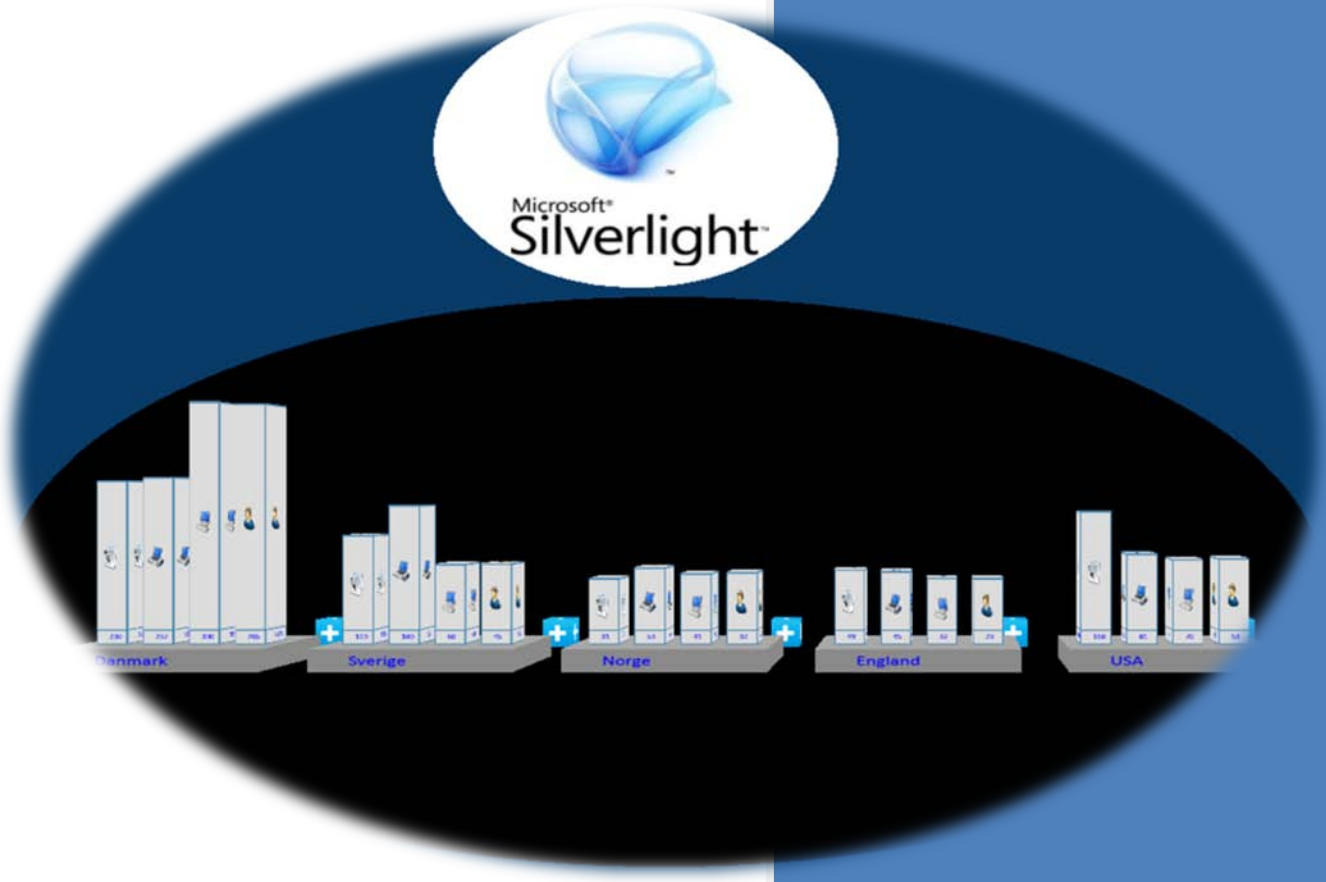
Forretningsgrafik (3D) Silverlight i segmenter



Forfatter:

Allan Sloth Christensen

IMM-B.Eng-2011-72

Vejledere:

DTU:

Bjarne Poulsen

LicenseWatch:

Lars Landbo

Forretningsgrafik (3D) Silverlight i segmenter

Vejledere

Bjarne Poulsen, DTU-IMM

Lars Landbo, LicenseWatch A/S

Studerende

Allan Sloth Christensen s083245

IMM-B.Eng-2011-72

Kgs. Lyngby 2012

Vejleder

Bjarne Poulsen, DTU IMM

Lars Landbo, LicenseWatch A/S

Studerende

Allan Sloth Christensen s083245

Kgs. Lyngby 2012

IMM-B.Eng-2011-72

Summary

LicenseWatch is a company in the software asset management (SAM) industry, where they offer their customers a product, that can help making the work of dealing with a lot of different software, licenses and assets more manageable.

As a help to their clients, LicenseWatch have made it possible for users in their SAM system, to divide their businesses into different segments, so it's easier to get the big picture, of where the company has the most assets or licenses, etc.

The introduction of the new 3D in Silverlight 5, and the lacks on their current segment page, where a regular "grid view" is used for viewing the segments, has made LicenseWatch wanting a 3D prototype of the segment page. This prototype is going to show LicenseWatch, whether 3D can be used to create more value on their segment page, and if Silverlight is a useful tool for this.

The prototype, that has emerged in this thesis, has been designed and implemented, based on a technology survey, which has looked on the general aspects of 3D and how these work in Silverlight, and an analysis of the value that the current segment page give and what value is desired on the new page.

Based on the prototype, LicenseWatch has assessed, that they will continue to work with 3D on their segment page, as they can see, that this gives them a lot of new opportunities and it provide the users with the desired value and functionality on the page. Silverlight as 3D tool, has also shown, that it fulfills the requirements needed for getting the desired segment page.

Resume

LicenseWatch er en virksomhed indenfor software asset management(SAM) industrien, hvor de tilbyder deres kunder et produkt, der kan hjælpe med at gøre arbejdet med en masse forskellige software, licenser og aktiver mere overskuelig.

Som en hjælp til deres kunder, har LicenseWatch, i deres SAM system, gjort det muligt for brugerne at opdele deres virksomheder i forskellige segmenter, sådan at det er nemmere at få det store overblik over, hvor i virksomheden der er flest aktiver eller licenser osv.

Introduktionen af det nye 3D i Silverlight 5, samt manglerne på deres nuværende segment side, hvor et almindelig "grid" bruges til fremvisning af segmenterne, har gjort, at LicenseWatch godt kunne tænke sig en 3D prototype af segment siden. Denne prototype skal vise om 3D kan bruges til at skabe mere værdi på deres segment side, og om Silverlight er et brugbart værkstøj til dette.

Prototypen, der er kommet frem i denne afhandling, er blevet designet og implementeret ud fra en teknologi undersøgelse, hvor der er blevet kigget på de generelle aspekter af 3D, og hvordan disse fungerer i Silverlight, samt en analyse af den værdi, den nuværende segment side giver, og hvilken værdi der ønskes på den nye side.

Ud fra prototypen, har LicenseWatch vurderet, at de vil arbejde videre med 3D på deres segment side, da de kan se, at dette giver dem en masse nye muligheder, og at brugerne får tilføjet den ønskede værdi og funktionalitet på siden. Silverlight som 3D værktøj har også vist, at det opfylder de krav, der skal til, for at få lavet den ønskede side.

Forord

Denne afhandling er udarbejdet ved Institut for Informatik og Matematisk Modellering, på Danmarks Tekniske Universitet, Danmark, i en delvis opfyldelse af kravene for at erhverve diplomingeniøruddannelsen(B.Eng)

Denne afhandling omhandler oprettelsen af en software prototype, der skal fungere, som en "proof of concept ", hvor det nye 3D i Silverlight 5 ønskes undersøgt. Fokus i denne afhandling er derfor på, om det nye 3D i Silverlight kan bruges til at skabe mere værdi for brugerne på segment siden, i det SAM system som LicenseWatch A/S har lavet.

Allan Sloth Christensen

Kongens Lyngby, Januar 2012

Anerkendelse

Der skal lyde stor tak til Bjarne Poulsen for nyttig vejledning gennem hele projektet og involveringen i projektet. Jeg vil også gerne takke virksomheden LicenseWatch A/S for først at have mig i praktik, og dernæst at ville have mig til at lave et projekt i samarbejde med dem. Der er dog især èn person fra LicenseWatch, jeg gerne vil takke, og det er Lars Landbo, der har været min vejleder i virksomheden, og som har været den person fra LicenseWatch, jeg har snakket med, når jeg har været i tvivl om noget, og derved har givet mig gode ideer og relevant kritik.

Derudover skal venner og familie som har rettet, kommenteret og delt deres tanker omkring rapporten og projektet have en stor tak.

Indholdsfortegnelse

Kapitel 1. Introduktion

Til at give læserne af denne rapport en bedre forståelse af projektet, kommer der en kort beskrivelse af baggrund og motivation for projektet, derefter en kort beskrivelse af virksomheden LicenseWatch A/S(LW) og dens produkt, da dette projekt skrives i samarbejde med denne virksomhed, og omhandler dele af dens produkt. Dette skal samtidig være med til, at give læseren forståelse for, på hvilke områder dette projekt er med til at hjælpe LicenseWatch A/S .

Til sidst vil der være en skitsering af projektets vision og beskrivelse, plus en oversigt over strukturen af denne rapport.

Baggrund og motivation

Efter et praktikforløb i virksomheden LicenseWatch A/S, blev jeg enig med dem om, at jeg skulle forsøge at forbedre deres nuværende produkt med 3D, sådan at der kan skabes bedre overblik for brugeren og derved mere værdi til deres system. Da LW er Microsoft partnere, ønskede de, at det blev lavet i Silverlight, og da Microsoft for nyligt har introduceret Silverlight 5 beta og efterfølgende Silverlight 5 RC, hvor noget af det nye er "Built-in 3D graphics support", passede det perfekt til dette projekt.

Licensewatch A/S og dens produkt

Licensewatch A/S er en virksomhed indenfor " Software Asset Management (SAM)" industrien, og er således med til at hjælpe andre virksomheder med at reducere omkostningerne på software licenser og den juridiske risiko. Til dette formål har LW udviklet et SAM system, som hjælper brugerne med at holde styr på softwarelicenser og installeret software på deres virksomheders PC'er samt andre aktiver, som virksomhederne gerne vil have mulighed for at få overblik over. LW produktet henvender sig mest til store virksomheder, som har 300+ PC'er at holde styr på, hvor overskueligheden af softwarelicenser er mangelfuld, og hvor eventuelle bøder kan blive meget store, hvis der ikke er styr på tingene.

Som beskrevet ovenfor er det et SAM produkt, som Licensewatch laver. Dette består af en masse dele, som tilsammen udgør produktet. Der er en "agent", der står for at opsamle data på virksomhedernes pc'er, og som bliver gemt i en sql database. Disse data er bla. software, hardware, filer og brugere. Der er et SPID(Software pattern identification) program, der bruges til

at identificere software strenge. Disse resulter fra SPID kan brugere af SAM produktet så få gavn af, så de selv slipper for at sidde og identificere tusindvis software strenge. Den del af Licensewatch's SAM produkt som brugerne benytter, består af en masse ASP.Net sider, der bruges til at vise alt data på en nem overskuelig måde. De fleste af disse ASP sider indeholder et datagrid, som får data fra databasen.

For dette projekts vedkommende er det segment siden som er vigtig. Det er denne sides data, som Licensewatch har ønske om, at 3D kan være med til at gøre mere overskuelig overfor brugerne. Det er de enkelte virksomheder, der selv opretter de data, som skal vises, da det er dem, der opretter de forskellige segmenter ud fra den struktur, de mener passer bedst til den enkelte virksomhed. Et segment indeholder computere, brugere, software, license, og aktiver. Kunderne bestemmer som sagt selv, hvordan de laver deres segmenter. Det kan f.eks. være efter afdelinger eller efter geografiske aspekter. Sådan som det er nu, kan man på segment siden ikke se antallet af licenser, computere eller nogle af de andre ting segmenterne indeholder, så derfor bliver kunderne nød til at trykke sig ind på hvert enkelt segment, for at få dette overblik. Det, som Licensewatch gerne vil have er, at kunderne, på siden nemt kan se, hvor meget segmenterne indeholder vha. af 3D effekt.

Figur 1.1-Segment side, lavet ud fra geografisk opdeling

Figur 1.1 viser her den nuværende segment side, med et simpelt geografisk eksempel. Her kan man tydelig se, at brugeren af systemet ikke har noget overblik over f.eks. hvilken af de 3 segmenter, der har flest licenser eller software.

For mere information omkring Licesewatch A/S og SAM, se http://www.licensewatch.com/

Vision

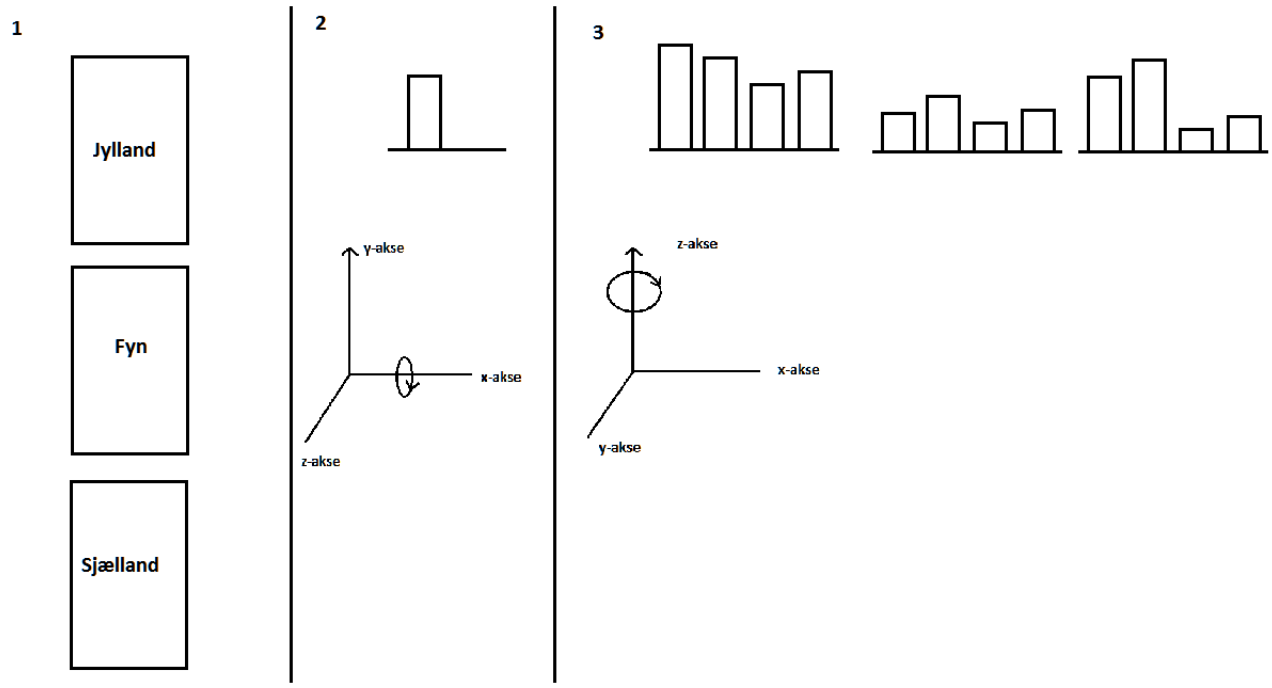
Det er dette projekts vision at lave en "proof of concept", der går ud på at få skabt en 3D scene til visning af segment data for brugere af systemet på en mere overskuelig måde, sådan at Licensewatch efterfølgende kan se, om 3D er noget, som de vil stræbe efter at få indarbejdet i senere udgivelser af deres SAM produkt.

Projekt beskrivelse

Som beskrevet tidligere, så går dette projekt ud på, at Licensewatch A/S har et SAM produkt, hvor virksomheden har ønske om at få 3D effekt ind på en af asp siderne, sådan at det manglende data på den nuværende side bliver vist, samtidig med at det er overskueligt for slutbrugerne.

Siden, som det handler om, er som sagt segment siden. Her har virksomhederne mulighed for at opdele deres virksomhed i forskellige segmenter for bedre overskuelighed. Dette kunne være: Opdeling efter afdelinger eller en geografisk opdeling, hvis virksomheden spreder sig over flere lande, eller det kunne være en økonomisk opdeling. Hvert segment har så tilhørende computere, brugere, licenser, aktiver. Sådan som det er nu, bruges en træstruktur til at vise segment-strukturen for brugerne af systemet, da man har mulighed for at lave undersegmenter. Brugerne kan så trykke på de forskellige segmenter, og få vist hvilke brugere, licenser eller computerer, der hører til de specifikke segmenter. Her kunne LicenseWatch godt tænke sig muligheden for at give brugerne en 3D fornemmelse af størrelsen af segmenterne, f.eks. antallet af licenser eller computere, sådan at brugeren af systemet, ikke bliver nødt til at gå ind på hvert enkelt segment, for at se dette. Dette giver også brugerne mulighedfor at sammenligne størrelsen af segmenterne imellem.

Nedenstående ses en ide til hvordan dette overblik for brugeren kan vises:



Figur 1.2-Ide til at give brugeren overblik vha. 3D

Ideen er her, at der startes med, at der på siden kun ses navnene på de enkelte segmenter, ligesom i det nuværende system, se stadie 1 på billedet. Foretillingen er her, at navnene på segmenterne står på rektangulære platforme, og på den modsatte side er der søjler, der kan bruges til at vise antal eller andre ting. Disse søjler kan brugeren dog ikke se, før der laves nogle rotationer i 3D scenen. I stadie 2 har brugeren roteret ca. 90 grader omkring x-aksen, og derved kan brugeren nu se en enkelt søjle for et enkelt segment. De resterende søjler og segmenter er skjult for brugeren, da de ligger gemt bagved hinanden. Derfor roterer brugeren så omkring z-aksen i stadie 3, som efter rotationen omkring x-aksen nu vender opad, og de resterende søjler og segmenter bliver nu synlige, og brugeren har nu et overblik over de 3 segmenter.

Da dette projekt er en proof of concept, vil implementeringen af en prototype blive lavet som en separat silverlight applikation, med henblik på at skabe en 3D scene, hvor der bliver vist segmenter, og hvor brugeren kan få overblik over licenser og computere osv. uden at skulle trykke sig ind på en ny side. Applikationen kommer til at hente data fra en sql server, hvor test data er blevet udarbejdet i samarbejde med Licensewatch. Grundet det juridiske i udlevering af deres

kunders databaser, har det desværre ikke været muligt for LicensWatch at levere en "virkelig" database fra en af deres kunder.

Hensigten med dette projekt er ikke, at man til sidst står med et produkt, som kan sættes direkte ind i det nuværende SAM system, da LicenseWatch ikke har som krav til deres nuværende kunder, at de har .NET 4.0, der er en nødvendighed for Silverlight 5 , men mere at lave et koncept, som Licensewatch bagefter kan konkludere på, om de vil bruge mere tid og penge på at få udviklet.

For at få en 3D effekt ind på segment siden, vil følgende opgaver blive lavet:

- Undersøgelse af 3D teknologi og herunder 3D i silverlight
- Der laves en undersøgelse omkring overstående ide, med henblik på at finde krav til siden ved bl.a. at få feedback og input fra Licensewatch.
- Designbeskrivelse ud fra de 2 foregående punkter.
- Beskrivelse af implementeringen.
- Udarbejdelse af unit test og bruger test.

Rapport struktur

Kapitel 2.

3D Teknologi

I dette kapitel vil jeg komme ind på hvordan 3D teknologi fungere og herunder, hvordan det fungerer i Silverlight

Kapitel 3.

Undersøgelse af ide til visning af segment siden.

Her vil der blive undersøgt, hvordan man bedst bruger 3D på segment siden, sådan at det bliver overskueligt for slutbrugerne. Dette vil foregå ved at diskutere overstående ide med Licensewatch, som har en stor viden omkring, hvad det er slutbrugerne efterspørger. Alt dette ender ud i, at der bliver opstillet forskellige krav til siden, både funktionelle og grafiske krav.

Kapitel 4.

Design beskrivelse.

Ud fra den fundne information omkring 3D teknologi og analysen af hvordan 3D effekten skal være, laves en design beskrivelse.

Kapitel 5.

Implemantation.

Beskrivelse af den endelige implementation af 3D scenen i silverlight, hvor nogle af de vigtigste metoder bliver beskrevet, samt hvordan de grafiske elementer på siden er opbygget.

Kapitel 6.

Test.

Brugertest af systemet, hvor Licensewatch afprøver og giver feedback, plus en unittest af den ikke grafiske funktionalitet i systemet.

Kapitel 7.

konklusion

En konklusion over projektet, hvor et resume af de forskellige opgaver vil blive opremset, samt evt. fremtidig udvikling af systemet bliver beskrevet.

Kapitel resume af introduktion

LicenseWatch A/S er en virksomhed indenfor SAM industrien, hvor den tilbyder store virksomheder et systemet, der hjælper dem med at få det store overblik over deres virksomhed, når det kommer til software, licenser, computere osv. En feature i LicenseWatchs SAM system er, at virksomhederne har mulighed for, selv at inddele deres virksomhed i forskellige segmenter, sådan at overskueligheden bliver bedre. På den asp. side der står for at vise segment strukturen i en virksomhed, ønsker LicenseWatch at give brugeren et bedre overblik over størrelsen af segmenterne, da dette er en mangel på den nuværende side. Derfor ønsker de at se, om 3D teknologi kan være med til at give brugeren dette overblik.

LicenseWatch er Microsoft partner og det nuværende SAM produkt er lavet på .NET platformen, og derfor ønsker de at se om Silverlight 5, hvor 3D nu er supporteret, kan bruges som udviklings miljø.

Projektet går derfor ud på, først at undersøge 3D teknologi, og derefter hvordan 3D fungerer i det nye Silverlight. Efterfølgende analyseres der på, hvordan 3D teknologi kan bruges til at give brugeren et større overblik over segmenterne. Derefter laves der et design, ud fra den fundne information omkring 3D teknologi og analysen af, hvordan 3D effekten skal være. Til sidst implementeres designet til en prototype, som LicenseWatch kan bruge som baggrund til en evt. fremtidig udvikling af segment siden.

Kapitel 2. 3D teknologi og Silverlight

I dette kapitel vil nogle fundamentale aspekter omkring 3D blive beskrevet, samt hvordan denne teori videreføres til Silverlight 5. Under 3D teknologi afsnittet vil der først komme en beskrivelse omkring, hvad en vertex er, hvordan det bruges i 3D og, hvordan billeder kan bruges som overflader. Dernæst kommer der lidt om shaders, og til sidst beskrives nogle vigtige matricer, herunder world, view og projection. Under Silverlight afsnittet kommer der først en beskrivelse af, hvordan silverlight kontrollen DrawingSurface, der er den grafiske enhed, som står for rendering, fungerer, og dernæst hvordan overstående 3D teknologi elementer overføres til et Silverlight projekt.

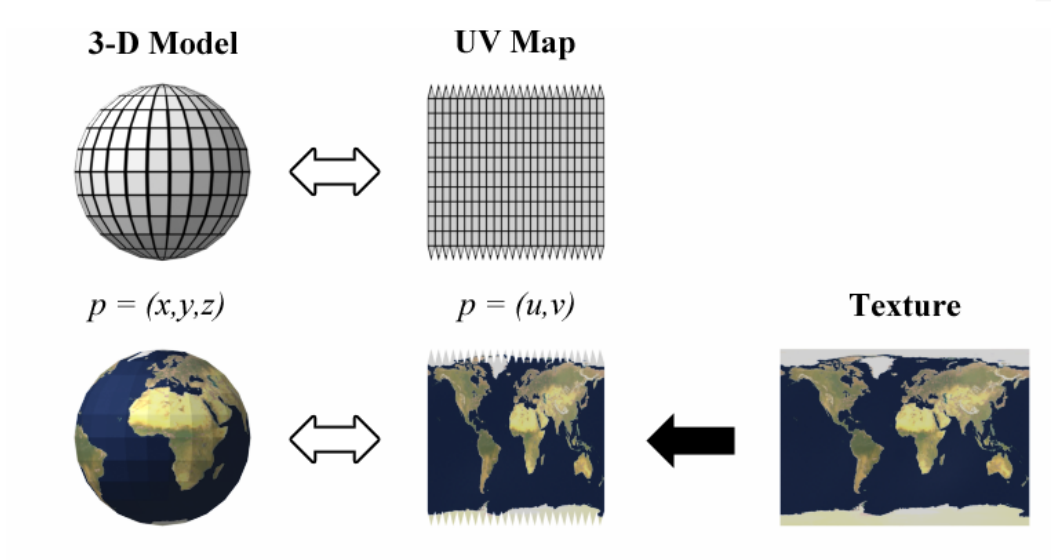
3D teknologi

Vertex

Den grundlæggende enhed til angivelse af geometriske objekter er en vertex. Simple geometriske objekter såsom linjer eller trekanter, kan angives med en liste af sådanne vertecies.
I computergrafik er objekter ofte repræsenteret som trianguleret polygoner, hvor objektets vertecies er forbundet, og ikke kun med de tre rumlige koordinater x,y,z, men også tit med andre grafiske oplysninger, der er nødvendige for at rendere objekter korrekt, såsom farver, teksturer og overflader normaler. Disse egenskaber anvendes af en vertex shader i renderingen, der er part af "vertex pipeline".
Som sagt, så bliver en vertex ofte beskrevet som en 3 dimensional vektor, hvor der skal angives x,y og z værdierne i et 3 dimensional koordinat system, som henviser til det 3D punkt, hvor en vertex skal placeres.

Texture

I 3D er der også mulighed for at give et objekt en overflade. Dette kan være som almindelig farve, eller der kan bruge en texture. Dette kan forstås som en slags tapet, der lægges på overfladen af et objekt. Denne texture er ofte et billede, og på den måde kan der nemt laves nogle flotte overflader på ens objekter. Måden hvorpå en texture lægges på en overflade kaldes "UV-mapping". Dette er altså processen, hvor der laves en gengivelse af et 2D billede på et 3D objekt. Bogstaverne U og V repræsenterer et 2D koordinat på et billede. Disse UV koordinater på et billede kan så mappes sammen med et 3D koordinat for et objekts vertex. Nedenstående billede er med til at vise, hvordan det skal forstås rent grafisk:



Figur 2.1 3D og UV mapping [2]

Som der ses på billedet, så er UV kortlægning med til at bygge bro fra 3D verdenen(vertex) til 2D verden(pixel). Ved at give trekanter eller andre polygoner UV koordinater, placeres de i princippet i billedrummet. Har man et billede, men ikke ønsker at vise hele billedet på en polygon, kan der bare tages den part af billedet der ønskes, ved at bruge de uv koordinater der repræsenterer den part, og så ”map” dem sammen med polygonens vertex koordinater.

Figur 2.2 3D og UV mapning [3]

Figur 2.2 viser et billede af en facade på en bygning. Det er dog kun døren der ønkes på ens polygon, og derfor angives de 4 UV koordinater på døren, som hjørnerne på døren svarer til. Derefter bliver de ”mapped” sammen med vertex koordinaterne på polygonen i 3D, som ses til venstre på billedet.

Shaders

Når der snakkes computer grafik, så er en shader et sæt af software instrukser, som bliver brugt til at beregne renderings effekt på hardware grafik. De bliver brugt til at programmere GPU’ens(graphics prossesing unit) programmerings ”pipeline” [4].

Direct3D og OpenGl bruger 3 slags shaders. En Vertex shader står for at tilføje specielle effekter til objekter i et 3D miljø. Den køres én gang for hver vertex, som bliver givet til grafik processoren. Dens formål er at omdanne hvert vertex 3D position i det virtuelle rum til 2D koordinaterne på skærmen. En vertex shader kan bruges til at manipulere med egenskaber, såsom position, farve og texture position og altså ikke oprette nye vertecies.

En geometry shader kan bruges til at tilføje og fjerne vertices fra en mesh, det vil sige overfladen på et 3D objekt, og til at tilføje volumetrisk detaljer til eksisterende mesh, som ville være for dyrt at behandle på en CPU. Men da det ikke er en shader, der vil blive brugt i dette projekt, så bliver denne ikke beskrevet yderligere og der springes til pixel shaders.

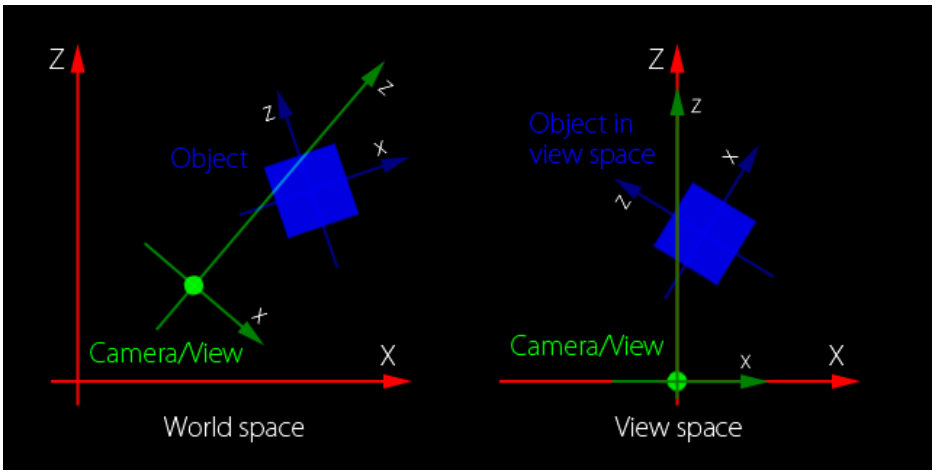
En pixel shader bliver, som navnet også hentyder til, brugt til at udregne farven af de enkelte pixels og andre attributter. Pixel shaderens job, kan spænde fra altid at udsende den samme farve, til at anvende en belysnings værdi, skygger, genspejling, transparens og andre fænomener.

World, View og Projection matricer

Når der snakkes om 3D, er det yderst vigtigt også at komme ind på de 3 matricer: world, view og projection.

