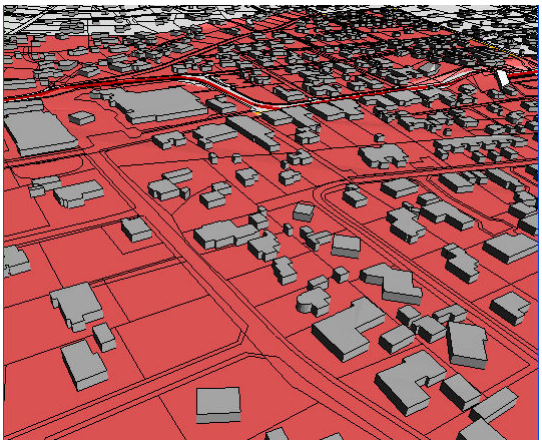
Level of Detail bliver brugt i GIS, til simplifikation af kortet, således at overblik altid bevares. I MapInfo-sammenhæng bliver LOD kun brugt til at definere, i hvilken højde et givent lag skal være synligt i. Eksempler på dette kan ses på Figur 23 - Figur 25. Basalt set fungerer det på den måde, at man nær ved jordoverfladen kan se alle kort-lag. Eller hvis der er flere versioner af et lag, så vil de mest detaljerede være synlige i lavt zoom-niveau. Jo højere man sætter zoom-niveauet, jo flere lag bliver fjernet fra kortet eller udskiftet med lav-resolutions-versioner.



Figur 23: I stor højde vises kun generelle elementer (billede fra GIS3D)

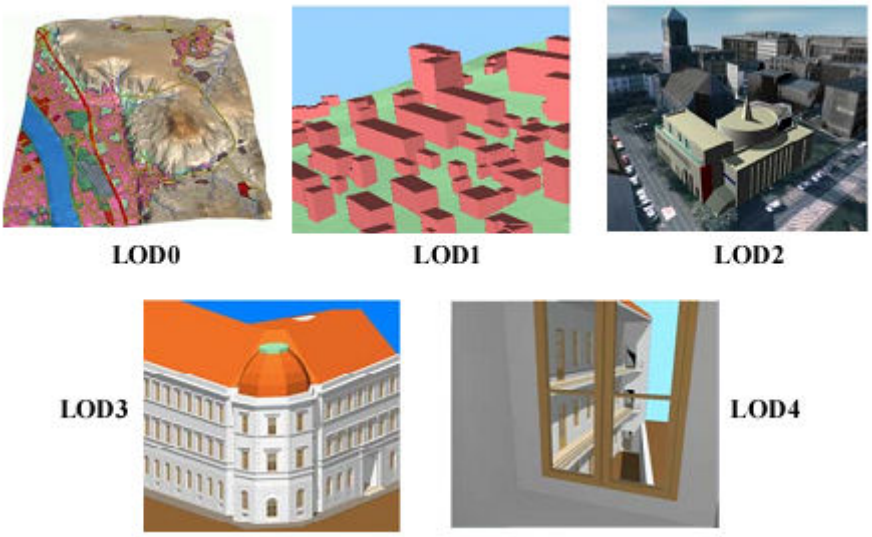


Figur 24: I mellemhøjde bliver flere elementer som eks. matrikler vist (billede fra GIS3D)



Figur 25: I lav højde bliver alle elementer vist (billede fra GIS3D)

[CITYGML] går et skridt videre og bruger LOD til at give flere grader af præcision til et givent objekt, således at hvert objekt indeholder flere versioner. [CITYGML] går også så langt, så de definerer graderne af præcision:



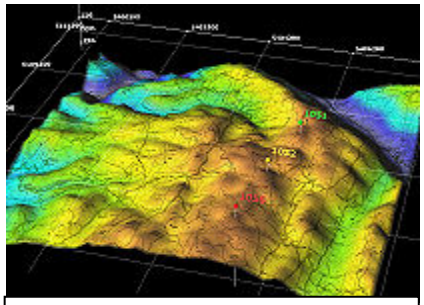
Figur 26: 5 grader af præcision illustreret i [CITYGML]

- LOD0 definerer den mindst informationsfyldte version, hvor kun DTM'en er synlig.
- LOD1 tilføjer elementer som bygninger, vist som blokke.
- LOD2 tilføjer ting som hustage og vegetation
- LOD3 forøger detaljegraden af LOD2, med eksempelvis bygningsudstyr som fotograferede textures, altaner, detaljerede vægge, detaljeret vegetation etc.
- LOD4 afslutter detaljegraden ved også at tilføje indmad og indvendig information til kortets objekter.

I MapInfo-sammenhæng har brugerne dog selv ansvaret for, at defineret LOD-niveauerne. Objekterne har dermed ikke nogen viden om andre detaljegrads-versioner.
Da projektet bygger på MapInfo-data, så vil [CITYGML]'s model kræve yderligere digitalisering fra brugerens side og jeg vælger derfor kun at forsætte tankegangen fra MapInfo, i første omgang.

4.8 Tematisering

På Figur 27 ses et eksempelvis på et tematiseret 3D-kort. I dette tilfælde er der tematiseret efter terrænhøjde, hvilket gør det let at udpege høje og lave punkter. Man kunne også have valgt at tematisere efter eks. undergrundsprøver, trafiktæthed mm., afhængigt af hvad man nu var interesseret i.



Figur 27: Tematiseret 3D-kort fra programmet Geo Surface3D PRO [SCIENCEGL]

Evnen til at analysere og tematisere kort, er et meget vigtigt emne indenfor GIS. Denne egenskab kan afgøre hvorvidt et kort er brugtbart til det givne formål eller ej. Desværre betyder dette også, at emnet er så omfangsrigt, at jeg af hensyn til tid, må udsætte det til senere udviklingstrin. Derudover findes der også

massere af eksisterende aplikationer, der kan bruges til at tematisere og analysere kort. Eksempelvis programmer som MapInfo og Arc, som projektet er ment til at være en hjælp til.

4.9 Plugins

For at gøre programmet interessant for erhvervslivet, er det en god ide at lave programmet så justerbart og fleksibelt som muligt. Eksempelvis er det en stor fordel for virksomheder (og for deres salg) hvis de kan tilpasse programmet mht. brugerinterface og funktionalitet, så de på den måde kan specialtilpasse programmet til en given kunde. Med funktionaliteten fra afsnit 4.9.1 i mente, vil en oplagt løsning derfor være at indbygge et plugin-system i programmet, således at man via DLL'er kan ændre og tilføje funktionalitet efter behov.
Et plugin-system vil også være til stor hjælp for en virksomheds udviklings-organisation, da denne dermed lettere kan opsplittes i autonome grupper, der via et fastlagt interface (plugin-systemet) kan udvikle og teste kode uafhængigt af hinanden.
Et plugin-system vil også give mulighed for sideløbende udvikling af GIS-systemernes mest normal værktøjer. Man kunne forestille sig, at der over tid kunne blive udviklet en decideret værktøjskasse, på samme måde som den man ser i Arc-produkter.

4.9.1 Adresse-opslag

En af de mere basale funktionaliteter i ethvert brugervenlig GIS-program, er adresse-opslaget. Altså evnen til at zoome til område på kortet, angivet af kombination af eksempelvis vejnavn, husnummer, postnummer, ejendomsnummer, matrikelnummer eller andre implicitte geo-nøgler, der hyppigt bruges og dermed er tilgængelige, i det anvendte arbejdsmiljø.
Den specifikke funktionalitet og udformning af disse opslag, afhænger dermed også ofte af den givne situation/arbejdsmiljø og det vil dermed være uhensigtsmæssigt, at forsøge at indfange alle tænkelige situationer, i en enkelt funktion til programmet. En mere hensigtsmæssig og fleksible tilgang vil være at lave disse opslagsfunktioner, som tilføjelses-moduler eller som ekstra-programmer til programmet. Det vil dermed være muligt for brugeren at tilføje den netop ønskede funktionalitet.

4.9.2 GIS-overblik

Et af MapInfo's svage punkter i forhold til eksempelvis Arc, er at det ikke har nogen indbyggede værktøjer, til at give overblik eller administrere kort generelt. Værktøjer til disse formål, bliver derfor i stedet lavet af virksomheder som Geograf A/S. Eksempelvis som plugins (MBX) til MapInfo.
For at give projektet helhed, skal der derfor præsenteres et bud på et værktøj til overblik og administration af kort.

4.10 Performance

Performancemæssigt er det vigtigt, som minimum at kunne følge med brugerens andre programmer. Hvis projektets resulterende applikation, er mærkbart langsommere end f.eks. MapInfo, så vil brugeren opleve den ekstra latency som pinefuld. Da projektet omhandler 3D-grafikbehandling, som i sig selv er beregningsmæssigt mere krævende end almindelig 2D, så vil den resulterende performance, derfor være et udtryk for, hvorvidt 3D-GIS i det hele taget er realistisk i almindelig sagsbehandling.
Performance-overvejelserne er ekstra vigtige at foretage, i forbindelse med den interne konvertering af 2D-kortene til 3D og i forbindelse med opbygningen af de nødvendige 3D-strukturer. Ud over at disse processer traditionelt set er beregningsmæssigt tunge, så er det også processer, hvortil der ikke eksisterer noget tilsvarende i 2D-programmerne. Faktisk så eksisterer der sandsynligvis heller ikke noget tilsvarende, i de deciderede 3D-programmer, da disse netop har fordelen af pre-konverteret data. Konvertering- og opbygnings-processerne er derfor rent handicap i forhold til begge miljøer.

4.11 Platform

For at vælge projektets udviklings- og afviklings-platform har jeg valgt at fokusere på implementations-hastighed og -organisation, samt generelle krav fra Geograf A/S's normale kundegruppe. Kundegruppen og projektets målgruppe er generelt Microsoft Windows-brugere. Geograf A/S's kundegruppe indeholder bl.a. ca. 25% af de danske kommuner, hvis styresystemer varierer imellem Microsoft Windows 98, 2000, 2003 og XP, med forskellige kombinationer af Citrix, Novell, VMWare mm. Det betyder at projektets programmel, som minimum skal kunne afvikles under Windows. Og i praksis behøves det heller ikke at kunne afvikles på andet. (Definitionen på hvorvidt et program kan køre under Windows, kræver ifølge min personlige holdning også, at programmet overholder standarder som *common dialogs* og *manifest*. Det er altså ikke nok at programmet kan køre i en prompt eller laver GUI ala *GLUT*.) Dog vil en smule forudseenhed ikke være af vejen, selvom alle kunderne er Windows-brugere. Eksempelvis så har Frankrigs regering netop besluttet, at udskifte alle platforme til licens-frie Unix-baserede-installationer (Linux) [HEISE]. Et lignende tiltag er ikke utænkeligt i Danmark.
Jeg har valgt at bygge projektet på Microsoft Visual Studio, OpenGL og C# (.NET). Dette opfylder mit fokus på implementations-hastighed, da Visual Studio er et *RAD*-værktøj og da OpenGL generelt er mere simpelt end DirectX og det opfylder brugernes krav om, at det skal kunne afvikles under Windows. Derudover sikrer valget af C# og OpenGL også, at det vil kunne afvikles på andre platforme end Windows. Dette klares via opensource-projektet Mono, der er en platformsuafhængig implementation af .NET. (Dette er dog pt. ret teoretisk, da Mono stadigvæk halter lidt mht. stabilitet.)
Af andre platforme, der ligeledes kunne have opfyldt kriterierne, kan nævnes Delphi/OpenGL og Eclips/Java. Delphi vil dog aldrig kunne blive platformsuafhængig og Eclips/Java lider af mindre handicaps i forhold til

.NET, mht. performance, framework-konsistens og implementationshastighed.

5 Kravspecifikation

Baseret ud fra analysen i afsnit 4 fås følgende krav til projektet:

- Projektet skal resultere i en GIS-applikation kaldet GIS3D.
- Programmet skal opfylde følgende:
    - Skal kunne afvikles selvstændigt eller som plugin til MapInfo Professionel
    - Skal være et MDI-windows-program med mulighed for flere åbne kort af gangen.
    - Vise og navigere i *native* MapInfo-kort
    - Kunne skifte frem og tilbage imellem MapInfo og GIS3D med samme kortudsnit.
    - Følgende navigation skal være til rådighed:
        - Pan
        - Zoom
        - Oribit
        - Reset
    - Vise følgende MapInfo-elementer:
        - Regioner / Polyregioner
        - Linier / Polylinier
        - Punkter
    - Vise kort i 3D
        - Kortet skal følge terrænet
        - Rummelige elementer skal vises som rummelige
        - Vise shadet terræn.
    - Skalere z-akse
    - Udpege elementer i kortet
    - Vise tilknyttet data til elementer
    - Vise flere lag i kortet og skal kunne gemme/vise (visibility) og ændre på rækkefølgen af disse
    - Vise lag med højdebestemt synlighed (LOD).
    - Konvertere MapInfo-2D-kort til 3D via følgende brugerinteraktion:
        - Udpegning af lag med terrænkoter
        - Udpegning af lag med højder
    - Indeholde plugin-system
        - Adresse-opslag
        - GIS-oversigt
    - Performance skal matche MapInfo's

6 Design

Følgende afsnit dedikeres til specifik teori, der bruges i projektet.
Afsnit 6.1 vil beskrive projektets struktur og give et overblik over dette og de efterfølgende afsnit vil udspecificere udvalgte emner.

6.1 Programstruktur

Følgende afsnit beskriver programmets planlagte opbygning via UML-diagrammer og screenshots.

6.1.1 GUI

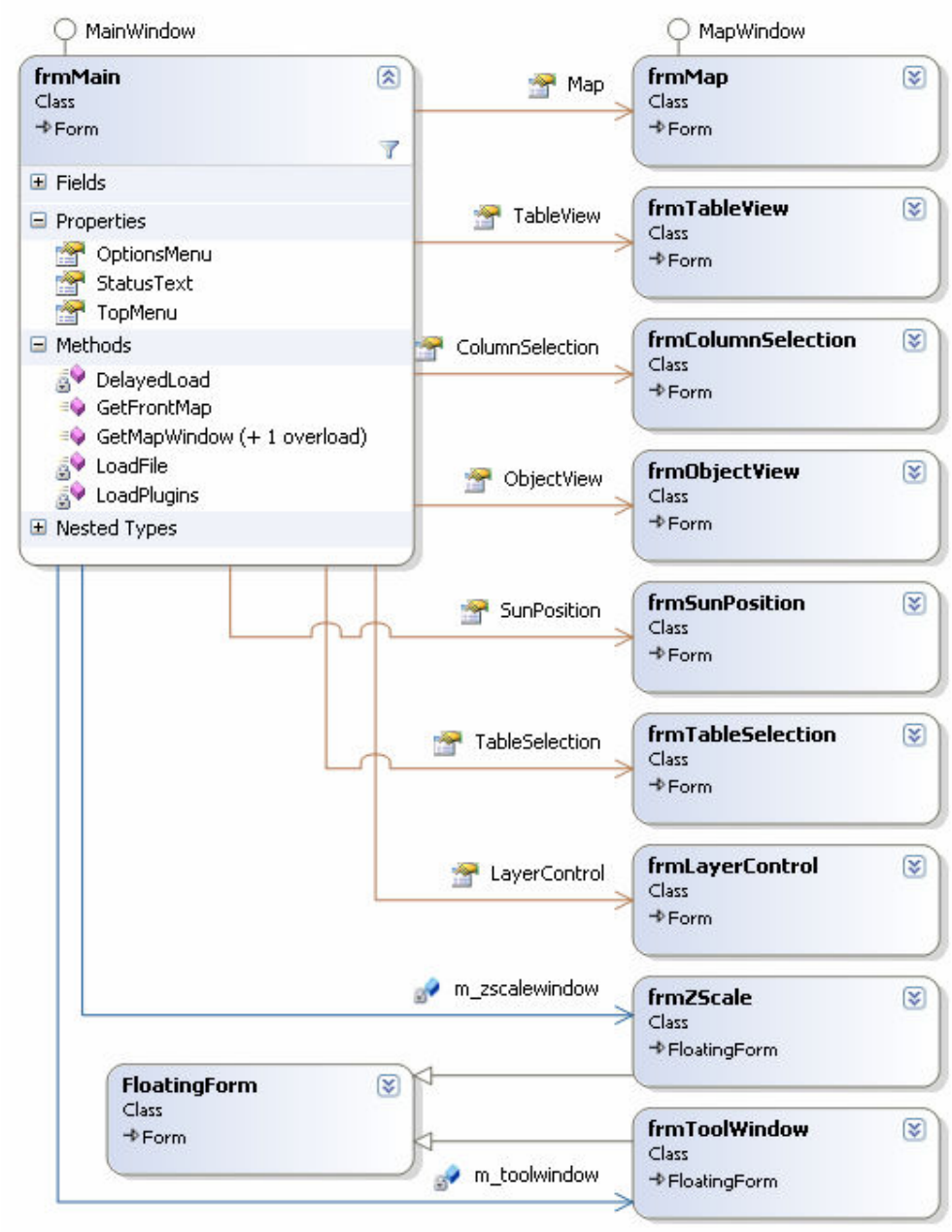
Programmet er ment til at fungere i to tilstande. En hvor programmet fungerer som standalone MDI-program og en hvor det fungerer som udvidelse af MapInfo. I situationen hvor programmet skal fungere som udvidelse, vil GUI'en kun bestå af et enkelt kort-vindue, så det på den måde vil virke som en naturlig del af MDI-programmet MapInfo. Når det derimod fungerer som standalone, skal programmet selv sørge for at forsyne MDI-interfacet.
På Figur 28 er vist et diagram over basis-delen af MDI-opbygningen. Her ses:

- frmMain som værende hovedvinduet
- frmZScale og frmToolWindow som værktøjskasser
- frmLayerControl, frmTableSelection, frmSunPosition, frmObjectView, frmColumnSelection, frmTableView og frmMap som underdialoger.

På Figur 29 ses programmets plugin-interface. Det bedømmes at fremtidige plugins vil få brug for følgende funktionalitet:

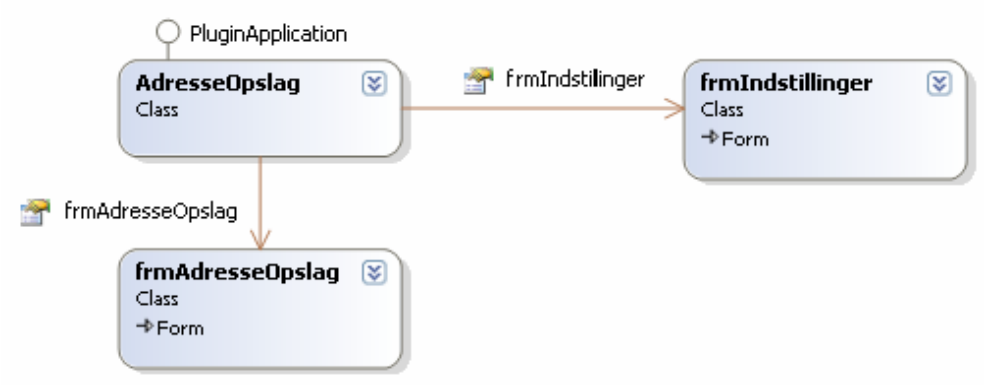
- Adgang til hovedvinduet for mulighed for tilføjelse af menuer, interface-tilpasninger og statusbeskeder.
- Adgang til hovedvinduets kort-vinduer for tilføjelser af funktionalitet til dette.
- Adgang til kort-vinduernes kort-lag og disse objekter og kolonner, for adgang til kortets egentlige struktur.