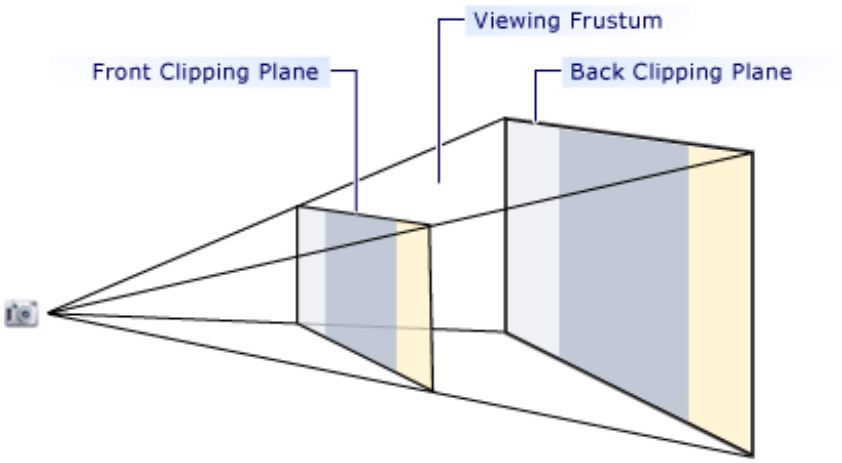
World matrix bruges til at transformere et objekts koordinater i dens lokale koordinat til verdens koordinatsystemet. Det vil sige, at er der specificeret et objekt, med en masse vertecies og tilhørende koordinater, og der ønkes at gøre det muligt for en bruger at rykke, rotere eller gøre den større, så man ikke skal ind og ændre objektets koordinater, men derimod ændre på hvordan den skal vises i verdenen, ved at ændre dens world matrix. World matricen består nemlig af 3 andre matricer: translation matrix(bruges til at rykke/flytte objekter), rotation matrix, og skalerings matrix. Så er der f.eks. lavet et objekt med en side med længden 1, og denne skal gøres dobbelt så stor, så ændres skalerings matricen for objeket.

View matricen bruges til at transformere verdenskoordinaterne ind i kameraets koordinat system baseret på kameraets position og retning. Dvs. at efter der er tilføjet en view matrix, så vil koordinater være relativ til kameraet og ikke (0,0,0) i verdens koordinat system. Nedenstående billede er med til at illustrere, hvordan et objekt er placeret i verdens koordinat systemet(i world space) og hvordan det bliver set fra brugeren, når en view matrix er tilføjet.



Figur 2.3 Camera/view [5]

Den sidste matrix der bliver beskrevet er en projection matrix. Denne matrix bliver brugt til at angive, hvor meget af verdenskoordinatsystemet, der skal være synligt, relativ til kameraet. Dette bliver også kaldt "field of view". Her skal der angives en vinkel af kameraet, en værdi for nær/forside og fjern/bagside, og en aspect ratio for skærmen. Når dette er gjort, får man det såkaldte viewing fustrum, hvor alt der er inde i den, er synligt, og det udenfor ikke er.



Figur 2.4 Viewing fustrum [7]

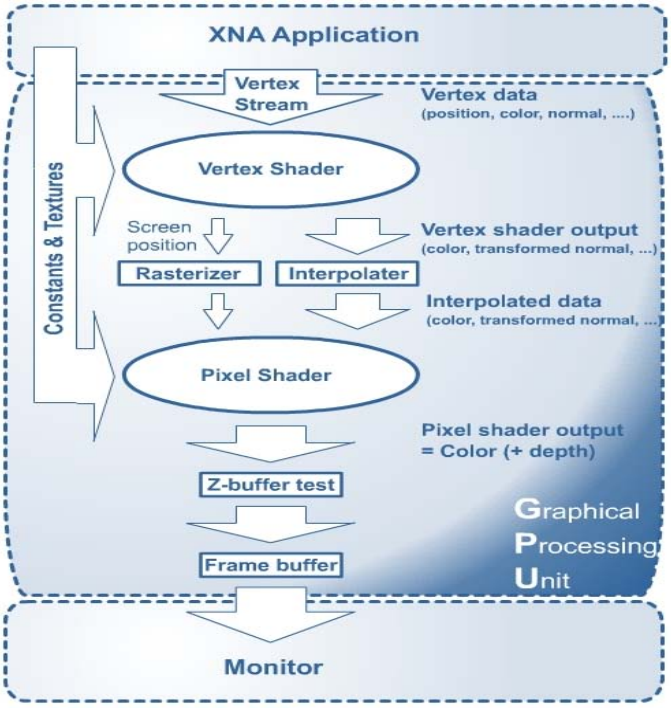
Silverlight

DrawingSurface

Det første der skal gøres, når der skal laves 3D i Silverlight 5 er, at der skal oprettes en DrawingSurface, som er den kontrol i Silverlight, der står for renderingen af 3D indholdet [7]. Det er en ny indfødt kontrol, der først er kommet her i Silverligt 5 og denne laves i xaml. Det primære fokus for DrawingSurface kontrollen er dens Draw event. Det er denne event, der bruges til at opdatere indholdet, der vises på DrawingSurface kontrollen. Det er altså i denne event, at der skal sørges for, at få GraphicsDevice enheden til at tegne ens 3D objekter. GraphicsDevice klassen er den, som indeholder alle de nødvendige elementer, der skal bruges for at lave og rendere objekter. Måden hvorpå denne GraphicsDevice kan tilgåes er vha. af GraphicsManageren, som kan skaffes ved GraphicsManager.Current, da denne holder styr på den nuværende GraphicsDevice. Her skal det bemærkes, at inden renderingen påbegyndes, skal enheden ryddes, og dette gøres med Clear metoden.

DrawingSurface kontrollens Draw event bliver kun fyret af en enkel gang, men dette kan omgås ved at bruge DrawEventArgs parameterens InvalidateSurface metode. Denne metode tvinger en opdatering af DrawSurface kontrollen, og der fyres så et nyt Draw event afsted en gang til. Derved skabes der et loop, som sørger for hele tiden at opdatere DrawSurface kontrollen, samt fyre Draw eventen afsted. Grunden til at det skal laves på denne måde er, at i forhold til XNA, hvor der er en konstant tikkende 3D renderings løkke, så er der ingen renderings løkke i silverlight 5, og derfor bruges en løkke lavet ved InvalidateSurface() [10].

I Silverlight er der mulighed for at gøre brug af ens hardware, og mere præcis maskinens Graphics Processing Unit(GPU). Dette kan gøres ved, at der i ens HTML- eller ASPX- fil tilføjes en ny parameter under "object tag". Parameterens navn skal være "EnableGPUAcceleration" og den værdi skal sættes til true. På nedenstående billede ses ”3D Graphics rendering pipeline”, som viser hvad der sker når der programmeres op mod GPU’en.



Figur 2.5 3D graphics rendering pipeline [8]

Da det er ens GPU, der står for at vertex data osv. bliver vist på skærmen, er det altså en nødvendighed for, at ens 3D side kan vises, at tagget ”EnableGPUAcceleration” er slået til.

Vertex

Som beskrevet i afsnittet om 3D teknologi, så bliver et 3D punkt kaldt en vertex. I silverlight er sådan en vertex defineret ved en struktur, som både gemmer vertex data, og en VertexDeclarations object, som indeholder information til GraphicsDevice enheden. Disse informationer omhandler størrelse og hvordan det skal bruges.

```
public struct VertexPositionColor
{
    public Vector3 Position;
    public Color Color;

    public VertexPositionColor(Vector3 position, Color color)
    {
        Position = position;
        Color = color;
    }

    public static readonly VertexDeclaration VertexDeclaration = new VertexDeclaration(
        new VertexElement(0, VertexElementFormat.Vector3, VertexElementUsage.Position, 0),
        new VertexElement(12, VertexElementFormat.Color, VertexElementUsage.Color, 0)
        );
}
```

Figur 2.6 VertexPositionColor

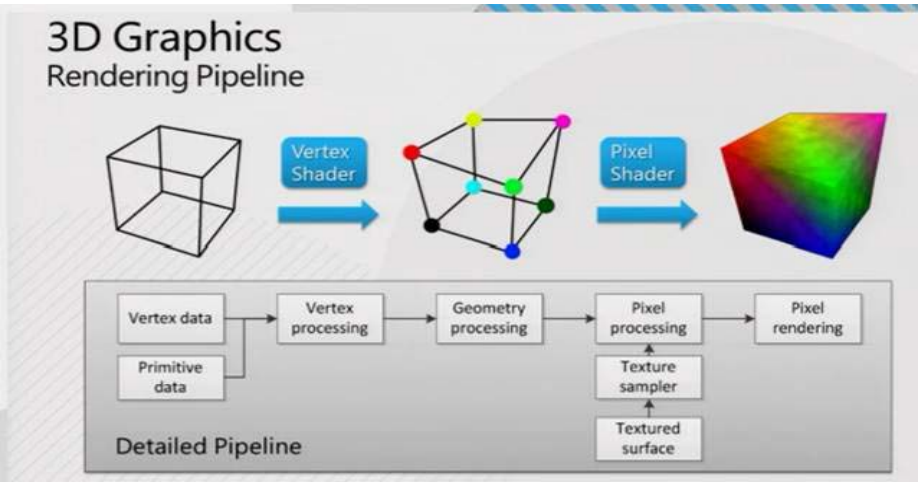
På dette eksempel får grafikenheden den information, at den vertex den skal vise, består af 2 vertex elementer. Den første er en Vector3 og denne vektor skal bruges som en Position. Det andet element er en Color, og denne skal bruges som farve på 3D punktet, som svarer til førnævnte Vector3 position. Denne vertex struktur, der indeholder denne vertexDeclaration, bliver ofte kaldt VertexPositionColor. En anden struktur, som der arbejdes meget med, kaldes VertexPositionTexture. Denne struktur adskiller sig fra den anden ved, at i stedet for en Color, indeholder den en Vector2 kaldet UV. Dette er som sagt en 2D postion, som henviser til et punkt på den valgte texture, der skal vises på Vector3 positionen i 3D.

Disse vertex strukturer, skal så gives til en VertexBuffer[12], der så er den, som grafik enheden læser fra i dens DrawPrimitives metode. Grafik enheden har en SetVertexBuffer metode, som skal bruges til dette.

Shaders

I 3D teknologi afsnittet blev der beskrevet vigtigheden af shaders, og en vigtig ting i Silverlight 5 er, at netop shader effekter kan anvendes på ens 3D objekter. Dette kan gøres ved at importere udarbejdede shaders skrevet i High-Level Shading Language(HLSL). Udarbejdelsen af disse filer er

mulige vha. Effect-Compiler Tool (FXC). Dette tool findes som en del af DirectX SDK [10] . To typer af shaders, som der oftes bruges, er Vertex shader(.vs filer) og Pixel shader(.ps filer). Som beskrevet ovenfor, så bruges en vertex shader til at transformere hvert vertex's 3D position i den virtuelle verden, til dens 2D koordinat på skærmen vha. world matricen. Pixel shaderen bruges til at beregne farven af individuelle pixels. I Silverlight bruges disse shaders af GraphicsDevice klassen til at rendere objekter sammen med DrawPrimitives metoden. På "3D Graphics rendering pipeline 2" billedet nedenunder, kan effekt shaderne ses i renderings pipelinen.



Figur 3.7 3D Graphics rendering pipeline [9]

Microsoft.Xna.Framework dll

Når der snakkes VertexDeclaration, VertexBuffer,PixelShader,VertexShader og vektorer som Vector2 og Vector3, så er det desværre ikke noget, som er indbygget i silverlight 5 RC(Release Candidate). Derfor er det en nødvendighed at lave en reference til nogle forskellige biblioteker, sådan at der opnås adgang til XNA's kerne grafik biblioteker, og derved muligheden for at arbejde med disse størrelser. Disse biblioteker er tilgængelige ved at downloade Silverlight toolkit og nogle af disse biblioteker, der så er til rådighed, er Microsoft.Xna.Framework, Microsoft.Xna.Framework.Graphics, Microsoft.Xna.Framework.Math og system.windows.Xna. Hvis der igen kigges lidt på VertexPositionColor og VertexPositionTexture, så var disse strukturer nogle, der selv skulle laves i Silverlight 5 beta, men med den nye release Silverlight 5 RC og Silverlight toolkittet, er disse tilgængelige ved Microsoft.Xna.Framework.Graphics.VertexPositionTexture og Microsoft.Xna.Framework.Graphics.VertexPositionColor, men de er lavet på samme måde som i Figur 2.6.

Tekst i 3D

Når der arbejdes med 3D, er der desværre ikke nogen mulighed for at arbejde med almindelige tekstbokse, som der kendes fra xaml. Dvs. at det ikke er muligt at sætte tekstbokse på 3D objekter, som består af vertecies. Denne kendsgerning giver selvfølgelig nogle vanskeligheder, men det er dog ikke umuligt at få tekst på sin 3D side. Eftersom det er muligt at mappe et texture billede på et 3D objekts vertecies, så er der altså mulighed for at få tekst på siden vha. af bitmaps, der indeholder den, tekst der skal bruges.

World, View og Projection matricer i Silverligt

Som beskrevet i 3D teknologi afsnittet, så er der 3 vigtige matrice,r som man skal have lavet, når der arbejdes med 3D. Er der lavet en reference til xna framework, så er der her en masse værktøjer, der kan hjælper en med at lave disse matricer [6].

Startes der med world matricen, så bestod denne, som beskrevet tidligere, af de 3 transformations matricer, translation, rotation og skalering. Som translation kan der f.eks. bruges Matrix.Identity, der giver en identitets matrix, og derved bliver objekterne altså ikke forskudt i forhold til deres lokale koordinater. Skalerings matricen fås ved at bruge Matrix.CreateScale(float value), hvor der skal angives, hvor meget man vil skalere. Rotations matricen fås ved Matrix.CreateFromYawPitchRoll(float yaw, float pitch, float roll), en rotation omkring x aksen kaldes yaw, pitch er omkring y aksen og roll er omkring z aksen. Når disse 3 matricer er oprettet, så ganges de bare sammen for at få den ønskede world matrix.

View matricen indeholder som sagt kameraets position, positionen for hvad kameraet kigger på, og en værdi der viser, hvordan kameraet vender, dvs. på hoved, oprejst eller på siden enten højre/venstre side. Her har xna igen et værktøj som kan bruges, nemlig Matrix.CreateLookAt(Vector3 cameraPosition, Vector3 cameraTarget, Vector3 cameraUpVector), hvor cameraUpVector er den vektor, der angiver om kameraet står på hoved eller oprejst osv. Angives der f.eks. en vektor (0,1,0), betyder det at kameraet er oprejst. Denne Matrix.CreateLookAt metode giver som return value den ønskede view matrix.

Projection maticen som angiver det såkaldte "field of view", kan laves ud fra metoden Matrix.CreatePerspectiveFieldOfView(float fieldOfView, float aspectRatio, float nearDistance, float farDistance). Den første parameter fieldOfView angives ofte som MathHelper.PiOver4, der svarer

til 0,785 radianer, som er omtrent 45 graders synsfelt. Aspect ratio værdien findes ved at dividere bredden med højden af DrawingSurface kontrollen. NearDistance og farDistance angives ud fra kameraets position, og som sagt beskriver disse, hvor tæt på/langt væk objekter ønskes synlige i forhold til kameraet.

Tråde

I Silverlight bruges der ofte mange tråde. En vigtig ting at vide er derfor, at når der arbejdes med 3D i Silverlight, så foregår Silverlight 3D's grafik renderingen ikke i applikationens UI tråd, der er hoved tråden. Selve renderingen foregår i en "composition" tråd. Det er derfor vigtigt, at når der ændres på ens data model, så skal dette ske på UI tråden og ikke på composition tråden, sådan at der bruges mindst mulig tid i Draw event handleren, og på den måde forbedres præstationen af programmet. Grunden til dette er, at når Draw event looper hele tiden, skal der derfor helst kun bruges tid på at tegne og ikke på at lave en masse beregninger. Det er så meningen, at når ens data er opdateret på UI tråden, så læses den opdaterede data på compositions tråden [7].

Kapitel resume af 3D teknologi og Silverlight

Ud fra dette kapitel omkring 3D teknologi, og hvordan dette fungerer i Silverlight, kan det ses at Silverlight, som værktøj for udarbejdelsen af en 3D scene, er et godt valg, da de vigtigste elementer i 3D er videreført til Silverlight eller tilgængelige vha. af referencer til forskellige Microsoft.Xna.Framework biblioteker. Dvs. at ønkes der bestemte billeder på 3D objekter, så er dette muligt vha. texture, og det er også muligt at manipulerer med sine world, view og projection matricer, og derved lave effekter såsom at rotere, zoome ind og ud og generelt bare at flytte sig i scenen.

Ydermere er der som sagt mulighed for, at der vha. texture, kan vises tekst på siden, og dette er en nødvendighed for en side, som den projektet omhandler, da der uden tekst ikke kan vises segmenter for slutbrugeren på en sådan måde, at der skabes overblik.

Når en virksomhed som Licensewatch A/S, der har et produkt, som indeholder en masse aspx. sider, og hvor der ønkes brug af 3D på en af disse sider til at vise overblik, er Silverlight 5 derfor et oplagt valg af teknologi. Der er mulighed for at virkeliggøre ideen, som kan ses på ”Figur 1.2-Ide til at give brugeren overblik” i projektbeskrivelsen, samtidig med at man bliver på .NET platformen.

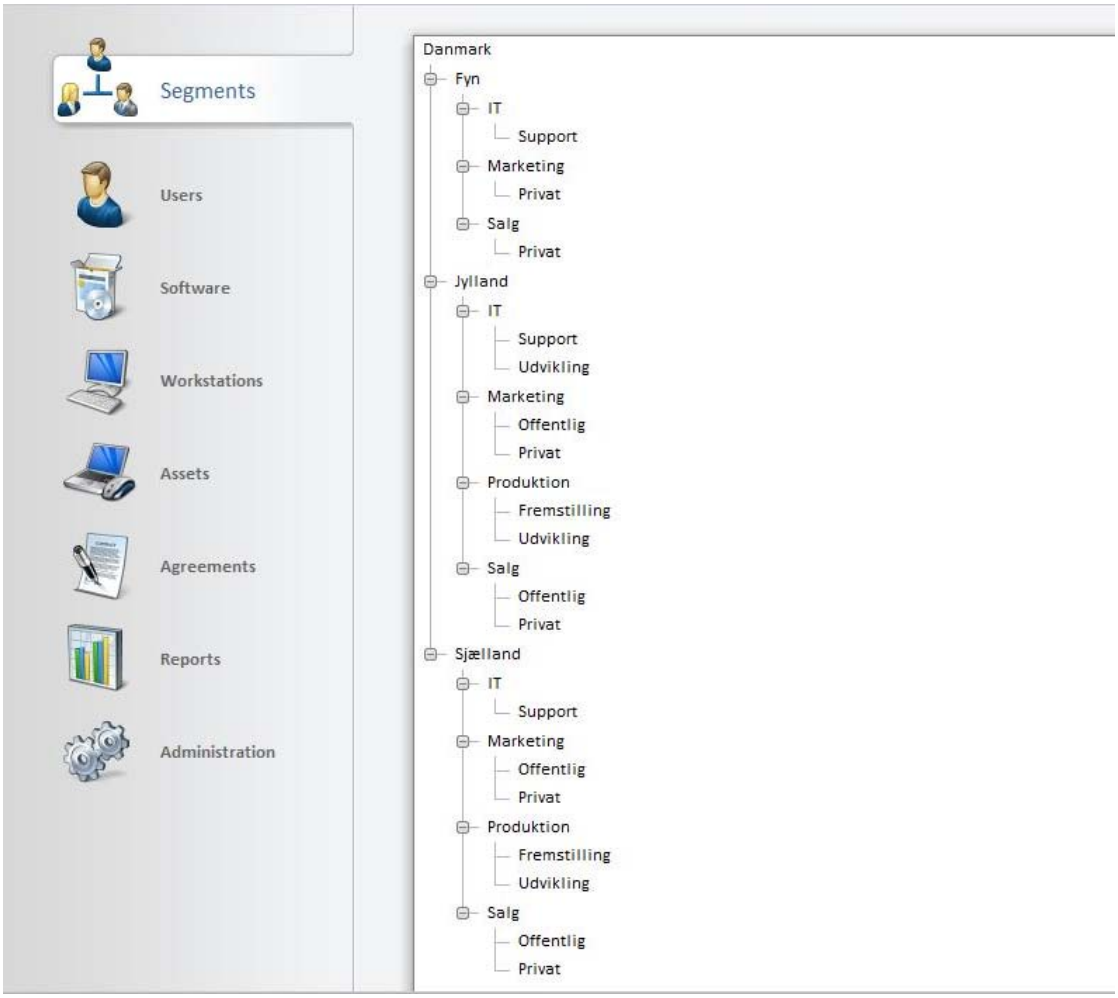
Kapitel 3. Analyse af ide og opstilling af krav

I dette kapitel beskrives der, hvad meningen med selve segment siden er, så der opnås forståelse for, hvilke behov denne side dækker hos brugerne, både som det er nu, og hvilke behov LicenseWatch gerne vil have dækket fremover på denne side. Derudover opstilles der også forskellige krav til prototypen ud fra denne analyse.

Analysen er lavet i samarbejde med Licensewatch A/S, da de har en stor viden omkring, hvad deres kunder gerne vil have, og hvad deres nuværende kunder bruger siden til.

Hvilken værdi giver segment siden

Grunden til at LicenseWatch (LW) har valgt at gøre det muligt for brugerne at opdele deres virksomheder i segmenter og på segment siden få vist et overblik over dem, er p.g.a. overskuelighed, som er en essentiel ting ved et SAM produkt. Overskuelighed er grunden til, at virksomhederne bruger et SAM produkt til at starte med, da opgaven med at holde styr på computere og deres software samt disse softwares licenser osv. er en uoverskuelig opgave for en stor virksomhed. I virksomheder, hvor antallet af computere er højt (100+ computere), kan man nemt forestille sig, bl.a. ved at kigge på antallet af installerede programmer på ens egen computer, at uden et værkstøj, som det LW kan tilbyde , bliver denne opgave nærmest umulig. Eftersom LW's produkt henvender sig til virksomheder af en vis størrelse, har disse virksomheder ofte brug for at organisere og inddele deres virksomhed, for at få indført større overskuelighed. Nogle vælger at opdele deres virksomhed i segmenter, der svarer til forskellige afdelinger, der allerede findes i virksomheden, mens andre finder på måder, der kan være mere hensigtsmæssige for den enkelte virksomhed. Segment siden, som den er nu, viser brugeren, hvilke segmenter der er, og hvordan segment strukturen er vha. en træstuktur, sådan at det nemt kan ses, om et segment har undersegmenter. På den måde er det nemt for en medarbejder, der f.eks. sidder i en økonomiafdeling, og har brug for at se, hvor mange licener der tilhører en bestemt afdeling/segment, at finde dette segment på segment siden og derefter komme ind på dennes summary side og derinde finde de oplysninger, han har brug for. Værdien af siden er altså, at der nemt opnås et overblik over, hvordan ens virksomhed er inddelt, og brugeren kan nemt klikke sig hen på et bestemt segment. Nedestående figur viser et billede af den nuværende segment side og måden, hvorpå en segmenstrukturen bliver vist.



Figur 3.1 Nuværende segment side

Hvem er brugerne af segment siden og hvorfor bruger de den

Ifølge LicenseWatch er brugerne af segment siden, oftest medarbejdere i økonomiafdelingen eller IT- afdelingen, da disse har brug for vide, hvordan virksomheden er inddelt med hensyn til software og licenser. Disse brugere har vidt forskellig baggrund og uddannelse samt forskellige formål med siden. Nogle bruger den bare til at klikke videre til summary siderne, mens andre er inde på siden for at se strukturen af segmenter, herunder hvilke segmenter der tilhører andre segmenter. Derfor er det vigtigt for LicenseWatch, at siden både kommer til at passe til de tekniske medarbejdere i IT-afdelingen, der ofte gerne selv vil styre indstillingerne, samt passe til økonomimedarbejderne, der hellere vil have en nem og hurtig brugergrænseflade, hvor der ikke skal laves alt for mange indstillinger, før de finder frem til søgeresultatet.

Hvad vil Licensewatch opnå ved at lave siden til 3D

Det LicenseWatch gerne vil opnå med at få 3D ind på segment siden, er at skabe mere overblik og give brugerne mere værdi på siden, sådan at det ikke kun er segmentstrukturen, der kan ses på siden, se figur 3.1. Eftersom segmenterne kan få licenser, brugere, computere og aktiver(assets) tilknyttet, er det disse elementer, som LW gerne vil have ind på siden, sådan at brugene ikke nødvendigvis skal ind på den enkelte summary side, for at få en fornemmelse af antallet . LW vil også gerne have, at det på en nem måde, er muligt for brugerne at sammenligne størrelserne af segmenterne på siden. Sådan at hvis der ønskes en viden omkring, hvilket segment der har flest licenser, så kan det undgås at gå ind på hver enkelt segments summary side og finde antallet og først derefter have muligheden for at sammenligne.

Hvad skal den nye 3D side kunne, såsom rotere og summere antallet

På den nye side vil LicenseWatch som sagt gerne have vist segmenterne på en måde, der minder om den nuværende side. LW vil dog til gengæld gerne udnytte 3D teknologien til at vise størrelsen på segmenterne angående brugere, computere, aktiverog licenser, så når siden åbnes, vises segmenterne som normalt, men ved at udnytte, at der på siden nu er en dybde, gøres det muligt for brugerne at rotere billedet på en sådan måde, at man kan vise antallet af de forskellige segmenter, uden at brugeren har været nød til at klikke sig ind på siden.

Da LicenseWatch også gerne vil udnytte 3D teknologien til at gøre det muligt for brugerne at sammenligne segmenterne imellem, så skal brugerne altid have muligheden for at se alle segmenter på samme tid, og derfor ønsker LW, at brugerne får mulighed for at bevæge sig fra side til side og op og ned på siden, samt at kunne zoome ind og ud. Ved at give brugeren mulighed for at rotere, bevæge sig og zoome ind og ud, bliver det altså muligt for brugeren enten at se alle segmenter eller kun at fokusere på de segmenter, der er relevante at sammenligne.

LicenseWatch vil også gerne have, at når siden viser antallet af f.eks. brugere tilknyttet et segment, så skal der tages hensyn til, om segmentet har undersegmenter, og om disse undersegmenter bliver vist, altså om "parent" segmentet er ekspanderet. Dvs. at har et segment selv 10 brugere tilknyttet, plus et undersegment med 5 brugere, så er antallet for segmentet 15, når denne ikke er ekspanderet, og 10 når den er ekspanderet, hvor antallet for undersegmentet så viser 5.

Visning af de grafiske elementer samt interaktions muligheder på siden

Da den nye segmentside i 3D, skal vise en træstruktur over segmenterne samt vise størrelsen af et segment vha. af forskellige 3D features, såsom muligheden for at vende et segment rundt ved rotation og derved vise brugeren en ny side af segmentet, er der også blevet analyseret på, hvordan dette bedst laves i 3D, så brugeren forstår, hvad der vises på siden. Derved er der kommet nogle krav/ønsker til, hvordan elementerne skal vises på siden og hvilke muligheder brugeren skal have.

I analysen er der blevet kigget på nogle gestalt love, samt på hvordan der kan bruges elementer, som der må forventes, at brugerne er bekendte med. På den måde skulle projektet gerne ende op med en prototype, som brugere af LicenseWatch systemet kan få gavn af med det samme, uden at de skal have den store træning i at bruge siden.

Startes der med at kigge på, hvordan antallet af de forskellige emner, som kan være tilknyttet et segment, bedst kan vises på siden, er der blevet analyseret frem til, at det vil være som søjler. Grunden til dette er, at det, som tidligere beskrevet i dette kapitel, er brugere med forskellig baggrund, som skal kunne bruge siden og forstå den. Derfor er søjler et forholdsvis sikkert valg, da man ofte forbinder søjler med et antal i et søjledigram, og især når segment siden bliver brugt af økonomiarbejdere, som må forventes at være bekendte med sådanne søjlediagrammer.

Det næste, der blev undersøgt var, hvordan sammenhængen mellem søjler, der viser antal, og det segment som søjlerne tilhører, skal vises, samtidig med at muligheden for at vise segmenterne på en måde, der minder om det nuværende, stadig beholdes intakt. Dvs. at vise navnene på de forskellige segmenter og vise træstrukturen uden at antallet er synligt for brugerne. Løsningen på dette blev, at hvert segment vises som en rektangel med dets navn indeni. På den måde kan man så bruge denne rektangel som en slags platform, hvor de søjler, der tilhører segmentet, kan stå på. Som begrundelse for, at brugerne vil forstå dette, henvises til gestalt loven "loven om nærhed", der siger, at hvis elementer er anbragt i nærhed af hinanden, så vil hjernen opfatte dem som hørende sammen [17].

Vælges der samtidig, at søjlerne står på bagsiden af segment platformen, og navnet på segmentet på forsiden, så vil det være muligt at vise segmenterne på en måde, som vil minde om den gamle

segment side. Her henvises igen til stadie 1. på figur 1.2 i kapitel 1, der kort viser ideen af den nye side og figur 1.1, der viser, hvordan dette ser ud på den nuværende side.