Derudover skal hovedvinduet også have et interface til plugins'ene, således at disse kan opstartes og afsluttes korrekt.
På Figur 30 og Figur 31 ses de to planlagte programmer, der udnytter det ovenstående plugin-interface. Disse er opbygget via en hovedklasse der fungerer som selve plugin'et og henholdsvis en plugin-hoveddialog og en opsætningsdialog.
På Figur 32 vises programmets opbygning, i situationen som udvidelse til MapInfo. Her ses MapInfo og dettes plugin, der indeholder programmets kort-vindue.

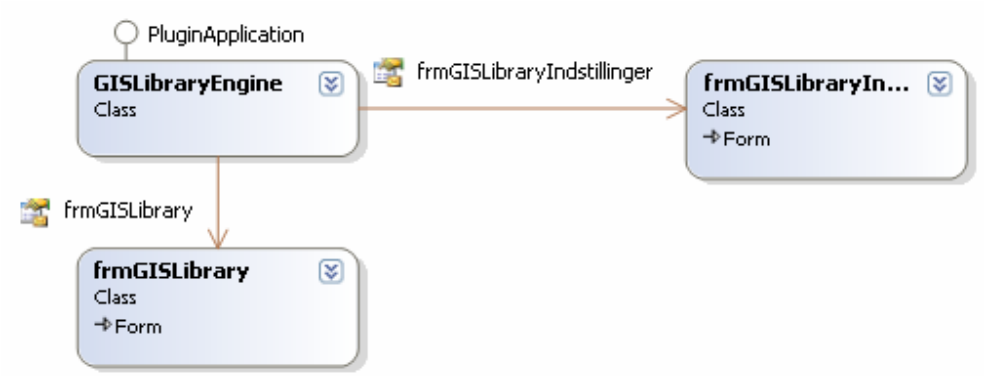


Figur 28: MDI-opbygningen

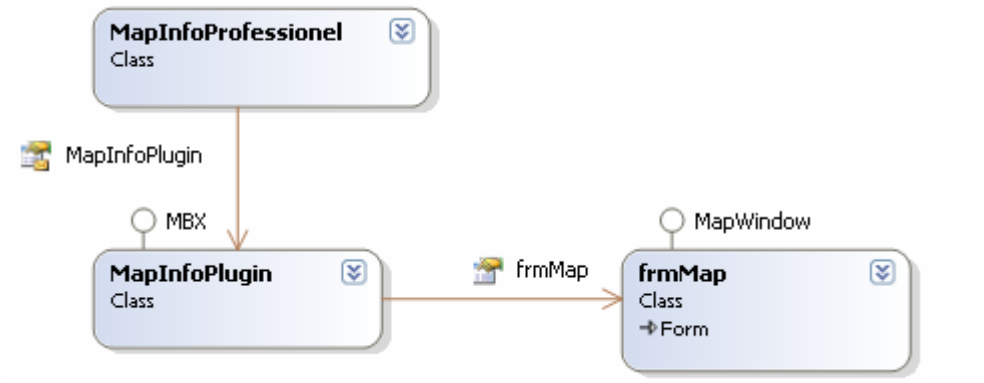
Figur 29: Projektets plugin-interface



Figur 30: Adresse-opslags-plugin'et



Figur 31: GISLibrary-plugin'et

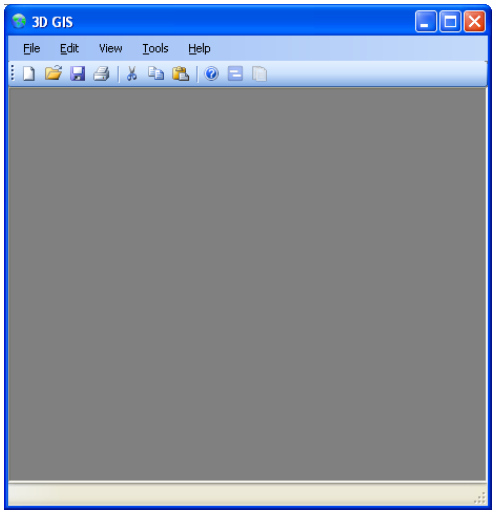


Figur 32: Programmet vist som udvidelse til MapInfo

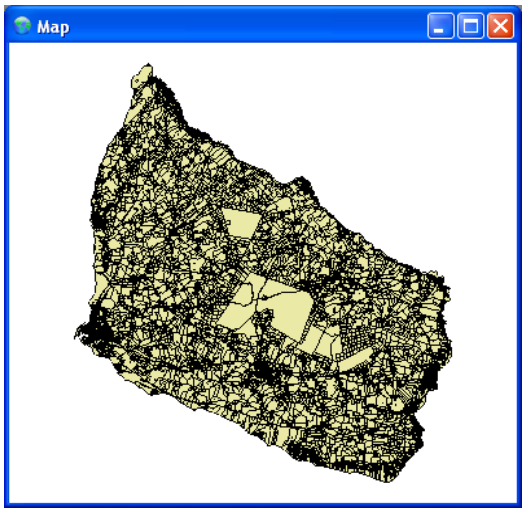
Følgende beskriver ovenstående GUI via screenshots:

- Figur 33, frmMain. Klassisk MDI-vindue med de mest almindelige menuer.
- Figur 34, frmMap. Kort-vinduet. Indeholder ingen yderligere kontroller.
- Figur 35, frmZScale. Toolbar med slider der kan variere z-skalerings-værdien fra 0,0 til 10,0.

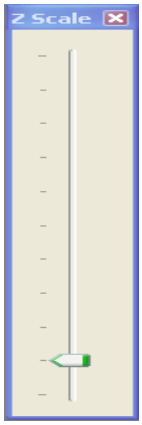
- Figur 36, frmToolWindow. Toolbar med navigations-/kort-funktionerne: Selection, Orbit, ZoomIn, ZoomOut, Pan, Info og Reset.
- Figur 37, frmSunPosition. 3 slidere til bestemmelse af breddegrad, dag på året og tidspunkt på dagen. Derudover en præsentation af resultatet fra den valgte beregning.
- Figur 38, frmTableView. Oversigtsliste over tabeller.
- Figur 39, frmColumnSelection. Dialog med combobox til udvælgelse af tabel/kolonne.
- Figur 40, frmAdresseOpslagIndstillinger. Dialog med udvælgelse af tabel, vejnavns-kolonne, vejkode-kolonne og husnr-kolonne.
- Figur 41, frmAdresseOpslag. Dialog med udvælgelse af vejnavn og husnr.
- Figur 42, frmGISLibrary. Liste over fundene kort. Kort-udsnit og præsentation af meta-data.
- Figur 43, frmGISLibraryIndstillinger. Liste med tilføjede fil-placeringer, hvori GIS-oversigten skal lede.
- Figur 44, frmObjectView. Dialog med propertybag over et givent kort-objekt.
- Figur 45, frmLayerControl. Oversigt over et givent kort og dets lag, rækkefølge af disse og synlighed.



Figur 33: frmMain

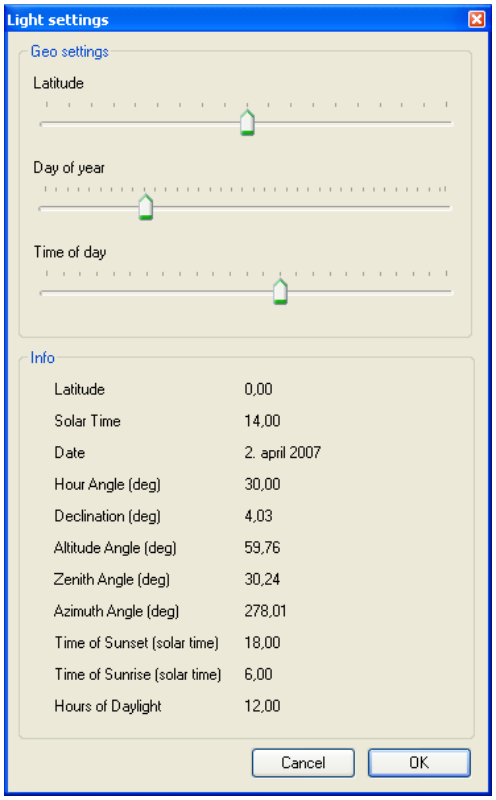


Figur 34: frmMap

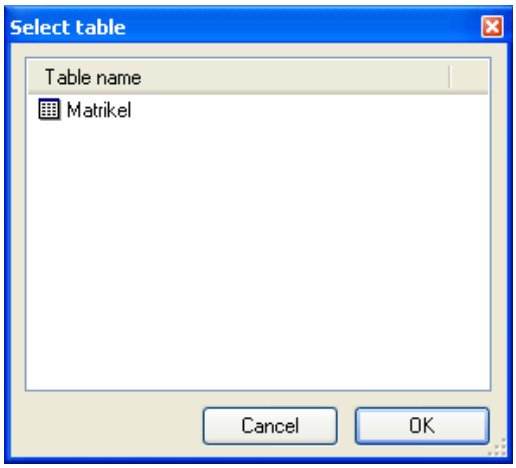


Figur 35: frmZScale

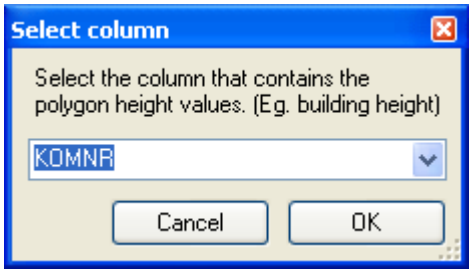
Figur 36: frmToolWindow



Figur 37: frmSunPosition

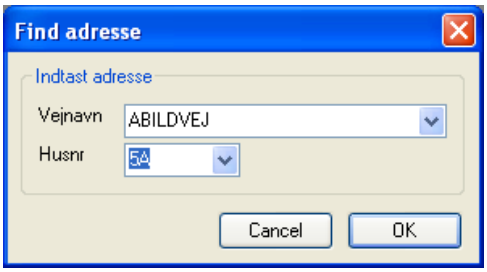


Figur 38: frmTableView

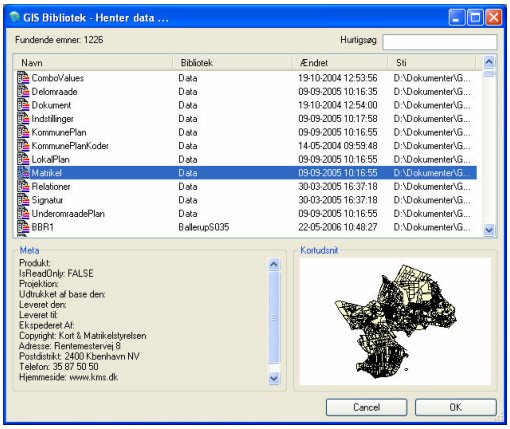


Figur 39: frmColumnSelection

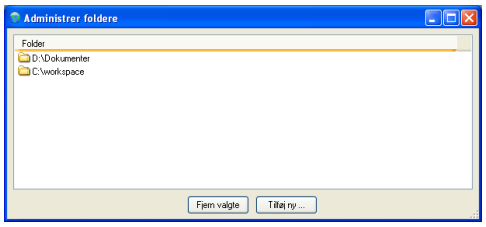
Figur 40: frmAdresseOpslagIndstillinger



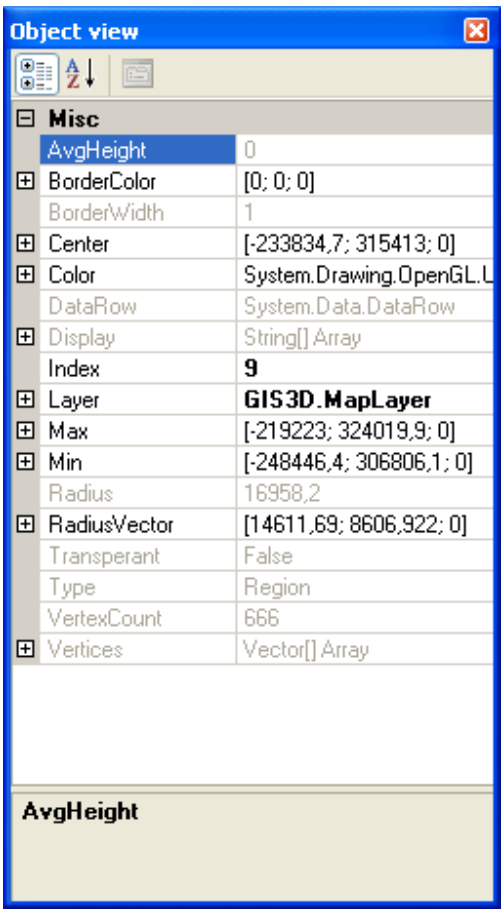
Figur 41: frmAdresseOpslag



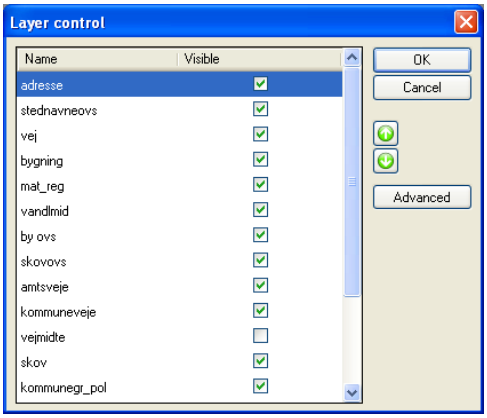
Figur 42: frmGISLibrary



Figur 43: frmGISLibraryIndstillinger



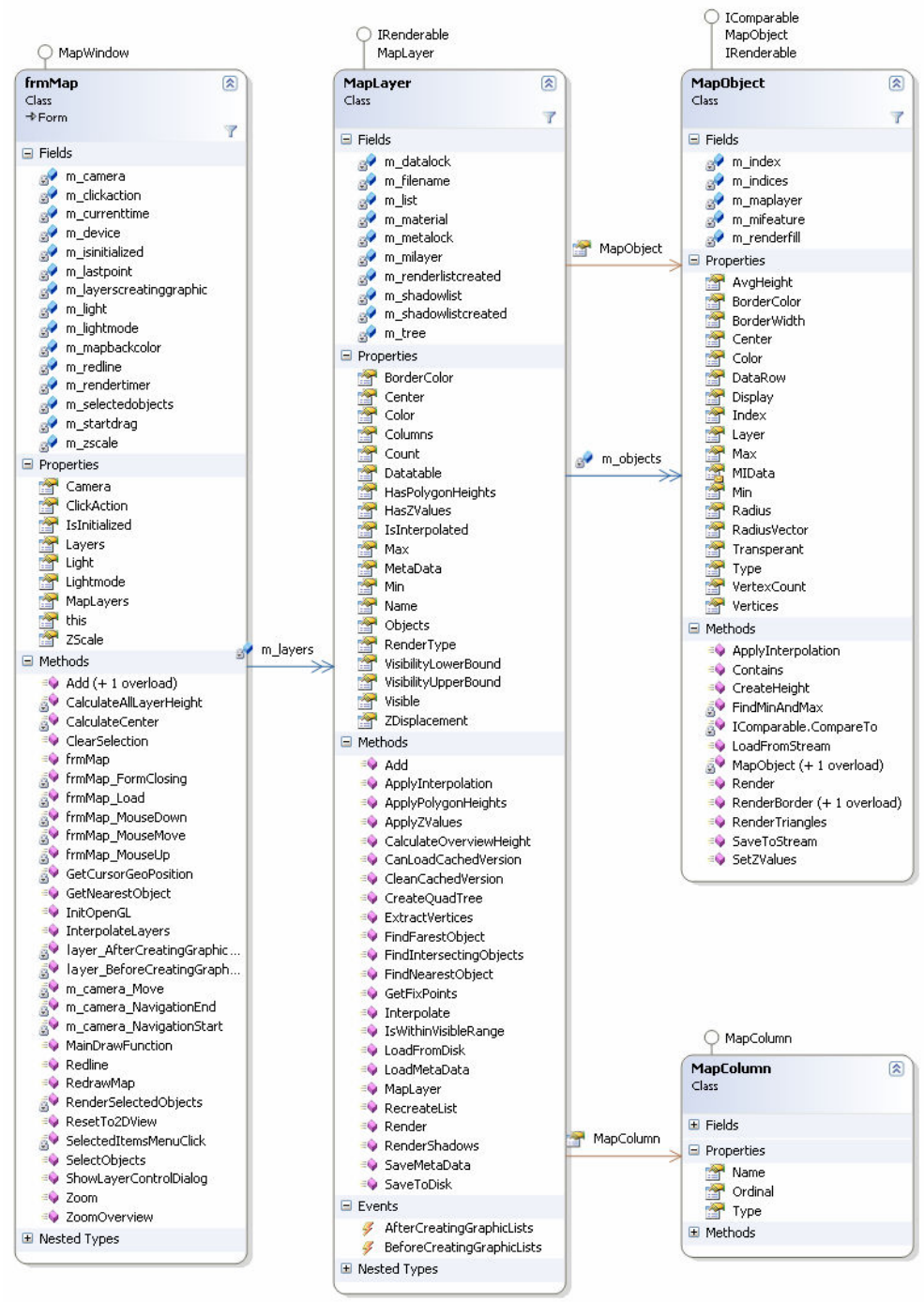
Figur 44: frmObjectView



Figur 45: frmLayerControl

6.1.2 Kort

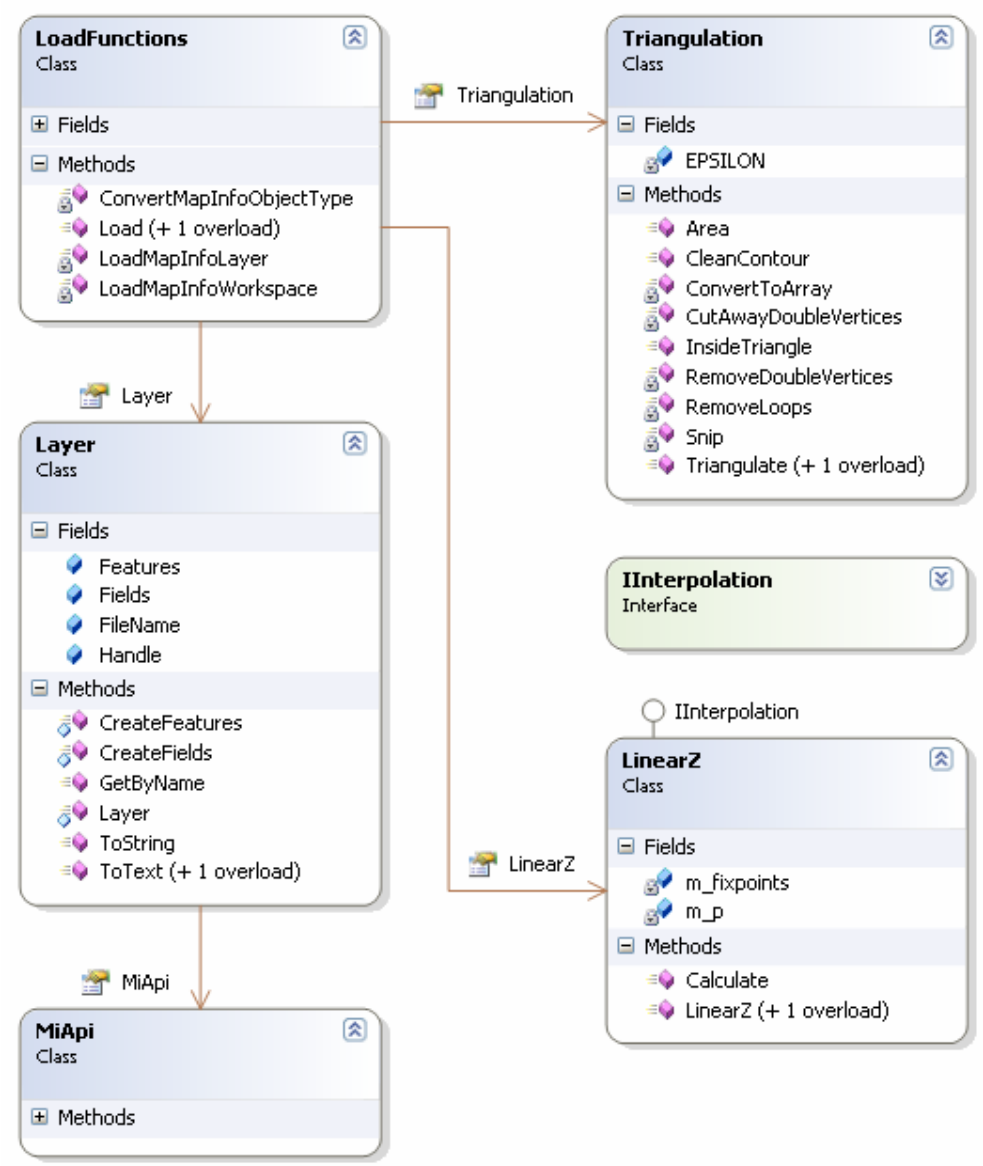
Projektets kortdata-struktur opbygges efter samme model som MapInfo. Kort-vinduet (frmMap) bruges til at rendere og indeholde hele kortet. Kort-vinduet indeholder derfor 0-∞ antal lag (MapLayer) eller tabeller, som de også figurerer som. Hvert lag har 0-∞ antal kortobjekter og hvert lags kortobjekter har 0-∞ antal fælles datakolonner.