Ved at give hvert segment platform fire søjler, der repræsenterer brugere, assets, computere og licenser, bliver søjlerne desværre ikke vist som i et normalt søjlediagram, hvor de søjler der skal sammenlignes står ved siden af hinanden. Derved opstår der et problem med, at få lavet det sådan, at brugeren nemt kan se, hvilke søjler der repræsenterer f.eks. computere og derved kan sammenligne disse søjler med hinanden. Løsningen er at bruge billeder, sådan at søjler der repræsenterer computere, får et billede med en computer, søjler der repræsenterer brugere får et billede af en person osv. På den måde opfatter brugeren søjler med samme billede som værende af samme type, og derfor mulige at sammenligne - her henvises til gestalt loven (loven om lighed)[17]. Valget af billeder blev bestemt ud fra, at billederne selvfølgelig skulle vise det søjlen står for, men det skulle også være billeder som brugerne er bekendte med. Derfor faldt valget på at bruge billederne fra den nuværende menu, som findes i LisenceWatchs SAM system. se nedenstående figur 3.2:

Figur 3.2 Søjle billeder

En vigtig ting ved den nuværende segment side er, som tidligere beskrevet, muligheden for at se segment strukturen vha. af en træstruktur, se igen figur 3.1. LicenseWatch vil som sagt gerne beholde muligheden for at vise denne struktur, og igen er det vigtigt at strukturen vises som i det gamle. Derfor skal en platform for et undersegment vises under dens "parent/forældre" samt forskydes til højre, og de skal forbindes med en streg, der går fra "forældre" segmentet ned til segmentet for et "barn".

Når man har med sådan en træstruktur at gøre, så skal der selvfølgelig også være en "expand" knap, så brugeren har mulighed for selv at bestemme, hvilke segmenter der skal ekspanderes. Da vi her arbejder med 3D, skal denne knap også have en dybde, sådan at brugeren hele tiden kan se knappen, selvom billedet er roteret på forskellige måder. Derfor er det blevet bestemt, at knappen er en kvadratisk kube, hvor siderne henholdsvis viser et + når det er muligt at ekspandere segmentet og modsat et - når den allerede er ekspanderet.

LicenseWatch vil som sagt gerne gøre det muligt for brugerne, at de helt selv kan bestemme, hvordan de gerne vil have segmenterne vist, sådan at det bliver nemmest for den enkelte bruger at overskue siden. De skal altså have mulighed for at rykke med kameraet og bestemme hvor meget, der skal roteres omkring de forskellige akser. Dog mener LicenseWatch, at det kun er en lille del af brugerne, der har lyst til at sidde og nørde med at rotere forskellige grader omkring akserne, og det er heller ikke alle brugere, der har erfaring med at navigere rundt i et 3D koordinat system. Ideen er derfor at have en knap, der sørger for at billedet roterer, sådan at siden ender med at se ud som i stadie 3, på firgur 1.2. Det skal dog være muligt for de mere erfarne at gå ind og bestemme disse rotationsindstillinger.

Det sidste krav der er kommet til systemet ud fra analysen er, at det skal være muligt at trykke på objekter i 3D scenen. Trykker en bruger på en segment platform, så skal brugeren sendes videre i systemet, til det valgte segments "summary side". Dette er grundet ønsket fra LicenseWatch om at bibeholde den feature fra den gamle side, hvor navnet på segmentet er et hyperlink.. Trykker brugeren derimod f.eks. på en søjle, der viser antallet af computere, så skal brugeren sendes hen til den side, der viser alle computere for det specifikke segment, da dette vil give siden en feature, som den gamle ikke havde, men som LicenseWatch mener, brugerne godt kunne tænke sig. Trykker brugeren på en "expand" knap så skal segmentet som beskrevet tidligere ekspanderes.

Kapitel resume af analyse

Efter denne analyse er der nu kommet nogle grafiske krav, samt krav der går på, hvilke interaktions muligheder der skal være for brugeren, og som LW godt kunne tænke sig indarbejdet i prototypen. Disse krav er som følgende:

- Antal af brugere og computere osv. skal vises som søjler, og selve antallet skal stå på søjlen.
- De forskellige antal for et segment der ikke er ekspanderet, skal være summen er dens egne og dens udersegmenter, og er den ekspanderet, skal den kun vise dens egne antal.
- Det skal være muligt for brugeren at roter billedet.
- Det skal være muligt at ændre på indstillingerne for rotationen.
- Navnet på et segment skal stå på en rektangulær platform.
- Brugeren skal have mulighed for at bevæge sig på siden. Dette indebærer mulighed for at zoome og flytte billedet fra side til side, op og ned.
- Søjlerne, der tilhører et segment, skal stå på denne platform.
- Det skal være muligt for brugeren, vha. flytning af kameraet, altid at se ”expand” knappen, og denne skal vise + og - .
- Brugeren skal kunne trykke på expand knappen, segment platformene og søjlerne.

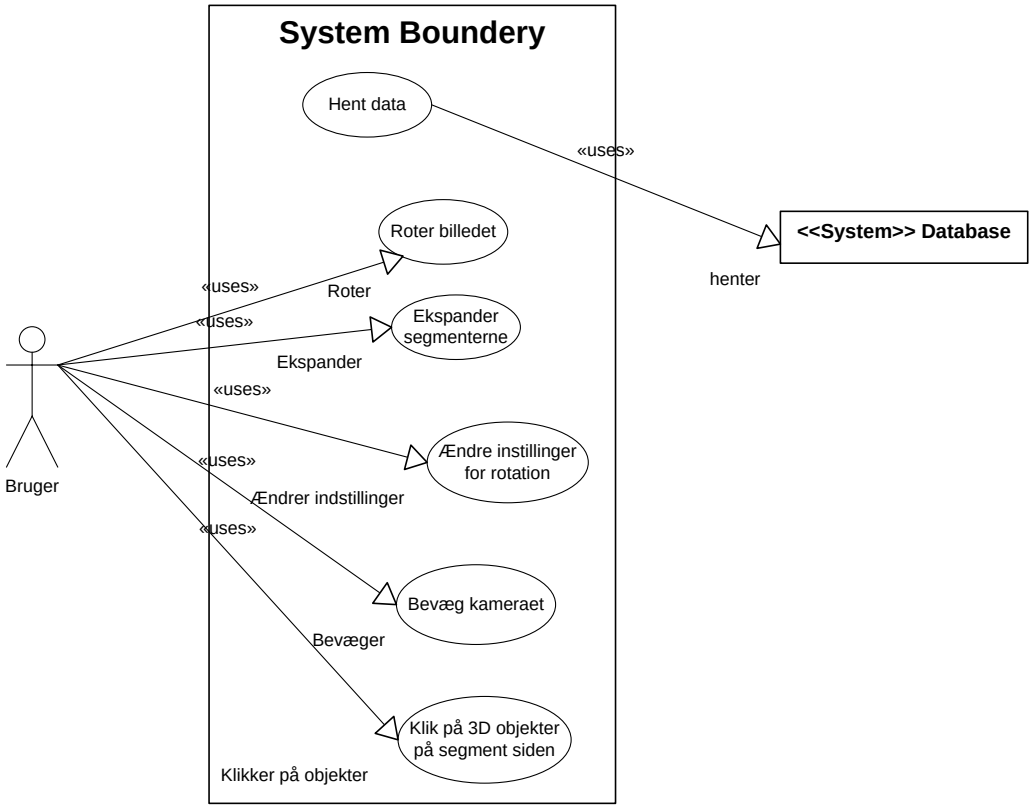
Disse krav skulle gerne ende ud i en prototype, der fungerer som ideen, der blev beskrevet i kapitel 1 under projektbeskrivelsen, se figur 1.2. Brugerne burde nemt kunne gå ind, og bruge den nye side uden at føle, at den er uoverskuelig, da netop overskueligheden er en af de helt store værdier, som segment siden er med til at give, og især den nye segment side, hvor fornemmelsen af størrelse på forskellige segmenter er synlig for brugerne. Kigges der på, hvad analysen kræver af siden rent 3D mæssigt, så ved vi allerede fra forrige kapitel, at langt de fleste af kravene er mulige at implemetere. Der er mulighed for at få billeder ind på søjlerne vha. texture, og texture giver også mulighed for putte tekst på segmenterne og søjlerne, se afsnit "Tekst i 3D" i forrige kapitel. Ved at lave forskellige rotations og translations matricer, er det muligt at rotere segmenter og søjler, samt at placere dem præcis, hvor man vil havde dem på siden. Ved at ændre på view matricen, der beskriver kameraet i 3D, (se forrige kapitel omkring matricer) så er det også muligt at give brugeren mulighed for at bevæge sig på siden.

Kapitel 4. Design

I dette kapitel vil projektets design blive beskrevet. Herunder vil der blive opstillet forskellige use cases, ud fra de fundne krav i kapitel 3. Disse use cases er med til at give en beskrivelse af de interaktions krav, som LW gerne vil have implementeret i prototypen. Efterfølgende ses på, hvordan projektet er struktureret i form af solution strukturen og forskellige klasse diagrammer. Til sidst vil nogle af de vigtigste forløb blive forklaret i form af sekvens diagram, herunder hvordan der arbejdes med composition tråden og UI tråden.

Use cases

Ud fra analysen omkring krav til segment siden i samarbejde med LW, er der fundet seks use cases, som her vil blive beskrevet som casual use cases. Ud af de seks fundne use cases, er 5 af disse, actions som brugeren af systemet skal have mulighed for. Den sidste omhandler en action fra systemet til en anden system aktør, nemlig databsen. De fem user actions er: "Roter billedet", "Ekspander segmenterne", " Ændr instillinger for rotation", "Bævæg kameraet" og " Klik på objekter i scenen". Den sidste og sjette use case er "Hent data".



Figur 4.1 Use case diagram

Casual use cases:

De første fem use cases, der involverer den " **Primary Actor** ", som er den almindelige bruger af systemet er:

1. *UC 1: Roter billedet*

Eftersom det i det forrige kapitel kom frem, at søjlerne, der skal repræsentere antallet af brugere og computere osv., skal stå på bagsiden af et segment platform, så skal brugeren have mulighed for at trykke på en roter knap, hvorefter objekterne på scenen roterer på en sådan måde, at man kan se søjlerne.

2. *UC 2: Ændr indstillinger for rotation*

Da LicenseWatch, som beskrevet tidligere, har forskellige typer af brugere til siden, hvor nogle af disse formentlig gerne vil have muligheden for selv at bestemme, hvordan objekterne i scenen skal roteres, så skal brugeren have mulighed for at ændre på værdierne for, hvor meget der skal roteres omkring akserne.

3. *UC 3: Ekspander segmenterne*

Da segment data, der bruges til segment siden, har en struktur, som passer til et "treeview", så skal 3D segment siden også indeholde et "treeview", hvor brugeren har mulighed for at ekspandere segmenterne, så undersegmenterne bliver vist under det segment, de hører til.

4. *UC 4: Bevæg kameraet*

Når det er muligt for brugeren at ekspandere alle segmenterne, så kan der opstå en situation, hvor der vises en masse segmenter på siden. Derfor skal brugeren have mulighed for at zoome ud, sådan at det store overblik stadig er muligt at opnå, og modsat hvis brugeren vil fokusere på enkelte segmenter, skal muligheden for at zoome ind på disse være til stede. Til at zoome bruges en slider. Derunder skal det selvfølgelig også være muligt at bevæge sig fra side til side, og dette gøres med musen.

5. *UC 5: Klik på 3D objekter på segment siden*

På den gamle segment side, har brugeren mulighed for at trykke på en knap, hvorved segmenterne bliver ekspanderet. På den nye side, skal denne knap være et 3D objekt, og derfor skal det være muligt at trykke på denne knap. Udover dette er brugeren også vant til, at navnet på et segment bliver vist som et hyperlink, som sender brugeren hen til en summary side for segmentet. Dette skal også virke på den nye side, så brugeren skal kunne trykke på en segment

platform og derefter sendes videre til en summary side. Derudover vil LicenseWatch, som beskrevet i forrige kapitel, gerne have muligheden for at sende brugeren direkte til den side, der f.eks. viser licenser for et specifikt segment, hvis vedkommende trykker på licens søjlen.

Den sidste use case involverer en " **Secondary Actor"** som er databasen, hvor virksomheden har sine data liggende, og som den nye segment side får dens data fra.

6. *UC 6: Hent data*

Systemet skal hente sit data i databasen og vha. af webservices skal disse data sendes til Silverlight delen af systemet.

Herunder er de 5 use cases opført i en tabel, hvor der er blevet givet værdier for, hvor komplekse de er, og hvilke prioriteter de har.

Use case UC:	**Kompleksitet (1-5),hvor 5 er det mest komlekse**	**Proiritet (1-5) hvor 5 er top prioritet**
1)Brugeren skal kunne rotere billedet så søjlerne for et segment kan ses.	2	5
2) Brugeren skal kunne ændre på indstillingerne for, hvor meget der skal roteres.	2	2
3)Da data vises som et treeview, skal man kunne ekspandere.	4	4
4) Brugeren skal kunne bevæge sig på siden	1	5
5) Brugeren skal kunne trykke på 3D objekter på siden.	5	3
6) Systemet skal hente data fra en database og Silverlight projektet bruger webservices til at få disse data.	3	5

Figur 4.2 Brief Use case tabel

Begrundelse for værdier i overstående use case tabel:

Til **UC 1** er der er angivet en kompleksitet på 2. Begrundelsen er, at der i kapitel 2 omkring 3D teknologi og Silverlight er beskrevet, hvordan man kan tilføje rotation vha. af Matrix.CreateFromYawPitchRoll(float yaw, float pitch, float roll) metoden. Da denne matrix metode findes, burde der ikke være de helt store problemer i at rotere. Derimod er prioriteten af denne use case sat til en top prioritet, da det er afgørende for ideen omkring, hvordan siden kan udformes, at denne use case bliver lavet. Er det ikke muligt at rotere, så bliver det meget svært at udnytte den ekstra dimension, man får givet i 3D.

Til **UC 2** er der angivet en kompleksitet på 2. Begrundelsen for dette er, at metoden Matrix.CreateFromYawPitchRoll(float yaw, float pitch, float roll) bare skal bruge tre float værdier, så derfor skal der bare laves en mulighed for brugerene til at ændre på tre simple værdier. Ændringen er derfor meget simpelt, men grunden til at den ikke har fået en kompleks værdi på 1, er at rotationen skal ske i en flydende bevægelse og ikke bare med en rotation, og her ligger der formentlig lidt arbejde, som ikke er helt så simpelt. Dette er ikke en top prioritet, som prioritets værdien også antyder, da det formentlig ikke er alle brugere af systemet, der vil udnytte denne feature.

Til **UC 3** er der angivet en kompleksitet på 4. Denne værdi er givet ud fra, at et treeview på segment siden kommer til at bestå af 3D objekter. Disse består kun af vertecies, hvor det ikke er muligt at binde på en position til et undersegment, som det ville være, hvis man arbejdede med almindelig view-model. Derfor skal der laves en algoritme, der går ind og finder ud af, hvor et "parent" segment er tegnet, og bliver denne ekspanderet, så skal den finde ud af hvor et "child" segment(udersegment) er tegnet, sådan at der kan laves en streg fra "parent" til "child". Derudover skal den forskyde segmenterne, der skal vises længere nede på siden, ud fra hvor mange nye segmenter der vises, nu hvor et segment er blevet ekspanderet. Denne use case har fået en prioritet på 4, hvilket næsten er en top prioritet. Dette kommer af, at det segment data, der vises på siden, er lavet til at blive afbilledet i et treeview, og det er det, som de gamle brugere af vant til. Hvis det f.eks heller ikke er muligt at ekspandere/skjule segmenter, så kan siden også blive meget uoverskuelig for store data set.

Til **UC 4** er der angivet en kompleksitet på 1. Grunden til dette er, at der fra tidligere kapitler vides, at måden hvorpå der kan bevæges på siden, er ved at ændre view matricen, da denne beskriver kameraets position og, hvor denne kigger hen. Så ved at ændre på kameraets z værdi, burde en zoom effekt være muligt. Til at bevæge kameraet fra side til side og op og ned, ændres på x og y værdierne. Til at ændre på disse, kan forskellige muse events og slider events bruges, og implematationen af disse burde være simpel. Prioritetet er derimod sat til 5. Dette kommer af, at der umuligt kan findes en kamera positon, der vil være tilfredsstillende for alle brugere af systemet, da antallet af segmenter er forskelligt fra virksomhed til virksomhed. Det at brugerne kan ekspandere, gør også, at antallet af segmenter på siden kan være forskelligt.

Til **UC 5** er der angivet en kompleksitet på 5. Dette skyldes problemet i, at ens muse koordinater, hvor en bruger trykker, er 2D koordinater, mens objekterne på siden har 3D koordinater, der beskriver, hvor i scenen deres vertecies(hjørner) er, og altså ikke koordinater for selve overfladen som brugeren trykker på. Der skal også tages hensyn til de tre matricer: world, view og projection, før man kan finde, hvor på 3D siden, der er blevet trykket. Prioriteten er sat til 3. Dette er fordi både brugere og LicenseWatch er vant til, at have denne feature, da det er normalt på den gamle side, at man kan trykke på hyperlinks og blive sendt videre. Det er også normalt for et "treeview", hvor man kan ekspandere, at der er en "expand" knap, der kan trykkes på, og denne vil være mest intuitiv at bruge, hvis den også er lavet som et 3D objekt på siden. Det er derimod ikke en feature, som er afgørende for prototypen, da man formentlig vil kunne komme på andre ideer til, hvordan man kan blive sendt videre i systemet, eller ekspandere segmenter. Der kunne f.eks. bruges en ListBox kontrol, hvor de synlige segmenter er vist, og hvor det så er muligt at trykke på segmenterne, og derefter vælge, om der skal ekspanderes eller sendes videre til deres summary side.

Til **UC 6** er der angivet en kompleksitet på 3. Dette forekommer måske lidt høj, eftersom selve sql kaldene formentlig ikke bliver særligt komplekse. Kompleksiteten på 3 begrundes med, at Silverlight delen af projekt, skal have data'en vha. webservices, som bliver kaldt asyncron, samt at det er UI tråden, der skal sørge for, at lave 3D view objekterne ud fra data'en. Denne skal derfor vide, hvornår data er hentet. Til sidst skal de view objekter, der er lavet ud fra det hentede data, tegnes på composition tråden. Derfor er der en vis kompleksitet i de asyncrone kald sammenlagt

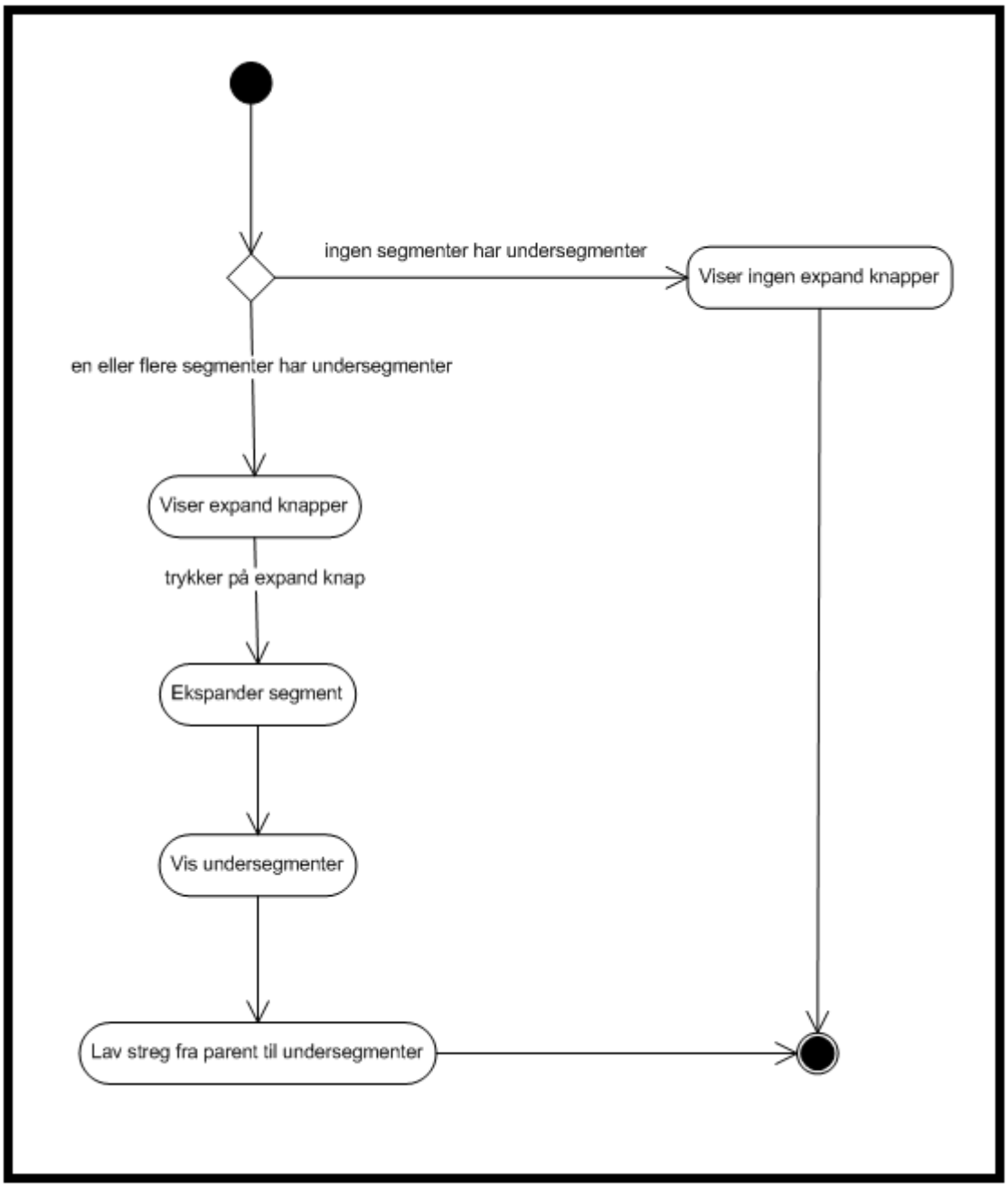
arbejdet med forskellige tråde, når de asyncrone kald er færdige. Prioriteten er sat til 5, da projektet ikke kan fuldføres, hvis det ikke er muligt, at hente det rigtige data, på en sådan måde, at det kan bruges, når man arbejder med 3D.

Fully dressed use case

Nedenstående er der udvalgt to vigtige use cases, og disse er beskrevet som fully dressed use cases. De resterende fully dressed use case beskrivelser kan findes i bilag 1.

ID:	UC 3
Titel	Ekspandere segmenter.
Beskrivelse	Segmenterne på siden er vist i et "treeview" og brugeren ekspandere et af segmenterne, sådan at undersegmenterne bliver synlige for brugeren.
Primær aktør	LicenseWatch bruger
"Precondition"	Brugeren er inde på segment siden, og data fra databasen er klar, samt der er mindst et segment, der har mindst et undersegment.
"Postcondition"	Segmentet, som brugeren har ønsket ekspanderet, er nu ekspanderet, og det er muligt at se undersegmenterne, og det er synligt at segmenterne hører sammen.
Hoved succes scenarie	1. Systemet viser en "expand" knap, ud for de segmenter der har undersegmenter. 2. Brugeren trykker på en af disse "expand" knapper ud for et segment. 3. Systemet viser nu undersegmenteter for segmentet. 4. Undersegmenterne er placeret under deres "parent" segment og rykket til højre. Udover dette, har systemet tegnet en streg fra "parent" segmentet til undersegmenterne
Udvidelser	1.a der findes ingen undersegmenter og brugeren kan derfor ikke ekspandere et segment.
Hyppighed af brug	Stort set altid. Det er sjældent, at der ikke er mindst et segment, som har et undersegment. Er der et undersegment, er det oftest den, der indeholder

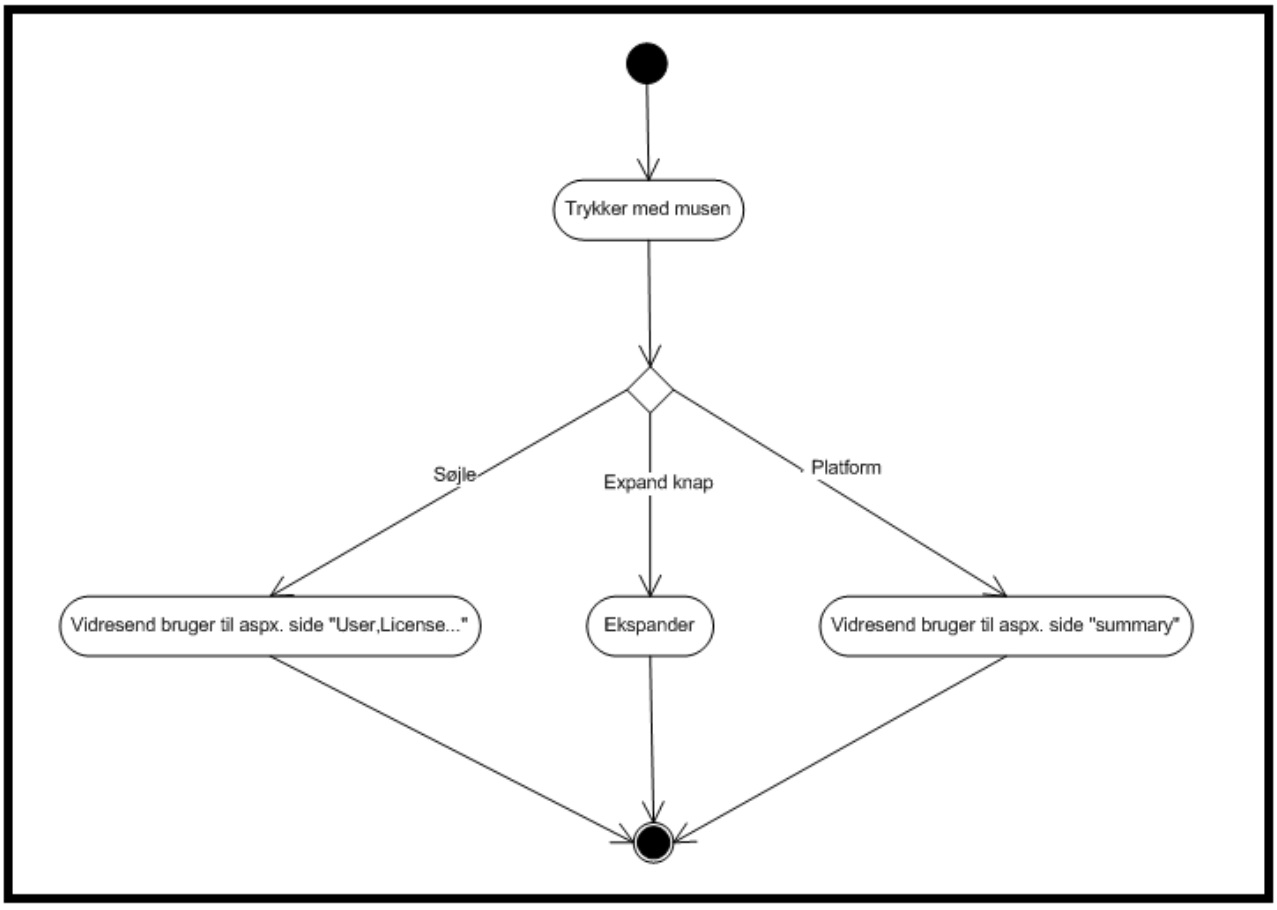
	data, og derfor vil brugeren ekspandere for at se undersegmentet.
Prioritet	4



Figur 4.3 use case UC 3 aktivitets diagram

ID:	UC 4
Titel	Klik på 3D objekter på segment siden
Beskrivelse	Brugeren trykker på f.eks. et segment og bliver sendt videre til en summary side for segmentet(Denne summary side findes ikke i denne prototype, men i det rigtige LW system. Brugeren sendes bare videre til en side, der viser segment id, så muligheden for at "redirecte" brugeren ses.
Primær aktør	LicenseWatch bruger
"Precondition"	Brugeren er på segment siden, og data er hentet fra databasen.
"Postcondition"	Brugeren er blevet videresendt til en ny aspx. side. Her kan brugeren se segmentID for det segment han har trykket på. I det rigtige system ville dette være en summary side.
Hoved succes scenarie	1. Brugeren flytter musen, så den er på et segment platform. 2. Brugeren trykker med venstre museknap. 3. System tjekker om der er blevet trykket på et objekt i 3D scenen. 4. Er der trykket på en platform for et segment, videresender systemet brugeren til en aspx. side, hvor segmentID bliver vist ,for den platform der er trykket på, samt at brugeren ønsker at se summary siden for segmentet.
Udvidelser	1.a Brugeren har også mulighed for at trykke på en søjle eller en "expand" knap. 4.a Der kan være trykket på andet end et segment platform. 4.a1 Er der trykket på en søjle, videresendes brugeren til en side, hvor segmentID på det segment søjlen tilhører bliver vist, sammen med beskrivelsen, om det er en user søjle eller license søjle osv. der er trykket på. 4.a2 Er det en "expand" knap, der er blevet trykket på, sker der det, som er beskrevet i use case UC 3.

Hyppighed af brug	Er det muligt at implementere, vil det blive brugt stort set hver gang. Den største grund til at brugerne benytter siden er for at klikke sig ind på et bestemt segment.
Prioritet	3



Figur 4.4 use case UC 4 aktivitets diagram

I test kapitlet vil disse use case beskrivelser, samt dem i bilag 1. blive taget frem og brugt i test, for at se om systemet er implementeret på en sådan måde, at de forskellige use case beskrivelser er opfyldt.

Overordnet struktur.

I dette afsnit vil visual studio solution strukturen blive beskrevet, samt hvordan de enkelte projekter i denne solution er lavet, og sammenhængen imellem dem vha. klassediagrammer og sekvens diagrammer.