Figur 46: Kortdata-struktur

6.1.3 Indload

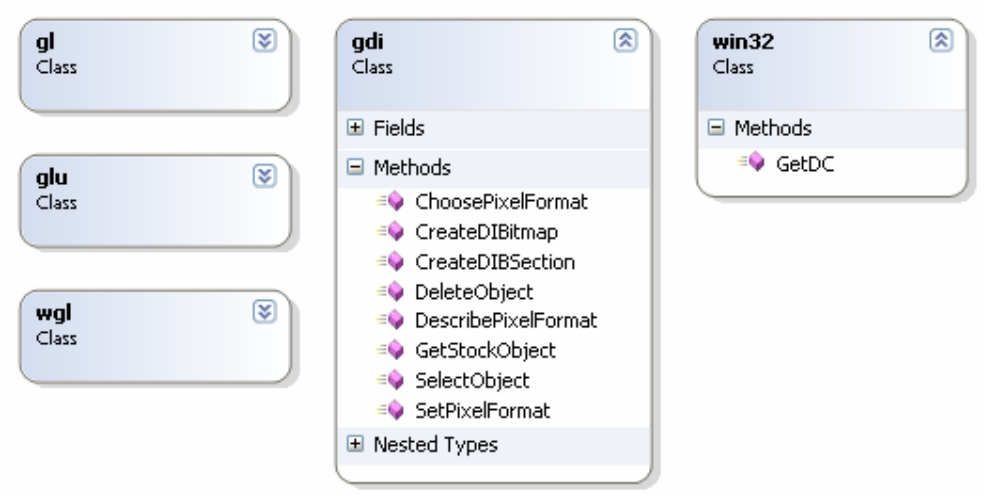
Indload og konvertering af kort, foregår i opstartsfasen af programmet. LoadFunctions.Load vil blive kaldt uanset filtype. Denne vil derefter identificere filen og selv udvælge den rette indloadsprocedure. Herefter vil data blive rengjort for geometriske fejl, trianguleret, tilføjet z-information og til sidst interpoleret. Første version af programmet vil bruge en simpel lineær interpolation-algoritme. Der laves derfor et ”interpolations”-interface, således at denne nemt kan udskiftes senere hen.



Figur 47: Indload og konvertering af kort.

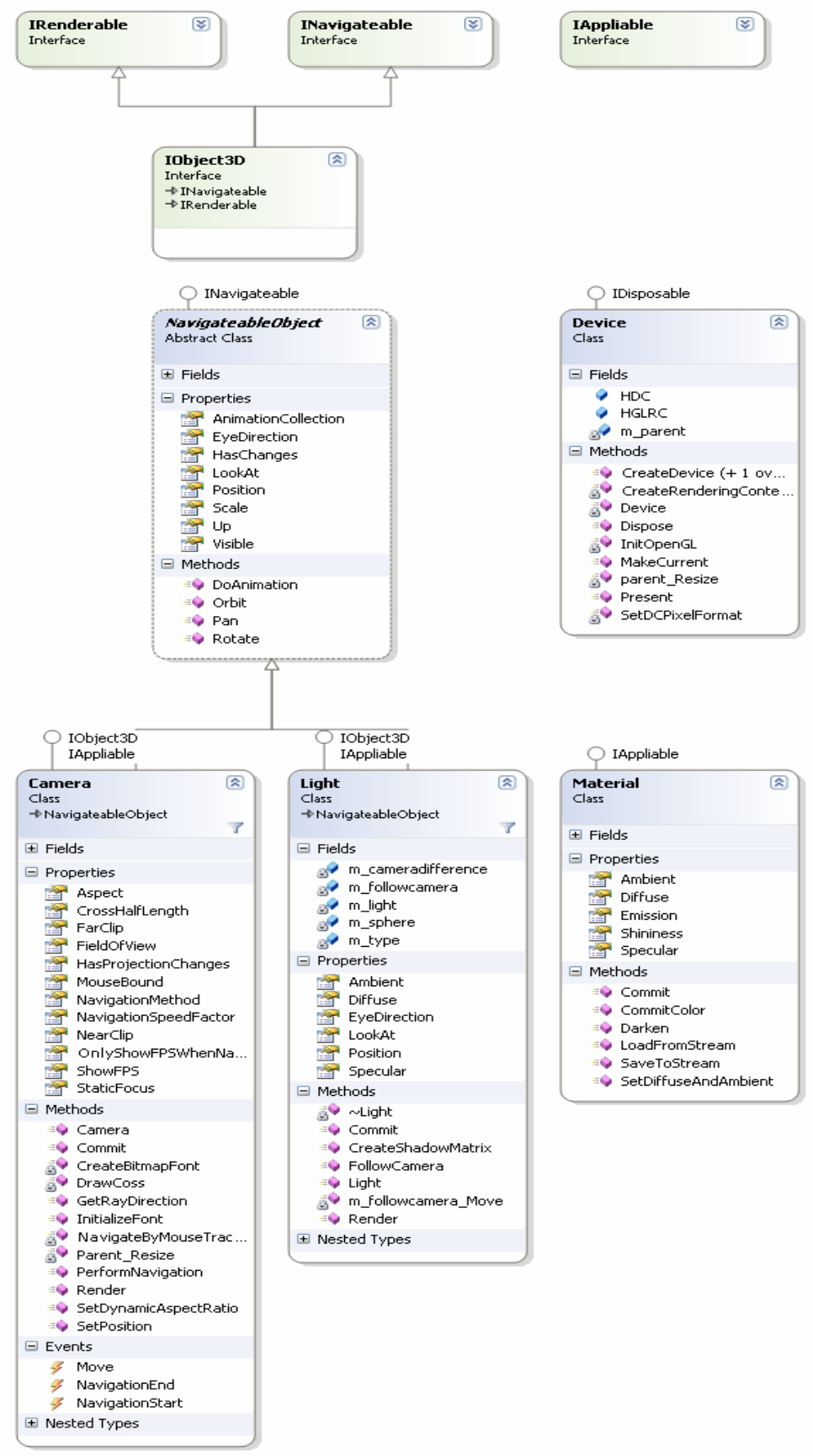
6.1.4 3D-rendering

Projektets OpenGL-wrapper er ment til at ligne det oprindelige ANSI-C-interface. OpenGL's prefixes er lavet til .NET namespaces, profixes er lavet til overloads og konstanter er lavet til enums. Eksempelvis så er funktionen "glVertex3f" lavet om til gl.Vertex. Klasserne gl, glu og wgl på Figur 48 er klassiske OpenGL-biblioteker. Klasserne gdi og win32 bruges til initialisering og opstart af OpenGL i Windows.



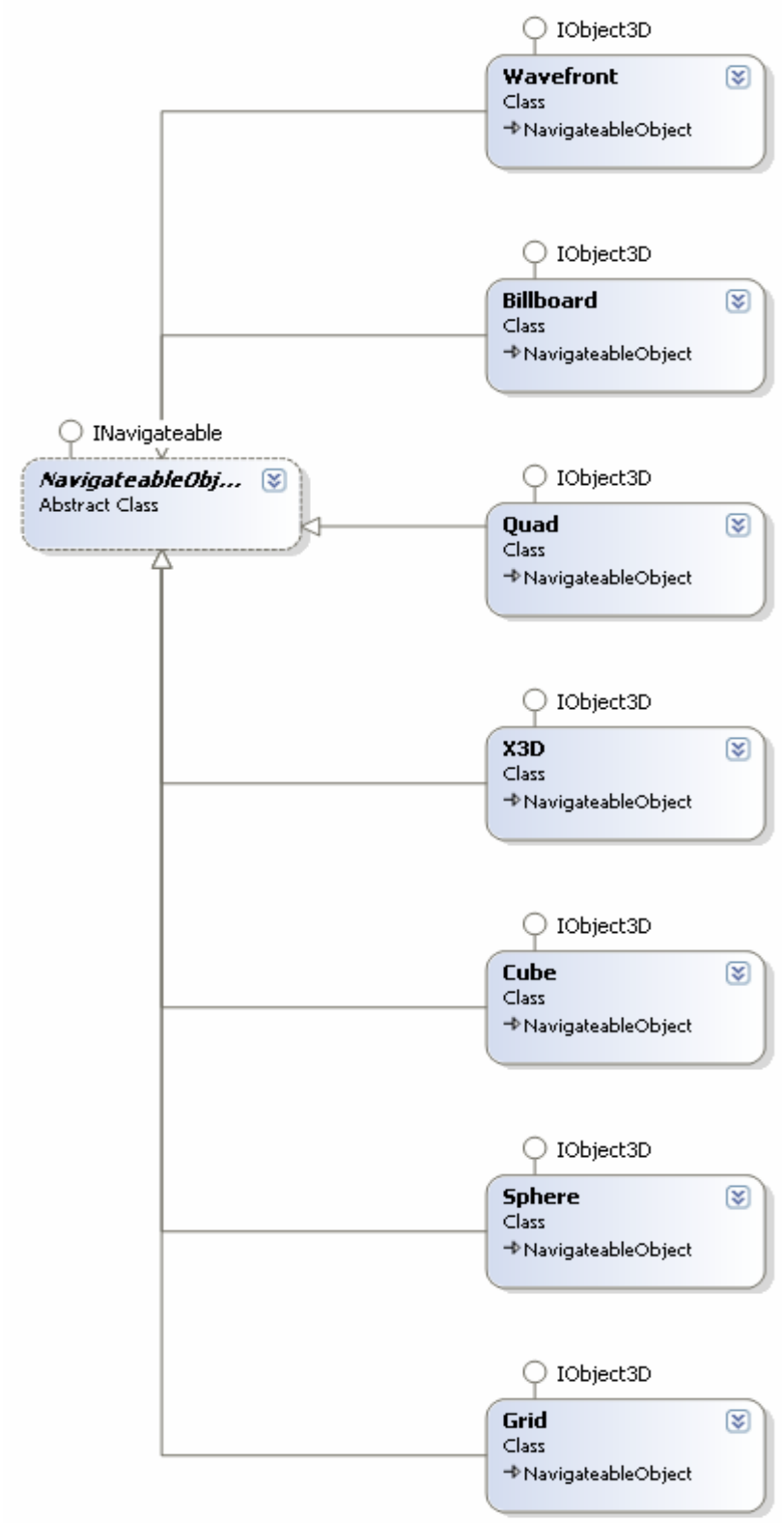
Figur 48: OpenGL-wrapper

Figur 49 beskriver det egentlige 3D-framework. Klassen ”Device” svarer til DirectX’s klasse af samme navn. Det er den der står for initialiseringen og for den endelige præsentation på skærmen. Hvis man ville lave en komplet skyggende OpenGL-Wrapper som eks. Tao [TAO], skulle alt kommunikation til OpenGL ligeledes ske via denne klasse. Klasserne ”Camera”, ”Light” og ”Material” (kamera, lys og materialefarve) tilføjer de mest basale værktøjer ved 3D-arbejde og den abstrakte klasse ”NavigateableObject” tilføjer basal navigation og positionering i 3D-rum.



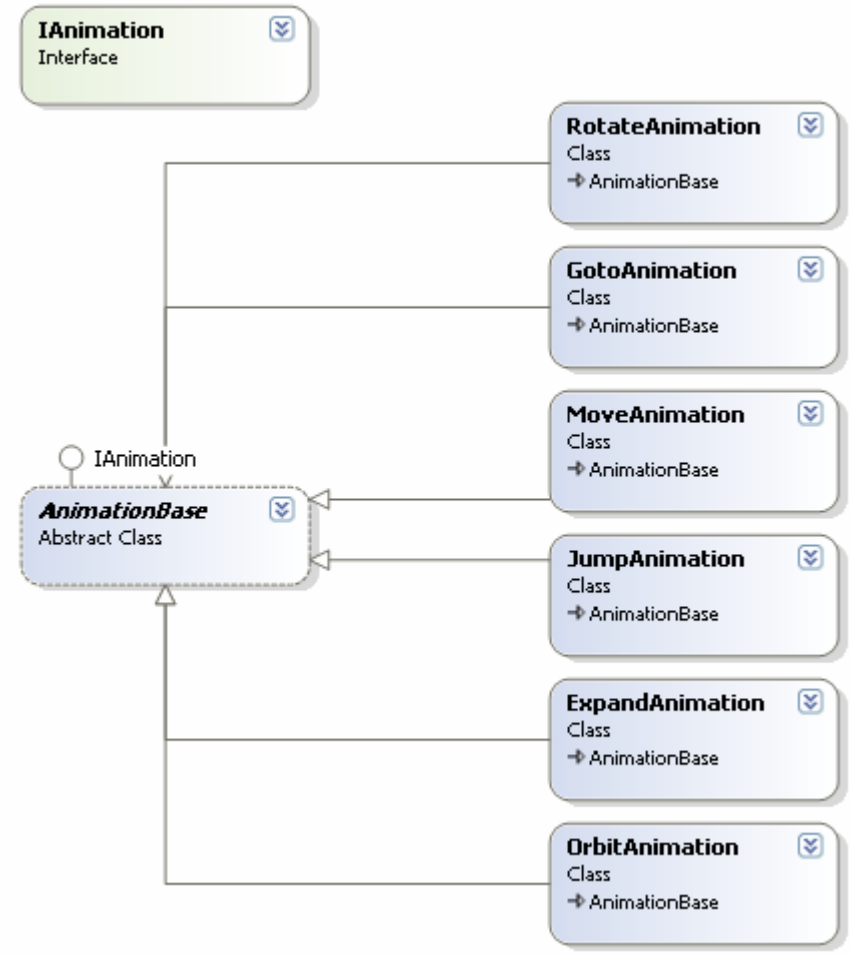
Figur 49: 3D-framework

Figur 50 beskriver projektets og 3D-frameworkets samling af 3D-primitiver og predefinerede objekter.



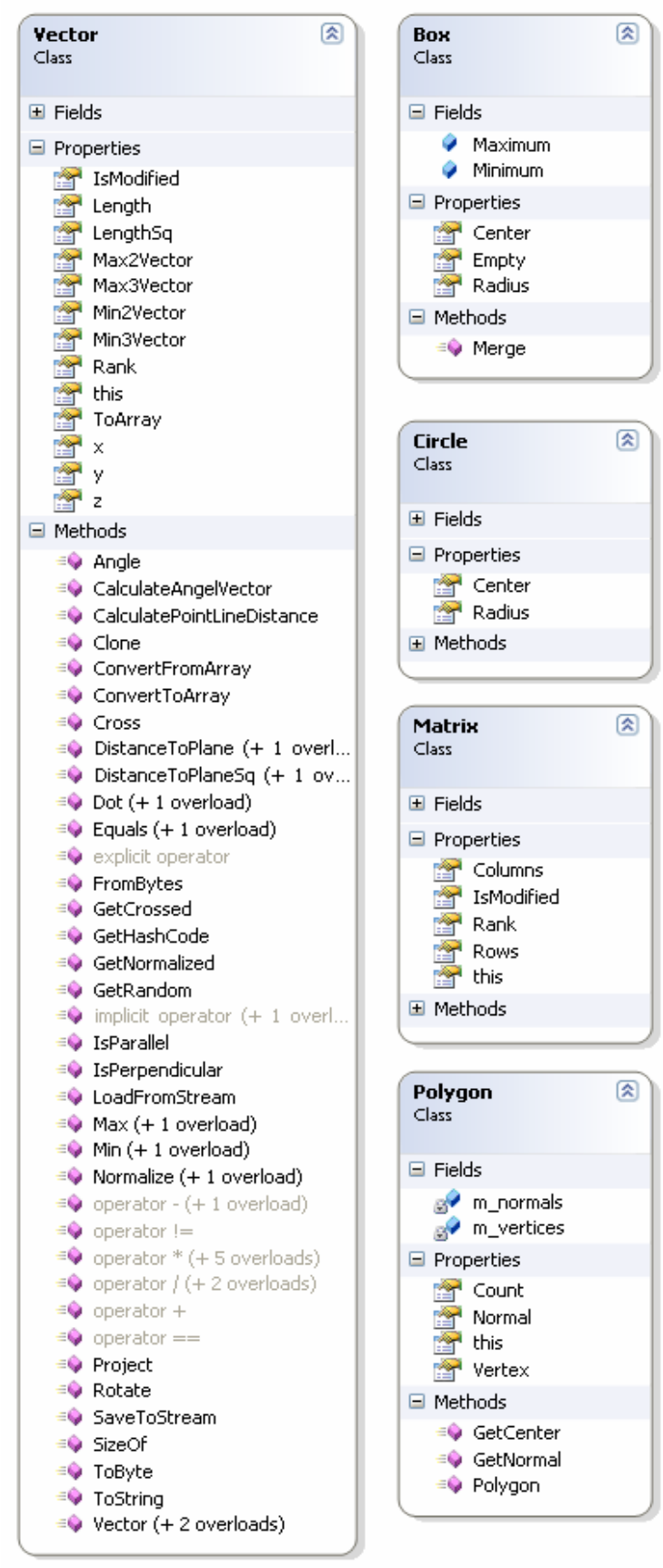
Figur 50: 3D-objektsamling

Figur 51 beskriver projektets og 3D-frameworkets animationsklasser.