Solution struktur

Solution strukturen i dette projekt er lavet sådan, at der er 4 indre projekter. De 4 projekter er:

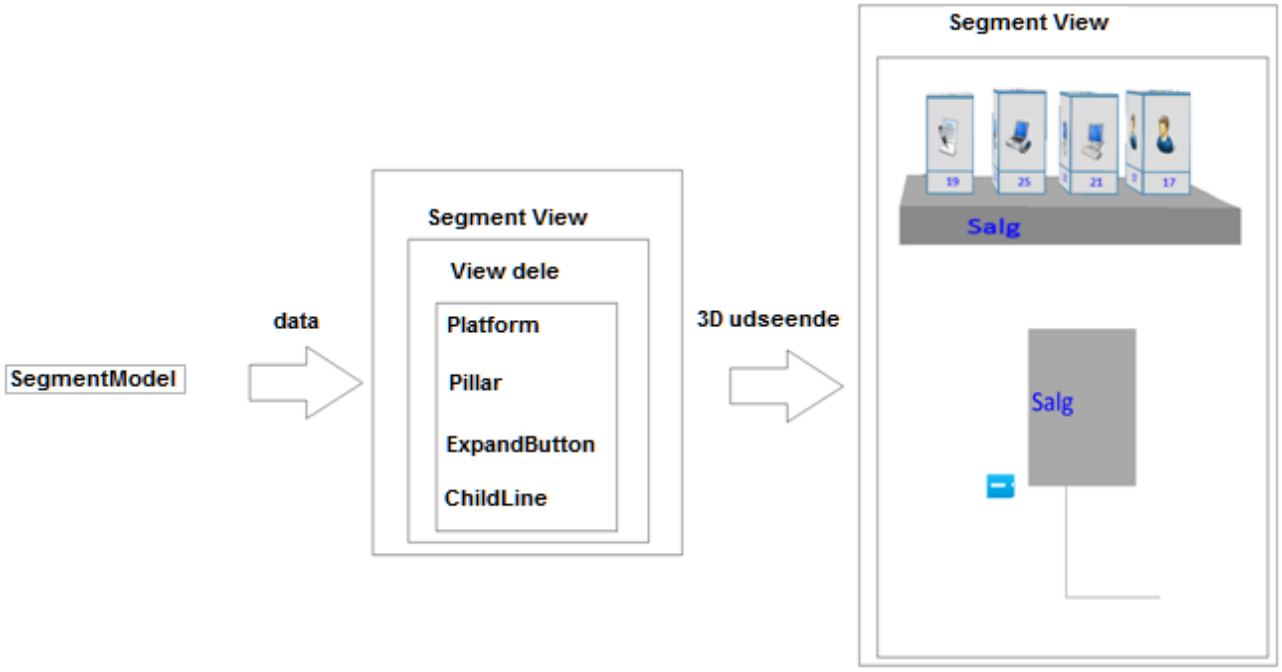
LicenseWatch3D.Web, der er et web projekt, der indeholder aspx. siderne, linq klassen som har forbindelse til databasen, og til sidst en webservice, der bruges til at give data til selve silverlight delen af projektet.

ShaderFilesLibrary, som er projektet, hvor alle shader filerne ligger, samt en klasse, der holder styr på de forskellige uniform resource identifier (URI), for de forskellige shader filer. Grunden til at dette er lavet som et separat projekt er, at skulle man have lyst til at skifte en eller flere af shaderne ud, så kan man nøjes med at opdatere dette projekts dll.

ModelProject, er projektet, der indeholder model klasserne. Da projektet handler om en aspx. side, der bruges til at vise data vedrørende segmenter, indeholder den ene model klasse "SegmentModel" alt data omkring et segment, som skal bruges til vise et segmentet i 3D.

Det sidste projekt *LicenseWatch3D*, er hoved projektet, hvor alt det grafiske sker. Dvs. de forskellige vinduer, der kan vises for brugeren. Udover vinduerne, er det også her, de forskellige views ligger, som bruges til at vise data fra model klassen, samt alt koden der bruges til at omdanne model data til noget, der kan bruges af de forskellige views. I dette projekt er der ikke en 1-1 relation mellem en model klasse og en view klasse, og grunden til dette er, at de forskellige 3D dele af et segment har fået hver deres klasse, og disse klasser udgør så et "samlet" view af segmentet.

Nedenstående billede viser, hvordan relationen mellem model og view er lavet i dette projekt.

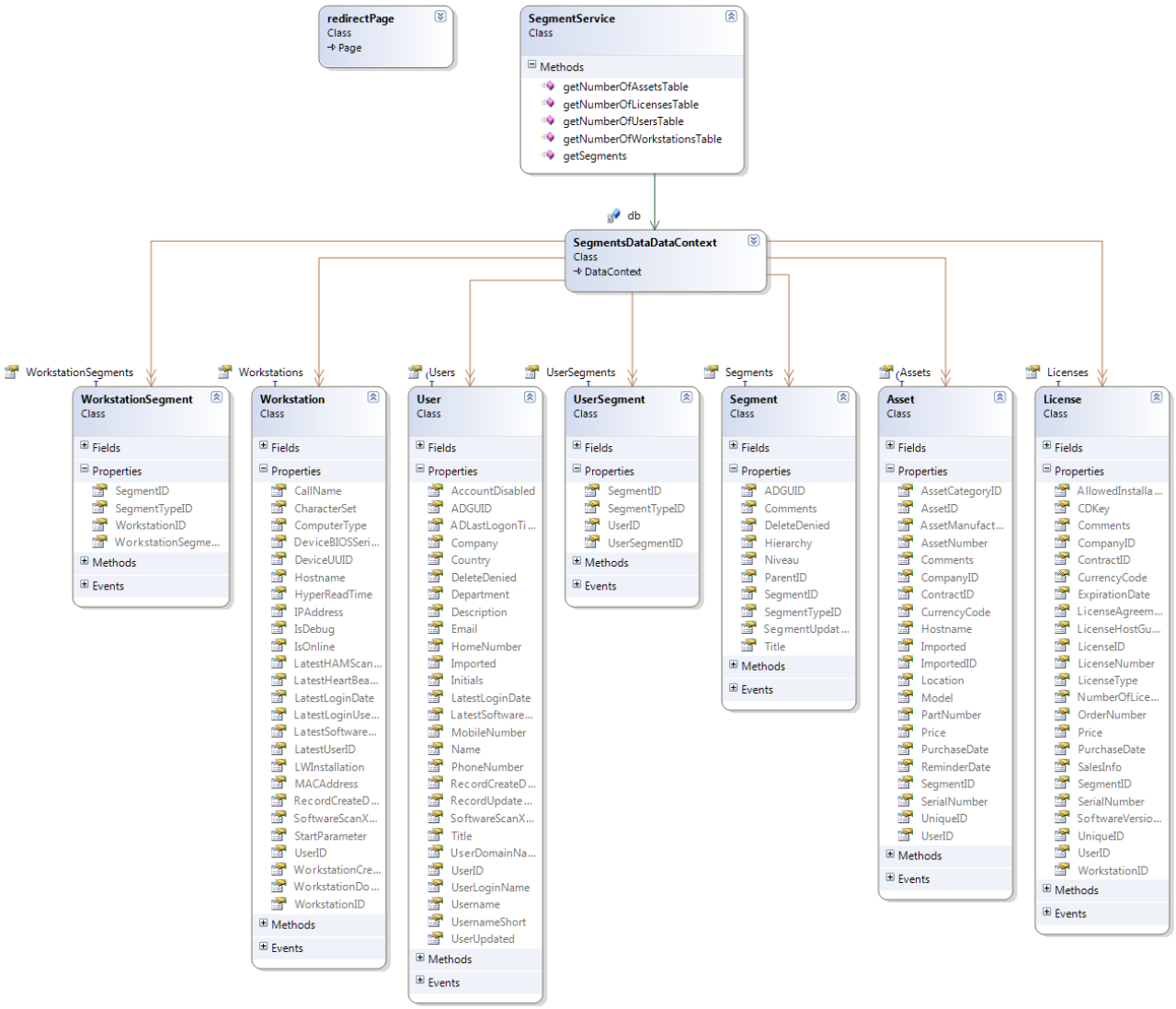


Figur 4.5 Relation mellem model og view klasser.

Overstående figur 4.5 skal forstås på den måde, at en SegmentModel klasse indeholder det relevante data, som skal vises på segment siden. View klasserne: Platform, Pillar, ExpandButton og ChildLine, bruges til visualition af dette data. I projektet er der ingen samlet view klasse, men disse view dele giver tilsammen det ønskede udseende for et segment. Udseendet ses helt til højre, hvor man kan se det fra forskellige vinkler.

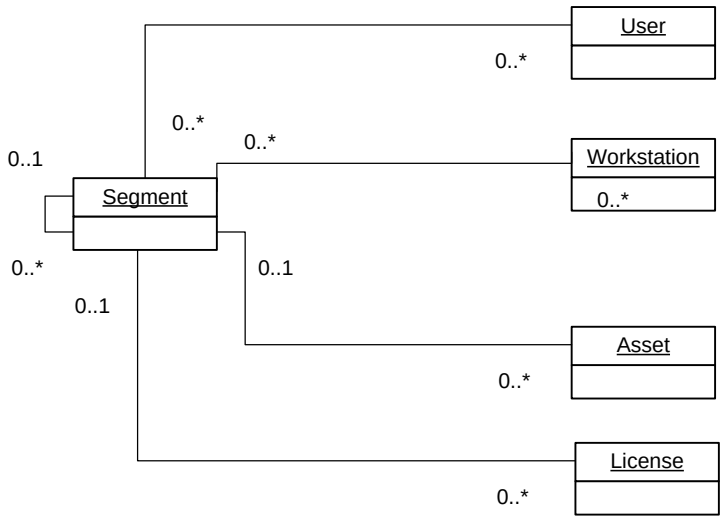
Næstefter tages et kig på opbygningen af de 4 projekter inde i solutionen. Herunder hvilke klasser der er, og sammenhængen mellem klasserne, samt beskrivelser af nogle af de vigtigste klasser.

LicenseWatch3D.Web



Figur 4.6 LicenseWatch3D.web klassediagram

På figur 4.6 ses strukturen i web projektet. Strukturen bærer meget præg af, at der i projektet bliver brugt "LINQ TO SQL" klasser, som forbindelse til databasen. Måden dette virker på er, at der bliver oprettet en klasse for hver af tabel, man vil forbinde til. Denne klasse indeholder så properties, der svarer til de kolonner, der er i tabellen, som den mapper til. I SegmentsDataDataContext klassen findes der så en tabel liste for hver af disse klasser. Til at forstå valget af de valgte tabeller, kigges der på domæne diagrammet for dette projekt, se figur 4.7.



Figur 4.7 Domæne model

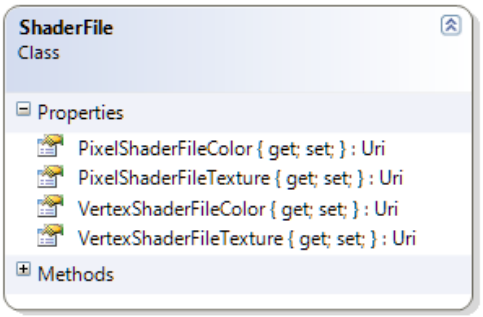
På domæne diagrammet ses, at et segment har 0..* brugere tilknyttet sig, og at brugerne kan være tilknyttet til 0..* segmenter og det samme er gældende for computere. Aktiver og licenser kan kun tilhøre et enkel segment, mens et segment igen kan have 0..* aktiver og licenser. Et segment kan også have undersegmenter og her er multiplicitet 0..*, dog kan et segment kun have én enkel "parent".

Kigges der igen på figur 4.6, ses det, at da brugere og computere har en 0..* relation, så har LicenseWatch lavet to tabeller, UserSegment og WorkstationSegment, der holder styr på disse relationer, og derfor er det vigtigt at have en forbindelse til disse tabeller. Det er dog også nødvendigt, at kende det samlede antal brugere i databasen, og User og Workstation tabellerne, er derfor også nødvendige at mappe til, da disse indeholder denne information. Da segmenter, aktiver og licenser kun kan tilhører et segment, findes deres relation til et segment i deres egne tabeller. I Asset og License tabellerne/klasserne er dette vha. af SegmentID at relationen findes, mens i Segment tabellen/klassen, er det vha. ParentID.

Måden hvorpå Silverlight projektet LicenseWatch3D får fat i data i databasen, er vha. WCF SegmentService klassen, der indeholder de nødvendige metoder for at få fat i det data, som projektet skal bruge. I disse metoder laves der så forskellige LINQ kald til SegmentsDataDataContext klassen.

Den sidste klasse på figur 4.6 er redirectPage, som er en simpel aspx side. Dens formål i projektet er, at efterligne de aspx sider, som LicenseWatch har i deres nuværende system, og som de gerne vil have, at brugeren kan gå direkte til fra segment siden. Dvs. en summary side for et segment, samt user,workstation,asset og license siderne for et segment. RedirectPage har ikke have et stort arbejdsområde, da den kun viser de QueryStrings, som bliver givet til den. Dette indebærer et segmentID og navnet på den side, brugeren ville se. Hvis dette projekt var koblet sammen med LicenseWatch nuværende system, ville denne side være dog være unødvendig.

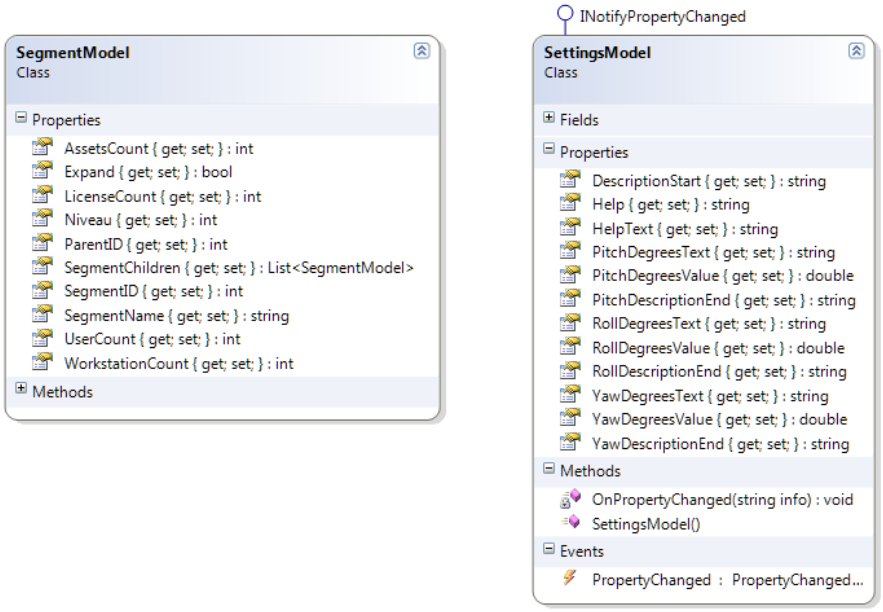
ShaderFilesLibrary



Figur 4.9 ShaderFilesLibrary klassediagram.

I ShaderFilesLibrary projeket er der kun en enkel klasse. Denne ShaderFile klasse holder som sagt styr på de forskellige uniform resource identifier (URI) for de forskellige shader filer, der også ligger i ShaderFilesLibrary projektet. Som beskrevet i kapitel 2 omkring 3D teknologi, så findes der flere forskellige slags shadere. Deriblandt en pixel shader og vertex shader, og disse shadere kan være forskellige, alt efter om de skal bruges sammen med en texture eller en almindelig farve. Derfor ligger der i dette projekt fire forskellige shadere. To pixel shadere, hvor en skal bruges sammen med texture, og en anden med farve. Det samme er gældende for vertex shaderne. I SharderFile klassen er der derfor lavet en Uri property for hver af disse shader filer.

ModelProject



Figur 4.8 ModelProject klassediagram

ModelProject projektet indeholder, som navnet hentyder til, de model klasser, der bruges i dette projekt. Den første model klasse er SegmentModel, og denne indeholder det data, som skal bruges for at lave view klasserne. Se figur 4.5 og 4.10.

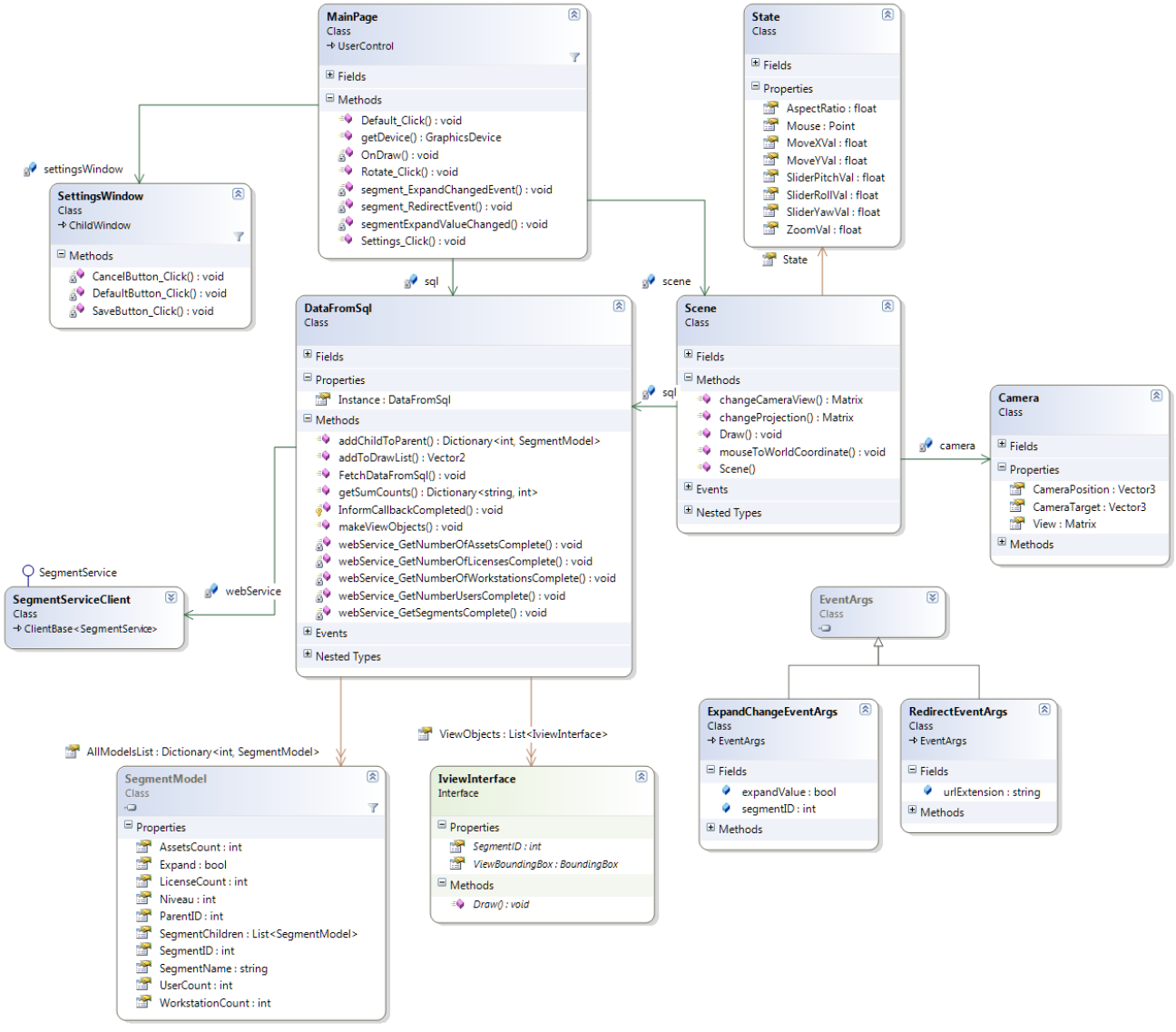
Som beskrevet i kapitel 3, så var en af de nye værdier, LicenseWatch gerne ville give til segment siden den, at brugerne nemt skal kunne se størrelsen på segmenterne. Her menes der antallet af brugere, computere, aktiver og licenser der er tilknyttet et segment. Derfor har SegmentModel en count porperty for disse fire størrelser.

De sidste properties i modellen skal bruges til at vise relationen mellem et segment og dens undersegmenter. Dette skal vises som et "treeview", der minder om den nuværende segment side se figur 3.1, hvor SegmentChildren er listen, hvor et segments "direkte" undersegmenter gemmes, dvs. de segmenter, hvis parentID svarer til segmentets egen segmentID, og derved er der en niveau forskel på segmentet og dens undersegmenter på 1.

Expand property bruges til at bestemme, om man på siden vil vise de segmenter, der ligger i segments SegmentChildren liste. Det er også denne property der bestemmer, hvilket udseende ExpandButton view klassen skal have. Se figur 4.5

Den sidste model klasse er SettingsModel. Denne bruges som model klasse for SettingsWindow klassen, hvor dennes slider værdier og tekster binder til SettingsModel klassens properties. SettingWindow klassen, se figur 4.9, bruges til at ændre på roterings indstillingerne, og derfor indeholder SettingsModel bla. disse roterings indstillinger, såsom yaw,pitch og roll værdierne, der beskriver, hvor meget der skal roteres om de forskellige akser.

LicenseWatch3D



Figur 4.9 LicenseWatch3D klassediagram del 1.

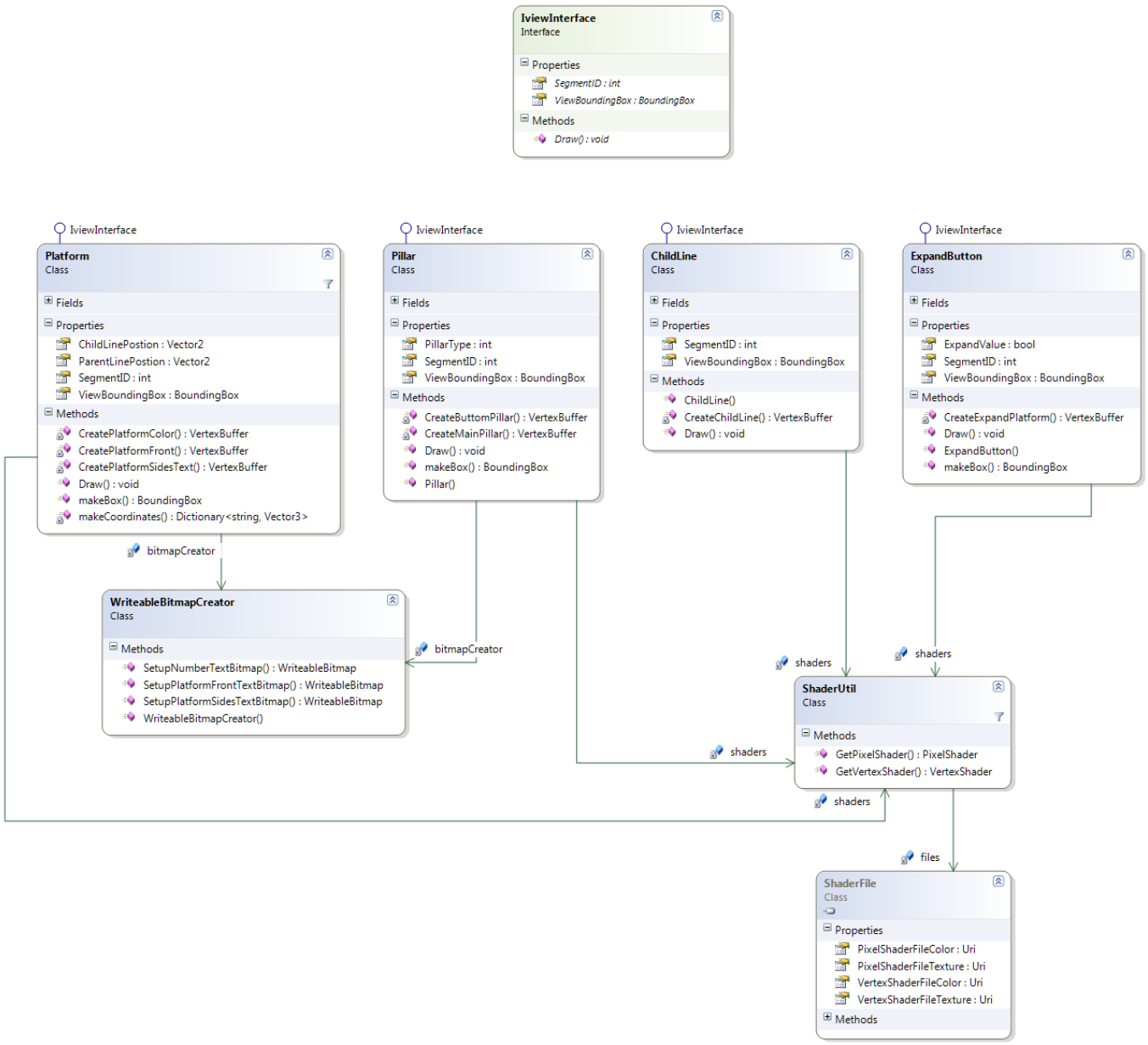
På overstående figur, ses del 1 af klassediagrammet for Silverlight projektet LicenseWatch3D. Da projektet omhandler 3D views, hvor databinding ikke indgår, så har det ikke været muligt at følge

MVVM pattern fuldt, og DataFromSql klassen fungerer derfor som en speciel udgave af en ViewModel klasse.

DataFromSql klassen skal som sagt forstås som projektets ViewModel. Det er denne klasse der får data fra sql vha. SegmentServiceClient klassen. Ud fra dette data laves der instanser af SegmentModel klassen og disse gemmes i AllModelList listen. Efterfølgende konverteres disse SegmentModel informationer om til informationer, der kan bruges til at oprette de forskellige view klasser, der alle implementerer IviewInterface interfacet. Alle view instancerne gemmes i ViewObjects listen, som derved indeholder alt det, der skal tegnes på siden. Eftersom DataFromSql klassen er den, der holder styr på de view objekter, som vises på siden, er det vigtigt, der kun findes en instans af klassen, sådan at der altid arbejdes med den samme ViewObejects liste. En måde hvorpå dette kan opnås, er ved at lave konstruktøren som en singleton. Propertien Instance i DataFromSql giver så adgang til den oprettede instans af klassen.

MainPage klassen indeholder DrawingSurface kontrollen. Denne kontrol har som beskrevet i kapitel 2. et Draw event. I dette event skal Scene klassens Draw metode kaldes, og den sørger så for at kalde Draw metoden for alle IviewInterface objecterne i ViewObjects listen fra DataFromSql klassen. DrawingSurface kontrollens Draw event køres, som beskrevet i kapitel 2 i afsnittet om tråde, på en separat tråd, kaldet composition tråden. Eftersom brugeren ændrer på rotations variablerne og kamera positions variablerne osv. på UI tråden, er der derfor lavet en State klasse, hvis formål er at holde styr på tilstanden af disse variabler, altså hvilken værdi de har. Ideen er så her, at på UI tråden ændres værdierne i State klassen, mens der læses fra State klassen på composition tråden. Da de fleste af disse variabler bruges i scenens draw metode, er det Scene klassen, der har fået en property kaldet State, hvor det så er meningen, at den eneste instans af State klassen er tilgængelig fra. Ændringerne af værdierne i State klassen, sker i MainPage klassen på UI tråden. Dette er muligt, da denne har en instans af Scene klassen.

Interface IviewInterface beskrivelse



Figur 4.10 LicenseWatch3D klassediagram del 2.

På figur 4.10 ses den sidste del af LicenseWatch projektet. Det er her de individuelle view klasser kan ses. De implementer, som beskrevet ovenfor, alle interfacet IviewInterface. IviewInterface består af to properties, som bruges til at implementere use case UC 5, der omhandler at brugeren skal kunne trykke på et objekt i scenen. Den første property ViewBoundingBox skal bruges til at undersøge om der er blevet trykket på et view. Dette vil blive beskrevet i kapitel 5. SegmentID skal bruges, når systemet har fundet ud af, at der er blevet trykket på et view i scenen. Er det et

"ExpandButton" view, brugeren har trykket på, så viser SegmentID, hvilket segment brugeren ønsker at ekspandere, og Expand værdien, i den tilsvarende SegmentModel klasse, som view'et blev lavet af, skal så ændres. Er det et view af typerne Pillar(søjle) eller Platform, så skal SegmentID værdien sendes videre til redirectPage og vises for brugeren.

Draw metoden er den metode, som beskrevet tidligere, bliver kaldt på composition tråden, og som sørger for at rendere selve viewet.

Tages der et kig på de fire view klasser, hvor interfacet er implementeret. Så ses det, at de alle bruger en ShaderUtil klasse. Denne util klasse har en instans af ShaderFile klassen fra ShaderFileLibrary projektet, og på denne måde får view klasserne adgang til de forskellige vertex og pixel shadere, de har behov for.