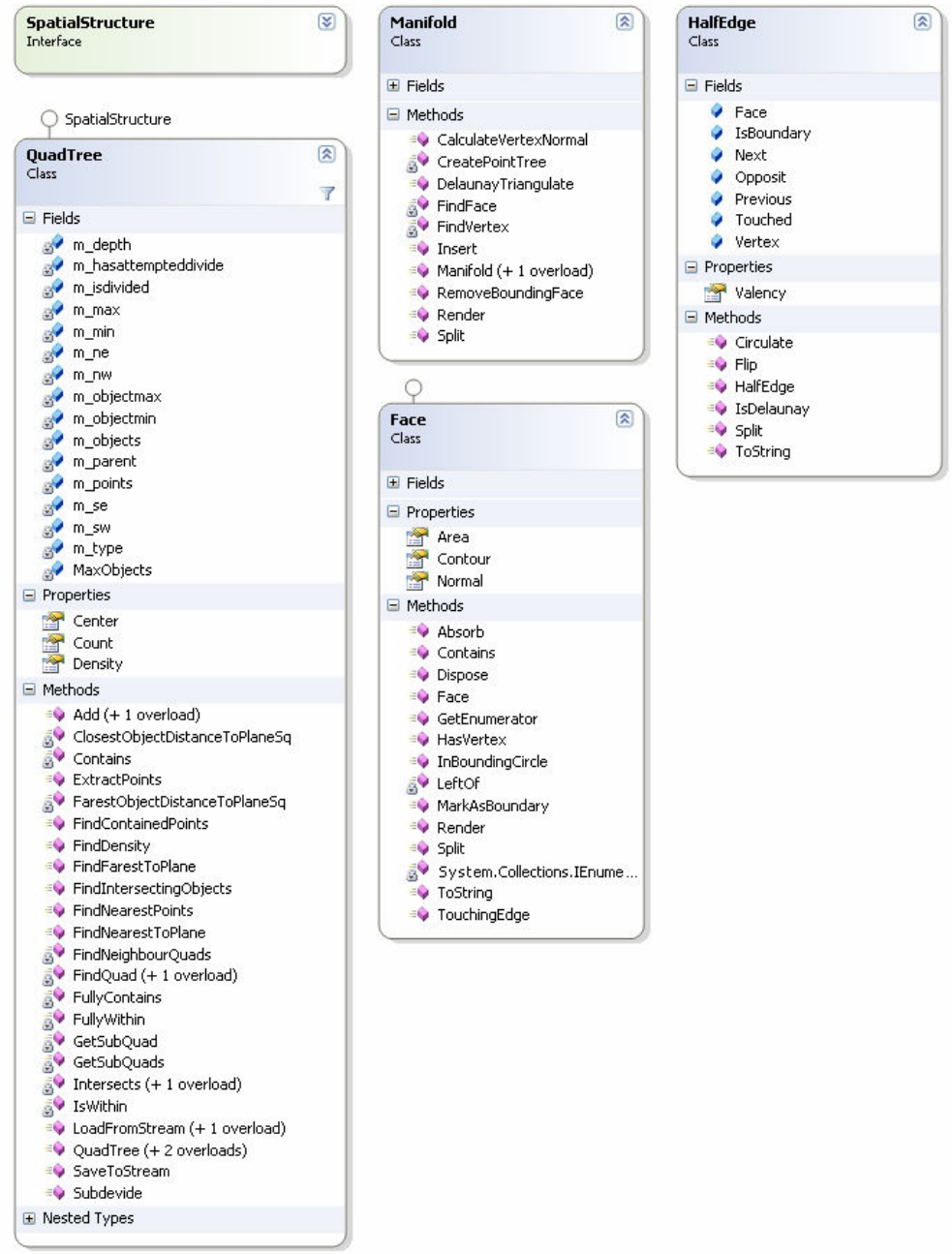
Figur 51: 3D-animation

Figur 52 beskriver klasser til brug for basale 3D-beregninger, såsom vektorer og matricer.



Figur 52: Lineært algebra

Figur 53 beskriver projektets spatielle datastrukture. Første version af programmet implementerer et simpelt QuadTree. Tilgang til dette sker derfor via et interface, således at senere udskiftning lettes. Programmet implementerer derudover også et *halfedge*-system, til geometriske beregninger.



Figur 53: Spatial datastruktur (QuadTree og HalfEdge)

6.2 3D-Framework

Projektet udvikles i .NET, som der ikke har nogen native implementation til generering af 3D-grafik. (Som f.eks. Java ellers har.) Dette betyder at der enten skal, udvikles et bibliotek til dette, udvikles et interface til et

eksisterende, eller der skal bruges et tredjeparts-interface. På nuværende tidspunkt findes der en række forskellige tredjeparts-interfaces til både OpenGL og DirectX.
Det mest kendte til DirectX er Microsofts eget DirectX.NET [MICROSOFT D3D]. Dette er en objektorienteret wrapper til ANSI-C-versionen og indeholder desuden et fyldigt vektor/matrix-bibliotek. Ulempen ved dette, er at det er Microsofts egne DirectX-folk der udvikler på det. Dette lyder umiddelbart som en god ting, men i praksis betyder dette, at det er folk der har arbejdet med ikke-objektorienteret ANSI-C-kode i de sidste 15 år, der også skal lave den objektorienterede .NET-version, hvilket sætter sine præg. Derudover har jeg efter lang tids arbejde med både DirectX og OpenGL, måttet konstaterer at DirectX er unødigt komplekst og dårligt dokumenteret, i forhold til OpenGL. DirectX.NET-versionen lider ydermere af endnu dårligere dokumentation, samt besværlig installation. Den pt. nemmeste løsning på 3D-rendering i .NET er derfor et interface til OpenGL.

6.2.1 OpenGL.NET

Der findes en række forskellige tredjeparts-.NET-interfaces og frameworks til OpenGL, hvoraf det mest kendte er Tao [TAO]. Kendetegnet ved mange af disse frameworks/interfaces, er at programmørerne importerer OpenGL-funktionerne, omdøber dem til noget mere sigende (ifølge dem selv), simplificerer brugen af dem og tilføjer en række hjælpeobjekter, som eks. kamera- og lys-objekter. Dette er ofte også en god ting, men efter at have arbejdet en del med 3D-rendering, er jeg kommet frem til følgende:

- OpenGL's navngivning er konsekvent og uden for mange spor af Hungarian-notation.
- Den er derudover (som regel) også meningsfuld og selve brugen af OpenGL er simpel, når man først har overstået initialiseringen.
- Performance i forbindelse med 3D-rendering af yderst vigtig karakter og bliver bl.a. påvirket af antallet af funktionskald, hvilke funktioner der bruges og rækkefølgen af disse.
- Til sidst så er 3D-rendering også ofte meget sart overfor kombinationen af funktionskald og initialisering. Den mest almindelige fejl i OpenGL og DirectX, er at der slet ikke renderes noget. Funktionaliets-dokumentationen er derfor yderst vigtig og fungerer i OpenGL ved at man indtaster funktionsnavnet i Google. OpenGL-interfaces/frameworks er derfor kritisable hvis de:
    - Ændrer funktions-navngivningen, da dette ødelægger muligheden for dokumentationssøgning.
    - Ænder navngivningen til noget er der mere uforståeligt end det eksisterende.
    - Laver simplifikationer af funktionalitet der i bund og grund ikke behøver simplificering og dermed tilføjer kompleksitet.
    - Laver hjælpefunktionalitet hvis brug af de oprindelige OpenGL-funktioner ikke er gennemskueligt.
    - Laver hjælpefunktioner som der af hensyn til garanteret funktion, afvikler for mange kald til OpenGL.

Frameworks som eks. Tao opfylder mange de kritisable punkter som jeg har opstillet og projektet vil derfor indeholde egen implementation af et OpenGL-interface. Dette er dog forholdsvis simpelt og foregår på følgende vis:

- Konverter OpenGL-header-filerne til C#-klasser, ved at bruge søg-og-erstat til at ændre funktionserklæringerne fra ANSI-C til C#. Eksempelvis så ændres følgende funktion:
  ```
  void glVertex3f(GL_FLOAT x, GL_FLOAT y, GL_FLOAT z);
  ```
  Til:
  ```
  [DllImport(DLL, EntryPoint="glVertex3f")] public static extern void glVertex3f(float x, float y, float z);
  ```
  Elementer som macro'er ændres enten til konstanter, funktionskald eller pakkes blot ud i resten af koden via søg-og-erstat.
- Fjern OpenGL's prefix. Prefixene bruges i forbindelse med ANSI-C-programmering til at adskille et sæt funktioner fra et andet. Dette gøres i objektorienterede miljøer, via namespaces eller klasser. For at bibeholde OpenGL-navngivningen, så evt. dokumentation ikke går tabt, kan "gl"-funktioner indsættes i en "gl"-klasse", således at kaldet til ovenstående funktion nu bliver til:
  ```
  gl.Vertex3f(x, y, z);
  ```
  Altså med et punktum, der blot skal fjernes ved dokumentationssøgning.
- Tilføj manglende overloads og fjern de unødvendige profix. OpenGL's funktioner er ikke overloadede. Ment på den måde at hver udgave af en funktion har et unikt navn. Eksempelvis så er ovenstående funktion en af mange "vertex"-funktioner, der alle gør det samme, men med forskellige argumenter. Dette er der også taget højde for i dokumentationssøgningen og søgeordet "glVertex" vil derfor føre til rette område. Dette kan derfor indbygges i wrapperen og på den måde forbedre IDE'ets *intelliSense*. Den ovenstående funktion vil derfor blive til:
  ```
  gl.Vertex(x, y, z);
  ```
- Konverter OpenGL's funktions-konstanter til enums. Eksempelvis så kan konstanten "`pname`" til funktionen "`glLight`" laves om til enumen:
  ```
  public enum LightParams: uint
  {
            AMBIENT               =       0x1200,
            DIFFUSE               =       0x1201,
            SPECULAR              =       0x1202,
            POSITION              =       0x1203,
            SPOT_DIRECTION        =       0x1204,
            SPOT_EXPONENT         =       0x1205,
            SPOT_CUTOFF           =       0x1206,
            CONSTANT_ATTENUATION  =       0x1207,
            LINEAR_ATTENUATION    =       0x1208,
            QUADRATIC_ATTENUATION =       0x1209,
  }
  ```
  Dette vil ligeledes give et stort boost til IDE'ens *intelliSense* og på den måde forøge implementations-hastigheden. Derudover vil det

også sikre at den omtalte funktion kun modtager understøttede værdier, så man eksempelvis ikke får lyst til, at give den konstanter som eks. ”`GL_LIGHTNING`” eller ”`GL_LIGHT1`”.

Den planlagte wrapper-struktur er illustreret i afsnit 6.1.4. Bemærk i denne forbindelse det manglende OpenGL-bibliotek ”glut”. Dette vil ikke blive indopereret i projektet, da ”glut” generelt er hjælpefunktioner til vinduegenerering og denne del af projektet klares langt bedre via .NET selv.
Ud over selve wrapperen, er det også rart med et par standardobjekter, til at hjælpe med renderingen. Dette er klassisk set et kamera-, lys- og et sceneobjekt. Kamera-objektet bruges til at holde styr på ting som øje-position, retning og måske endda også skærm-størrelse og resolution. Lys-klassen bruges til at pakke funktionalitet til dette ned i og scene-klassen kan bruges til at opbygge, indsætte og styre visuelle objekter i. Jeg har valgt ikke at bruge noget sceneobjekt i projektet, da dette i vores tilfælde er selve kortet. Ud over kamera og lys er det også ofte behjælpeligt med en række primitiver som f.eks. en boks, cylinder, kegle o.s.v. Ud over at kunne bruges til at opbygge mere komplekse former, bruges de også til testscener. (Man bruger meget tid på tests i 3D-udvikling.)
Til sidst vil det også være rart med nogle hjælpefunktioner til de mest platformsspecifikke dele af initialiseringen. Det er denne del som glut ellers normalt tager sig af. 3D-bliblioteker som OpenGL og DirectX har forholdsvise komplicerede initialiseringer, da begge lægger sig tæt op af computerens hardware.
Det ovenstående framework og wrapper er illustreret i afsnit 6.1.4.

6.2.2 Lineær algebra

.NET’s basis-framework indeholder ikke nogen former for vektor/matrix-biblioteker og OpenGL’s behandling af disse, er heller ikke altid så gennemskuelig eller omfattende, som man godt kunne tænke sig. Igen så er løsningen derfor, enten at bruge et tredjeparts-bibliotek, eller at lave et selv. Jeg har endnu ikke været i stand, til at finde noget ordentligt tredjepart-programmel til .NET. DTU bruger ofte et bibliotek kaldet GEL [GEL] til disse beregninger, hvilket også ville være yderst brugbart i dette projekt. GEL er dog desværre for indfiltret i C++, til at være hurtigt porterbart til .NET. Dog har løsningen med at lave det selv, også vist sig at give flere fordele. For det første, så er lineære algebra ikke det sværeste at implementere. Man kommer virkelig langt med en vektor-klasse, der implementerer de mest almindelige operatorer som plus, minus, gange, dividere, prik og kryds. Derudover så er det også en stor hjælp, at have mulighed for at tilpasse og tilføje ekstra funktionalitet til biblioteket, undervejs i udviklingen.
Projektet vil derfor selv implementere det fornødne vektor/matrix-funktionalitet og vil følge strukturen fra afsnit 6.1.4.

6.3 3D-beregninger

Projektet generelt, er opbygget af 3D-beregninger. Dog indbefatter dette dog mest, vektor-beregninger og lineært algebra. Jeg har derfor udvalgt 2 beregninger til at fremvise.

6.3.1 Navigation

Ethvert 3D-projekts første problemstilling, går ud på at placere og navigere objekter.
Objekter bliver givet deres eget lokale koordinatsystem, hvori objektet bliver defineret. Eksempelvis udgør følgende punkter en pyramide:

(0, 0, 0), (2, 0, 0), (2, 2, 0), (0, 2, 0), (1, 1, 2)

Hvis pyramiden skal placeres i punktet α, så flyttes hele det lokale koordinatsystem (translation):

$$p' = Tp \tag{6.3.1.1}$$

Hvor det gælder at:

$$T = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3.1.2}$$

Hvis objektet skal skaleres, gøres det på samme måde:

$$p' = Sp \tag{6.3.1.3}$$

Hvor det gælder at:

$$S = \begin{bmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3.1.4}$$

Skalering bliver kun brugt i projektet til at skalere z-aksen, efter ønske fra brugeren (afsnit 4.6.2). Matricen er derfor i dette tilfælde:

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3.1.5}$$

Hvis objektet skal roteres, gøres det på følgende måde:

$$p' = R_x R_y R_z p \tag{6.3.1.6}$$

Hvor:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3.1.7}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.3.1.8)$$

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.3.1.9)$$

Den endelige metode til at placere et givent objekt i projektet er derfor:

$$p' = TRSp \quad (6.3.1.10)$$

Og hvis eksempelvis et objekt (eks. kamera) skal foretage en orbit (uden skalering) omkring punktet α, i afstanden γ så ser ligningen således ud:

$$p' = T(\alpha)R_{orbit}T(\gamma)Rp \quad (6.3.1.11)$$

6.3.2 Slagskygger

Jeg vælger at implementere den simple form for slagskygger, der laves ved at projektere alle punkter fra et objekt, ned på et givent plan. Dette giver en nem implementation med god performance. Dog vil der opstå problemer, hvis skyggen der projekteres, ikke befinder sig på et fladt plan.
Jeg vælger at bruge formlen fra [PHAETOS] til denne udregning:

$$Z = \begin{bmatrix} L_x n_x & L_x n_y & L_x n_z & -cL_x - dL_x \\ L_y n_x & L_y n_y & L_y n_z & -cL_y - dL_y \\ L_z n_x & L_z n_y & L_z n_z & -cL_z - dL_z \\ n_x & n_y & n_z & -d \end{bmatrix} \quad (6.3.2.1)$$

Hvor det gælder at:

$$d = Ln \text{ og } c = En - d$$

Hvor L er lysets retning, n er normalvektor til planet og E er et punkt i planet.
Skyggeversionen af et objekt er derfor:

$$p' = ZTRSp \quad (6.3.2.2)$$

En bedre skyggeberegning kan dog laves via shadow volumes [BEAM]. En teknik der går ud på, via grafikkortet, først at rendere billedet set fra lyskilden og derefter fra kameraet af. Ved at kombinere de to billeder, kan man da skabe shadow volumes, der er uafhængige af de projekterede flader og objekter. Dette ville komme til udtryk i projektet, ved at skyggerne ville kunne kastes på arbitrært terræn og på arbitrære objekter, modsat de ovenstående projektionsskygger.

Men som nævnt, så giver projektionsskyggerne en bedre performance, end volume shadows og første version af programmet vil derfor implementere projektionsskygger.
Lysets retning til brug for skyggerne beregning iflg. [STINE & GEYER] på følgende vis:
Solar-vinkel:

$$\omega = 15(t_s - 12) \tag{6.3.2.3}$$

Hvor t_s er solar tid, i timer.
Declination-vinkel:

$$\delta = \sin^{-1}(0{,}39795 \cdot \cos[0{,}98563(N - 173)]) \tag{6.3.2.4}$$

Hvor N er dag på året.
Højde-vinkel:

$$\alpha = \sin^{-1}(\sin\delta\sin\phi + \cos\delta\cos\omega\cos\phi) \tag{6.3.2.5}$$

Hvor φ er breddegraden.
Azimuth-vinkel:

$$A'' = \cos^{-1}\left(\frac{\sin\delta\cos\phi - \cos\delta\cos\omega\sin\phi}{\cos\alpha}\right)$$
$$A = \begin{cases} 360 - A'', \omega > 0 \\ A'', \omega <= 0 \end{cases} \tag{6.3.2.6}$$

Herefter beregner jeg da L på følgende vis:

$$L = -\begin{bmatrix} \sin A \\ \begin{cases} -\cos A, \phi > 0 \\ \cos A, \phi <= 0 \end{cases} \\ \tan\alpha \end{bmatrix} \tag{6.3.2.7}$$

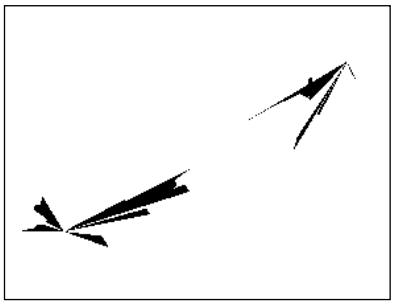
3D-rendering af 2D

Grundet computerskærmens udformning (2D-billede), den menneskelige abstraktionsevne og computergrafikkens historie, så er 2D-rendering stadigvæk den mest udbredte renderingsform og har af disse grunde en række indbyggede forenklinger, antagelser og smuthuller, set i forhold til 3D-rendering. Data der er lavet til at blive renderet i 2D, er derfor sjældent egnet til at blive renderet i et af de eksisterende 3D-frameworks. Følgende afsnit vil derfor beskrive nogle metoder til at forberede 2D-data til 3D.

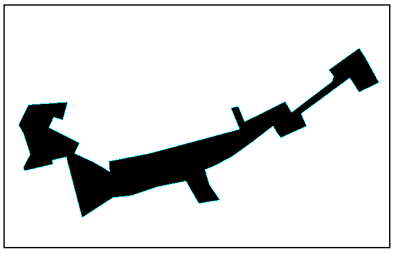
6.3.3 Rengøring af polygoner

Første fælde i 3D finder man, når ens tegnede objekter ikke bliver vist eller kun bliver vist delvist. Dette kan skyldes at 3D-flader har en for- og en bagside og som standard bliver kun forsiden vist. Grunden til at de har dette, skyldes at man i lysberegningerne (shading) skal kende opretningen (normalvektor) på objektet. Man kan dog vælge at definere begge sider på en flade, som værende forsider, men i 3D-regi betyder dette at grafikkortet skal behandle dobbelt så mange sider som ellers. Det giver også god mening, at det kun er ydersiderne på objekterne, der skal være synlige. Hvis ikke man selv vil definere, hvad der er op og ned på ens vertices, skal man derfor holde styr på, hvilken omløbsretning man tegner sine objekter med. I OpenGL skal ens objekter som standard, tegnes "mod uret" [OPENGL]. Dette betyder at man ved 2D-data, først skal have styr på, hvilken retning figurerne er tegnet i. Figur 57 er et eksempel på en 2D-figur, hvor noget af den, er tegnet i en anden retning. Som nævnt kan dettes også overkommes ved blot, manuelt at definere opretningen for hele figuren, hvilket er relativt nemt for en 2D-figur. Dog er det af hensyn til senere behandling en god ting, både at have styr på omløbsretningen af sine figurer, samt hvad der er "inde" og "ude" på dem. Især "inde" og "ude"-konceptet kan være problematisk i 3D, hvis man samtidig ikke har styr på sin omløbsretning.