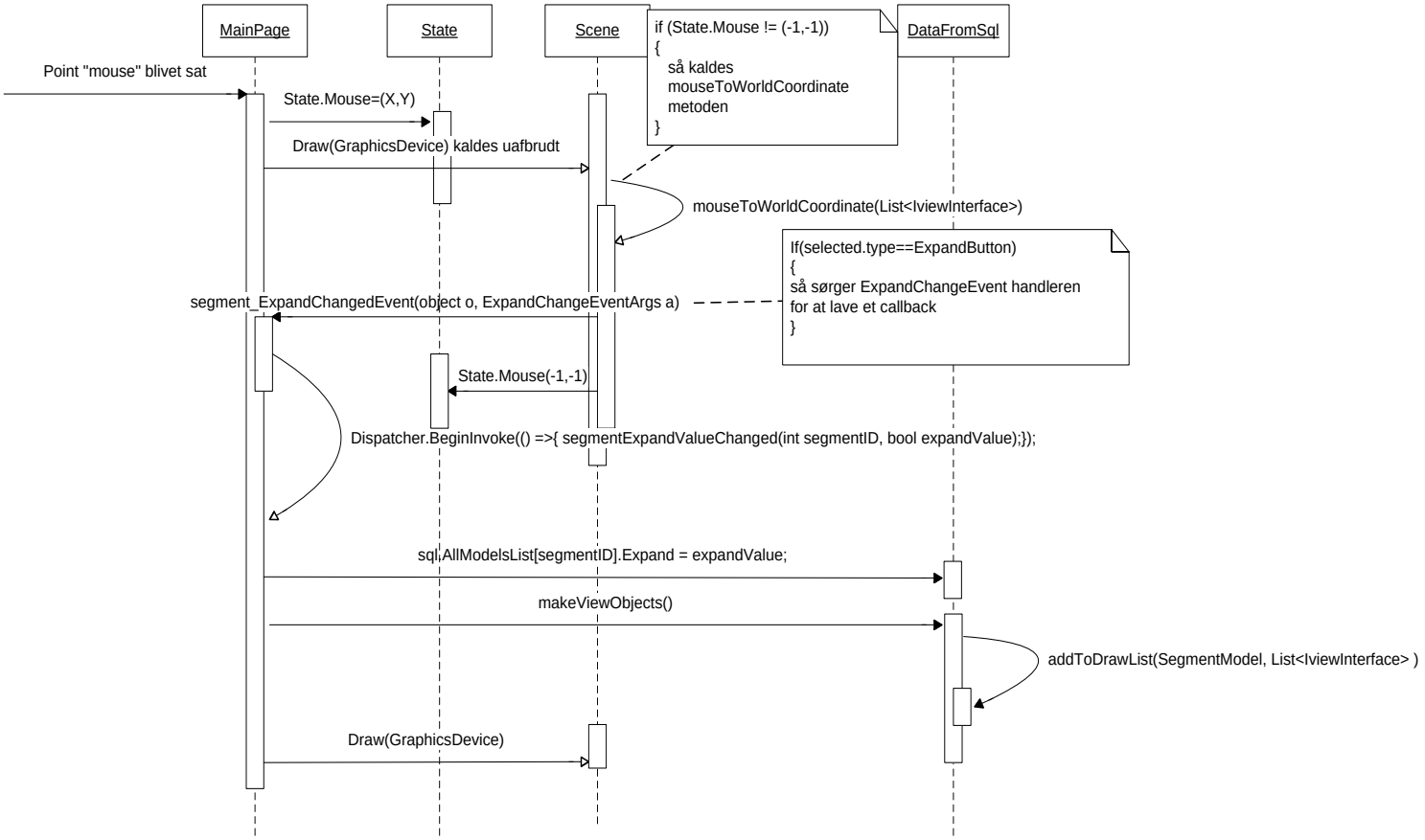
Som det kan ses på figur 4.5, så skal man vise tekst på både platformen og de forskellige søjler for et segment. På platformen skal navnet på segmentet stå, mens der på på søjlerne skal skrives antallet. Som beskrevet i kapitel 2, så kan tekst på 3D objekter kun laves vha. bitmaps, der så bliver brugt som texture og mappet på objektets vertecies. Derfor bruger view klasserne Platform og Pillar en WriteableBitmapCreator klasse. Denne klasse bruges til dynamisk at lave bitmaps ud fra navnet på et segment eller antallet. Da der både skal bruges tekst på platformens forside og sider, se igen figur 4.5, og da disse sider har forskellige dimensioner, så har WriteableBitmapCreator klassen fået en metode, for hver af disse billeder samt en for oprettelse af et billede med antal.

Sekvens diagrammer

I kapitel 2 blev det beskrevet, hvordan DrawingSurface kontrollens Draw event køres i en tråd kaldet composition tråden, og at det er i denne tråd, at selve renderingen foregår. Det blev også beskrevet, at hvis man vil forbedre præstationen af programmet, så skal man sørge for, at det eneste der sker på compositions tråden er, at den tegner de forskellige "primitives" som ens views består af. Alle beregninger og ændringer af data modellerne eller views skal ske på UI tråden eller en anden separat tråd, da composition tråden loopes hele tiden.

Måden hvorpå dette løses i projektet, kan vises ved at se på sekvens diagrammet for use case UC 3. Her er der en bruger, der først trykker på en expand knap, og derefter skal et segment vise dens

undersegmenter. Dette vil sige, at der skal laves beregninger for, hvordan det/de nye views skal se ud, samt hvordan det påvirker de gamle views, med hensyn til forskydning osv. Nedenstående sekvensdiagram viser, se figur 4.11, hvordan dette forløb skal laves, og det viser, hvordan der tages hensyn til, at der skal ske mindst muligt på compositions tråden.



Figur 4.11 Sekvens diagram over UC 3 og UC 4. .

Sekvens diagrammet figur 4.11 er udformet ud fra, at expand knappen er lavet som et view kaldet ExpandButton, bestående af vertecies med 3D koordinater. Derfor er UC 3 magen til UC 4, hvor en bruger trykker på et 3D objekt.

Sekvensen starter ved, at en bruger trykker på DrawingSurface kontrollen med musen. Derefter bliver State klassens Mouse property sat til at være lig musens X og Y koordinater i stedet for default værdien (-1,-1). På compositions tråden, hvor DrawingSurface kontrollens Draw event

håndteres, kaldes Draw metoden i Scene klassen fra MainPage. Det første denne metode gør, er at tjekke, om Mouse Propertien i State, har en anden koordinat end (-1,-1). Har den det, vil det betyde, at brugeren har trykket med musen, og herved kaldes mouseToWorldCoordinate metoden i Scene klassen.

Denne metode udregner vha. muse koordinaterne og de 3 matricer world, view og projection, om brugeren har trykket på et view objekt på siden. Viser det sig, at der er trykket på et view af typen "ExpandButton", så sker der et callback til metoden segment_ExpandChangedEvent i Mainpage klassen vha. event handleren ExpandChangeEvent. Denne metode får som paramenter et ExpandChangeEventArgs argument. Denne indeholder et segmentID og en boolean værdi expandValue, se figur 4.9. Som segmentID bruges værdien af propertien SegmentID, som alle views har implementeret, se firgur 4.10. Som expandValue værdi bruges den modsatte bolean værdi af propertien ExpandValue for det ExpandButton view, der er blevet trykket på.

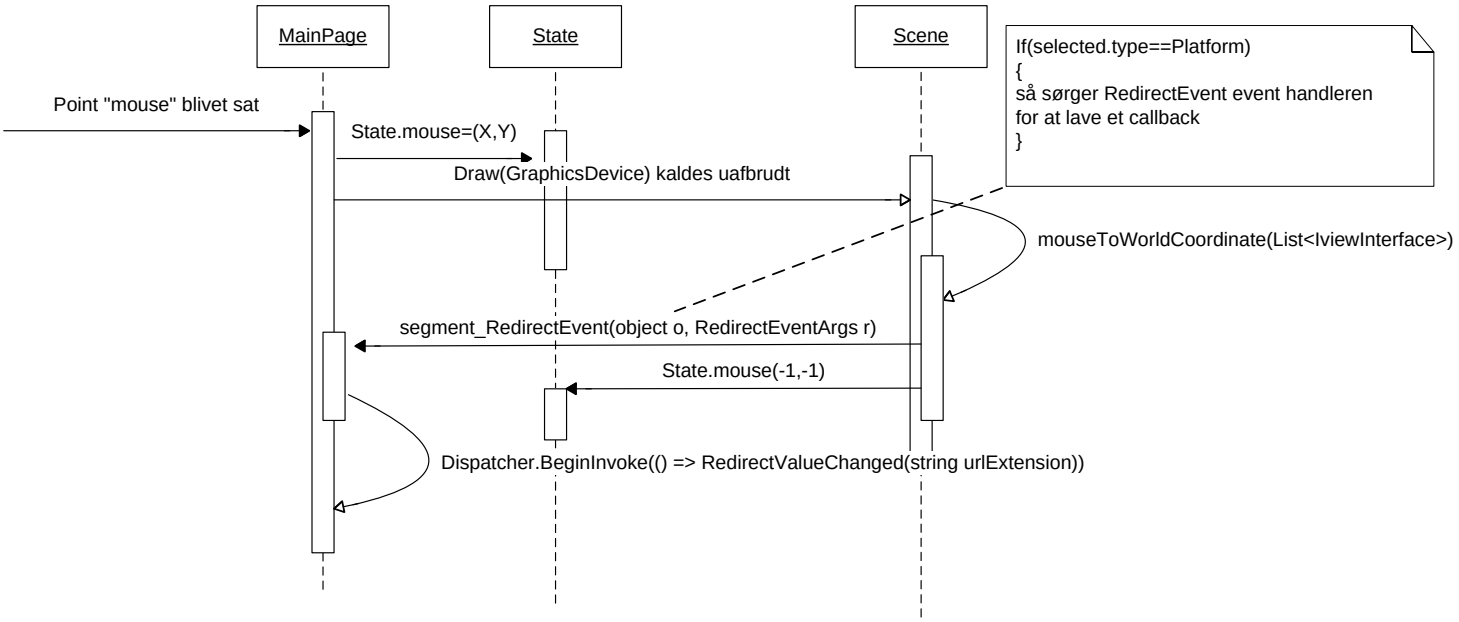
Metoden segment_ExpandChangedEvent bliver som beskrevet tidligere håndteret tilbage i MainPage klassen, hvor der sørges for, at segmentExpandValueChange metoden køres på UI tråden. I metoden sørges der først for, at Expand værdien ændres for den SegmentModel, som ExpandButton view'et er lavet ud fra. AllModelsList er en dictionary liste, der befinder sig i DataFromSql klassen, og som indeholder alle SegmentModel klasserne. Da Dictionary listen er Silverlights svar på en hash tabel, så har listen en key og en value. For denne AllModelsList er SegmentID key, og value er selve SegmentModel klassen for denne SegmentID, se figur 4.9. Derfor kan man,ved at bruge SegmentID fra det view, der er trykket på som nøgle i AllModelsList listen, få adgang til SegmentModel klassens Expand property, og ændre den til en ny værdi. Efterfølgende bliver makeViewObjects metoden kaldt. Denne metode, starter hele forløbet i DataFromSql med at lave view objekter og gemme dem i ViewObjects listen.

Scene klassens Draw metode kaldes uafbrudt i rendering loopet, og så længe Mouse propertien i State klassen er (-1,-1), så laver den ikke ret meget andet, end at kalde alle view objekternes Draw metode, og derved tegne alle primitiverne, som de forskellige view består af.

På denne måde opnås det, at der på compositions tråden ikke laves alt for meget andet arbejde, end at tegne primitiver, som er blevet lavet på UI tråden. Grunden til at mouseToWorldCoordinate metoden dog køres på compositions tråden er, at nogle af de brugte metoder fra GraphicsDevice

enheden, der er krævet til udregningen, kun virker på compositions tråden, samt at det kun sker, når brugeren har trykket med musen, og altså ikke i hver renderings loop. I implementerings kapitlet vil mouseToWorldCoordinate metoden blive beskrevet mere omhyggelig.

Eftersom figur 4.11 viser sekvens diagrammet over, hvad der sker, når der trykkes på en ekspanderings knap lavet i 3D, så er sekvens diagrammerne for, hvad der sker, når en bruger trykker på et andet view, stort set ens. Figur 4.12 viser et sekvens diagram over, hvad der sker, hvis der istedet for en ekspanderings knap, er blevet trykket på et segment.



Figur 4.12 Sekvens diagram over UC 4 med Platform view

I starten sker der det samme som i figur 4.11, og forskellen kommer nede i mouseToWorldCoordinate metoden. Finder metoden ud af, at der er trykket på et segment, dvs. et Platform view, så sker der igen et callback, men denne gang til metoden segment_RedirectEvent i MainPage klassen vha. event handleren RedirectEvent. Denne metode tager som parameter et RedirectEventArgs argument, se figur 4.9, der indeholder en urlExtension variabel. Når det er et Platform view, der trykkes på, indeholder urlExtension SegmentID fra view'et. Måden hvorpå RedirectEvent behandles i MainPage, er ligesom ved ExpandChangedEvent, se figur 4.11. Her sørges der også for, at der skiftes fra compositions tråden

over til UI tråden, da det ikke er meningen, at composition tråden skal sørge for at åbne en ny side. Siden der åbnes er RedirectPage.aspx, se figur 4.6, og denne side læser så de parameter, som urlExtension indeholder. Var det et Pillar view, der blev trykket på, så ville urlExtension indeholde en parameter mere kaldet "Page", og denne henviser til, om det er "User,Asset,Workstarion eller License" siden brugeren vil se.

Kapitel resume af design

I dette kapitel er der blevet kigget på projektets use cases, og disse er både blevet analyseret og beskrevet. Det er beskrevet hvilken prioritet og kompleksitet, som hører til de forskellige use cases. UC. 3, der omhandler ekspandering af segmenter, og UC 4, som omhandler klik på 3D objekter på siden, er derudover blevet vist som fully dressed use cases, da disse er nogle af de mere komplekse ud fra analysen af dem.

Udover at kigge på use cases, er den ønskede struktur af programmet også beskrevet vha. af klasse diagrammer over de forskellige solution projekter. Det blev her vist, at da der bliver arbejdet med 3D view, så kunne MVVM design pattern ikke følges fuldt ud, og DataFromSql klassen, der er projektets ViewModel, er derfor en lidt speciel ViewModel.

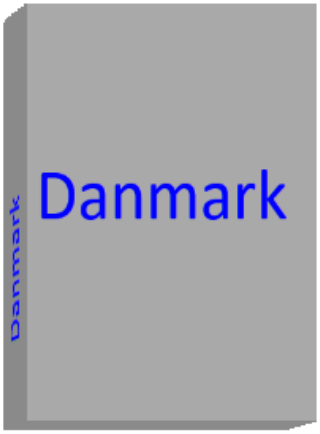
Til sidst blev der kigget på, vha. sekvens diagrammer, hvordan arbejdet med de to tråde: UI tråden og compositions tråden, håndteres i prototypen. Her blev det vist, at compositions tråden ikke laver andet end at tegne de forskellige views som gemmes i ViewObject listen, og tjekke om brugeren har trykket på noget på scenen. UI tråden sørger derimod for at oprette nye views samt at ændre data i SegmentModel klasserne og herved ændre udseende på eksisterende views. Herved opfylder designet arbejdet med de to tråde, sådan at der opnås en optimal præstation af systemet, som beskrevet i tråde afsnittet i kapitel 2 omkring Silverlight 3D.

Kapitel 5. Implementering

I dette kapitel vil nogle af de vigtigste implementeringsemner i projektet blive beskrevet. Som det første beskrives, hvordan de forskellige views bygges op, og hvordan disse repræsenteres vha. vertexPositionTexture og vertexPositionColor. Efterfølgende beskrives det, hvordan visningen af tekst på siden laves vha. WriteableBitmap klassen. En beskrivelse af opsætningen af de 3 vigtige matricer, world view og projection vil også blive givet i forhold til, hvodan disse laves i projektet. Derudover beskrives, hvordan værdierne fra model klasserne bruges til at lave tilhørende view objekter. Til sidst vises også, hvordan det i prototypen er gjort muligt for brugerne af siden, at trykke på 3D objekter på siden.

Opbygning af 3D views

Platform



Figur 5.1 Plartform view

På x-aksen er den 9 bred. På y-aksen er den 13 høj. På z-aksen er den 1.5 dyb.

Den består af 3 vertexbuffere, da dens vertecies bliver mappet sammen med to forskellige texture billeder samt en farve.

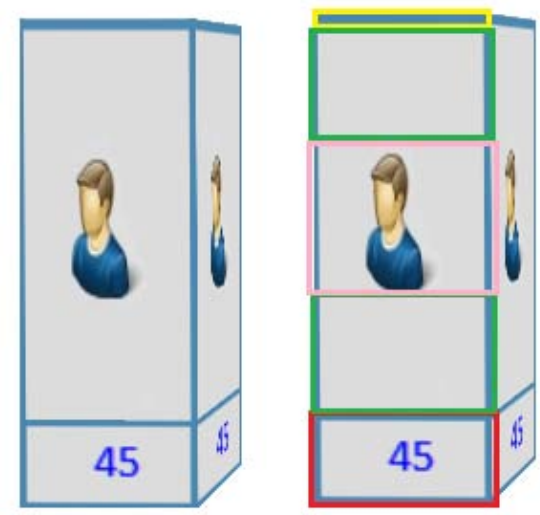
Den første vertexbuffer er den, som står for at mappe forside koordinaterne sammen med et billede, hvor navnet på platformen står.

Den anden vertexbuffer står også for at mappe vertecies med et billede, hvor navnet på segmentet står, men dette er et andet billede. Grunden til dette er, at siderne, hvor billedet skal bruges, har andre dimensioner end forsiden, og derfor er billedet lavet til at passe med sidernes

dimensioner. Derudover har det en mere mørkegrå farve, sådan at side kanterne bedre kan ses, i forhold til hvis de havde samme farve som forsiden.

Den sidste vertexbuffer mapper ikke vertecies med en texture, men derimod en farve. Den står for at give farve til bagsiden, top og bund. Bagsiden får samme grå farve som forsiden, mens top og bund får samme mørkegrå farve som siderne, hvor navnet på segmentet står.

Pillar (søjle)



Figur 5.2 Pillar view

Det første der skal siges om søjlerne er, at koordinat systemet for denne er roteret, da de står på bagsinden af segment platformene. Dvs. at højden er på z-aksen, bredden på y-aksen og dybden er angivet på x-aksen.

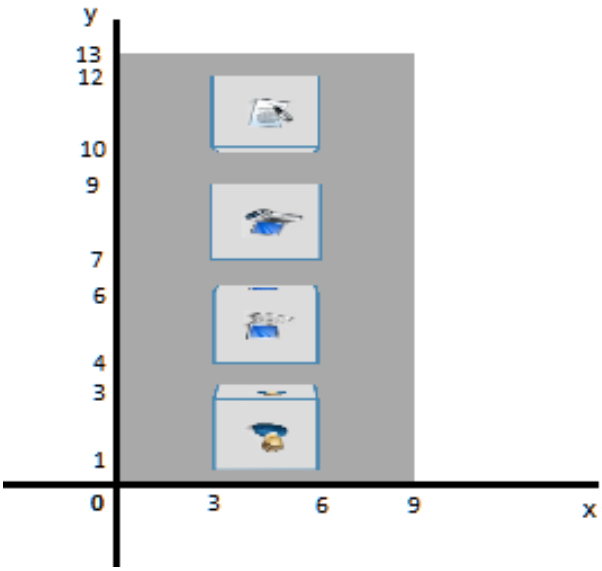
De har en dynamisk højde, som bliver beregnet ud fra, hvor stor en procent del de har af det samlede antal. Dog skal det siges, at højden på søjlen som minimum, så længe procentdelen er >0, vil være 3,5. Ser man på søjlen til højre på figur 5.2, med de farvede firkanter, så viser disse firkanter, hvordan en søjle er inddelt. Bunden, som er den røde firkant, er konstant 1 høj, den lyserøde firkant, der indeholder user ikonet, er altid 2 høj. På den måde blive ikonet ikke strukket eller krympet, selvom søjlen er meget høj eller lav. Den gule firkant i toppen, er så der, hvor de sidste 0.5 ligger. Dette gør, at den blå kant altid har den samme størrelse. Herved har søjlen altså en minimum højde på 3,5. Det er så de grønne firkanter, der dynamisk bliver strukket, alt efter hvor høj søjlen skal være. Ved 100 % udgør de grønne firkanter en højde på 20. Bruges et eks. med

en søjle, som har 50 % af antallet, har denne en samlet højde på 13.5. Dette kommer af, at de grønne firkanter hver er 5 høje, og søjlen bliver i alt:
1(rød)+5(grøn)+2(lyserød)+5(grøn)+0.5(gul)=13.5 høj.

Bredden på y-aksen er sat til en konstant værdi på 2, mens dybden på x-aksen er sat til en konstant værdi på 3.

I forhold til platform view'et arbejdes der her med to vertexbuffere. Den ene vertexbuffer står for bunden af søjlen, da bundens vertecies her skal mappes sammen med en texture bestående af et WriteableBitmap billede, hvor antallet af tilhørende segmentet er skrevet. Den anden vertexbuffer står så for at mappe de resterende vertecies, dvs. de vertecies som udgør den lyserøde, den gule og de grønne firkanter, sammen med en texture bestående af det billede, der passer til søjle typen. Se figur 3.2.

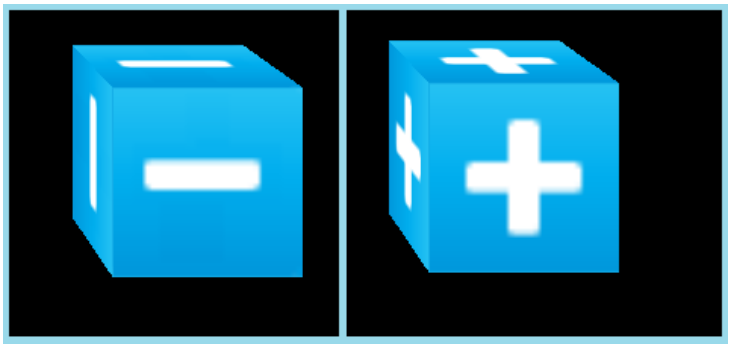
Måden hvorpå søjlerne er placeret på platformen kan ses på nedenstående billede:



Figur 5.3 Pillar view placering på platform

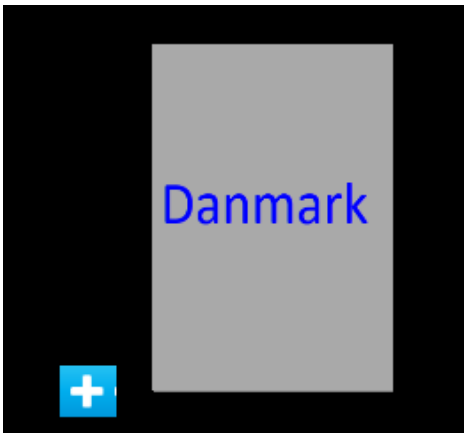
På billedet figur 5.3, ses en platform bagfra. Søjlerne ses derved fra oven, og her ses, hvordan søjlerne er placeret på platformen. Platformen har en bredde på 9 på x-aksen, og søjlen har som sagt en længde 3 på x-aksen. Derfor er søjlerne placeret på midten af platformen i forhold til x-aksen. På y-aksen er søjlerne placeret med en enhed på 1 imellem sig, og en enhed på 1 fra platformens top til den øverste søjle og fra den nederste søjle til platformens bund.

ExpandButton



Figur 5.4 ExpandButton View

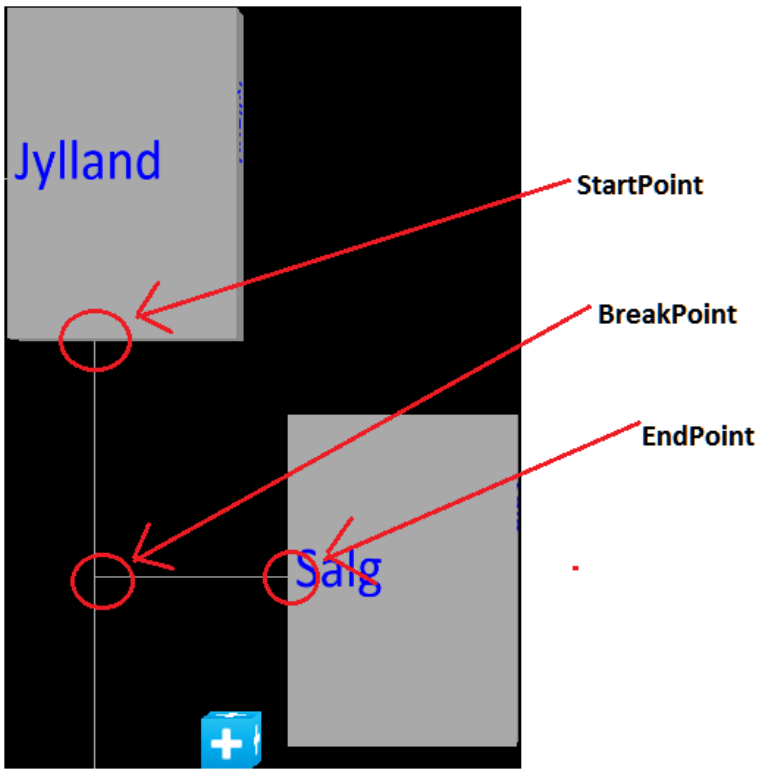
Dette er en kvadratisk kube med en sidelængde på 2. Placeringen af "expand" knappen er til venstre for den platform den hører til, og forskudt en smule ned på y-aksen, og en smule tilbage på z -aksen, sådan at den er nemmere at se, når der er roteret. På nedenstående billede, se figur 5.5, kan forskydningerne på x og y akserne ses:



Figur 5.5 placering af ExpandButton View

ExpanButton består af en enkel vertexbuffer. Dog er det forskelligt, hvilken texture dens vertecies bliver mappet sammen med. Er segmentet udvidet, vises der et minus, og er segmentet ikke udvidet, vises et plus, se figur 5.4.

ChildLine



Figur 5.6 ChildLine view

Denne linje går fra et parent segment til et undersegment(child). Start positionen er sat til at gå fra bunden af parent segmentet, med en forskydning på 3 på x-aksen, i forhold til nederste venstre hjørne af segmentet. Den rammer så undersegmentet midt på dennes venstre side. Break positionen findes så ud fra BreakPoint(StartPoint.X,EndPoint.Y), se figur 5.6.

ChildLine view'et består som ExpandButton, kun af en enkel vertexbuffer, men i forhold til ExpandButton, bliver dennes vertecies mappet sammen med en farve. Som det er nu, er denne farve sat til at være den samme grå farve, som bruges på for og bagside af Platform view.

Vertex positioner

Eftersom overstående views tilsammen udgør udseendet for ét segment på 3D siden, så gav det mest mening, at have én Vector3 variabel kaldet segmentPosition, som bestemmer placeringen af de forskellige view objekter i 3D verdenen tilhørende et segment. Metoden addToDrawList, der står får at oprette de individuelle views, er så metoden, der sørger for at opdatere denne Vector3 variabel alt efter, hvor mange segmenter, der skal vises på scenen. Helt præcis hvordan metoden

opdaterer variablen og laver views, bliver beskrevet senere i kapitlet, mens dette afsnit vil koncentrere sig om at vise, hvordan de forskellige view objekters vertecies er lavet i forhold til denne segmentPosition variabel, i deres lokale koordinatsystem.

Da platform view'et er det eneste view, der med sikkerhed skal laves for alle segmenter, som vises på siden, var det oplagt at tage udgangspunkt i et punkt fra dette view. Eftersom søjlerne vises på platformens bagside, og expand knappen vises nederst til venstre for en platform, så blev det valgt, at segmentPosition variablen skulle repræsentere, hvor platformens nederste venstre hjørne på dennes bagside er placeret i verdenskoordinatsystemet. De resterende vertecies/punkter for platformen og de andre views, bliver så sat ud fra, hvor de skal placeres i forhold platformens nederste venstre hjørne på bagsiden.

Nedenstående figur 5.7 viser, hvordan platformens 8 vertex positioner i dens lokale koordinatsystem bliver sat.

```
//forside
Vector3 platformLeftBottomFront = new Vector3(0.0f, 0.0f, 1.5f);
Vector3 platformLeftTopFront = new Vector3(0.0f, 13.0f, 1.5f);
Vector3 platformRightBottomFront = new Vector3(9.0f, 0.0f, 1.5f);
Vector3 platformRightTopFront = new Vector3(9.0f, 13.0f, 1.5f);
//bagside
Vector3 platformLeftBottomBack = new Vector3(0.0f, 0.0f, 0.0f);
Vector3 platformLeftTopBack = new Vector3(0.0f, 13.0f, 0.0f);
Vector3 platformRightBottomBack = new Vector3(9.0f, 0.0f, 0.0f);
Vector3 platformRightTopBack = new Vector3(9.0f, 13.0f, 0.0f);
```

Figur 5.7 Platform view

Her ses at platformLeftBottomBack vertex positionen bliver sat til (0,0,0) i dens lokale koordinatsystem. Dette gør, at når der senere laves en translation matrix vha. Matrix.CreateTranslation(segmentPosition), og denne bruges i world matricen, så bliver platformLeftBottomBack placeret på segmentPosition i verdenskoordinatsystemet. De andre vertex positioner er så lavet udfra, at der ønskes en platform, der er 9 bred, 13 høj og 1.5 dyb, som beskrevet tidligere i kapitlet.

Vertex positionerne for et Pillar view er lidt mere besværlige, da der, som det ses på figur 5.2, bruges to forskellige texture billeder, og disse billeder skal ikke mappes sammen med de samme vertecies. Derfor bliver disses vertex positioner lavet for hvert billede, samt oprettet i to forskellige metoder, der står for at lave to forskellige vertexbuffere, hvor vertecies for de to billeder gemmes. På figur 5.8 ses, hvordan vertex positionerne, som de to billeder skal mappes sammen med, bliver udregnet. Øverst i den blå firkant ses punkterne, som det specifikke søjle billede, dvs. det af en user, workstation, asset eller license, mappes sammen med. Nederst i den røde er det punkterne, der bruges til bunden af søjlen, hvor et WriteableBitmap bruges til at vise det tilhørende antal.