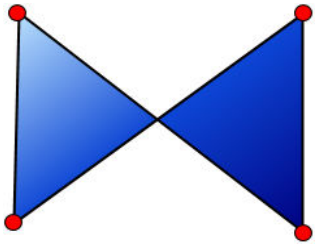
Næste problem man støder på, kan være objektet på Figur 55. Her ses en figur med 4 punkter, der danner en butterfly. Objektet krydser sig selv. Situationen er ikke ualmindelig for MapInfo-data. De fleste 3D-frameworks vil dog kløjes i figurer som disse, da man i 3D-sammenhæng altid renderer trekanter. Figur 55 er derfor i bedste fald en firkant set med 3D-øjne. Løsningen på dette er at finde stederne hvor figurerne krydser og her indsætte ekstra punkter. Jeg vælger



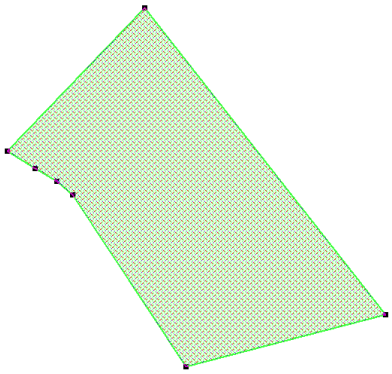
Figur 57: Figur kun tegnet delvist



Figur 56: Figur tegnet korrekt (billede fra GIS3D)



Figur 55: 2D-figur der slår krølle på sig selv

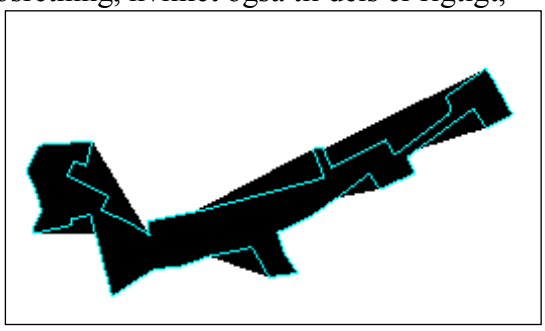


Figur 54: MapInfo melder om 11 linier i dette objekt, der har 7 synlige punkter (billede fra MapInfo)

dog i stedet for, i projektet, blot at informere brugeren om disse objekter, da objekter der krydser sig selv, også defineres som fejl-objekter i korrekt GIS. En anden situation i samme boldgade som de krydsende objekter, er objektet på Figur 54. Her ses et objekt med 7 punkter og kanter. Hvis man dog undersøger data, eller får MapInfo til vise egenskaberne for den, fortæller den at objektet har 11 kanter. Dette skyldes at flere af punkterne/kanterne ligger dobbelt. Igen er dette ikke et ualmindelig objekt i kommunalt MapInfo-data. Objektet kan muligvis godt tegnes af et 3D-framework, men sandsynligheden for at det ikke kan, eller at en tilhørende algoritme bryder sammen, på grund af den, er stor. Løsningen er derfor blot at fjerne de dobbelte punkter.
En sidste ting man skal tage stilling til i 2D-data, er hvorvidt objekterne er lukkede eller åbne. Altså hvorvidt det sidste punkt er placeret samme sted som det første. Hvilket man vælger, er for så vidt lige meget, så længe det blot er konsekvent. Programmer som eks. MapInfo har dog en tendens til ikke at være konsekvent med den slags og det er derfor nødvendig at checke for det.

6.3.4 Triangulering

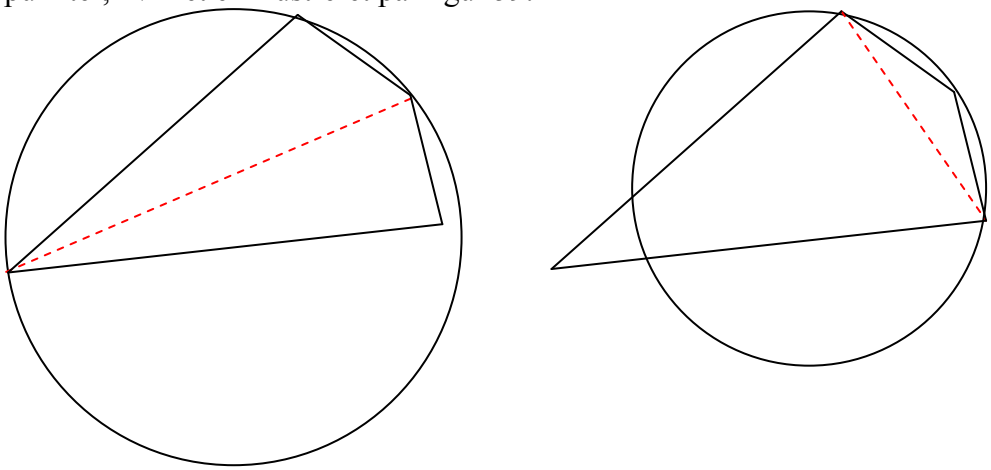
En sidste ting man skal gøre ved 2D-data, før det kan vises i 3D, er at det skal trianguleres. Dette skyldes ikke fejl i data, men derimod snarere fejl i 3D-frameworkene. I afsnit 6.3.3 nævnte jeg, at det er nok blot at tegne sine objekter med en konsekvent omløbsretning, hvilket også til dels er rigtigt, da OpenGL derefter selv kan klare resten. På Figur 58 ses objektet fra Figur 56 renderet via OpenGL's egen triangulering. Omløbsretningen på objektet er forinden sorteret korrekt og som det kan ses er der heller ikke noget af objektet der vender på hovedet. Til gengæld er der generelt "for meget" objekt. Løsningen er derfor selv at triangulere det. En måde at triangulere på, kan eksempelvis være Delaunay-algoritmen [WIKI], hvilket der forsøger at maksimere trekanternes minimums-vinkler og på den måde skabe pæne trekanter. Denne kan udføre på følgende måde:



Figur 58: Førviste objekt, trianguleret af OpenGL

```
Punktsættet P haves
Indsæt bounding trekant i T, indeholdende P
For hvert punkt p i P:
     Indsæt p i indeholdende trekant
     Indsæt de 3 nye trekanter i T
     For hver trekant t i T:
          Check om alle kanter i t er Delaunay
          Hvis ikke så flip
     Loop t
Loop p
Fjern bounding trekant fra T
```

Metoden til at teste hvorvidt en kant er Delaunay, udføres ved at teste hvorvidt den ene trekants omkransende cirkel, indeholder alle den andens punkter, hvilket er illustreret på Figur 59.



Figur 59: Den samme kant (stiplede) i to forskellige situationer. Den første (venstre) er Delaunay, da den omkransende cirkel indeholder den modsatte trekant. Den anden kant (højre) er af samme grund ikke.

Hvis ikke kanten opfylder Delaunay, så "flippes" den. Et eksempel på dette ses ligeledes på Figur 59, hvor eneste forskel på de to situationer er, at den røde kant er "flippet".
I projektets hovedtriangulering, vælger jeg dog ikke at implementere Delaunay-algoritmen. Delaunay's mål er at skabe pæne trekanter, hvilket jeg ikke er decideret interesseret i, da jeg ikke har planer om at bruge de efterfølgende trekanter, til andet end OpenGL-rendering. Den ovenstående algoritme har derudover en afviklingstid på $O(n^2)$, hvis der ikke foretages optimeringer. Jeg vælger derfor at bruge koden fra [RATCLIFF], der ikke garanterer Delaunay, men som til gengæld er hurtig. Delaunay-algoritmen vil jeg dog derimod bruge til at beregne vertex-normaler med, da det i denne forbindelse, er en fordel at have "pæne" trekanter.

6.4 Konvertering af 2D til 3D

Når data først er behandlet, således at det kan renderes via et 3D-framework, skal det derefter behandles således at det bliver til faktisk 3D-data, som beskrevet i afsnit 4.6.1. Dette gøres i 3 omgange:

- Objekter der indeholder z-information, skal have dette konverteret til koordinater
- Z-koordinater udtrækkes fra alle åbne kort og bruges efterfølgende til, at interpolere z-koordinater ud til alle manglende punkter.
- Objekter der er defineret som rummelige, skal konverteres til dette.

6.4.1 Interpolation af terræn

Som interpolationsmetode til projektet vælger jeg at implementere en simpel linear interpolation, hvor hver punkts vægt er proportional, med den inverse afstand til det ønskede interpolationspunkt [NIELSEN]:

$$w_i = \frac{\frac{1}{d_i}}{\sum_{j=1}^{N} \frac{1}{d_j}} \qquad (6.4.1.1)$$

Hvor det naturligvis gælder at:

$$d_i = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \qquad (6.4.1.2)$$

Den interpolerede værdi fås derfor som følgende:

$$z_i = \sum_{j=i}^{N} w_j z_j \qquad (6.4.1.3)$$

Dog vil ovenstående udregning, resultere i terræn med høje energi-niveauer (skarpe kanter og takker). For derfor at imødekomme lidt af dette, kan beregningen udvides til at vægte med forskellig potens:

$$w_i = \frac{\frac{1}{d_i{}^p}}{\sum_{j=1}^{N} \frac{1}{d_j{}^p}} \qquad (6.4.1.4)$$

Høje potenser (p > 1) vil resultere i, at de nærmeste punkter vægter mere end ellers, hvilket i praksis vil resultere i en vis *curvature minimizing*.
En endnu bedre metode, ville dog være eksempelvis Kriging-interpolation, da disse søger at minimere fejlvariansen og i praksis derfor giver flottere, mere naturligt og statistik mere korrekt terræn. Kriging er dog en smule kompliceret at implementere og vil sandsynligvis også være at skyde over målet i første udgave af dette projekt.
Som tidligere nævnt, så fås fixpunkterne til beregningerne, fra eksisterende data i kortene. Hvis den tilgængelige data derfor kun består af eksempelvis kloakkort, så vil den efterfølgende interpolation sandsynligvis også indbefatte en del ekstrapolation. Selv hvis et højdepunktskort eller et boringskort er til rådighed, vil der kunne forekomme ekstrapolation, da det ellers vil kræve, at der forefindes højdeinformation, udenfor det pågældende kortudsnit. Dette er desværre uundgåeligt, eftersom vi ikke har muligheden for at "mangle højdedata" i vores 3D-præsentation.
I den ovenstående interpolationsmetode vil ekstrapolation resultere i at terrænet vil følge de nærmeste kendte punkter og på den måde give et fladt yderområde.
En sidste problemstilling i interpolationen, handler om de punkter der bliver interpoleret. Jeg vælger kun at interpolere eksisterende vertices i lagene, hvilket betyder at hvis fladerne er for store og dermed indeslutter lokale minima eller maksima, så vil fladerne ikke illustrere dette. Dette kan også resultere i situationer hvor indesluttet geometri, vil svæve over terrænet, grundet manglende interpolation. Løsningen på dette er at indsætte yderligere punkter, i terrænfladerne, hvor der er brug for det. Indtil omfanget af dette problem er bedre belyst, venter jeg dog med at implementere dette.

6.4.2 Dannelse af rummelige objekter

Brugeren definerer en eller flere kortlag, til at indeholde rummelige objekter. Det kan eksempelvis være bygnings-lag. Metoden til at lave flader til 3-dimensionelle, rummelige objekter er simpel. Forinden er fladerne behandlet på samme måde som kortets andre flader, med triangulering, rengøring og interpolation. Det betyder at en rummelig udgave, kan laves ved at:

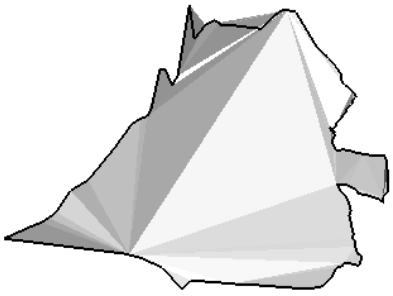
- Kopiere fladen og dens triangulering
- Transponere den kopierede flade i z-retningen
- Triangulere verticene mellem de to flader
- Plan den kopierede flade, så den står i vatter

Det sidste punkt er en smagssag og bygger på en forudsætning om at objekter som bygninger o.l. rent faktisk står i vatter. Jeg har derfor valgt ikke at udføre dette punkt, indtil omfanget af de potentielle fejl er nærmere undersøgt.

6.4.3 Beregning af lyssætning

Efter konverteringen til 3D, opstår et nyt 3D-rederingsspecifikt problem. 2D-data indeholder ikke eksplicit information om, hvad der er over- og under-side på en figur. (Det ligger i definitionen, eftersom denne information tilhører den 3. dimension.) 3D-rendering kræver dog dette, for at kunne beregne lys (shading) på objekterne. Informationen kommer i form af objekt-normaler, hvilket der derfor skal beregnes for de nye 3D-objekter. Uden normalerne vil en 3D-flade se flad ud, uanset vinkel.
Da projektets terrænmodel består af "region-objekter", der tilsammen udgør terrænet, vil projektet implementere to former for lyssætning. Flat shading og Gouraud shading [ANGEL2]:

- Flat shading er den nemmeste og hurtigste performancemæssigt set og kan udregnes separat for hvert objekt, uden kendskab til naboobjekter. Den beregnes ved at udregne normalen for hver trekant i objektet.
- Gouraud shading kræver at hver vertex har en tilhørende normal. OpenGL vil derefter interpolere lyssætningen fra hver vertex og på den måde give en flydende shading. Projektets datastruktur kommer i denne forbindelse lidt i vejen, da beregningerne til en vertex-normal, kræver normalerne fra alle tilhørende nabotrekanter. Derudover vil det bedste resultat, også fås via Delaunay-trekanter og ikke dem som projektet laver i indlæsnings-fasen. Løsningen er derfor at lave en separat udregning af vertex-normalerne på følgende vis:
    - Udtræk region-vertices fra alle lag
    - Indsæt vertices i samlet nabo-struktur
    - Udfør Delaunay-triangulering



Figur 60: Region-objekt med Flat shading (billede fra GIS3D)

- Beregn normaler til alle vertices
- Gem beregnede normaler i de oprindelige objekter

Den omtalte nabo-struktur er blot et udtryk for en struktur, der indeholder information om alle nabo-objekter. Til dette formål vil jeg derfor implementere en *HalfEdge*-struktur, der inspireres af den fra [GEL].

6.5 MapInfo

Grundet programmets standalone struktur, er interaktionen med MapInfo, begrænset til læsning af MapInfo-data og arbejdesområder og til et MapInfo-plugin der opstarter projektets program GIS3D. Følgende afsnit beskriver disse.

6.5.1 MapInfo Professionel-plugin

Der vil blive udviklet en MapInfo-MBX, som kan autostartes i MapInfo. Dette plugin vil tilføje en ekstra menu til MapInfo's "Vindue"-menu. Via denne vil brugeren kunne aktive pluginet. Pluginet vil derefter gemme en midlertidig kopi af brugerens nuværende kort-situation (også kaldet et arbejdsområde). Det midlertidige arbejdsområde vil derefter blive afleveret som parameter til GIS3D-programmet, der da vil vise samme kortudsnit, blot i 3D.

6.5.2 Load af native-MapInfo-data

MapInfo's eget dataformat, er et binært format med en aldrende protokol. Derudover har MapInfo heller ikke udgivet nogen former for dokumentation over denne og læsningen af disse er derfor ikke ligetil. Jeg vælger derfor at bruge et subset af GDAL-biblioteket [MITAB], til at håndtere denne opgave. GDAL og dets underbiblioteker er ANSI-C- og C++-biblioteker. Der skal derfor bruges et .NET-interface til dette, på samme måde som til OpenGL. Udviklerne bag biblioteket, har dog i dette tilfælde, selv vedlagt et sådant og jeg vælger derfor at bruge dette. Dog har udviklerne ikke udbygget interfacet fuldt ud og denne del vil jeg derfor selv udføre.

6.5.3 Load af MapInfo-arbejdsområder

MapInfo's arbejdsområder er skrevet i ASCII-format og med MapBasic-syntax. De er derfor forholdsvis lette at læse og fortolke. Følgende informationer vil blive udlæst af arbejdsområderne:

- Tabeller/kort der skal åbnes
- Rækkefølgen af lag
- Kameraets startposition og zoom-niveau
- Lagenes synlighed og LOD-synlighed

For at undgå at lave en decideret parser til arbejdområde-syntaxen, så nøjes jeg med, blot at søge data igennem for foruddefinerede kendte sætninger. Eksempelvis så vil følgende regulære udtryk (for beskrivelse af regulære udtryk i almindelighed, kan jeg henvise til [GOYVAERTS]), finde et arbejdsområdes start zoom:

```
set\smap\s+.*?zoom\s+([\+|-]?[0-9][0-9]?[\.[0-
9]*]?)\s+units\s+"(.+?)
```

6.6 Performance

3D-programmer har generelt meget fokus på performance, da 3D-rendering hurtigt kan blive en smertefuld oplevelse for brugeren. Det er også ofte den praktisk opnåelige performance, der sætter grænsen for, hvor mange og hvilke ekstra features, der kan indbygges. Tidlige tests med programmet har bl.a. vist, at det med de nedenstående performance-tiltag, ikke vil være behageligt for brugeren, med zoom-animationer, grundet en for lav framerate. Ligeledes er features som eks. *antialiasing* også udsat til senere implementations-trin grundet samme målinger. Dette afsnit vil derfor beskrive de tiltag, der implementeres i projektet og også hvilke tiltag der er nødvendige for evt. forbedring af samme.

6.6.1 Afstandsberegninger

Afstandsberegninger er noget der bliver brugt mange steder i projektet. Eksempelvis så er interpolationen i afsnit 6.4.1 næsten udelukkede bygget på det. Afstandsberegningerne har den ulempe, at de (den Euclidiske version) bruger kvadratrødder, hvilket der selv på moderne computere kan være problematisk, performancemæssigt set. En forbedring af afstandsudregningerne vil derfor kunne forbedre performance i store dele af programmet. I [BAPTISTE] fremstiller Rafael Baptista følgende approksimerende afstandsfunktion:

$$f(x,y) = \frac{1007}{1024}\max(|x|,|y|) + \frac{441}{1024}\min(|x|,|y|) - if\left(\max(|x|,|y|) < 16\min(|x|,|y|), \frac{5\max(|x|,|y|)}{128}, 0\right) \quad (6.6.1.1)$$

Denne funktion vil jeg derfor implementere i projektet og lave en kort test af dets virke, for at fastslå om dets brug er relevant i .NET-miljøer, eller om dettes *SSE*-implementation overflødiggør lignende tiltag.
Udover den ovenstående approksimation, så vil jeg i samme test også afprøve 1-norms afstanden, som udregnes som følger:

$$f(x,y) = x + y \quad (6.6.1.2)$$

Derudover tilføjer jeg også mit eget bidrag, der bygger på at afstanden kan approksimeres på baggrund af relationen mellem dimensionerne:

$$f(x,y) = (x+y)\left(\frac{\sqrt{x^2+y^2}}{x+y}\right) \rightarrow$$
$$f(x,y) \approx (x+y)\cdot 0.8022\left(\frac{y}{x}\right)^{0.0483} \quad (6.6.1.3)$$

Ovenstående kan for yderligere performanceforbedring, bruges til specialtilfældene:

$$f(x,y)_{2/1} = (x+y)\cdot 0{,}745355992 \qquad (6.6.1.4)$$

Og:

$$f(x,y)_{3/1} = (x+y)\cdot 0{,}790569415 \qquad (6.6.1.5)$$

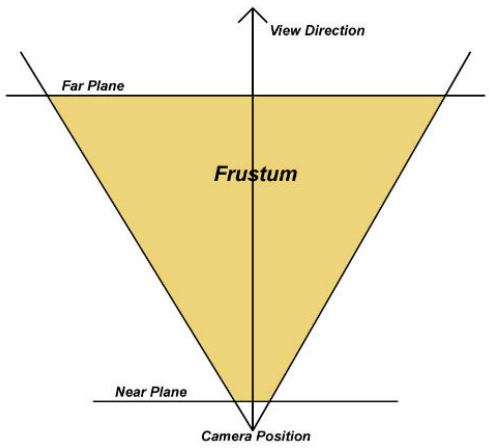
De to ovenstående beregninger bygger på en antagelse om, at afstande i praksis ofte er i x/y-forholdet 3/1, eller deromkring. En konstant til imødegåelse af dette, kan derfor udregnes på forhånd.

6.6.2 OpenGL-grafiklister

OpenGL som projektet bliver udviklet til, har modsat DirectX, en mulighed for at lave en serie af renderingstrin, om til en kompileret grafikliste. Disse lister bliver gemt og hentet af OpenGL og er hurtigere at rendere, end en tilsvarende ikke-kompileret serie. De bliver ikke ofte nævnt i forbindelse med professionelle 3D-applikationer som spil, da disse ofte udvikles til DirectX, men også fordi de kompilerede lister nemt kan komme i sammenstød med andre optimeringsteknikker, som eks. Culling. Projektet har dog en fordel af grafiklisterne, frem for andre applikationer, da dette er udviklet i .NET og ikke i C++. .NET-applikationer der kobler op til *unmanaged code,* kan blive udsat for en lille performance-hæmsko i form af ekstra *marchaling,* i forbindelse med disse kald. Det vil sige, at hvis man kan udføre samme algoritme, via færre kald til (i dette tilfælde) OpenGL, så vil man sandsynligvis få, en lille forbedring. Dette giver indvirkning på grafiklisterne, da de netop kan reducere renderinger med tusindvis af kald, til blot et enkelt kald.
Projektet vil derfor udnytte grafiklister til rendering af kortlag, således at et lag også er en grafikliste. Dette gør at lag hurtigt kan gøres enten synlige eller usynlige, uden at der er noget der skal genkompileres.

6.6.3 Frustum Culling

En af de mest effektive performance-forbedringer i 3D-applikationer, som eks. spil, er Frustum Culling [PICCO]. Faktisk er det i applikationer med større scener utænkeligt, ikke at bruge en form for Frustum Culling. På Figur 61 er vist en illustration af et frustum. Bemærk at man i 3D-hardware-rendering arbejder med en ”*near clip”*- og en ”*far clip”*-grænse, også af hensyn til performance. Dette bevirker at det synlige område for et kamera, tager form som en pyramide uden top. Frustum Culling går ud på,



Figur 61: Illustration af frustum (billede fra [PICCO])

kun at rendere objekter, der er indenfor pyramiden. Eller rettere sagt, kun at sende objekter indenfor pyramiden til grafikkortet. I både hardware- og software-rendering, bliver alle scenens objekter behandlet med samme algoritmer, med mindre dette netop forhindres. Ikke-synlige objekter vil derfor bruge næsten lige så mange resurser, som synlige. Hvis scenen derfor indeholder mange ikke-synlige objekter, kan performance forbedres drastisk, hvis man inden renderingen har frasorteret disse.
Situationen i dette projekt, er dog en del anderledes, end den i spil-applikationer. Den typiske scene i GIS, er en udvalgt del af et geografisk område. Eksempelvis er det i kommunal sagsbehandling, kun den pågældende kommune der udgør scenen. Brugeren vil derfor også meget ofte være interesseret i at se hele eller store dele af scenen. Dette betyder at målgruppens GIS-scener, ikke vil indeholde mange ikke-synlige objekter. Eller sagt på anden måde, så skal alle scenes objekter kunne renderes på samme tid.
Derudover gælder følgende for Frustum Culling:

- Frustum Culling-beregninger kan risikere at forværre performance, hvis algoritmens elementer ikke er optimerede eller hvis for mange objekter viser sig at være synlige.
- Frustum Culling kræver højeffektive spatialle partitions-træer, der ofte kan tage lang tid at implementere og debugge.
- Frustum Culling kræver *Collision detection* beregninger fra alle objekter, hvilket i sig selv kan være krævende. Den simpleste metode til dette er via bounding boxes, der dog kan indføre store fejlberegninger i forbindelse med komplekse objekter.
- Performance-optimeringer i Frustum Culling, indbefatter bl.a. simplificeringer af Frustum-pyramiden, afstandsudregninger og objekternes bounding objects, hvilket igen resulterer i flere (fejl) synlige objekter.

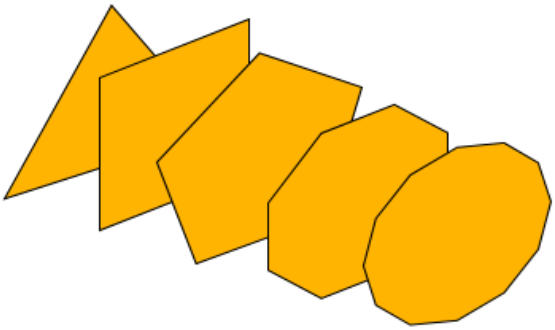
Derudover er projektets elementer primært opdelt i regioner og samlinger af regioner (kort-lag). Projektets grafiklister indeholder derfor, hvert et helt kort-lag, da der ikke er nok tilgængelige lister, til at have en separat liste for hver region. Frustum Culling vil derimod kræve, at lagene skal kunne renderes partielt, hvis det skal have nogen effekt. Dette betyder derfor, at man enten skal opgive grafiklisterne i projektet eller lave en subopdeling af kort-lagene.
Alt dette gør at jeg vælger, ikke at implementere Frustum Culling i første version af projektet. Hvis målgruppens data havde indbefattet hele landsdele eller kontinenter, ville man være tvunget til det, men i første omgang vurderer jeg, at det ikke er den ekstra implementationstid værd.

6.6.4 LOD-optimering

En anden effektiv performance-forbedring kan opnås, ved automatisk at justere vertex-detaljegraden for et objekt, afhængigt af hvor langt væk eller hvor småt objektet er. Dette skal ikke forveksles med LOD-niveauer, som dem der ses i [CITYGML], hvis formål er at forbedre overblik.

Figur 62: Cirkel i lav detaljegrad

Figur 62 illustrerer en vertex-simplificeret cirkel. Hvis figuren havde været på størrelse med en prik, ville den have lignet en cirkle. Nu ligner den i stedet en firkant, hvilket den også er. Hvis figuren skulle være opbygget af forbundene punkter (polylinie) og samtidig være en perfekt cirkle, så skulle cirklen består af uendelig mange små linier, hvilket ville være en hård belastning for grafikkortet. Grafikkortets performance (dets framerate) afhænger i høj grad af hvor mange vertices, man hælder ned i det.
Set på lang afstand, hvor figurerne vil optræde som små, vil der ikke værre synlig forskel på figuren med 4 linier og på den med uendelig mange linier. Selv på tættere afstande, hvor der kan skimtes en svag forskel, vil denne muligvis stadigvæk være acceptable. Hvis ikke, så kan firkanten i stedet opdeles, således at den i stedet får 8 linier o.s.v.
Projektets polygon-objekter vil ligeledes på autonom vis, kunne simplificeres. Dog vil der opstå potentielle problemer, hvis eks. to naboliggende cirkler i kortet bliver simplificeret til firkanter, da der herved kan skabes et synligt mellemrum eller andre artifacts, afhængigt af simplificeringsmetoden.
En mere interessant simplificering i projektet, vil derfor være at rendere kort-lag, som *billboards* ved store afstande. Dette gøres ved, først at rendere laget på normal vis og efterfølgende blot rendere det første renderede billede igen. Dette kan gøres, fordi terræn set fra høj afstand, generelt ser fladt ud. Et fladt *billboard,* vil derfor ikke ødelægge denne illusion.
Jeg har dog valgt helt at undvære objekt-simplificering i projektet. Projektets objekter kan muligvis godt simplificeres uden synlige tab, men ligesom autonom LOD-optimering heller ikke ofte optræder i CAD-applikationer, så mener jeg at arbejdet, der skal til for at give gode autonome objekt-simplificeringer i GIS, kræver for meget, i forhold til hvad det vil tilføje projektet.
En bedre og nemmere tilgang til simplificering, ville være at implementere LOD-modellen fra [CITYGML]. Denne har som tidligere nævnt ikke til formål at forbedre performance, men vil dog stadigvæk resultere i den ønskede forbedring, ved at kræve, at alle objekter skal indeholde flere simplificerede versioner. Disse simplificerede versioner, skal dog digitaliseres, mere eller mindre manuelt, således at der ikke opstår artifacts eller fejlagtige udseende objekter. Denne ekstra, manuelle digitalisering går dog imod projektets mål og jeg vil derfor, som også nævnt i afsnit 4.7, ikke implementere [CITYGML]'s model i første omgang.



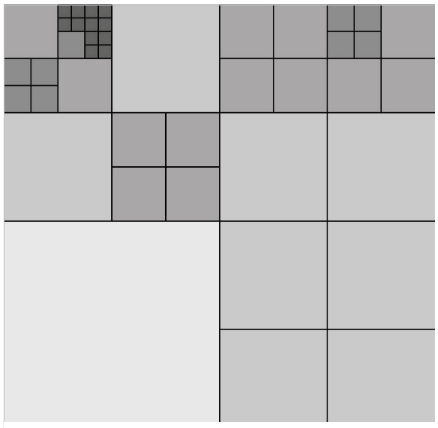
Figur 63: Et cirkel-objekt vil kunne indeholde flere digitaliserede LOD-niveauer af sig selv iflg. [CITYGML].

6.6.5 Occlusion Culling

Occlusion Culling omhandler teknikker, til at undgå at rendere objekter, der ikke er synlige grundet andre objekter. Eksempelvis så er det unødvendigt at rendere et objekt, hvis dette er placeret bag en væg og dermed ikke er synligt for kameraet, selvom objektet er indenfor Frustum. Disse teknikker er beregningsmæssigt ret tunge og implementationen og udnyttelsen af dem er derfor ofte undværet.
Situationen er dog anderledes i projektet. Her består scenen af coplanar, relativ flade objekter. Beregningerne til at udregne om et objekt skygger for et andet, er derfor forholdsvis simple. Ydermere så er alle objekter statiske og deres indbyrdes skyggeforhold ligeledes. Dette betyder at objekternes indbyrdes occlusion-forhold kun skal udregnes 1 gang.
Igen kommer projekts grafiklister dog i vejen her, da denne optimering vil kræve, at objekterne skal kunne renderes individuelt, hvis man ikke vil genskabe listerne hver gang. Og i praksis har tidlige tests med programmet også vist, at der er relativt få renderingsmæssige tunge objekter, der rent faktisk bliver skåret væk, ved denne metode. Denne er derfor heller ikke implementeret i første version af projektet.

6.6.6 Spatial partition tree

I de fleste 3D-applikationer, bruger man specialiserede datastrukturer, til at organisere applikationens spatialle objekter. Beregningerne det bruges til, kan enten være situationer som *collision detection*, frustum culling, occlusion culling, nærmeste nabo, selection e.l. Fælles for situationerne og datastrukturerne, er at man forsøger at undgå situationen med udviklingstiden O(n) eller værre. Beregningerne der bruges i forbindelse med 3D-rendering, er meget følsomme, da deres samlede afviklingstid helst skal kunne foretages på mindre end 1/50 sekund, grundet evt. animation. Projektet implementerer ikke nogle culling-optimeringer eller *collision detection* til at starte med. Til gengæld vil der være et stort behov for selection-optimering, da GIS netop skal forsyne brugeren med selection- og nærmeste nabo-muligheder. Derudover kræver funktionerne fra afsnit 6.7.1 bl.a. også mulighed for at fremsøge nærmeste og fjerneste objekt.



Figur 64: Illustration af projektets QuadTree (billede fra [DPI])

Jeg har valgt, at der skal implementeres en 2-dimensional datastruktur i projektet. Dette skyldes at GIS-scener primært foregår i 1 plan, også selvom det er 3D. Det er derfor lettere og giver bedst performance, hvis man begrænser sig til 2 dimensioner. Jeg vælger derfor at implementere et simpelt quad tree [PICCO], med *lazy balacing*. Træet er opbygget efter princippet på Figur 64. Hvert blad er her en firkant (quad), der kan opdeles i 4 lige store dele hvis nødvendigt. Træet er *lazy balanced,* fordi en vigtig del af projektet handler om, hurtigt at kunne skifte til 3D-kortene. Da et normalt

træ vil blive *balanced* umiddelbart efter at det er fyldt op, så vil dette være en ekstra faktor til programmets opstart. Jeg vælger derfor at lade træet pakke sig selv ud, når det får brug for det. Derudover skal det kun udpakke de dele ud af træet, som den pågældende søgning berører. På den måde vil man få en smule længere søgninger, de første par gange man søger i et nyt område. Til gengæld vil træet ikke bruge tid til *balancering*.
Quad-træet er dog grundet dets symmetriske quad-opdelings-metode ikke optimalt. Bedre performance kan opnås ved bl.a. R-trees [BECKMANN & KRIEGEL & SCHNEIDER & SEEGER], der ikke bruger reel opdeling, men blot laver et hierarki af bounding boxes. Jeg har dog valgt quad-træet til at starte med, grundet dets simple implementation.

6.6.7 Load performance

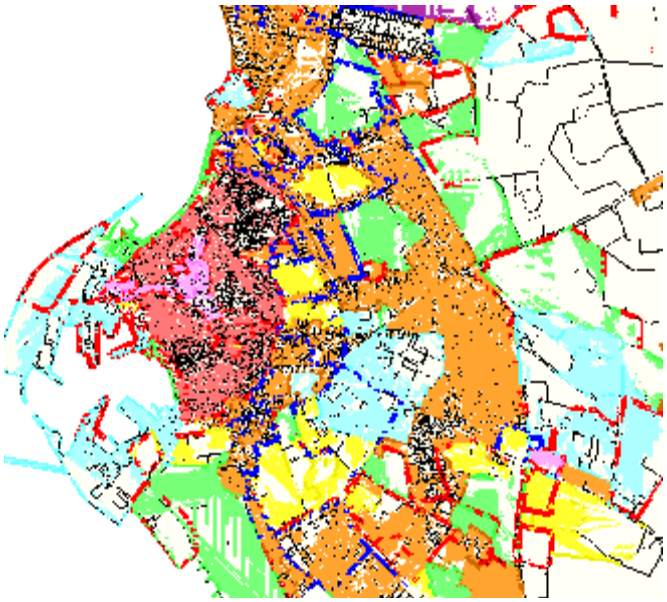
For at projektet skal være en succes i praksis, så skal performance være i orden i alle henseender. En af projektets akilleshæle i denne forbindelse, er selve konverteringen af 2D-data til 3D. Programmet er nødt til ved indload af nye kort, at udføre en række algoritmer, for at data passer ned en struktur der kan tegnes af 3D-rendere. Derudover skal data også tilpasses, så det får en reel z-akse. Begge dele tager tid under indload, hvilket kan give unaturlige pauser i arbejdsrytmen, hvis der er tale om store eller mange kort. For at forbedre denne loadtid, har jeg valgt at cache de beregnede data, så disse ikke behøves at blive genberegnet, ved hver eneste load. Problemet ved dette, er naturligvis at det cachede data skal genskabes, hver gang det oprindelige kort bliver ændret. Der kan derfor opstå en situation, hvor selve cachingen kommer forværre performance, hvis det oprindelige kort bliver ændret hyppigt. Erfaring fra vores kunder viser dog, at fordelene ved caching af beregnet data, langt overstiger de gener det måtte medføre, selv i dynamiske kort der reelt ikke er egnet for caching. Derudover viser det sig også, at det tid/resurser som bruges på at gemme et kort i det binære format, er minimalt og ofte ikke bemærkes.
Programmet indeholder derfor funktionalitet til at gemme og hente loadet data, i et binært format, med en protokol der er optimeret til programmets datastrukturer.

6.7 *Artifacts*

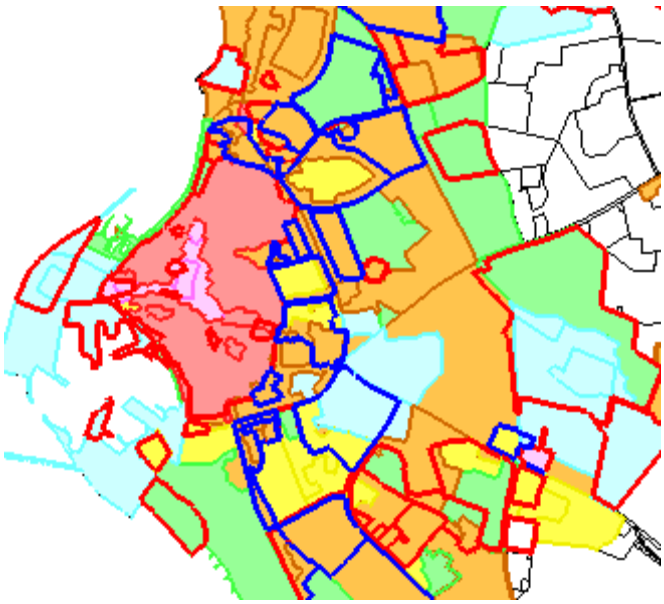
Undervejs i implementationen støder man på flere former for *artifacts* i renderingen. Disse er ofte specifikke for den givne platform, framework eller hardware. Fejlene har ingen direkte relevans for projektet, ud over at de er nødvendige at løse, for at opnå vores resultat.

6.7.1 Upræcision i dybde-bufferen

Efter implementeringen af flere lag i samme kortvindue, fremkom store områder med grafikfejl ved ikke-ortogonale synsvinkler.
På Figur 66 ses et kortudsnit over Rønne. Kortet indeholder et matrikel-, et kommuneplan- og et lokalplan-lag i den angivne rækkefølge og er positioneret ortogonalt på øje-linien. På Figur 65 ses det samme kort, men er her en smule tippet i forhold til øje-linien. Her ses det, at de 3 lag nu bliver blandet sammen på tilfældig vis. Dette skyldes at grafikkortets opfattelse af de tre lags dybde (afstand fra øjet) ikke er præcis nok, numerisk set.
Denne dybde-præcision kan forøges, ved at formindske scenes dybde-spænd. Dybdebufferen er dog ikke lineær, så den største forbedring fås ved at sætte *near clip* værdien så tæt på scenens egentlige interesse som muligt, da det er de "forreste" værdier der har størst præcision.



Figur 65: Kort med grafikfejl. Læg mærke til at de underliggende matrikler, kan ses igennem de øvre lag.



Figur 66: Kort uden grafikfejl.

En anden upræcision i dybdebufferen, kan komme til udtryk i forbindelse med coplanar polygoner. Hvilket er et hyppigt scenario i projektet, da vektor-repræsentation af GIS-data, netop er opbygget af *coplanare* lag. Løsningen på dette er en smule mindre dybsindig, da man her er nødt til at ty til explicit styring af grafikkortet. OpenGL indeholder funktioner, der kan fortælle kortet, at de efterfølgende polygoner har et givent *offset* i forhold til de forrige. Derved undgår man at afrundingsfejl på verticerne, giver *bleed-through* artifacts, da man explicit har angivet rækkefølgen på polygonerne. En sidste metode jeg har måttet ty til er simpelthen, ikke at tegne de steder hvor der er flere lag oven på hinanden. Altså at undlade at tegne de underliggende lag. Dette gøres via *stencil-bufferen*.