Anvendes sammen med det specifikke søjlebillede

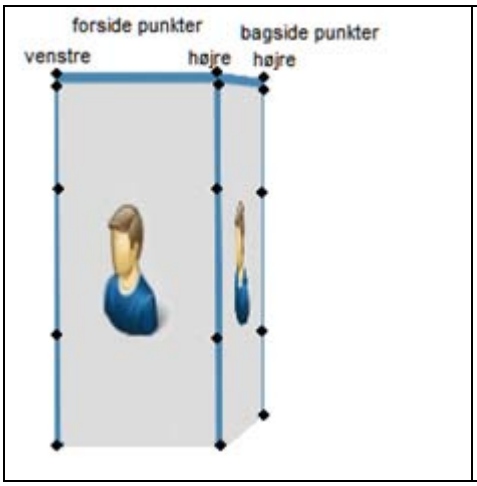
```
//Forside
Vector3 topLeftFront = new Vector3(3.0f, 3.0f, -1 - (height / 2) - 2 - (height / 2) - 0.5f);
Vector3 topMidLeftFront = new Vector3(3.0f, 3.0f, -1 - (height / 2) - 2 - (height / 2));
Vector3 midTopLeftFront = new Vector3(3.0f, 3.0f, -1 - (height / 2) - 2);
Vector3 midBottomLeftFront = new Vector3(3.0f, 3.0f, -1 - (height / 2));
Vector3 bottomLeftFront = new Vector3(3.0f, 3.0f, -1);

Vector3 topRightFront = new Vector3(3.0f, 1.0f, -1 - (height / 2) - 2 - (height / 2) - 0.5f);
Vector3 topMidRightFront = new Vector3(3.0f, 1.0f, -1 - (height / 2) - 2 - (height / 2));
Vector3 midTopRightFront = new Vector3(3.0f, 1.0f, -1 - (height / 2) - 2);
Vector3 midBottomRightFront = new Vector3(3.0f, 1.0f, -1 - (height / 2));
Vector3 bottomRightFront = new Vector3(3.0f, 1.0f, -1);
//Bagside
Vector3 topLeftBack = new Vector3(6.0f, 3.0f, -1 - (height / 2) - 2 - (height / 2) - 0.5f);
Vector3 topMidLeftBack = new Vector3(6.0f, 3.0f, -1 - (height / 2) - 2 - (height / 2));
Vector3 midTopLeftBack = new Vector3(6.0f, 3.0f , -1 - (height / 2) - 2);
Vector3 midBottomLeftBack = new Vector3(6.0f, 3.0f, -1 - (height / 2));
Vector3 bottomLeftBack = new Vector3(6.0f, 3.0f, -1);

Vector3 topRightBack = new Vector3(6.0f, 1.0f , -1 - (height / 2) - 2 - (height / 2) - 0.5f);
Vector3 topMidRightBack = new Vector3(6.0f, 1.0f , -1 - (height / 2) - 2 - (height / 2));
Vector3 midTopRightBack = new Vector3(6.0f, 1.0f, -1 - (height / 2) - 2);
Vector3 midBottomRightBack = new Vector3(6.0f, 1.0f, -1 - (height / 2));
Vector3 bottomRightBack = new Vector3(6.0f, 1.0f, -1);
```

Anvendes sammen med writeableBitmap billedet med antallet

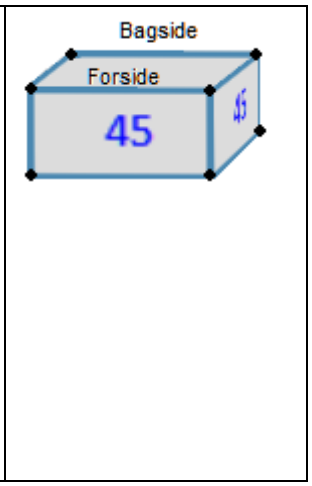
```
//Forside
Vector3 topLeftFront = new Vector3(3.0f, 3.0f, -1.0f);
Vector3 bottomLeftFront = new Vector3(3.0f, 3.0f, 0.0f);
Vector3 topRightFront = new Vector3(3.0f, 1.0f, -1.0f);
Vector3 bottomRightFront = new Vector3(3.0f, 1.0f, 0.0f);
//Bagside
Vector3 topLeftBack = new Vector3(6.0f, 3.0f, -1.0f);
Vector3 bottomLeftBack = new Vector3(6.0f, 3.0f, 0.0f);
Vector3 topRightBack = new Vector3(6.0f, 1.0f, -1.0f);
Vector3 bottomRightBack = new Vector3(6.0f, 1.0f, 0.0f);
```

Figur 5. 8 Pillar vertex positioner

Kigger man igen på figur 5.2, så ses det, at søjlens øverste del, hvor det specifikke søjle billede bruges, er delt i fire dele. Derfor skal der bruges 20 Vector3 positioner, og placeringen af disse kan ses på figur 5.8(blå firkant) og 5.9 (minus bagsidens venstre punkter).



Figur 5.9



Figur 5.10

Forklaringen på vertex positionerne for den øverste del er, se igen figur 5.8 (blå firkant), at igen bliver de sat ud fra, hvor de skal være i forhold til platformens nederste venstre hjørne på dens bagside. Da søjlen skal have en dybde på 3 på x-aksen, som beskrevet tidligere i kapitlet, og som det kan ses på figur 5.3, hvor søjlernes placering på platformen ses, så har søjlen en x værdi på enten 3 eller 6.

Søjlen skal derudover have en bredde på 2, og bredden for søjlen er, som beskrevet, vist på y aksen. Derfor har y værdierne en værdi på 1 eller 3, da dette svarer til, hvor den nederste søjle skal placeres bag på en platform, se igen figur 5.3. Måden hvorpå de resterende søjler placeres ovenover hinanden i verdenskoordinatsystemet er, vha. af en offset værdi. Denne bruges sammen med segmentPosition variablen i oprettelsen af en translation matrix, der sørger for at placere søjlerne rigtigt i forhold til hinanden i verdenskoordinatsystemet.

Offset værdien bruges som sagt til at rykke søjlens lokale koordinater, sådan at den bliver placeret det rigtige sted på platformen i verdens koordinatsystemet, For en user søjle er offset 0 og derfor placeres denne i bunden, se figur 3. For en workstation søjle er offset 1, og den placeres ovenover user søjlen. Dernæst kommer asset søjlen med en offset på 2, og til sidst license søjlen med en offset på 3. Det er addToDrawList metoden der styrer, hvilke søjler der skal laves, og derfor er det

den, der sørger for at give de rigtige offset værdier videre til de forskellige Pillar view typer, når disse oprettes.

Z værdien for vertex positionerne er, som beskrevet, den, der beskriver højden på søjlen, og her skal det bemærkes, at det er ud af den negative z-akse, at søjlen placeres. Da søjle delen på figur 5.9, skal placeres ovenpå en søjle bund, se figur 5.10, hvor antallet er skrevet, og hvis vertex positioner kan ses nederst på figur 5.8 i rød firkant, så er det lavet sådan, at bottomRightFront for den øverste del, er lig med topRightFront for den nederste del, og det samme er gældende for de andre bottom positioner. Da bunden skal have en højde på 1 på den negative z-akse, dvs. -1, så har alle z værdierne for bottom positionerne i den blå firkant en værdi på -1. De efterfølgende vertex positioner højere oppe på søjlen, udregnes så ud fra den overstående Pillar view beskrivelse, der blev beskrevet tidligere i kapitlet, se under figur 5.2.

Nederst på figur 5.8 (rød fikant), ses det som sagt, hvordan vertex positionerne for søjle bunden er sat, se figur 5.10. Beskrivelsen af de valgte X og Y værdier er magen til den for den øverste del. Da søjle bunden som sagt, skal have en højde på 1, så skal der være en forskel på 1 mellem z værdierne for ”bottom” positionerne og ”top” positionerne for bunden. Da søjlen skal stå på en platform view, hvis bagside har en z-værdi lig 0, så har søjlen altså z-værdien 0 for bottom positionerne, og derfor får top positionerne altså en z-værdi på -1.

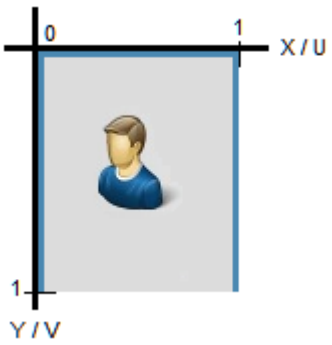
De to view ExpandButton og ChildLine, hvis vertex positioner ikke er blevet beskrevet her, kan ses i bilag 2 "vertex positioner".

UV og color mapping

I kapitel 2, blev det beskrevet, at i silverlight bliver en vertex defineret ved en struktur, der både gemmer vertex data, og en VertexDeclarations object, der indeholder information til GraphicsDevice enheden. Vertex data informationerne afhænger af, om strukturen er en VertexPositionTexture, hvor data er en Vector3 Position og en Vector2 UV eller en VertexPositionColor, hvor data er Vector3 Position og en Color Color.

I forige afsnit "udregning af vertex positioner", blev det beskrevet, hvordan Vector3 Position propertien blev udregnet/sat for de forskellige views. I dette afsnit vil UV koordinaterne og Color værdierne blive beskrevet, og det vises hvordan disse sættes sammen med positionerne.

I kapitel 2 i "Texture" afsnittet, blev det beskrevet, at når man skal lægge et texture billede på en 3D overflade, så kaldes dette for ”UV mapping”, og UV koordinaterne, skal forstås som x og y koordinaterne for det billede, der bruges som texture, og disse går altid fra 0-1. På figur 5.11 ses, hvordan UV koordinaterne er for et billede.



Figur 11 UV koordinater for et søjle billede.

Billedet viser UV koordinaterne for user søjle billedet og på figur 5.9, ses søjle delen, hvor texture billedet er puttet på overfladerne. Her kan man også se, at der bliver brugt 10 vertex positioner til at beskrive en søjle overflade. Dvs. at der også skal bruges 10 UV koordinater, som vertex positionerne kan blive ”mapped” sammen med. For Pillar view delene, der vises på figur 5.9 og 5.10, kan UV koordinaterne ses på nedenstående billeder.

```
//2D koordinater til søjlens billede
Vector2 topLeftUV = new Vector2(0.0f, 0.0f);
Vector2 topRightUV = new Vector2(1.0f, 0.0f);

Vector2 topMidLeftUV = new Vector2(0.0f, 0.1f);
Vector2 topMidRightUV = new Vector2(1.0f, 0.1f);

Vector2 midTopLeftUV = new Vector2(0.0f, 0.25f);
Vector2 midTopRightUV = new Vector2(1.0f, 0.25f);

Vector2 midBottomLeftUV = new Vector2(0.0f, 0.75f);
Vector2 midBottomRightUV = new Vector2(1.0f, 0.75f);

Vector2 bottomLeftUV = new Vector2(0.0f, 1.0f);
Vector2 bottomRightUV = new Vector2(1.0f, 1.0f);
```

Figur 5.12 UV koordinater for specifikt søjle billede

```
//2D koordinater til søjle bundens billede
Vector2 topLeftUV = new Vector2(0.0f, 0.0f);
Vector2 topRightUV = new Vector2(1.0f, 0.0f);
Vector2 bottomLeftUV = new Vector2(0.0f, 1.0f);
Vector2 bottomRightUV = new Vector2(1.0f, 1.0f);
```

Figur 5.13 UV koordinater til WriteableBitmap billed

Måden hvorpå disse UV koordinater og vertex positioner bliver ”mapped” sammen, er så ved hjælp af VertexPositionTexture, hvis konstruktør kræver en Vector3 position værdi og en Vector2 UV værdi. Her er det vigtigt at vide, at GraphicDevice enheden i silverlight hovedsagelig arbejder

med retvinklede trekanter. Dvs. at skal der laves en firkant, som består af 4 punkter, så skal de retvinklede trekanter, der tilsammen udgør firkanten, gives til GraphicsDevice enheden. Eftersom søjledelen på figur 5.9, består af fem overflader, hvor fire af dem: forside, bagside, venstre og højre side, hver består af fire retvinklede firkanter, se figur 2, der igen udgøres af to retvinklede trekanter, og som hver skal angives med tre VertexPositionTexture objekter, så er resultatet altså, at der skal bruges $4 \cdot 4 \cdot 2 \cdot 3 = 9$ (VertexPositionTexture objekter for disse fire sider. Udover dette, så har søjledelen også en top overflade, der består af en enkelt firkant. Denne udgøres igen af to retvinklede trekanter, der skal repræsenteres med tre VertexPositionTexture objekter dvs. der skal bruges$1 \cdot 1 \cdot 2 \cdot 3 = 6$ VertexPositionTexture objekter. Alt i alt skal der altså 102 VertexPosition-Texture objekter til, før at graphicsDevice enheden kan finde ud af at tegne søjle delen på figur 5.9. Da det er muligt, at der skal tegnes fire søjler for hvert segment, der vises på scenen, kan man altså ende op med adskillige VertexPositionTexture objekter, men da disse objekter er af typen struct, som er et letvægt objekt, der ikke bruger så meget hukommelse, så ender man ud med et bedre slutprodukt, end hvis VertexPositiontexture var lavet som en klasse.

Alle disse VertexPositionTexture objekter gemmes så i et array, hvor det er vigtigt, at de tre objekter, der udgør en retvinklet trekant, indsættes efter hinanden. Det er så dette array, der udgør VertexBufferens data. I bilag 3 UV ”mapping” med VertexPositionTexture, kan der ses et uddrag fra koden, hvor denne ”uv mapping” sker vha. VertexPositionTexture. Skal der ikke laves en ”mapping” mellem vertex positioner og en texture, men derimod med en farve, så gøres det på samme måde bare med VertexPostionenColor objekter.

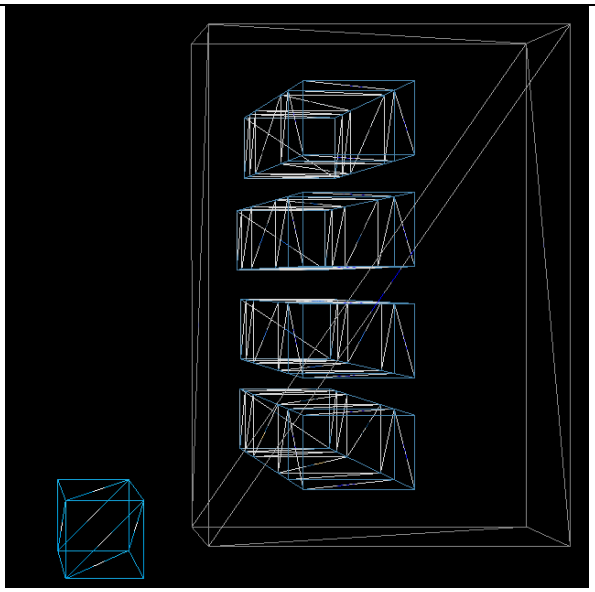
VertexBufferen indeholder altså et array med VertexPositionTexture objekter eller VertexPositionColor objekter, og det er dette data, der gives videre til en vertex shader. Tages der udgangspunkt i en vertex shader for VertexPositionTexture objekter, så er det i dette projekt lavet sådan, at shaderen modtager en vertex position koordinat, samt den tilhørende UV koordinat. Udover disse input, har shaderen også WorldViewProj(de tre matricer ganget sammen) matricen som en global variabel, og denne bliver så ganget sammen på position koordinaten. Derved bliver vertex positionen placeret det rigtige sted på skærmen. Vertex shaderens output er så denne position på skærmen, samt UV koordinaten, hvormed der skal ske en ”mapping” . Disse outputs er så input til pixel shaderen, og denne har desuden en global texture variabel, som så er det billede,

der skal bruges. Vha. af UV koordinaten laves der så et opslag i texturen, og der findes en rgba værdi. Denne værdi er pixel shaderens output, og denne bestemmer så farven for en pixel på skærmen.

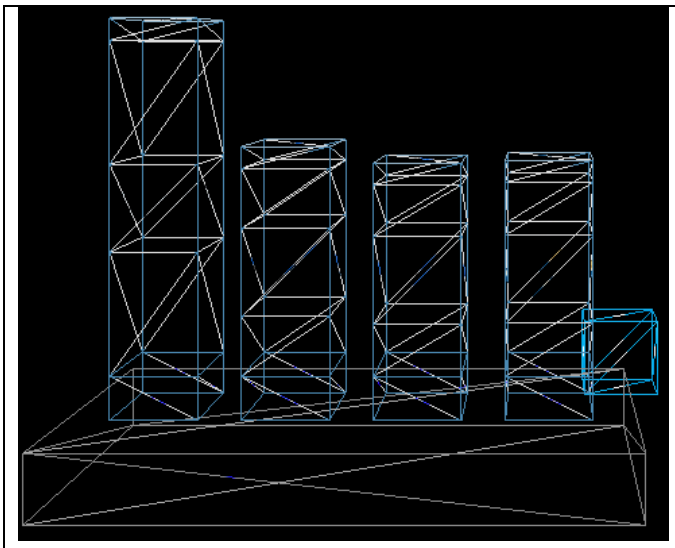
I bilag 5 kan de to shadere, der bruges med en texture, ses.

Endelig resultat

Figur 5.14 Views med "solid" overflader



Figur 5.15 Views med "wireframe" overflader



Figur 5.16 Views med "wireframe" overflader

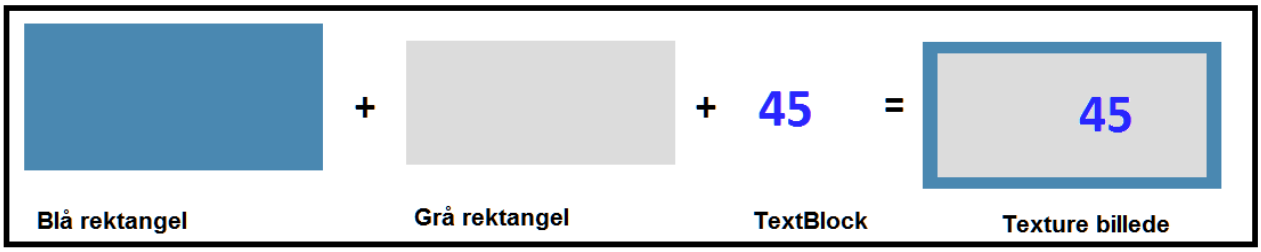
Figur 5.14, 5.15 og 5.16 viser det endelige udseende for et segment på 3D siden, bestående af de forskellige views. På figur 5.14, er view objekterne blevet tegnet med solide overflader, og da man ser segment platformen forfra, er søjlerne gemt bagved, og derved ikke synlige. På figur 5.15 er view objekterne tegnet med wireframe overflader, som i princippet betyder, at det kun er kanterne af de trekanter, som repræsenterer de forskellige views, der vises. Her kan alle de tegnede trekanterne, der bruges for at vise et segment i 3D, altså ses. Da overfladerne ikke er solide, kan søjlernes placering på platformens bagside nu også ses. På figur 5.16 er segmentet roteret, og Pillar view objekteres trekanter kan nu tydeligt ses.

WriteableBitmap som texture

I det forrige afsnit, er det blevet beskrevet, hvordan de forskellige views er opbygget. Her blev det bl.a. vist, at Platform view'et og Pillar(søjle) view'et bruger WriteableBitmap billeder som texture til deres overflader. Dette afsnit beskriver, hvordan disse WriteableBitmap billeder laves? projektet, hvor der bliver taget udgangspunkt i det WriteableBitmap billede, der viser antallet tilhørende en bestemt søjle, og som bruges til bunden af et Pillar view.

Grunden til at der overhovedet er et behov for et WriteableBitmap er, at der på et Pillar view og Platform view skal vises en dynamisk tekst, dvs. de forskellige antal for et segment eller segmentets navn. I kapitel 2 blev det beskrevet, at i 3D kan tekst kun vises i form af texture billeder, og derfor er det nødvendigt med WriteableBitmap billeder, da dette giver mulighed for at lave texture billeder med forskellig tekst løbende i programmet.

Kigges der specifikt på det billede, der bruges på bunden af en søjle, så ses det, at det består af en grå rektangel med en blå kant, hvor der i midten er skrevet et antal. Måden, hvorpå dette er opnået, er ved at bruge 3 UI elementer, og derefter tegne dem i det samme billede. De 3 elementer er en blå rektangel, en grå rektangel og til sidst en TextBlock, hvor antallet er skrevet.



Figur 5.47 texture billede til brug på Pillar(søjle) view

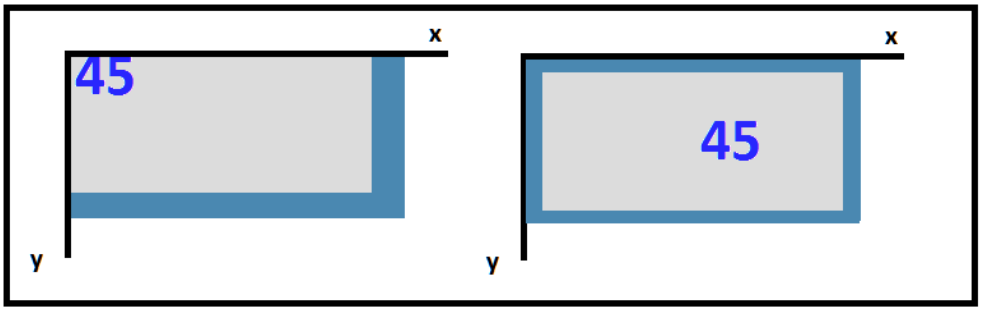
På overstående billede ses de 3 elementer, og hvordan de tilsammen giver det ønskede billede, som kan bruges som texture på søjle bundens overflader. Måden, hvorpå dette laves i C#, er som sagt vha. WriteableBitmap klassen. Denne klasse giver mulighed for at kombinere UI elementer til et billede, der har de ønskede dimensioner, dvs. den højde og bredde, som bedst passer til de dimensioner, som den overflade, hvor billedet skal bruges som texture, har. Nedenstående kode udsnit viser, hvordan Ui elementerne bliver tegnet på billedet:

```
WriteableBitmap bitmap = new WriteableBitmap(bitmapWidth, bitmapHeight);

bitmap.Render(rectangleBack, new TranslateTransform());
bitmap.Invalidate();
bitmap.Render(rectangleFront, new TranslateTransform() { X = 2, Y = 2 });
bitmap.Invalidate();

txt1.Foreground = new SolidColorBrush(Colors.Blue);
bitmap.Render(txt1, new TranslateTransform() { X = 25, Y = 4 });
bitmap.Invalidate();
```

Figur 5.18 Rendering af WriteableBitmap billede

På dette kode udsnit ses, at først puttes en rectangleBack (blå rektangel) på billedet vha. af Render metoden efterfulgt af en Invalidate metode, der sørger for en "redraw" af bitmap billedet. Derefter puttes rectangleFront (grå rektangel) på billedet, og da det sker efter at den blå rektangel er kommet på billedet, så puttes den grå ovenpå den blå. Denne rækkefølge er meget vigtig, p.g.a. at hvis det skete modsat, ville den grå ikke være synlig, eftersom den blå ville ligge ovenpå, og denne rektangel er større. Udover at den grå rektangel lægges ovenpå den blå, så forskydes den også 2 på både x og y aksen i forhold til origo, som er øverst i venstre hjørne. Til sidst lægges en TextBlock txt1 på billedet. Denne textblock indeholder så string værdien, svarende til det antal, der skal vises. Denne forskydes på x og y aksen, sådan at det minder om en højre

centrering, som man er vant til, når man arbejde med tal. Nedenstående billede viser hvor billedet ville se ud uden disse forskydninger på akserne, i forhold til når de anvendes:



Figur 5.19 Resultat uden/med indrykning

Implementering af world, view og projection matricerne

I kapitel 2. 3D teknologi og Silverlight beskrives det, at når der arbejdes med 3D, så er det umuligt at komme udenom de tre matricer kaldet world, view og projection. Det blev også beskrevet, at ved at have en reference til xna framework i ens silverlight projekt, så fik man en masse værktøjer, der kan bruges til at oprette disse matricer. I dette afsnit, vil det blive beskrevet, hvor og hvordan disse matricer bliver oprettet i projektet vha. disse værktøjer.

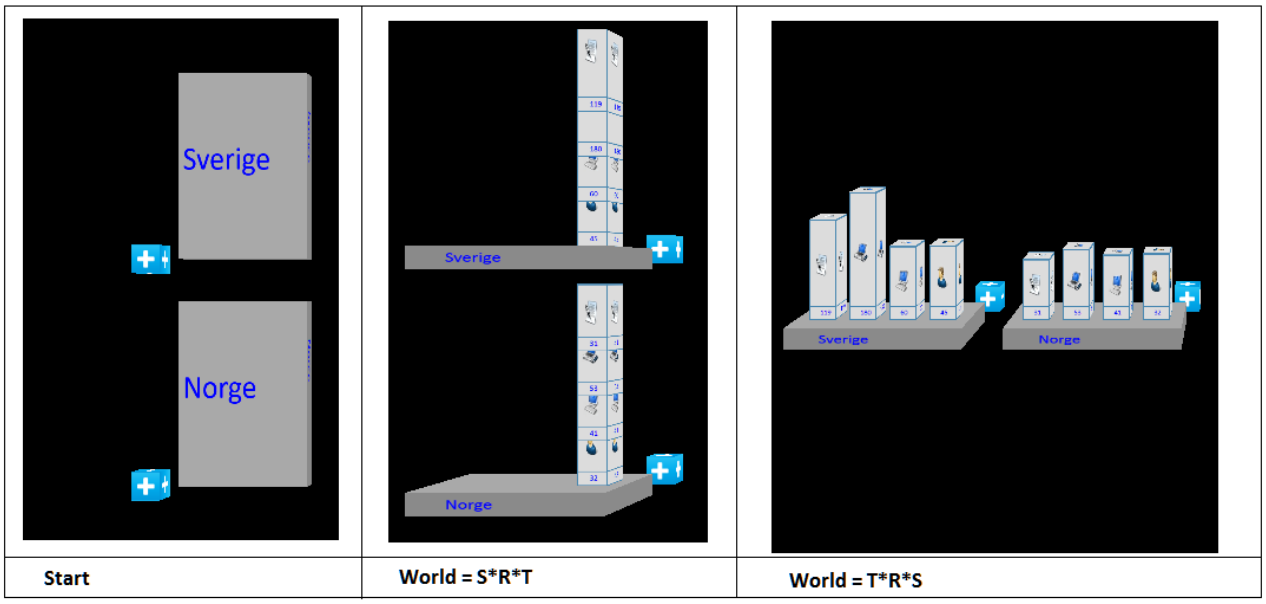
World matrix

Da world matricen bruges til at transformere et view objekts koordinater i dens lokale koordinat system til koordinater i verdenskoordinatsystemet, sker oprettelsen af denne world matrix inde i de forskellige view klasser, og mere bestemt deres draw metode. Grunden til at denne matrix oprettes i de enkelte klasser, er fordi world matricen består af de tre andre matricer: translation, scale og rotation. Disse matricer er individuelle for hvert view objekt, da det er forskelligt hvor i verdenskoordinatsystemet, de placeres. Måden hvorpå matricerne oprettes ses på nedenstående billede, se figur 5.20.

```
Matrix translation = Matrix.CreateTranslation(segmentPosition);
Matrix scale = Matrix.CreateScale(1.0f);
Matrix rotation = Matrix.CreateFromYawPitchRoll(state.SliderYawVal, state.SliderPitchVal, state.SliderRollVal);
Matrix world = translation * rotation * scale;
```

Figur 5.20 Oprettelse af world matrix

Forklaringen af matricerne figur 5.20 er, at for oprettelsen af translation matricen, bruges segmentPosition værdien, som et view har fået fra addToDrawList metoden. Denne flytter de lokale koordinater for et view, til de ønskede verdenskoordinater. Som skalerings matrix bruges den matrix, der kommer ved Matrix.CreateScale(1.0f). Denne betyder at de lokale koordinater bliver ganget med en factor 1, som gør, at objekternes koordinater ikke ændrer sig, og view'et beholder sin størrelse. Til sidst oprettes rotations matricen. Denne laves vha. Matrix.CreateFromYawPitchRoll(float yaw, float pitch, float roll), og som beskrevet i kapitel 2, bruges yaw værdien til at bestemme hvor meget der skal roteres omkring y-aksen, pitch bruges til rotation omkring x-aksen og roll omkring z-aksen. Disse værdier hentes fra den State klassen?, som bliver givet med som parameter i Draw metoden. Når disse 3 matricer så er lavet, ganges de sammen og resultatet er view objekets world matrix. Rækkefølgen af, hvordan de 3 matricer bliver ganget sammen, har en stor indflydelse på world matricen, og derved positionen af de forskellige views, grundet at der arbejdes med matrix multiplikation. På nedenstående billede ses de to rækkefølger, som der normalt arbejdes med: scale*rotation*translate(SRT) eller translate*rotation*scale(TRS).



Figur 5.21 SRT vs. TRS

Billedet viser, at når der ikke er roteret, så er der ingen forskel på SRT og TRS, mens at når der er roteret, så er det en stor forskel på, om world matricen er lavet ved SRT eller TRS. Ved SRT roter objekter omkring sig selv, før de bliver flyttet hen på deres ønskede placering. Ved TRS bliver

objekterne først flyttet og derefter roteret. Som billedet viser, så giver SRT ikke det ønskede resultat for dette projekt vedkommende, og TRS rækkefølgen er derfor den, der bruges.

View matrix

View matricen er som beskrevet i kapitel 2 den matrix, der indeholder informationer omkring kameraets position, positionen for hvor kameraet kigger hen og omkring, hvordan kameraet vender. Da view matricen bruges til at beskrive kamera informationer, er der i projektet oprettet en Camera klasse, der indeholder tre properties. Disse properties er CameraPosition, CameraTarget og View, hvor CameraPosition og CameraTarget er Vector3 værdier, der indeholder x, y og z værdierne for, hvor de er i verdenskoordinatsystemet se figur 2.3 i kapitel 2. View propertien indeholder så den matrix, der kommer ved at bruge Matrix.CreateLookAt(Vector3 cameraPosition, Vector3 cameraTarget, Vector3 cameraUpVector). I dette projekt er cameraUpVector værdien altid sat til Vector3.Up, hvilket betyder at kameraet altid står oprejst. Kode usnit for view matrix kan ses på figur 5.22.

```
public Matrix View
{
    get
    {
        view=Matrix.CreateLookAt(CameraPosition, CameraTarget, Vector3.Up);
        return view;
    }
}
```

Figur 5.22 oprettelse af view matrix

Da det er muligt for brugeren at flytte på kameraet, samt at zoome ind og ud, er der lavet en metode kaldet changeCameraView. Denne sørger for at ændre på CameraPosition og CameraTarget ud fra værdierne i State klassen, og den returner View propertien.

```
public Matrix changeCameraView()
{
  camera.CameraPosition = new Vector3((34.0f + State.MoveXVal), (23.0f + State.MoveYVal), 120.0f + State.ZoomVal);
  camera.CameraTarget = new Vector3((34.0f + State.MoveXVal), (23.0f + State.MoveYVal), 0);
  return camera.View;
}
```

Figur 5.23 changeCameraView

Forskellen på CameraPosition og CameraTarget ligger i deres z værdi. Grunden til at de har samme x og y værdier er grundet et ønske om, at kameraet kigger ligeud, da dette giver det bedste billede for brugeren. Som default har de begge x værdien 34 og y værdien 23, og disse værdier er fundet ved at prøve sig frem, indtil det ønskede resultat blev fundet. Z værdien for CameraTarget er sat til 0, da bagsiden af et platform view har z-værdien 0, og det er på denne bagside at søjlerne står, og denne værdi giver derfor det bedste resultat. Default værdien for CameraPositions z værdi er sat til 120, og dette er igen grundet, at det giver det bedste resultat.

Værdierne moveXVal og moveYVal udregnes i et MouseMove event, og zoomVal i et ValueChanged event for en slider kontrol. Disse værdier er så tilgængelige igennem State klassen. Grunden til at det er lavet på denne måde er, at changeCameraView metoden kaldes på compositions tråden, mens de to events MouseMove og ValueCanged sker sideløbende på UI tråden.

Projection matrix

Projection matricen er den matrix, der indeholder ens viewing fustrum informationer. Viewing fustrum bliver som beskrevet i kapitel 2, brugt til at angive, hvor meget af ens verdenkoordinatsystem, der skal være synligt. Dette betyder samtidig, at det kun er de objekter, der er inde i ens viewing fustrum, der er synlige på siden. Måden hvorpå denne projection matrix laves i silverlight er vha. metoden Matrix.CreatePerspectiveFieldOfView(float fieldOfView, float aspectRatio, float nearDistance, float farDistance). Beskrivelse af denne metode kan ses i afsnittet "World,View og Projection matricer i Silverlight" i kapitel 2.

```
projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4, State.AspectRatio, 1.0f, 400.0f);
```

Figur 5.24 oprettese af projection matrix

På figur 5.24 billedet ses kode udsnittet for oprettelsen af projection matricen. Værdien MathHelper.PiOver4 svarer, som tidligere beskrevet i kapitel 2, til en synsvinkel på 45 grader. Værdien State.AspectRatio bliver udregnet på UI tråden i et SizeChanged event for drawingSurface kontrollen. Dette sker, hver gang ens browser vindue skifter størrelse og her bliver bredden divideret med højden, og denne værdi er så ens aspectRatio. Denne gemmes i State klassen, hvorfra værdien kan læses på compositions tråden, hvor projection matricen bliver oprettet. De

sidste to værdier er ens near clipping plane og far clipping plane værdier. Disse værdier skal ses i forhold til ens kamera position. Dvs. near clipping plane er 1 fra kamera positionen og far clipping plane er 400 fra kamera positionen og alt derimellem er synligt.

Illustration af viewing fustrum med near og far clipping plane, se figur 2.4 i kapitel 2.

Model til View

Kapitel 4 beskrev, at DataFromSql er klassen der står for hele forløbet med oprettelsen af model og view objekter, da denne fungerer som en speciel form for ViewModel. Dette sker ved, at den modtager data fra en sql database vha. asynkrone webservice kald, og omdanner dette data til SegmentModel objekter. Efterfølgende oprettes view objekter ud fra model objekterne. Derudover sørger den for, når model klasserne ændres/opdateres, at oprette nye opdateret views. Opbevaringen af både SegmentModel objekterne og view objekterne sker også i DataFromSql. Til SegmentModel objekterne bruges to Dictionary lister, mens view objekterne gemmes i en almindelig Liste<T>.

Som beskrevet tidligere, er en Dictionary liste silverlights svar på en hash tabel, og består derfor af nøgler og dertilhørende værdier. Grunden til at valget faldt på Dictionary lister til opbevaring af SegmentModel objekterne er, at når der, som i dette projekt, er brug for hurtigt at kunne finde et SegmentModel objekt og ændre dens Expand værdi, så dur det ikke, at der i worst case skal køres et helt array igennem for at finde det specifike objekt. Her tilbyder Dictionary listen, at man kan finde et objekt i konstant tid, ved brug af den tilhørende nøgle. Det er her også vigtigt, at man kan give de forskellige SegmentModel objekter en individuel nøgle i listen, og her er det meget simpelt, da der fra databasen, gives en brugbar nøgle, kaldet SegmentID, der er individuel for hver SegmentModel.

De to Dictionary lister som SegmentModel objekterne bliver lagt i, er lavet sådan, at i den ene gemmes alle objekterne som værdi, med deres SegmentID som nøgle. Denne bruges til at ændre Expand værdien i konstant tid. Listen er kaldt allModelList og DataFromSql klassen har derfor en property AllModelList, hvor denne liste kan tilgås fra, se figur 4.9. i kapitel 4. Den anden liste indeholder kun de SegmentModel objekter, der ikke har nogen parent segment, og disse objekter kan kaldes for "top" forældrene. Her kan de resterende objekter så tilgås fra ”top” forældres SegmentChildren lister og igen fra deres undersegmentes SegmentChildren lister osv. Denne liste

med ”top” forældrene bruges, når SegmentModel objekternes data, skal laves om til view objekter. Her ønkes der som default nemlig kun vist ”top” forældrene på siden, og det er kun views tilhørende dem, der skal tegnes. Er disse ekspanderet, så kigges der så i deres SegmentChildren lister, og der laves view objekter ud fra deres undersegmenter osv. Det er her vigtigt at forklare, at det er de samme SegmentModel objekter, der gemmes i de to lister. Derfor, når der ændres i Expand værdien for et objekt i allModelList, er denne også ændret i modelsToViewList, som den anden liste kaldes.

Når de to Dictionary lister er blevet oprettet og fyldt med SegmentModel objekter, bliver makeViewObject metoden kaldt. Denne står for at kalde addToDrawList metoden, for de objekter der ligger i modelToViewListen, dvs. "top" forældrene, da disse altid skal laves til views og vises på siden. Metoden addToDrawList tager som parametre et SegmentModel objekt, og en List<IViewInterface> liste, hvor alle view objekterne bliver lagt over, når de er oprettet. Metoden er lavet som en rekursiv algoritme, der sørger for at kalde sig selv, for hvert SegmentModel objekt i SegmentChildren listen for det SegmentModel objekt, den får ind som parameter, hvis dennes SegmentModel objekts Expand property er sat til "true". Udover dette returner metoden en Vector2 værdi, og denne værdi bruges til at tegne et ChildLine view objekt. På nedestående billede vises det rekursive kald, samt hvordan retur værdien bruges.

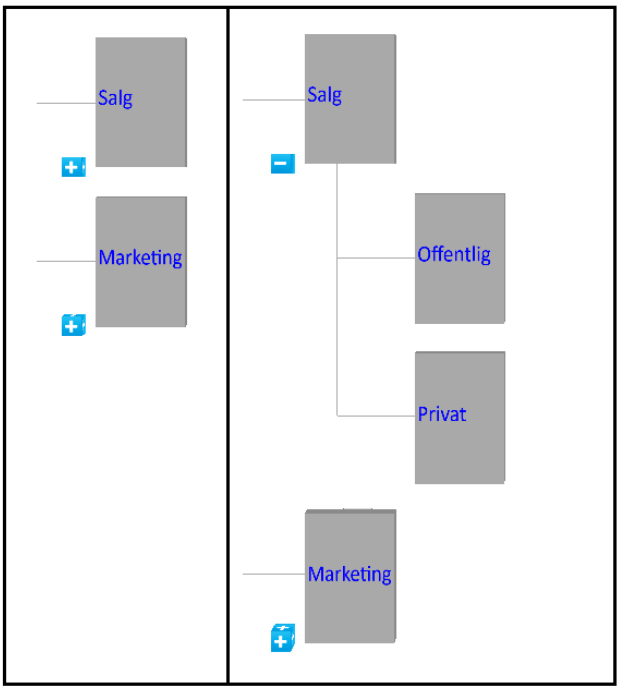
```
foreach (SegmentModel childSegment in segment.SegmentChildren)
{
    segmentPosition.X = 0;
    Vector2 childLinePosition=addToDrawList(childSegment, tempViewObjects);
    ChildLine childLine = new ChildLine(platform.ParentLinePostion, childLinePosition);
    childLine.SegmentID = segment.SegmentID;
    IviewInterface viewChildLine = childLine;
    tempViewObjects.Add(viewChildLine);
}
```

Figur 5.25 rekursiv kald i addToDrawList metode

Forklaringen på overstående billede er, at en foreach løkke, går alle SegmentModel objekterne i SegmentChildren listen igennem, for det SegmentModel objekt, der er ved at blive kigget på, hvis Expand value er true. Inden det rekursive kald til addToDrawList metoden sker, nulstilles Vector3 værdien segmentPositions x værdi til 0. Grunden til dette er, at i starten af addToDrawList metoden, sættes x værdien ud fra den Niveau property, som et SegmentModel har, og y værdien

får trukket 16 fra, og derved overlapper view objekterne ikke hinanden. Efter dette laves det rekursive kald, med et af undersegmenterne som den nye SegmentModel parameter, og der laves Platform view, Pillar views, ExpandButton view og ChildLine views for dette SegmentModel objekt. Har denne også "børn" og er denne ekspanderet, så laves der også rekursive kald med dens børn som parameter osv. Dette gør, at alle view objekter er oprettet, for undersegmenterne af et ekspanderet segment, og placeringen af dem på scenen er derved bestemt og segmentPosition værdien opdateret. I oprettelsen af et Platform view, får platformen segmentPosition værdien med som parameter i dens konstruktør, og dens properties ChildLinePosition og ParentLinePosition bliver så udregnet ud fra denne. Det er så ChildLinePosition, der bliver sat som return værdien for addToDrawList. På figur 5.6 ChildLine view ses, at ChildLine har et StartPoint, og denne sættes til at være lig ParentLinePosition for det platform view, der er blevet lavet ud fra den SegmentModel, der kigges på, mens EndPoint sættes til at være lig den ChildLinePosition som addToDrawList returner.

Grunden til at de rekursive kald er så vigtige er, at positionen for et segment på siden er bestemt ud fra hvor mange segmenter, der skal vises over den, og derfor er disse nødt til at blive oprettet først. På nedenstående billede er der en illustration.



Figur 5.26 illustration af vigtigheden af de rekursive kald

Billedet på figur 5.26, viser først, at når "Salg" segmentet ikke er ekspanderet, så skal "Marketing" segmentet vises lige under. Vælger brugeren af systemet at ekspandere "Salg" segmentet, så skal "Marketing" segmentet ikke længere stå lige under, men rykkes ned, så der er plads til "Salg" segmentes undersegmenter. Da "Salg" og "Marketing" er undersegmenter af det samme segment, ligger de i den samme SegmentChildren liste, og de bliver derfor kaldt efter hinanden i foreach løkken. Havde addToDrawList ikke været en rekursiv metode, der selv sørger for at oprette views for "Offentlig" og "Privat" segmenterne samt at opdatere segmentPosition værdien, så ville det være svært at vide, hvad positionen for "Marketing" segmentet skulle være, og især hvis "Offentlig" eller "Privat" segmenterne også havde børn og var ekspanderet. Her ville det f.eks. ikke være nok bare at kigge på, hvor mange børn "Salg" segmentet har.

Når addToDraw metoden skal lave et segments Pillar(søjle) views, så skal den først finde ud af, hvor mange brugere, computere ovs. segmentets søjler skal vise. Dvs. at er segmentet ikke ekspanderet, så skal antallet som vises, for enten brugere, computere osv. være lig segmentets eget antal lagt sammen med antallet af dens undersegmenter, og er segmentet ekspanderet, så skal den kun vise det antal, der er tilknyttet segmentet selv. Måden hvorpå addToDrawList metoden finder det rigtige antal, er så selvfølgelig først at tjekke for, om segmentet er ekspanderet. Er den det, så bruges de forskellig count properties i SegmentModel instans, der indeholder segmentets informationer, se figur 4.8 i kapitel 4.

Er et segment ikke ekspanderet, bruges addToDrawList getSumCounts metoden til at udregne de forskellige summer af brugere og computere osv. for et segmentet og dens undersegmenter. Metoden er lavet som følgende:

```
public Dictionary<string, int> getSumCounts(SegmentModel segment, Dictionary<string, int> sum)
{
    sum["Users"] += segment.UserCount;
    sum["Workstations"] += segment.WorkstationCount;
    sum["Assets"] += segment.AssetsCount;
    sum["Licenses"] += segment.LicenseCount;


    foreach (SegmentModel childSegment in segment.SegmentChildren)
    {
        getSumCounts(childSegment, sum);
    }
    return sum;
}
```

Figur 5.27 getSumCounts metode

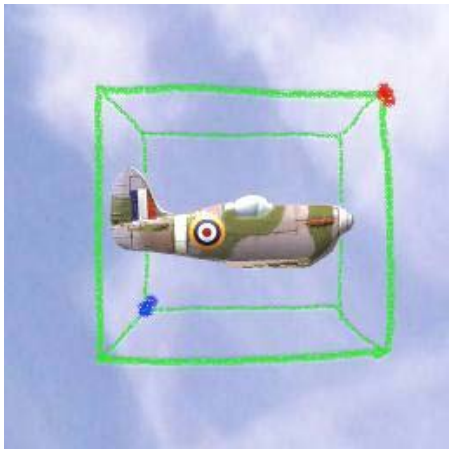
Metoden figur 5.27 tager en SegmentModel og en Dictionary<string,int> liste som parametre, og retur listen igen, efter at den er blevet opdateret. Dette er også en rekursiv metode, ligesom addToDrawList, og denne sørger for først at lægge et SegmentModel objekts count properties til de nuværende sum værdier, og derefter kalder den getSumCount metoden for alle dens undersegmenter, så også deres count properties bliver lagt til de samlede summer. Dvs. at når getSumCounts kaldes fra addToDrawList med en SegmentModel og en Dictionary liste, hvor int værdierne er 0 til at starte med, så retuneres efter de rekursive kald, altså en opdateret Dictionary liste. Denne indeholder de totale sum værdier for det ikke ekspanderede segment. I bilag 4 ses, hvordan getSumCounts kaldes, og hvordan sum værdierne bagetefter bruges, når der oprettes et Pillar view objekter.

Trykke på objekter med musen

Da dette projekt omhandler en 3D side, hvor brugeren skal have mulighed for at trykke på forskellige objekter på 3D siden, såsom at trykke på en expand knap, for at få vist et segments undersegmenter, eller trykke på et segment og blive sendt videre til dens summary side, så opstår der et dilemma. Dette dilemma går ud på at løse problemet med muse koordinater i 2D og 3D objekter med 3D koordinater, og mere præcis, hvordan tjekkes der for, om en bruger har trykket på noget i scenen, og hvad er der blevet trykket på.

Måden hvorpå dette dilemma blev løst i implementeringen af prototypen, var vha. metoden mouseToWorldCoordinate(List<IviewInterface> drawList). I denne metode laves muse koordinaten først om til to nye Vector3 værdier, hvor forskellen ligger i z værdien. Den ene værdi skal have en z værdi lig 0(ligger ved "near clipping plane") og den anden en værdi lig 1(liggende ved "far clipping plane"). På denne måde får man to værdier (muse punkt X, muse punkt Y, 0) og (muse punkt X, muse punkt Y, 1). Derefter bruges et kald til unproject metoden for hvert punkt, og unproject retuner værdien for deres placering i 3D rummet[13]. Metoden skal bruge en source, hvor en af de to nye vector3 værdier bruges, sammen med de tre vigtige matricer view, projection og world. Da unproject metoden kaldes èn gang, for hver af de to vector3 værdier, der blev lavet ud fra musekoordinaten, ender vi ud med en vector3 nearPoint og vector3 farPoint. Det næste der skal gøres er at finde en retning. Denne retning findes ved at trække nearPoint vektoren fra farPoint vektoren og derefter normalisere denne vektor. Dette gør at retnings vektoren laves til en "unit-

length direction vector". Denne retnings vektor bruges til at lave en ”ray”(stråle), der skal forstås som en ray lige ind i skærmen, der går igennem den pixel som brugeren trykkede på, og alt bag den pixel. Til at lave sådan et ray skal der brugs et start punkt, og her bruges nearPoint, og den retnings vektor, der lige er lavet. Derefter kan en ”intersects” metode i Ray klassen nu bruges. Insertects metoden kræver enten en BoundingBox eller en BoundingSphere, men da de objekter det skal være mulige at trykke på, er firkantede objekter, så vælges en BoundingBox.



Figur 5.28 BoundingBox [8]

På figur 5.28 ses en Boundingbox omkring en 3D model, der i dette tilfælde er et fly. Oprettelsen af en BoundingBox kræver en min. værdi og en max. værdi. Disse vector3 værdier repræsenterer så to diagonale hjørner af en kube. Da alle view objekter, som er tegnet på 3D siden, implementer IviewInterface, har de en property kaldet ViewBoundingBox, se figur 4.10 i kapitel 4. Denne property indeholder den BoundingBox, der omkranser view'et, og denne kan så bruges i ray.insersect metoden.

Måden hvorpå en BoundingBox for et view bliver lavet, er vha. af metoden makeBox. Koden for denne metode kan ses på nedenstående billede.

```
public BoundingBox makeBox(Vector3 leftBottomBack, Vector3 rightTopFront, Matrix world)
{
    BoundingBox box;
    Vector3 min = Vector3.Transform(leftBottomBack, world);
    Vector3 max = Vector3.Transform(rightTopFront, world);
    box = new BoundingBox(min, max);
    return box;
}
```

Figur 5.29 Oprettelse af BoundingBox

Her ses at makeView metoden kræver to vector3 værdier, der er de lokale min. og max. koordinater. Udover disse skal den bruge world matricen, da BoundingBox'en skal laves i verdenskoordinatsystemet.

For at se om der er blevet trykket på et view, køres en løkke, der går alle objekterne i drawList, som er parameteren metoden kræver, igennem. Denne liste indeholder view objekterne fra ViewObject listen, der er listen, som indeholder de objekter, der er tegnet på DrawingSurface kontrollen. Da det ikke er meningen, at der skal kunne trykkes på et view objekt af typen ChildLine, så bliver der i løkken sørget for, at der ikke bliver lavet en bounding box for denne type. Vha. ray.intersect metoden, hvis return værdi angiver, hvor langt væk ray'et skærer en BoundingBox, holdes der hele tiden styr på, hvilket view der, hvis BoundingBox skærer ray'et, er nærmest, da denne må være den forreste, og derfor den, brugeren har trykket på.

Når løkken så er færdig, står man tilbage med det view, der er blevet trykket på. Da det er muligt at trykke på forskellige view typer, og det er ikke meningen, at der skal ske det samme, tjekkes der for, hvilken type der er blevet trykket på. Er det et view af typen ExpandButton, se sekvensdiagram 4.11, så bruges SegmentID og ExpandValue fra view'et til at oprette en instans af ExpandChangeEventArgs. Efterfølgende sker der et callback til segment_ExpandChangeEvent metoden i MainPage. Denne metode tager som parameter et ExpandChangeEventArgs argument, og her bruges den nyoprettede instans af ExpandChangeEventArgs, som blev lavet ud fra view'et. Metoden sørger for at kalde metoden segmentExpandValueChanged på UI tråden. Måden dette sker, er vha. Dispatcher.BeginInvoke(System.Action a), der sørger for at udføre den angivne System.Action, der er en delegate, asynkron på tråden, som Dispatcher klassen er associeret med. Da MainPage er associeret med UI tråden, får vi altså udført vores Action delegate på UI tråden. Action parameteren er som sagt en delegate, og denne laves vha. lambda expression, og mere præcis en statement lambda. I beskrivelsen omkring BeginInvoke, på Microsoft hjemmeside[18], bliver det beskrevet, at Action parameteren ikke må have nogle input værdier, samt at det skal være en void metode. Derfor laves Action delegate metoden på denne måde : () => { /* body of void function here */ } [19], hvor ( ) betyder ingen input værdier [19]. Inde i tuborg klammerne kaldes metoden segmentExpandValueChanged med segmentID og expandValue fra

ExpandChangeEventArgs argumentet som parametre. Dette gør at segmentExpandValueChanged køres på UI tråden.

I metoden segmentExpandValueChanged ændres først Expand værdien for den SegmentModel, som hører til det view, der blev trykket på, og efterfølgende kaldes makeViewObjects metoden. Grunden til at segmentExpandValueChanged metoden skal køres på UI tråden er, som beskrevet i kapitel 2, at man helst skal undgå at ændre på model data eller arbejde med en masse beregninger og oprette nye views osv. på composition tråden. Compositions tråden skal kun læse view data og tegne de tilhørende primitiver, som de forskellige view er lavet af.

```
void segment_ExpandChangedEvent(object sender, ExpandChangeEventArgs expandArgs)
{
    this.Dispatcher.BeginInvoke(() => { segmentExpandValueChanged(expandArgs.segmentID, expandArgs.expandValue); });
}
private void segmentExpandValueChanged(int segmentID, bool expandValue)
{
    sql.AllModelsList[segmentID].Expand = expandValue;
    sql.makeViewObjects();
}
```

Figur 5.30 event for expand button

På figur 5.30 ses, hvad der sker, når der sker et callback til segment_ExpandChangedEvent metode. Udover at makeViewObjects() skal kaldes på UI tråden fra et design mæssigt synspunkt, så er der også en implementerings grund. Denne er, at når de forskellige WriteableBitmaps bliver lavet, så bruges der bl.a. en TextBlock, der er en UI kontrol, og bliver denne ikke oprettet på UI tråden, så sker der en " Invalid cross-thread acces exeption ".
Er det ikke et view af typen ExpandButton, der bliver trykket på, er det meningen, at man skal sendes hen til en ny side i LicenseWatch SAM produkt. Da dette projekt er en prototype, videresendes brugeren derfor til en "redirectPage.aspx" side, og alt efter om der er blevet trykket på et segment eller en søjle tilhørende et segment, bliver der sendt forskellige værdier til redirectPage siden.
Derfor laves der, hvis der bliver trykket på et view af typen Platform eller Pillar, se sekvens diagram 4.12, et callback til segment_RedirectEvent metoden i Mainpage klassen vha. RedirectEvent event handleren. Denne metode tager som parameter en instans af RedirectEventArgs klassen, der indeholder en string urlExtension variabel. På figur 5.31 ses,

hvordan et callback til segment_RedirectEvent sker og hvordan RedirectEventArgs instansen bliver lavet ud fra et Pillar view eller Platform view.

```
if (selectedType.FullName.Equals("LicenseWatch3D.Views.Pillar"))
{
    Pillar selected = (Pillar)nearestView;
    RedirectEventArgs pillarRedirectArgs=null;
    string urlExtension = "";
    switch (selected.PillarType)
    {
        case 1://workstation
            urlExtension = "?SegmentID=" + nearestView.SegmentID + "&Page=Workstation";
            pillarRedirectArgs = new RedirectEventArgs(urlExtension);
            break;
        case 2://Users
            urlExtension = "?SegmentID=" + nearestView.SegmentID + "&Page=User";
            pillarRedirectArgs = new RedirectEventArgs(urlExtension);
            break;
        case 3://Assets
            urlExtension = "?SegmentID=" + nearestView.SegmentID + "&Page=Asset";
            pillarRedirectArgs = new RedirectEventArgs(urlExtension);
            break;
        case 4://Licenses
            urlExtension = "?SegmentID=" + nearestView.SegmentID + "&Page=License";
            pillarRedirectArgs = new RedirectEventArgs(urlExtension);
            break;
    }
    RedirectEvent(this, pillarRedirectArgs);
}
if (selectedType.FullName.Equals("LicenseWatch3D.Views.Platform"))
{
    string urlExtension = "?SegmentID=" + nearestView.SegmentID;
    RedirectEventArgs platfromRedirectArgs = new RedirectEventArgs(urlExtension);
    RedirectEvent(this, platfromRedirectArgs);
}
```

Figur 5.31 RedirectEvent for Platform og Pillar view

På nedestående billede figur 5.32 ses, hvordan segment_RedirectEvent metoden er lavet, og igen sørger den får at kalde en anden metode på UI ligesom på figur 5.30.

```
void segment_RedirectEvent(object sender, RedirectEventArgs rediretArgs)
{
    this.Dispatcher.BeginInvoke(() => { RedirectChanged(rediretArgs.urlExtension); });
}
private void RedirectChanged(string urlExtension)
{
    string url = HtmlPage.Document.DocumentUri.Scheme + "://" +
                 HtmlPage.Document.DocumentUri.Host + ":" +
                 HtmlPage.Document.DocumentUri.Port + "/" +
                 "redirectPage.aspx" + urlExtension;

    HtmlPage.Window.Navigate(new Uri(url), "blank");
}
```

Figur 5.32 Håndtering af RedirectEvent

Da mouseToWorldCoordinate metoden ligger i et DrawEvent callback, som sker uafbrudt, puttes en if sætning rundt om metoden, så den kun bliver kaldt, når State.Mouse punktet ikke har værdien (-1,-1). State.Mouse punktet bliver sat til musens koordinater i et MouseLeftButtomDown Event og, når punktet er blevet kigget på i mouseToWorldCoordinate metoden, sat til tilbage til default værdien (-1,-1).
Grunden til at mouseToWorldCoordinate metoden kaldes på compositions tråden, når denne helst ikke skal lave andet en at tegne primitiver er, fordi metoden unproject kun virker i et Draw event callback på compositions tråden. Dette gør dog ikke så meget rent præsentationsmæssigt, da det kun sker hver gang brugeren har trykket med musen, og altså ikke i hvert renderings loop.

Kapitel resume af implementering

I Kapitlet er det beskrevet, hvordan de forskellige views bygges op. Da der her er tale om silverlight 3D, så bliver view objekterne ikke lavet vha. XAML, som man normalt gør. I stedet angives de forskellige views vha. af en masse positioner og en værdi, der henviser til, hvad positionen skal blive ”mapped” sammen med. Dette kan enten være en farve eller en UV værdi, der henviser til et punkt på et texture billede. Positionerne og dertilhørende ”mapping” værdier, bliver så brugt til enten at oprette instanser af VertexPositionTexture eller vertexPositionColor. Antallet af vertexPositionTexture og vertexPositionColor instanser afhænger af, hvor mange retvinklede trekanter der skal til for at vise de enkelte views, da graphicDevice enheden i Silverlight hovedsagelig arbejder med retvinklede trekanter.

I kapitel 3 blev der analyseret frem til nogle krav, som den nye segment side skal opfylde. Nogle af disse krav kræver, at der skal kunne vises dynamisk tekst på siderne i form af navnet på et segment, samt de tilhørende antal for computere, licenser osv. der hører til segmenterne. I dette kapitel er der beskrevet, hvordan dette opnås ved at bruge WriteableBitmap. Her er der mulighed for at bruge forskellige UI elementer, såsom rektangler og tekstbokse, og derefter lave et bitmap ud fra disse. Herved kan der laves dynamiske billeder, der passer til segment Platform view'et og Pillar view'et, sådan at man kan få vist navne og antal.

Da view objekterne som sagt ikke laves vha. XAML, så bruges der i projektet ikke bindings mellem model klasserne og dertil hørende views. Derfor sker oprettelsen af views ikke automatisk ud fra oprettede model objekter. I projektet er det så lavet sådan, at DataFromSql er klassen, der sørger

for at omdanne model data til views. Dette sker i metoden addToDrawList, der sørger for at lave views for et enkelt segment, og er segmentet ekspanderet, laver den et rekursiv kald til sig selv, sådan at et segments undersegmenter også bliver tegnet. Alle view objekterne ligges efterfølgende over i en liste, der så læses fra på compositions tråden.

Kapitlet sluttede med at vise løsningen på problemet omkring omdannelsen af 2D muse korrdinater til 3D koordinater i scenen, og derved gøre det muligt for brugeren at trykke på objekter. Løsningen blev metoden mouseToWorldCoordinate i Scene klassen. Kort sagt så sørger metoden for at omdanne musekoordinaten til to koordinater i 3D scenen, som derefter bruges til at lave et Ray. Et Ray har en intersect metode, som kan bruges sammen med en BoundingBox. Da alle view objekterne har en BoundingBox, der omkranser view'et, så køres listen med view objekterne igennem, og det view, hvis BoundingBox giver den laveste intersect værdi, er så det view objekt, der er trykket på.

Kapitel 6. Test

Med henblik på at kontrollere implementationen af segment siden i 3D, er der blevet udformet forskellige tests, heriblandt unit test af nogle af de vigtigste ” ikke” grafiske metoder. Derudover en form for brugertest, hvor siden er blevet afprøvet af medarbejdere fra LicenseWatch A/S, ved at de har udført forskellige opgaver på siden, hvorefter resultaterne er blevet analyseret. Dette skulle gerne være med til at vise, at implementationen af de 6 use cases er opfyldt.

Unit test

Formålet med en unit test er, som navnet hentyder til, at teste mindre dele(units) af et software system. I dette projekt er det "Unit Test Framework for Microsoft Silverlight", der er en del af Silverlight Toolkit, som er blevet brugt. Dette framework gør det muligt at unit teste en silverligt application i forhold til det indbyggede test miljø i visual studio. I dette unit test framework til silverlight, bliver unit tests kørt inde i en web browser, men måden hvorpå test koden laves, minder om det indbyggede test framework i visual studio. Frameworket giver desuden også mulighed for, at teste asynkrone metoder, og da der i dette projekt arbejdes med asynkrone webservice kald, så er dette en nødvendighed for at teste disse.

Måden unit test normalt laves på, er vha. 3 skridt.

1. Initialisere test ved at opsætte de nødvendige parametre, klasser og afhængigheder, der skal bruges for at testen kan fuldføres.
2. Kør koden som skal testes.
3. Påstå forskellige egenskaber omkring resultatet.

De to første skridt siger sig selv, hvorimod den sidste er en unit test term. I de fleste frameworks er det muligt at lave forskellige assertions omkring resultatet, og dette framework er ikke anderledes. De forskellige assertions kan f.eks. være null-checks eller værdi sammenligning, se figur 6.1

```
Assert.AreEqual(viewObjects.Count, correctCount, "addToDrawList dit not make the wanted amount of view objects");
Assert.IsNotNull(vertexShaderColor, "The vertex shader for color could not be loaded");
```

Figur 6.1 Brug af forskellige assertions i unit test framework.

En total på 18 unit-test er blevet skrevet til at teste funktionaliteten af udvalgte elementer i systemet. De 18 test omhandler 4 forskellige områder.