7 Implementation

Det er i implementationsdelen at projektets fokus har været placeret. Generelt har jeg forsøgt, at lave så meget egen-implementation som muligt, da genbrug af eksisterende kode, i bedste fald er problematisk, i forbindelse med kommerciel videresalg. Bemærk dog følgende undtagelser:

- Mitab [MITAB] er et opensource projekt og er en del af et større projekt [GDAL]. .NET-wrapperen til denne kommer ligeledes fra samme projekt, men er dog blevet modificeret.
- Triangulereringen [RATCLIFF] er hentet fra FlipCode og er lavet af John W. Ratcliff. Koden er dog blevet porteret og modificeret.

7.1 Kode

Koden til projektet er vedlagt som bilag, sektion 2.

8 Test

Følgende afsnit vil beskæftige sig med test og verifikation af projekts resultat.
Projektets test og verifikation består i en kort performancetest, hvori projektet vil blive målt og sammenlignet mod MapInfo og i nogle bruger-interviews, hvori nogle GIS-relaterede personer vil give deres besyv.

8.1 Performance

8.1.1 Testspecifikation

Programmernes opstartstider er decideret målbare og giver et billede af projektets potentielle succes. Følgende tests vil derfor blive udført:

- Mål af opstartstid for MapInfo, med givent arbejdsområde
- Mål af opstart for GIS3D, med samme arbejdsområde, uden brug af cachet data
- Mål af opstart for GIS3D, med samme arbejdsområde, med brug af cachet data
- Mål af opstart for GIS3D i light-udgave. (Opstart fra MapInfo) Hvilket vil tvinge programmet til at loade alle lag.

Datasættet (arbejdsområdet) der vil blive brugt, vil med vilje blive valgt, ud fra mængden af data. Datasættet vil indeholde lidt flere data, end hvad der er normalt at bruge og på den måde give et bud på et worst case scenario.
Udover opstarts-performance, opererer de fleste 3D-programmer også med en målbar FPS-faktor. Denne er dog ikke tilgængelig for MapInfo, da denne ikke har nogen form for animation. Den har dog stadigvæk en vis navigationsperformance og denne vil derfor blive evalueret i afsnit 8.2.
Derudover er følgende processer, ikke en del af opstartsfasen:

- Interpolation
- Goroud shading

Deres afviklingstid vil derfor blive målt separat, men med samme datasæt, som i ovenstående test.
Afstandsudregningerne fra afsnit 6.6.1 vil derudover også blive opmålt mod hinanden. Dette vil blive gjort via et virkelighedsrelateret datasæt, samt summering af 100 gennemløb af hver punktlængde målt til midten af datasættet. Følgende udregninger vil blive målt:

- Euclid
- Kvadreret Euclid
- Raphael-approksimation
- 1-norm
- Approksimeret via arbitrær relation
- Approksimeret via 2/1-relation
- Approksimeret via 3/1-relation

For hver udregning vil følgende blive målt:

- Gennemsnitlig fejl-procent
- Maksimale fejl-procent
- Udregningstid for 100 iterationer

Alle tests vil blive udført på en 1,7 MHz Intel Centrio, 512 Mb DDR, 256 Mb ATI 9700.

8.1.2 Testresultater

Det brugte datasæt indeholder følgende antal vertices: 881730

Opstartstider	
MapInfo Professionel	Ca. 4-10 s
GIS3D, med cachet data	Ca. 2-17 s
GIS3D, uden cachet data	Ca. 25-30 s
GIS3D, via MapInfo (load alle lag)	Ca. 9-18 s

Afviklingstider	
Interpolation	Ca. 15 min
Goroud shading	> 1 t

Afstandsudregninger				
Funktion	**Avg. fejl**	**Max fejl**	**Tid (ms)**	**Index**
Euclid	0	0	2514	1
Kv. Euclid	5229,74	11713,88	1622	0,65
Raphael	0,01849877	0,03790269	2233	0,89
1-norm	0,221543	0,4142136	1612	0,64
Aprox. relation	0,07693624	0,5492448	4917	1,96
2/1-relation	0,1073428	0,2546391	1642	0,65
3/1-relation	0,09341341	0,2094254	1643	0,65

8.1.3 Testkonklusion

Af opstartstiderne kan man se, at GIS3D generelt er langsommere end MapInfo til at starte op. Dog afhænger dette meget af .NET-programmets givne kompileringsniveau. I situationen hvor .NET-programmet bliver afviklet for 2.-3. gang, med cachet data, starter det hurtigere op end den tilsvarende MapInfo. Dette skyldes at .NET-programmerne bliver kompileret on demand (JIT) og bliver derudover også bliver optimeret under runtime. Dette giver langsommere opstart første gang programmet startes, men kan også, som det ses, give et boost i de efterfølgende opstarter.
Testen med ikke-cachet data viser, at den allerførste opstart, hvor ingen kort før er blevet loadet med programmet, kan være en tålmodighedskrævende affære. Undersøger man programmets opførelse, vil man se at den tidskrævende proces ligger i selve load'en af MapInfo-filerne. Altså ikke i konverteringen eller behandlingen af data. Dette kunne skyldes at mitab-biblioteket ikke var specielt optimeret. Dette kan Geograf dog afkræfte, da de via nogle C++-tests, har kunnet konstatere loadtider, svarende til dem man ser med de cachede data. Flaskehalsen ligger derfor muligvis i den *marshaling* der foregår mellem *managed* og *ummanaged* kode, eller muligvis i noget .NET *boxing*. Begge dele vil ikke være urealistisk at tilrette, til at opnå samme resultat som det der kan fås med C++.

Den sidste test, med opstart fra MapInfo og startet i et lavt zoom-niveau, viser at processen der kompilerer de interne OpenGL-grafiklister og processen der overfører disse til grafikkortet (de to processer kan afvikles separat), ikke er uden betydning. I situationen hvor programmet opstartes fra MapInfo, kan denne ekstra ventetid endda gøre brugeren lidt utålmodig. Begge tidskrævende processer kunne sandsynligvis reduceres kraftigt, eller helt afskaffes, hvis programmet i stedet for grafiklister, havde implementeret Frustum Culling.
Af de to ekstra afviklingstests, kan ses at uoptimeret lineær interpolation er realistisk for det givne dataset. Processen er ikke ment til at blive afviklet ofte og 15 min for alle lag i arbejdsområdet, er derfor acceptabelt. Afviklingstiden for Gouraud shadingen er derimod ikke acceptabel. Processen nåede aldrig at blive afviklet til ende. Nærmere undersøgelser viste at flaskehalsen lå i *halfedge*-trianguleringens uoptimerede "find indeholdende trekant"-funktionalitet, samt den uoptimerede Delaunay-triangulering. Det nuværende stadie for Goroud shadingen, er dermed ikke brugbart.
Ud fra afstandsberegnings-resultaterne, ses det at Raphael-approksimationen har den mindste fejlprocent og derudover også er hurtigere at udregne end klassisk Euclid, om end kun 11% hurtigere. Hvis det skal være endnu hurtigere, så scorer 3/1-relation-beregningen et index på 0,65 i forhold til Euclid og har derudover en gennemsnitlig fejl på 9,3%. Testens andre funktioner har alle enten højere fejlprocenter eller er langsommere end Euclid. 3/1-relationsberegningen har dog en maxfejl-procent på ca. 21% og derfor vil jeg ikke bedømme denne udregning, som værende brugbar. Raphael-approksimationen derimod giver kun en maxfejl på ca. 4% og denne vil derfor, afhængigt af situationen være anvendelig. Dog er en konstant forbedring på 11% ikke videre imponerende og en løsning med kvadreret Euclid til sammenligningsformål (eks. afstand1 > afstand2) og almindelig Euclid i andre sammenhæng, vil derfor stadigvæk være at foretrække.

8.2 Useablility

8.2.1 Testspecifikation

For at evaluere programmets egentlige succes, har jeg valgt at lade det afprøve, af 3 udvalgte GIS-brugere. De 3 brugere har jeg udvalgt ud fra deres professionelle GIS-niveau og jeg har med vilje valgt, at bruge GIS-folk, hvis kundskaber strækker sig udover MapInfo og projektets egentlige målgruppe. Med dette håber jeg at kunne få en objektiv evaluering, baseret på GIS og på GIS-programmer i almindelighed.
De 3 personer i testen er følgende:

- Hasse Hauch, Geograf, GIS-konsulent ansat ved Geograf A/S
- Sussi Larsen, Digitalisør, Projektassistent ansat ved DONG Energy
- Carina Høj, Agronom- og GIS-studerende ved KU, tidligere KVL.

Testen vil foregå ved, at jeg giver en præsentation af programmet, hvorefter personen selv får lov at afprøve programmet. Herefter vil jeg interviewe personen med følgende spørgsmål, som udgangspunkt:

Spørgsmål	Uddybning
Lang opstart	Har programmet en for lang opstart eller kan man leve med det?
Navigations-hastighed	Er projektets navigationshastighed bedre eller værre end MapInfo og er den tilfredsstillende?
Er det acceptabelt med gættede tagrygge o.l.	Ville det være acceptabelt hvis programmet gav et gæt på hvorvidt et hus havde et skråtag eller ej?
Kan det viste 3D erstatte 2D	Kan 2D noget, som der ikke vil kunne laves via det viste 3D?
Kan GIS3D det den skal kunne	Programmet er ment som supplement til MapInfo og til brug for almindelige GIS-brugere. Kan den udfylde dette?
Kan GIS3D erstatte MapInfo	Kunne GIS3D blive en konkurrent til MapInfo, hvis det blev udbygget?
Hvilke funktioner mangler	Hvilke funktioner mangler før programmet kan udfylde sin rolle og hvad mangler før det kan erstatte MapInfo?
Er der brug for GIS3D	Er programmet brugbart eller en forbedring på nogen måder?
Kan tematisering i virkeligheden klare det	Kan højde- og slagskygge-tematisering i virkeligheden klare sagen? Og vil det være en bedre eller dårligere løsning i så fald?
Sammenligning med andre	Kan andre programmer som f.eks. Arc i virkeligheden allerede løse opgaven bedre end GIS3D.
Fremtiden	Fremtiden ser i øjeblikket ud til at gå mod internetbaserede kort. Er GIS3D brugbar for superbrugere eller vil det være brugbart som en WFS-service i stedet? Er 3D brugbart for ikke-GIS-folk?
Andet	Er det andet at tilføje?

8.2.2 Testresultater

Udtalelserne fra testpersonerne, er præsenteret som stikord og korte sætninger. Dette skyldes at der ikke blev brugt diktafon eller lignende hjælpemidler ved interviewene og at integriteten derved kun tillader, det faktisk noterede.

8.2.2.1 Udtalelser fra Carina Høj

Lang opstart – *Opstart via MapInfo er tålelig. Den må dog ikke være længere. Opstarten som standalone-program er god.*
Navigations-hastighed – *Den er fin nok. Ikke forstyrrende. Det samme som Arc.*
Er det acceptabelt med gættede tagrygge o.l. – *Hvis man kunne vælge at slå det fra og til, så ja. Men hvis jeg skal vælge, vil jeg sige nej.*
Kan det viste 3D erstatte 2D – *Ja.*
Kan GIS3D det den skal kunne – *Ja, hvis det er ment som supplement til MapInfo, så vil jeg mene det.*
Kan GIS3D erstatte MapInfo/Arc – *Ja, hvis der blev ændret på nogle ting*
Hvilke funktioner mangler der – *Jeg kunne godt tænke mig, hvis lagkontrolen var en venstre-bar, ligesom i Arc. Arc havde også en masse værktøjer og en værktøjskasse. Og der mangler noget funktionalitet til udprint og layout.*
Er der brug for 3D – *Kommer an på situationen. Nogle situationer, ja. I andre er 3D ikke en fordel.*
Kan tematisering i virkeligheden klare det – *Til nogle ting måske, men det er rart at man kan dreje kortet.*
Er GIS3D/3D brugbart nok, til at ytre ønske om tilkøb af denne funktionalitet – *Ja, men jeg er ikke den der laver den slags beslutninger, så jeg ved det ikke.*
Sammenligning med andre – *Det virkede nogenlunde som det der er i Arc. Her skulle man dog åbne specielle 3D-kort.*
Fremtiden – *3D er nok ikke brugbart for ikke-GIS-folk. Det er nok forvirrende. Men hvis man sidder med bygninger o.l., så er det.*

8.2.2.2 Udtalelser fra Hasse Hauch

Lang opstart – *Man kan sagtens leve med opstartstiden. Ingen problemer.*
Navigations-hastighed – *Heller ikke nogen problemer her. Den er faktisk hurtigere til at opdatere kortet, end MapInfo.*
Er det acceptabelt med ”gættede” tagrygge o.l. – *Ja, konstruerede tagrygge vil være bedre end ingenting. Men kunne evt. lave en kontrol ved at beregne skygger og sammenligne skyggerne med skygger fra et ortofoto. Hvis skyggerne er ens vil de konstruerede tagrygge nok være korrekte.*
Kan det viste 3D erstatte 2D – *Jeg er i tvivl om 3D kan erstatte 2D, pga. manglende hastighed og funktioner i 3D, men med det forbehold vil jeg sige JA.*
Kan GIS3D det den skal kunne – *Den mangler nogle funktioner til udprint og til at gemme kamera-positionen og hente den igen. Jeg kunne godt tænke mig hvis man kunne dreje rundt om center-punktet. Hvis programmet skal kunne erstatte MapInfo, så skal der være mulighed for analyser.*
Kan GIS3D erstatte MapInfo/Arc – *Ja, hvis der kommer analyse-funktionalitet og udprint-funktionalitet.*
Er kortet ”nok” 3D – *Nej, ikke helt. Der mangler bl.a. nogle tagrygge mm. Det ser lidt stilistisk ud lige nu.*
Er der brug for 3D – *Ja, der er stor brug for 3D. F.eks. til præsentation for almindelige mennesker, men også til brug i sagsbehandling.*

Kan tematisering i virkeligheden klare det – *Nej, overhovedet ikke. Hvis man presser ekstra ting som skyggepolygoner og højder ind på et almindeligt kort, så kommer der alt for meget information. Man er nødt til at tage den 3. dimension i brug, hvis man vil bevare overblik.*
Er GIS3D/3D brugbart nok, til at ytre ønske om tilkøb af denne funktionalitet – *Ja, hvis manglerne blev lavet.*
Sammenligning med andre – *Til MapInfo findes der jo Vertical Mapper. Men her skal man dog bruge et grid. Og det er heller ikke særligt brugervenligt. Og man kan heller ikke navigere i tingene på samme måde og det er langsommere.*
Fremtiden – *Der er stor fremtid i 3D. Det ER fremtiden. Udviklingen går mod web-løsninger, men her vil der også være brug for 3D. Og superbrugerne har også brug for den.*
Andet – *Tilgangen med at interpolere direkte på koordinaterne er ret interessant. Ens x- og y-koordinater bliver ikke mappet ud på et grid, så det er stadigvæk de rigtige data.*

8.2.2.3 Udtalelser fra Sussi Larsen

Lang opstart – *Nej, det er ikke noget. Mine egne programmer er meget værre.*
Navigations-hastighed – *Det har jeg ikke bemærket. Der er fin navigationshastighed.*
Er det acceptabelt med "gættede" tagrygge o.l. – *Hvis jeg skal vælge, vil jeg sige, helst ikke. Jeg synes ikke at det gør noget, at der ikke er tagrygge.*
Kan det viste 3D erstatte 2D – *Ja, jeg kan ikke se, hvorfor det ikke skulle kunne det.*
Kan GIS3D det den skal kunne – *Ja, det vil jeg mene. Jeg har ikke fantasi der at forestille mig, ting den mangler.*
Kan GIS3D erstatte MapInfo/Arc – *Grafisk set, ja.*
Hvilke funktioner mangler – *Ja, der jo udprint-funktionaliteten og tematiseringen.*
Er der brug for 3D – *Det synes jeg.*
Kan tematisering i virkeligheden klare det – *Man kan godt bruge tematiseringen til noget af det, men det bliver aldrig helt godt.*
Er GIS3D/3D brugbart nok, til at ytre ønske om tilkøb af denne funktionalitet – *Ja, det vil jeg mene. Bl.a. har vi ledningskort, med store mængder af information, hvor deres indbyrdes forhold er meget vigtige. Her ville vi nok være parate til at købe, uanset pris.*
Sammenligning med andre – *Jeg har desværre aldrig arbejdet rigtigt med 3D. Jeg har dog brugt mange 2½D kort. Men jeg kender ikke rigtigt nogle jeg kan sammenligne med.*
Fremtiden – *Almindelige folk vil nok synes at 3D er spændende. Det skal faktisk helst være 3D, hos almindelige folk.*
Andet – *3D ville være smart til at lave kvalitetssikring af det digitaliserede.*

8.2.3 Testkonklusion

Ingen af testpersonerne var afskrækkede af opstartstiderne i programmet. De udtalte at ventetiden var fin og at man for øvrigt ikke lagde mærke til den.

Kun i situationen hvor programmet startede op via MapInfo og startede i et allerede indzoomet kort, bemærkede de den. Heller ikke her, mente de at det var et problem. I denne forbindelse skal det måske bemærkes, at testes 3 personerne netop er superbrugere indenfor CAD og GIS og at de derfor sandsynligvis er vant til, at arbejde med tunge programmer. Det kan derfor tænkes, at de besidder mere tålmodighed end almindelige brugere.
Med hensyn til programmets navigationshastighed, så var alle 3 personer lidt uforstående overfor spørgsmålet i starten. Jeg måtte derfor vise dem, at man i navigations-situationer kunne se at scenen "hakkede". (FPS-værdier på under 40, giver hakkende bevægelser i 3D-rendering.) Dette var de dog alle ret upåvirkede af og mente at det var ret naturligt. Geografen var derudover tilfreds med, en ifølge ham, hurtig opdatering af kortet. (Flere 2D-kort-viewer bruger tid på at hente og rendere ekstra kort, ved f.eks. panorering af kortet.)
Til spørgsmålet om hvorvidt "gættede" tagrygge o.l. er acceptabelt, var testpersonerne lidt uenige. De var alle enige om, at muligheden for fiktive tagrygge var interessant, men generelt var holdningen dog, at man godt kan undvære dem.
Alle testpersonerne var derudover enige om, at det viste 3D godt kunne erstatte 2D. Ingen af personerne kunne komme på noget, som 2D kan og som ikke også ville kunne laves i det viste program.
I forbindelse med, om programmet kunne det skulle, forklarede jeg at programmet, i hvert fald til at starte med, var ment som et supplement til MapInfo og jeg spurte dem derfor om det da kunne det den skulle. Her bemærkede personerne, at det manglede nogle analyse-værktøjer. Geografen fokuserede bl.a. hvorvidt programmet var ment til præsentation eller til analyse. I denne forbindelse forklarede jeg dem derfor, om programmets plugin-system og deraf mulighed for sideløbende opbygning af værktøjskasse ala dem i ArcGIS og MapInfo. I denne forbindelse viste jeg dem også Adresse- og GISoversigt-pluginet, hvilket de var tilfredse med. Efter noget granskning konkluderede testpersonerne dog, at der burde være noget funktionalitet til udskrivning og dettes layout. Derudover mente Geografen også at der burde være noget funktionalitet til at gemme og hente den præcise situation, mht. kamera-vinkel, lyssætning o.s.v. Han kunne derudover også godt tænke sig at der havde været en navigationstype der kunne dreje kortet. (Han havde aldrig været udsat for Orbit-navigationen før og var ikke helt imponeret over den.)
Til spørgsmålet om hvorvidt programmet kunne erstatte eks. MapInfo, svarede alle personer ja til, på forudsætning af, at der kom udprint-funktionalitet og at der blev opbygget flere analyse-værktøjer.
Til spørgsmålet om, hvorvidt der i det hele taget er brug for 3D i GIS, eller om det blot er en gimmick, var den aggronom-studerende lidt neutral og udtrykte, at det jo kom an på situationen, men at der i nogen situationer var brug for 3D. Geografen var mere overbevist og svarede derfor med et rungende "ja" og digitalisøren var her lidt uforstående overfor spørgsmålet, som hun mente var en selvfølge.
Til spørgsmålet på om, hvorvidt tematisering i virkeligheden kan klare 3D-situationerne, var den aggronom-studerende igen lidt neutral og fortalte at

hun da havde lavet skyggeberegninger i 2D. Dog mente hun ikke at tematisering kunne være en erstatning for 3D. Igen var Geografen mere bestemt på dette punkt og mente ikke, at tematisering kunne være løsningen. Han nævnte bl.a. at informationsmængden ville blive for sammenpresset, hvis man ikke tog den 3. dimension i brug.
Spørgsmålet om hvorvidt testpersonerne mente at programmet var brugbart nok, til at ytre ønske om deciderede tilkøb af programmet, blev stillet i håb om at belyse, hvor brugbart de mente tingene var. Blot fordi personer er interesserede i, eller venligt stillede, overfor nye teknologier, så betyder det ikke at de er nok imponerede, til rent faktisk at ville betale penge for det. Spørgsmålet skulle derfor forsøge at afklare, om personerne i virkeligheden blot var høflige i besvarelsen af spørgsmålene.
Spørgsmålet havde dog ikke helt den grublende effekt på testpersonerne, som jeg havde håbet på. Den aggronom-studerende mente ikke, at hun havde forstand på den slags og Geografen og Digitalisøren, der begge havde et lidt nonchalant forhold til dette, mente at man sagtens ville købe/sælge det, hvis de få mangler blev bragt i orden.
Af sammenligning med andre programmer, kunne personerne bl.a. fortælle om udvidelser til ArcGIS og MapInfo. I disse fortalte personerne dog om, at de skulle åbne specielle 3D-kort, før det virkede. Graden af performance og brugervenlighed i disse imponerede generelt heller ikke testpersonerne.
Med hensyn til fremtiden, så var alle testpersonerne enige om, at 3D er fremtiden indenfor GIS. Men med hensyn til GIS-brug til almindelige personer, var de lidt uenige. Den aggronom-studerende holdt på, at ikke-GIS-folk nok vil finde 3D uoverskueligt i forhold til 2D. Geografen derimod mente at 3D netop var en fordel, til GIS-præsentation for ikke-GIS-folk. Derudover var alle testpersonerne fascinerede over, at kortets ledninger og undergrundsobjekter var synlige, når man drejede kortet på hovedet. Og de var også generelt interesseret i projekts tilgang til 3D-generering af kortene, der afviger fra grid-løsningerne, som de ellers var vant til.

9 Konklusion

Første del af projektet er nu færdiggjort. Projektet vil blive fuldt op efterfølgende, hvis resultaterne bedømmes brugbare erhvervsmæssigt set. Projektet har opnået, at implementere et program, der efter den nævnte 2-trins-model, kan konvertere 2D-MapInfo-data til 3D-GIS-data og visualisere det.
Implementationen er lavet via en let tilgængelig programmerings-platform og via relativt hurtigt implementerbare algoritmer, hvilket på den måde har gjort det muligt at opbygge et program, der med få undtagelser, kan matche det veletablerede MapInfo Professionel. Trods de simpelt implementerede algoritmer og platform, viser en undersøgelse, lavet ud fra interviews af udvalgte erfarne GIS-folk, at det resulterende program er fuldt ud funktionelt performance- og erhverv-mæssigt set, dog med forbehold for de ikke-implementerede funktioner. Programmet er derudover i stand til, ifølge de interviewede personer, på tilfredsstillende vis at konvertere og visualisere 2D-MapInfo-data, til tilsvarende LOD1-niveau [CITYGML] 3D-GIS-data.

Programmet er derudover i stand til, at foretage flere arbejdsopgaver, der er umulige at udføre i praksis, i MapInfo og i 2D-GIS generelt. Eksempler på programmets formåen indenfor disse, kan bl.a. ses i afsnit 4.2, hvor der er indsat flere screenshots fra programmet.
Til sidst har projektet også resulteret, i en række stabile kode-biblioteker og algoritmer, til videre brug og udvikling i erhvervsmæssige sammenhæng. Algoritmer og kode hvis natur, ikke er naturligt forekommende traditionelt 2D-GIS og derfor er værdifuldt for disse virksomheder.
Undervejs i projektets forløb, har jeg fundet frem til flere andre projekter, der arbejder med noget af det samme. Bl.a. har interviewene nævnt programmet Vertical Mapper [NORTHWOOD], der på samme måde, som dette projekts resulterende program, er en udvidelse til MapInfo. Dog fokuserer Vertical Mapper ikke på, at konvertere og visualisere 2D-kort i 3D, på samme måde som dette projekt. I stedet fokuserer [NORTHWOOD] på generering af en komplet DTM, opbygget via et grid, med efterfølgende analyse af dette. Vertical Mapper giver mulighed for 3D-renderinger af DTM'en, men visualiserer generelt ikke kortene i 3D, ved almindelig brug. Til gengæld vil Vertical Mapper af disse grunde, muligvis kunne benyttes til generering af mere præcise, eller blot andre typer af DTM'er til brug i projektets program.
Af andre værker kan bl.a. nævnes ArcGIS, der også bliver nævnt i interviewene. Dette fokuserer dog, ligesom Vertical Mapper og programmerne der listes i [LO & YEUNG2], ligeledes mest på terræn og analyser af dette. Et andet projekt der mere omhandler det samme som dette projekt, er derfor f.eks. [KOPPERS]. Koppers fokuserer på generering af 3D-objekter (bymodel-objekter). Modsat dette projekt, så fokuserer han dog mest på bygninger og træer og skaber på den måde, ikke en konvertering af et komplet kort.
Et andet produkt man også kan sammenligne det med, er Google Earth [GOOGLE]. Dette er et program, lavet af Google, der pt. indeholder et orthofoto-kort, samt DTM, vejnavne og bynavne over det meste af jorden, hvis ikke hele. Ligesom i projekts program, har man i Google Earth, muligheden for at se kortet i 3D-synsvinkel. Derudover findes der også allerede en række bymodel-objekter i kortet, der er digitaliseret af brugerne. Ifølge specifikationerne vil brugerne kunne digitalisere objekter, i helt op til LOD4-præcision. På Figur 67 kan ses en digitaliseret model af Øresundskollegiet i LOD2. Google Earth beskæftiger sig ikke med konvertering af eksisterende kort til 3D, eller i det hele taget med GIS, som arbejdsværktøj. Google Earth er mest ment, som en geografisk søgemaskine. Dog kan man købe professionelle udgaver af både Google Earth og dettes digitaliseringsværktøj, som der netop er ment til at give professionelle brugere flere muligheder. Ifølge specifikationerne indeholder disse dog stadigvæk ikke muligheder for ting, som plugins, tematisering, analyser, fjernelse af orthofotoet og import af 3D-modeller fra andre kilder, end deres eget digitaliseringsprogram. Hvilket dog er ting, der hurtigt ville kunne tilføjes, hvis Google ønskede det. Med disse ville Google Earth kunne blive til et reelt alternativ til både dette projekts visualiseringsdel, men også til GIS-løsninger som MapInfo og Arc i almindelighed.

Figur 67: Øresundskollegiet i Google Earth

Projektet har på nuværende stadie, stadigvæk en række mangler og ukomplette implementationer. Eksempelvis så nævner de interviewede personer de to åbenlyse mangler:

- Tematisering/analyser
- Udprint

Hvis projektet skulle have erhvervsmæssigt interesse, skulle der først arbejdes med disse to emner. I forbindelse med udprint-funktionaliteten, ville et oplagt ikke-trivielt emne, bl.a. også være *antialiasing* af de visualiserede 3D-kort. Af andre punkter der ligeledes også har været nævnt tidligere i rapporten, kan nævnes:

- Interpolations-metoden: Som Vertical Mapper også illustrerer, så er Lineær-interpolation ikke altid bedste løsning. Mulighed for andre interpoleringer, som bl.a. Kriging [NIELSEN], ville være at foretrække.
- Subdevision: Datasæt'enes større flader lider af, at de kun har omkransende vertices. Dette resulterer bl.a. i at lokale minima og maxima ikke illustreres, med mindre andre overliggende flader, er til stede. Strategisk indsættelse af ekstra vertices, ville derfor være at foretrække.
- Volumetriske skygger: Skyggerne i projektet er et fantastisk værktøj til brugerne. De implementerede projektionsskygger er dog ikke fyldestgørende og volume shadows ville derfor være at foretrække.
- Flere kort-formater: Projektet bruger på nuværende tidspunkt [MITAB] til at læse MapInfo-kort med. [MITAB] er dog et subset af [GDAL] og det vil derfor umiddelbart være nemt at skifte til dette og på den måde, komme til at understøtte alle GDAL's understøttede formater.
- Culling: Undervejs i projektet, har den valgte tilgang via kompilede grafiklister, vist sig værende uheldig. På trods af, at de interviewede personer alle var fint tilfredse med den grafiske hastighed, så har

programmet alvorlige rendering-performance-problemer. At test-personerne er vant til at arbejde med værre programmer, gør ikke tingene bedre. Med den nuværende rendering, vil programmet heller ikke kunne rendere kort, der er større end det testede, med mindre man tager LOD-funktionalitet i brug. Alt dette skyldes programmets manglende culling-optimering.

- CityGML: Et fantastisk tiltag for projektet, ville være at udskifte datastrukturen, til den der defineres af [CITYGML]. Projektet kræver på nuværende tidspunkt kun, at brugerne angiver hvilke lag der indeholder z-information og hvilke der indeholder rummelige objekter. CityGML giver derimod mulighed for specifik klassificering, af alle objektelementer og giver indbygget mulighed for automatisk definering og brug af LOD. En udvidelse af projektet der på en effektiv måde kan hjælpe brugeren til at klassificere og konvertere de nuværende kommunale kort-data til CityGML, ville kunne give store forbedringer for både brugere og programmel. (Om end det grundet den ekstra arbejdsbyrde med klassificering og digitalisering af data, ikke nødvendigvis ville være noget, kommunerne ville hilse velkommen.)

Med hensyn til fremtiden, så er firmaer som Geograf A/S, grundet størrelse og virksomhedssammensætning, ikke nødvendigvis de mest egnede til at gennemføre et nationalt skift, fra traditionel 2D- til 3D-GIS. Heller ikke selvom projektet viser, at skiftet godt kan udføre partielt og over tid. Dog er alle GIS-personer enige om, at 3D-GIS er fremtiden og folk undrer sig generelt over, at vi ikke allerede har det. Myten om at den manglede udbredelse af 3D-GIS, skyldes højde computerkrav, bliver ligeledes afvist projektet. Det er derfor også min overbevisning, at vi indenfor relativ kort tid i Danmark, vil se lignende eller tilsvarende kortløsninger, som dette projekt præsenterer. Dette projekt viser blot, den realistiske mulighed for partiel indførelse af 3D-GIS i etablerede arbejdsmiljøer.

10 Referencer

[STINE & GEYER]
William Stine, Michael Geyer, 2001: Power From The Sun, Chapter 3, http://www.powerfromthesun.net/chapter3/Chapter3Word.htm

[NIELSEN]
Allan Aasbjerg Nielsen - Geostatistik og analyse af spatielle data

[MITAB]
Open Source Geospatial Foundation: MITAB, MapInfo .TAB and .MIF/.MID Read/Write Library, http://mitab.maptools.org/

[OGC]
Open Geospatial Consortium Inc, 2006, version 1.2.0: http://portal.opengeospatial.org/files/?artifact_id=18241

[SCIENCEGL]

ScienceGL: Geo Surface3D PRO, http://www.sciencegl.com/gis_dem/index.html

[GDAL]
Geospatial Data Abstraction Library: http://www.gdal.org/

[RATCLIFF]
John W. Ratcliff, 2000: Efficient Polygon Triangulation, http://www.flipcode.com/cgi-bin/fcarticles.cgi?show=63943

[PICCO]
Dion Picco, 2003: Frustum Culling, http://www.flipcode.com/articles/article_frustumculling.shtml

[BAPTISTE]
Rafael Baptiste, 2003: Fast Approximate Distance Functions, http://www.flipcode.com/articles/article_fastdistance.shtml

[ANGEL]
Edward Angel, Third Edition: Interactive Computer Graphics – A Top-Down Approach Using OpenGL

[ANGEL2]
Edward Angel, Third Edition: Interactive Computer Graphics – A Top-Down Approach Using OpenGL, Chapter 6.5, Polygonal Shading

[GAFFGA]
Stefan Gaffga, 2003: Shadow Projection in OpenGL - A mathematical explanation of implementing shadow projection, http://www.devmaster.net/articles/shadowprojection/

[CITYGML]
Special Interest Group 3D (SIG 3D), 2006: Open Geospatial Consortium Inc - Candidate OpenGIS CityGML Implementation Specification (City Geography Markup Language), https://portal.opengeospatial.org/files/?artifact_id=16675

[BEAM]
Josh Beam, 2004: Tutorial - Stenciled Shadow Volumes in OpenGL, http://www.3ddrome.com/articles/shadowvolumes.php

[ASSARSSON & AKENINE]
Ulf Assarsson, Tomas Akenine-Moller: A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware

[NORTHWOOD]
Northwood Geoscience Ltd., Canada, 1999: Vertical Mapper – Contour modeling & display software – Installation and Tutorials

[HEISE]
Heise Online, 12-03-2007: Französische Nationalversammlung nimmt Ubuntu, http://www.heise.de/newsticker/meldung/86578

[TAO]
The Tao Framework, 2007: http://www.taoframework.com/

[MICROSOFT D3D]

Microsoft, MSDN, 2007: DirectX 9.0 for Managed Code, http://msdn2.microsoft.com/en-us/library/ms804944.aspx

[GEL]
IMM: GEL: GEometric and Linear algebra tools, http://www2.imm.dtu.dk/projects/GEL/GEL-docs/index.html

[OPENGL]
The Khronos Group consoritum, OpenGL Working Group: OpenGL API Documentation Overview, http://www.opengl.org/documentation/

[WIKI]
Wikipedia: Delaunay Triangulation, http://en.wikipedia.org/wiki/Delaunay_triangulation

[GOYVAERTS]
Jan Goyvaerts: Regular-Expressions.info http://www.regular-expressions.info/

[KOPPERS]
Lothar Koppers, 2002: Generierung von Objekten fûr 3D-Stadtmodelle

[LO & YEUNG1]
C.P. Lo, Albert K.W. Yeung, 2007, Chapter 2: Concepts and Techniques in Geographic Information Systems

[LO & YEUNG2]
C.P. Lo, Albert K.W. Yeung, 2007, Chapter 9: Concepts and Techniques in Geographic Information Systems

[LO & YEUNG3]
C.P. Lo, Albert K.W. Yeung, 2007, Chapter 7: Concepts and Techniques in Geographic Information Systems

[PLANDK2]
Servicefællesskabet for Geodata, 2007: PlanDK2, http://www.plansystemdk.dk/Emner/PlanDK2/

[KIM & LEE & LEE]
Kyong-Ho Kim, Kiwon Lee og Jong-Hun Lee: 3D Geographical Analysis within JAVA/VRML-based GIS: "Lantern" Operation, http://www.geocomputation.org/1998/31/gc_31.htm

[GOOGLE]
Google: Google Earth, http://earth.google.com/

[SINNAKAUDAN & GHANI & KIAT]
Shanker Kumar Sinnakaudan, Aminuddin Ab Ghani & Chang Chun Kiat, 2002: Flood Inundation Analysis Using HEC-6 and ArcView GIS 3.2a, http://redac.eng.usm.my/html/publish/2002_10.pdf

[BECKMANN & KRIEGEL & SCHNEIDER & SEEGER]
Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger, 1990: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles+, http://dbs.mathematik.uni-marburg.de/publications/myPapers/1990/BKSS90.pdf

Alle ovenstående URL'er, er testet fungerende den 19. juni 2007.

11 Anerkendelser

Følgende personer og organisationer vil jeg gerne takke, for at have bidraget til projektet:

- Geograf A/S
- Hasse Hauch, Geograf og GIS-konsulent hos Geograf A/S
- Flemming Nissen, Landinspektør og Salgschef hos Geograf A/S
- Carina Høj, Aggronom-studerende ved KVL
- Sussi Larsen, Digitalisør og Projektassistent hos DONG
- Gerhard Joos, Associate Professor ved DTU
- Nikolaj Randorff, Sagsbehandler hos Jammerbugt Kommune
- Allan Aasbjerg Nielsen, Associate Professor, DTU

12 Ordliste

Antialiasing
: Teknik til at minimisere distortion eller grafikfejl grundet pixelering.

Artifact
: Udtryk brugt om grafikfejl.

Axis faced billboards
: Billboard-teknik hvor det indsatte billede er fæstnet til en eller flere akser, hvilket giver mulighed for kameraet, at se billboardet fra den flade vinkel.

Billboard
: Teknik brugt i hardware-3D-rendering, hvor et flat billede indsættes i scenen og agerer 3D-objekt.

Boxing
: .NETs teknik til at konvertere mellem value types og reference types. Boxing giver performance-tab og sker implicit i koden.

Camera faced billboards
: Billboard-teknik hvor det indsatte billede altid vises ortogonalt på øje-retningen.

Collision detection
: Kollisionsberegninger og beregninger indbefattende object intersection, både partiel og fuld mm.

Common dialogs
: En samling af standard-dialoger og kontroller, designet og implmenteret af Microsoft. Bør benyttes af alle programmet af hensyn til brugervenlighed.

Curvature minimizing
: Emne der omtales i forbindelse med triagulering og subdevision, til at minimisere energi-niveauer i objekter.

Fly through
: Kamera-animeret 3D-scene.

GLUT
: Udvidelse til OpenGL, der tilføjer dialog og hjælpe-funktionalitet.

HalfEdge
: Datastruktur bestående af retningsbestemte kanter. En kant i et mesh har derfor to halfedges. En i hver retning.

IntelliSense
: Microsofts navn for den teknik, der forudser og foreslår en programlinies næste element. Eksempelvis ved indtast af punktum e.l.

Lazy balacing
: Brugt i rapporten som udtryk for at søgetræet bliver balanced ved brug og kun i de områder hvor denne bliver brugt.

Managed code
: Microsofts udtryk for kode, der eksekveres af en CLI-kompilant VM.

Manifest
: I rapporten henviser dette til det manifest, der skal indbygges i Windows-programmer, for at disse følger det givne styresystems grafiske versioner af common dialogs og controls.

Marchaling
: Microsofts udtryk for den teknik, der forbinder managed code med unmanged code

Near clip
: Forreste kamera-grænse i hardware-3D-rendering. Mellem kameraet og denne bliver der ikke renderet nogle objekter.

RAD
: Rapid Application Development. Bruges ofte om værktøjer, hvor udvikling kan ske ved at ”tegne og hive” komponenter sammen.

Stencil-buffer
: Ekstra buffer i hardware-3D-rendering, der kan bruges til valgfrie værdier.

SSE – Streaming SIMD Extensions
: Streaming Single instruction, multiple data Extensions. Bruges bl.a. til optimering af instruktioner som ”sqrt” o.l.

Subdevision
: Teknik/emne der beskæftiger sig med opdeling/indsættelse af flere vertices i objekter.

TIN
: Triangulated Irregulary Network. Wireframe mesh.

Unmanaged code
: Microsofts udtryk for kode, der ikke eksekveres af en CLI-kompilant VM.