- Hentning af data fra sql server
- Omdannelse af model data til view objekter
- Oprettese af shadere
- View og Projection matricerne

Hentning af data fra sql server

Dette område handler om den funktionalitet, der i systemet sørger for at hente data fra sql databasen, og vha. webservicekald får det ind i silverlight projektet, hvor der så dannes SegmentModel instanser ud fra det hentede data. Det er altså funktionaliteten i use case 6, der bliver testet, se bilag 1 use case 6.

Måden, hvorpå denne test er lavet, er ved at kigge på de fem callback metoder i DataFromSql klassen, der køres, når de asynkrone webservices kald er færdige. Disse er:

- webService_GetSegmentsComplete
- webService_GetNumberUsersComplete
- webService_GetNumberOfWorkstationsComplete
- webService_GetNumberOfAssetsComplete
- webService_GetNumberOfLicensesComplete

Disse fem metoder har hver især til opgave, at hente forskelligt data fra databasen, hvor den første webService_GetSegmentComplete, skal hente data fra segment tabellen, dvs. SegmentID, ParentID, SegmentName og Niveau.
Unit-testen, der er blevet opstillet for denne metode, sørger for først at kalde den tilhørende asynkrone metode, og når denne er færdig og metoden webService_GetSegmentComplete har kørt, testet der for, om der er kommet det rigtige data op fra databasen. Her tages der en stikprøve af de SegmentModel instanser, der er oprettet, og der tjekkes for, om disse indeholder det rigtige data, samt om der er blevet lavet en SegmentModel instans for hver række i segment tabellen. I bilag 6 i figur 12.1 kan test metoden ses.
De resterende metoder sørger for at kalde de tabeller i databasen, der indeholder sammenhængen mellem segmenterne og dens brugere, computere, aktiver og licenser, dvs. hvor mange antal èt segment har af de forskellige elementer. De fire test metoder for disse metoder?,

er lavet stor set på samme måde som testen på figur 12.1 bilag 6. Dog er det her de forskellige count properties i SegmentModel instanserne, hvis korrekthed der tjekkes. Her er der igen taget en stikprøve, da der i testen regnes med, at hvis resultaterne er rigtige for en tilfældig stikprøve, så er de også rigtige for de resterende SegmentModel instanser.
Hvis de fem testede metoder består deres tilhørende test, så betyder det altså, at der bliver hentet det rigtige data, samt at der bliver oprettet det rigtige antal SegmentModel instanser med dette data. Derved er use case 6, ”Henting af data” implementeret korrekt i prototypen.

Omdannelse af model data til view objekter

Dette område handler om at teste de metoder, der er med til at omdanne SegmentModel objekternes data til view objekter, som så kan tegnes på siden. I alt bruges der tre metoder, og der er derfor oprettet en test metode til hver af disse.

Den første metode der testes, er metoden addChildToParentList. Metoden bruger Dictionay listen med alle SegmentModel objekterne, og sørger for at placere de forskellige SegmentModel objekter, der har en ParentID forskellig fra 0 i deres parent SegmentModel objekts ChildrenList. Top forældrene, dvs. de SegmentModel objekter, som ikke har nogen forældre, puttes over i en Dictionary liste med SegmentID som key og med SegmentModel som value. Denne liste kan så senere bruges af metoden addToDrawList, men det er her vigtigt, at listen kun indeholder top forældrene, og at deres børn/undersegmenter kan findes i deres SegmentChildren liste, og det er det, der testes for i metoden TestChildToParent.

Test metoden sørger selv for at oprette et vist antal SegmentModel objekter og tildele propertierne SegmentID og ParentID de rigtige værdier. Den sørger også for at lægge alle objekterne over i en Dictionary liste, der skal symbolere dictionary listen allModelList i DataFromSql klassen. Ved selv at oprette SegmentModel objekterne og tildele dem data, så kan metoden addChildToParent testes uafhængig af andre metoder, og dette giver den bedste unit test. Test metoden går den Dictionary liste igennem, som addChildToParent returner. Dette sker vha. af en rekursiv algoritme, der sørger for at gå top forældrenes SegmentChildren lister igennem, samt deres børns SegmentChildren lister osv. Når listerne køres igennem, tjekkes der hele tiden for, om SegmentModel objekterne i de forskellige SegmentChildren lister, har en ParentID, der svarer til SegmentID'et for det SegmentModel objekt, som listen tilhører. Udover

dette holdes der også styr på, hvor mange SegmentModel objekter, test metoden får kigget på. Dette er grundet, at man skal kunne tilgå alle SegmentModel objekterne, når Dictionary listen køres igennem, enten direkte fra listen, eller igennem de forskellige SegmentChildren lister. Derfor skal antallet af modeller der er kigget på, altså være lig antallet af oprettede modeller. Dvs. hvis alle SegmentModel objekterne der direkte ligger i Dictionary listen har ParentID=0, og der ikke findes fejl i nogle af SegmentChildren listerne, samt at der bliver kigget på alle SegmentModel objekterne, så består addChildToParent metoden testen.

Den næste metode der testes er addToDrawList metoden. Det er denne metode, der skal bruge den liste, som addToParent metoden returner, og omdanne SegmentModel objekterne til view objekter, der implementere IViewInterface. I test metoden TestAddToDrawList, der tester metoden, bliver der først oprettet en masse SegmentModel objekter, og der bliver manuel oprettet en liste, der svarer til den liste som addToParent ville returne. Igen er grunden til alt det manuelle arbejde, at denne test handler om addToDrawList metoden, og derfor undgås brugen af andre metoder så vidt som muligt.
Måden testen foregår på er, at da antallet af view objekter, der skal laves for hvert SegmentModel objekt, som vises på siden, er kendt, og hvor mange børn de ekspanderede segmenter har osv., så udregnes der et antal for, hvad det korrekte antal af view objekter skal være, og der laves en assertion på, om antallet af view objekter, der er lavet vha. af addToDrawList metoden, er lig det korrekte antal.

```
Assert.AreEqual(correctCount,viewObjects.Count,  "addToDrawList dit not make the wanted amount of view objects");
```

Figur 6.2 Assert for addToDrawList metode

Den sidste metode der testes indenfor området "omdannelse af model data til view objekter" er metoden getSumCounts, der bruges i addToDrawList til at se, hvor mange brugere, computere osv. der tilhører et ikke ekspanderet segment, og som bruges til at udregne højden på søjlerne. Som beskrevet tidligere, så skal antallet, der vises på en søjle af et ikke ekspanderet segment, være lig dens eget antal, lagt sammen med undersegmenternes antal, for de segmenter, der tilhører det ikke ekspanderede segment. Metoden er en rekursiv metode, der sørger for at gå SegmentChildren listen igennem for et SegmentModel objekt, samt dens børns lister og deres lister osv.

Metoden der tester getSumCounts, er TestgetSumMethod. I denne oprettes der også først en masse SegmentModel objekter, og alle disse får tildelt en værdi til deres count properties, se figur 4.8 i kapitel 4. og de bliver random lagt i hinandens ChildrenList lister, med en enkel top forældre øverst i segment strukturen.
Metoden getSumCounts får så denne top forældre med som parameter, samt en Dictionary liste kaldet sum, hvor de forskellige count værdier gemmes. Disse værdier opdateres så ved hvert kald af getSumCounts metoden. Det er denne opdaterede liste, som metoden returner. Da man kender de korrekte antal af brugere, computere osv., da de manuelt er blevet givet til SegmentModel objekterne, tjekkes der for, om værdierne i den opdaterede sum liste, er lig de korrekte værdier. I bilag 6 under "omdannelse af model data til view objekter" på figur 12.2 ses, hvordan test metoden er lavet.

Oprettese af shadere

Da det grafiske 3D i systemet kun kan blive vist på siden, hvis der bliver oprettet VertexShader og PixelShader objekter, så testet der for, om de forskellige .vs og .ps filer, se shader afsnittet i kapitel 2 under Silverlight, kan findes i systemet, samt om disse kan bruges til at oprette PixelShader og VertexShader objekter.

Til at teste om de fire shader filer kan findes i projektet (Color.vs, Texture.vs, Color.ps og Texture.ps), oprettes først en instans af ShaderFile klassen. Denne indeholder URI properties for hver fil, og der testes for, om en ResourceStream er null, ved at bruge en ? disse properties, se nedenstående figur:

```
ShaderFile files = new ShaderFile("Color.vs", "Color.ps", "Texture.vs", "Texture.ps");
[TestMethod]
public void TestGetVertexShaderColorFile()
{

    Assert.IsNotNull(Application.GetResourceStream(files.VertexShaderFileColor), "The vertex shader file for color could not be found");
}
```

Figur 6.3 Assert for oprrettelse af shader stream

Billedet på overstående figur viser, at der bliver testet for, om det er muligt at lave en ResourceStream til filen Color.vs. De resterede tre filer bliver testet på samme måde i metoderne TestGetVertexShaderTextureFile, TestGetPixelShaderColorFile og TestGetPixelShaderTextureFile.

Det næste der testes for er, om disse shader filer kan bruges til at oprette henholdsvis VertexShader og PixelShader objekter. Til at teste for dette kræves en instans af ShaderUtil klassen, som er den klasse i LicenseWatch3D projektet, der har en instans af ShaderFile klassen, se figur 4.10 i kapitel 4. Udover ShaderUtil klassen, er det nødvendigt at have en instans af MainPage klassen, da denne har fået lavet en speciel metode til denne test. Denne metode er en simpel get metode, der retuner en GrapichsDevice instans, da denne skal bruges i VertexShader og PixelShader konstruktørerne, og som get metoderne i ShaderUtil klassen derfor kræver som parameter. Test metoderne er lavet som følgende:

```
#region getting shades
ShaderUtil shaders = new ShaderUtil();
MainPage mainpage = new MainPage();
[TestMethod]
public void TestGetVertexShaderColor()
{
    VertexShader vertexShaderColor = shaders.GetVertexShader(mainpage.getDevice(), 1);
    Assert.IsNotNull(vertexShaderColor, "The vertex shader for color could not be created");
}
```

Figur 6.4 test oprettelse af shader objekter

På figur 6.4 ses, at der testes for, om det VertexShader objekt, der fås ved at bruge ShaderUtil klassens GetVertexShader metode er NotNull. GetVertexShader metoden kræver som sagt en GraphicDevice, der fås fra mainpage klassen, samt en int type, der bruges til at se, om VertexShader objektet skal laves ud fra Color.vs eller Texture.vs, hvor 1 er Color og 2 er Texture.

De resterende TestGetVertexShaderTexture, TestGetPixelShaderColor og TestGetPixelShaderTexture metoder, der tester de andre muligheder for oprettelse af shader objekter, er lavet på samme måde.

Som det ses på figur 6.3 og 6.4, så testes der kun for, om filerne eksisterer og om det er muligt at oprette en VertexShader eller PixelShader, og altså ikke om en shader giver det ønskede grafiske resultat, når vertex data bliver givet til den. Da VertexShader og PixelShader objekterne i dette projekt ikke skal gøre andet end at mappe en texture eller color på en position og, vha. af worldViewProj matricen, placere denne vertex på det ønskede sted på skærmen, så er korrektheden af shader filerne fundet ved simpel studering af resultatet på skærmen, og her bliver

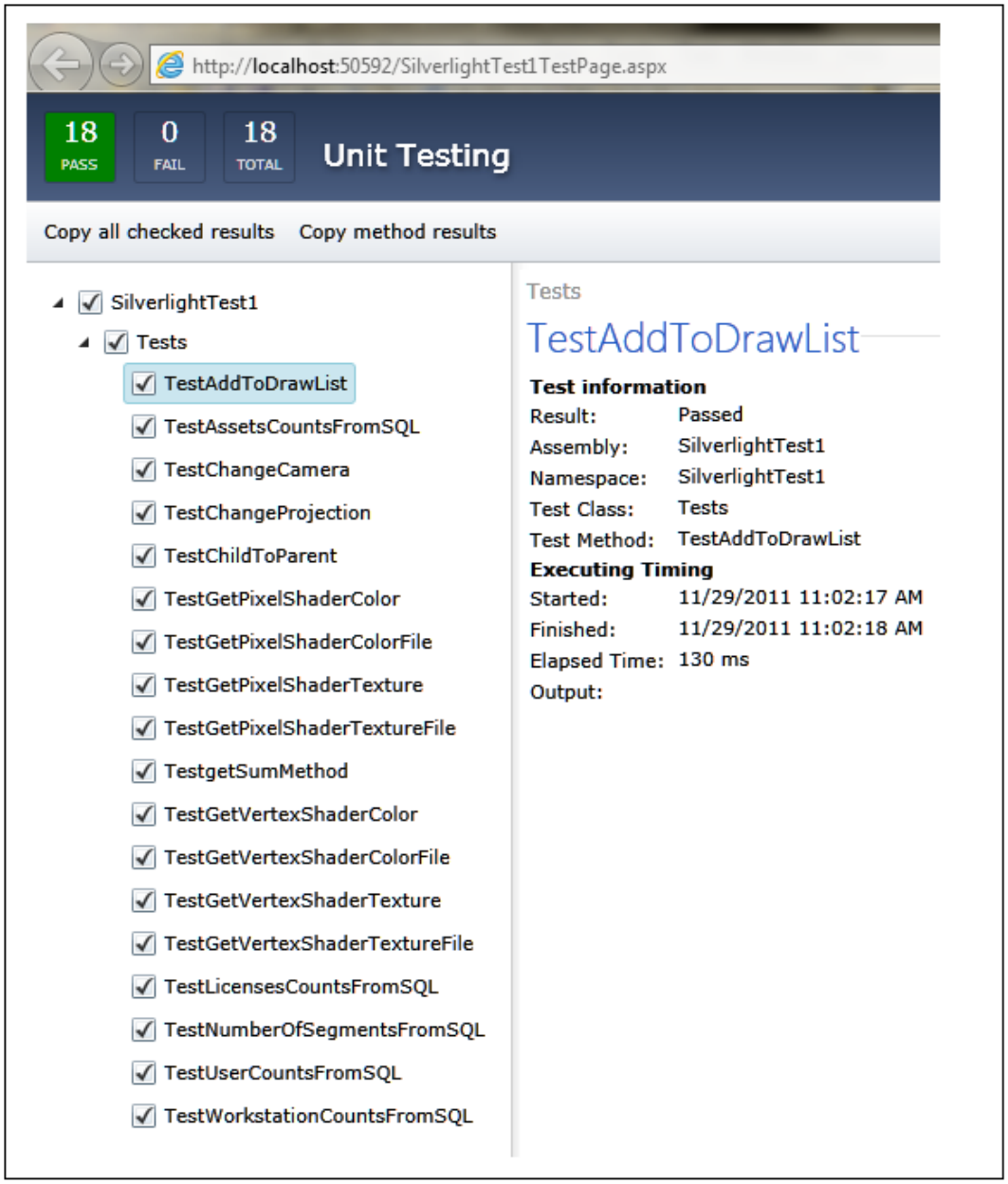
vertex positionerne placeret rigtigt, samt korrekt mappet sammen med et texture billede eller farve.

View og Projection matricerne

Til at ændre de to matricer View og Projection, når brugeren enten bruger musen eller zoom slideren, til at flytte rundt på siden, eller når brugeren minimerer eller maksimerer browser vinduet, er der i systemet oprettet to forskellige metoder. Deres formål er at ændre de to matricer alt efter, hvad brugeren har gjort.

Disse to metoder er kaldes changeCameraView og changeProjection, og disse tager informationerne fra State klassen, som er nødvendige for den specifikke matrix, og bruger dem til at oprette og returne en ny matrix, som så kan blive brugt. Det er disse metoder, der testes, og der testes for, om deres return matrix er som forventet. Måden hvorpå dette er gjort kan ses på figur 12.3 i bilag 6. Unit testen der sørger for at teste changeCameraView metoden, har altså til formål at verifecere elementer af use case 4, da det er i denne metode at kameraets position bliver ændret.

Resultat



Figur 6.5 resultat af unit test

Som det ses på figur 6.5, så kører alle unit-tests uden at producere nogen fejl. Unit-testning er en god måde at verificere funktionaliteten af de individuelle kode dele, og det er en meget værdifuld ressource at have i implementeringsfasen. Sammen med debuggeren i visual studio, har dette givet et yderst værdifuld værktøj til problem løsning, da det herved har været nemt at se, hvilke kodedele der fejlede, og hvilke der virkede, og derved muligt at finde frem til, hvor eventuelle problemer lå.

Ovenstående unit-test har hovedsagelig handlet om at teste de ikke grafiske elementer af systemet, såsom at hente data og omdanne det til view objekter. Elementer, som at trykke på objekter på siden, eller teste om de forskellige view objekter bliver placeret på det rigtige sted på siden, har ikke været mulige at få verificeret i en unit-test, da disse elementer er afhængig af compositions tråden, og denne fungerer ikke i test miljøet. Dette er også grunden til, at det kun er view og projection matricerne, der bliver testes. Deres change metoder bliver godt nok kaldt på compositions tråden, men det er ikke en nødvendighed, at de bliver kaldt der, og derfor kan de testes. World matricen derimod, som ikke bliver testet i unit-testen, bliver oprettet og ændret i draw metoderne for de forskellige view objekter, og view objekternes draw metode fungerer kun, hvis de bliver kaldt inde i et draw callback på compositions tråden fra DrawingSurfaces.

Black-Box test

De elementer, som ikke har været mulige at teste i en unit test, er dog blevet testet i en anden form for funktionalitets test, hvor en person er blevet sat til, f.eks. at trykke med musen på og omkring expand knappen 100+ gange, og tjekke om knappen fungerer som forventet. Det samme er sket med tryk på segmenterne og søjlerne. Denne test fungerer altså som black-box test, hvor der kun kigges på, om resultaterne er som forventet.

For at se om de forskellige view objekter bliver placeret rigtigt på siden, dvs. om world matricerne for objekterne bliver lavet rigtigt, er testen derudover blevet udført ved, at personen ekspanderede forskellige segmenter. Dette gør at positionen for undersegmenterne for et ekspanderet segment ændres, se figur 5.26 i kapitel 5, samt ved at lave forskellige rotationer, og se om objekterne bliver placeret som forventet på siden.

Da det ikke er meningen, at programmøren, som har implementeret koden, selv skal lave sådan en test, så er en person udenfor projektet blevet brugt. I stedet for at bruge en person til at teste ovenstående, kunne en automatiseret test program være brugt, men det har ikke været muligt at skaffe sådan et. Det optimale havde selvfølgelig været, når nu der bruges en virkelig person, at denne var fra LicenseWatch A/S, men grundet det store ressource forbrug af sådan en test, har dette ikke været muligt. Derfor blev der fundet en person på DTU, der var villig til at hjælpe. Personen læser til bygningsingeniør, og er derfor vant til at arbejde med 3D programmer, og har derfor en vis forståelse for, hvordan rotationer i 3D fungerer osv. Derudover er personen bror til

undertegnede, og har derfor ikke haft noget problem med at sige, hvis noget ikke var som det skulle være.

Resultatet af denne test var, at test personen konkluderede, at objekterne blev placeret rigtigt, både ved ekspandering og ved rotationer. Angående funktionaliteten med at trykke på objekter på siden, fandt personen en enkel fejl. Fejlen lå i, at når objekterne blev roteret, så var det ikke muligt at trykke på søjlerne. Fejlen var nem at rette, da det var en simpel ombytning af to parametre, der skulle fikses i Pillar klassens makeBox metode. Ved denne test blev implementationen af use case 1,3 og 5 altså verifeceret.

Herved er funktionaliteten af de grafiske elementer også blevet testet, og sammen med resultatet af unit-testen, så er en rigtig stor del af funktionaliteten blevet testet og verificeret.

Bruger test

Brugertesten, der er blevet lavet i til dette projekt, er ikke, som man normalt kender det, en test med mange brugere. Det var desværre ikke muligt her, men alligevel kunne en test, hvor to personer fra LicenseWatch blev sat til at svare på spørgsmål, tjene et formål, da de vha. af deres store ekspertise, kan være med til at vise, om det er muligt at bruge siden til det ønskede formål, og om den er forståelig for folk, der er vant til den gamle side.

Fremgangsmåde

Måden testen blev lavet på var, at der blev opstillet forskellige opgaver/spørgsmål, som kun var mulige at svare på ved at bruge den nye segment side med dertilhørende test database. Dette er med til at verifecere, at prototypen har implementeret de seks use cases korrekt, da opgaverne kræver implementationen af disse. Testen var derudover opdelt i to, hvor den første del kun gik ud på at se, om testpersonerne svarede rigtigt. I den anden del, hvor personerne var blevet mere bekendte med siden, blev der udover svarene også kigget på, hvor mange handlinger/skridt personerne brugte på dette. Antallet af handlinger kan så sammenlignes med det mindst mulige antal, som er fundet ved at afprøve alle tænkelige måder for, hvordan disse svar kan findes.

Den første testperson er supporter i LicenseWatch, så han er vant til at arbejde med brugen af SAM systemet, og han har også en god forståelse for, hvordan brugerne af systemet bruger siden, samt hvilke problemer de kan have. Den anden testperson er leder af udviklingsafdelingen, og

han er en af de personer i LicenseWatch, som har mest viden omkring, hvad kunderne ønsker sig af segment siden.

Til testen var der også lavet en brugervejledning til siden, og denne blev gennemgået sammen med hver enkel testperson, før testen blev startet. Derved havde personerne den fornødne viden til at finde svarene.

Test spørgsmålene og brugervejledningen kan ses i henholdsvis bilag 7 Brugertest og bilag 8 Brugervejledning.

Resultat

Test person nr. 1 support medarbejderen:

For support medarbejderen var resultatet af test 1, hvor der kun kigges på svarene, at han fik 9 ud af 10 rigtige. Spørgsmålet, som blev svaret forkert på, var spørgsmål 3, se bilag 7 under test del 1. Her glemte personen, at hvis et segment ikke er udvidet, så viser den summen af dens og dens undersegmenters antal. Dog svarede han i spørgsmålene 1,6,7 og 10 rigtigt, og disse spørgsmål omhandler den samme problemstilling, om hvorvidt et segment skal ekspanderes eller ikke, for at finde det rigtige svar.

I test 2 klarede personen det også rigtig godt, og her svarede han rigtigt på 6 ud af 6, og i spørgsmål 4 fik han siden tilbage til default, som han skulle. Kigges der på antallet af handlinger/skridt, som personen udførte, dvs. tryk på knapper, tryk på objekter på siden såsom at ekspandere, bevægelse på siden med musen og brug af slidere, så brugte han ikke mange flere skridt, end det minimum antal, som opgaverne krævede. I gennemsnit brugte han 1.7 skridt mere end det var krævet af ham, og oftest var disse ekstra skridt bevægelser på siden, dvs. flyt kameraet op og ned osv.

Test person nr. 2 udviklings lederen

For udviklingslederen var resultatet i test 1, det samme som supportmedarbejderen, nemlig 9 ud af 10, og faktisk var det igen spørgsmål 3, der blev svaret forkert på, og grunden var den samme, da personen også her glemte at ekspandere segmentet.

I test 2 var der lidt større forskel på supportmedarbejderen og udviklingslederen. Hvor support medarbejderen svarede rigtigt på alle spørgsmålene, havde udviklingslederen en enkelt forkert.

Dette var spørgsmål 3, se bilag 7, under test del 2, der omhandler ændring af roterings indstillingerne. Personen kunne godt finde ud af at rotere 180 grader om y aksen, men glemte at læse, at det *kun* var y aksen der skulle roteres om, og derfor blev svaret forkert. I forhold til antal skridt, var udviklingslederen dog bedre end supportmedarbejderen med kun 0.5 skridt for meget i gennemsnit, og igen var det nogle få ekstra bevægelser på siden, der gjorde det.

Alt i alt havde de to personer tilsammen kun 3 forkerte svar, hvor to af dem var til det samme spørgsmål, hvis problemstilling de ikke havde problemer med at løse i andre spørgsmål. Udover de høje antal af rigtige svar, så brugte test personerne ikke ret mange unødvendige skridt, for at komme frem til svarene. Testen viser altså, at vha. en kort brugervejledning, er det nemt for personer, der er vant til at bruge den gamle side, at gå over til den nye side med 3D.

Kapitel resume af test

Prototypen er blevet testet på 3 forskellige måder. Den første test, der blev beskrevet, og som er blevet brugt løbende gennem implementeringen, er unit test af det non grafiske funktionalitet. Resultatet af unit testen viste, at alt det non grafiske funktionalitet bestod dets unit test, og er dermed blevet verificeret som fungerende kode. Grunden til at unit testen kun omhandler det non grafiske funktionalitet er, at det ikke er muligt at teste det funktionalitet, som køres på compositions tråden.

Den anden test havde derfor til opgave at teste den grafiske funktionalitet, såsom at trykke på objekter, og se om view objekter bliver placeret rigtigt. Dette skete vha. en black-box test. Testen blev udført ved, at en person f.eks. trykkede en masse gange omkring et objekt på siden og noterede resultatet , og efterfølgende en vurdering af, om dette resultat var som forventet. Denne test viste, at også det grafiske funktionalitet virkede som ønsket, på nær en enkel fejl, der blev fundet ved tryk på pillar view objekter, men denne fejl blev hurtigt løst.

Den sidste test, der blev kørt på prototypen, var en form for bruger test, hvor to personer fra LicenseWatch blev sat til at løse forskellige opgaver på siden. Her blev der samtidig holdt øje med, hvor mange skridt, der blev brugt. Denne test viste, at siden var nem at gå til for gamle brugere, da næsten alle opgaver blev løst og antallet er unødvendige skridt var minimale.

I alt er prototypens funktionalitet blevet verificeret, og overstående tests har derudover vist, at kravene, der blev opstillet i kapitel 3 og i de 6 use cases, opfyldes af prototypen.

Kapitel 7. Konklusion

Introduktion

LicenseWatch A/S er en virksomhed, som skaber overblik for deres kunder, og dette gør de vha. deres SAM produkt. I dette SAM produkt som LicenseWatch leverer, er der mulighed for, at kunderne kan opdele deres virksomhed i bestemte segmenter. Her ønsker LicenseWatch at se, om det nye 3D i Silverlight 5 kan bruges til at forbedre overblikket på den asp. side, der viser segment strukturen.

Det første kapitel af denne rapport beskrev bl.a. visionen af denne rapport:

"*Det er dette projekts vision at lave en ”proof of concept", der går ud på, at få skabt en 3D scene til visning af segment data for brugere af systemet på en mere overskuelig måde, sådan at Licensewatch efterfølgende kan se, om 3D er noget, som de vil stræbe efter at få indarbejdet i senere udgivelser af deres SAM produkt."*

Processen med at udarbejde denne 3D prototype af segment siden blev inddelt i fem opgaver, som hver især har bidraget til resultatet af dette projekt. Det første afsnit i dette kapitel opremser kort disse 5 punkter.

I det efterfølgende afsnit, vil der blive kigget på fremtidig arbejde på projektet. Heri vil der blive beskrevet, hvad der skulle laves på prototypen, hvis der havde været et halv år mere, til udarbejdelsen. Derudover en beskrivelse af hvilke planer LicenseWatchs har med segment siden, efter at de har studeret prototypen.

Rapport resume

Ud fra de enkelte kapitel resuméer, kommer der en kort beskrivelse af resultatet fra de enkelte opgaver, der blev beskrevet i kapitel 1, samt en konklusionsbemærkning på dette.

Kapitel 2. 3D teknologi og Silverlight

Med udgivelsen af den nyeste version af Silverlight "Silverlight 5 RC" der indeholder 3D, har man fået et nyt værktøj, til udarbejdelse af 3D på en asp. side. Her får man mulighed for at lave Graphics Processing Unit (GPU) accelereret 3D applikationer, som man kender det fra bla. opengl. Vha. af referencer til forskellige xna biblioteker, der er tilgængelige ved download af Silverlight

toolkit, er det muligt at arbejde med de vigtigste elementer indenfor 3D, såsom texture og matrix manipulationer.

Herved har man muligheden for at vise tekst på siden, samt at ændre på world, view og projection matricerne , og især muligheden for tekst på siden er altafgørende for en side som segment siden i LicenseWatch's SAM system.

Silverlight 5 RC er som værktøj til udvikling af en asp. side med 3D altså et godt valg, og ideen på figur 1.2 indeholder dermed ikke nogle elementer, der ikke er mulige i Silverlight, og denne kan altså udvikles på .NET platformen som ønsket af LicenseWatch.

Kapitel 3. Analyse af ide og opstilling af krav

Til den nye segment side, er der kommet forskellige krav til verdenen, såsom at brugeren skal kunne bevæge sig på siden, ekspandere segmenterne, se de forskellige antal, rotere billedet, og rent grafisk at segmenterne og deres antal skal vises som rektangulære platformer og søjler. Kravene er kommet frem ved bl.a. at kigge på funktionaliteten på den nuværende side, og hvilken værdi denne giver. LicenseWatch ønsker ikke at fjerne funktionalitet og værdi fra brugerne, som de er vant til, da det skal være nemt for brugerne at gå fra den nuværende side, over til at bruge den nye side med 3D. Udover dette er der også kigget på gestalt lovene for at se, hvordan man bedst hjælper brugerne af systemet, til at forstå sammenhængen mellem de 3D elementer, der vises på siden. Her er bl.a. ideen med, at søjlerne skal står på de platforme, hvor segment navnet står, fundet vha. af loven om nærhed.

De fundne krav til prototypen, burde ud fra analysen give segment siden et bedre overblik og derved mere værdi, og det burde være nemt for både nye og eksisterende brugere at bruge den nye side.

Kapitel 4. Design

De fundne krav i kapitel 3 har ledt frem til alt 6 use cases, som prototypen skal implementere. Fem af disse omhandler interaktion med brugeren, mens den sidste omhandler at få data fra databasen. Eftersom prototypen omhandler 3D, hvor databinding ikke kan bruges mellem model og view objekterne, så har det ikke været muligt at følge MVVM fuldt ud, som det kendes fra wpf, og ViewModel klassen i projektet er derfor en lidt speciel udgave. Her er det DataFromSql klassen, som er ViewModel

klassen, og som sørger for både oprettelsen af model og view objekter, og opdatering af view objekter, når model objekterne opdateres/ændres. Fra kapitel 2 vides der, at når der arbejdes med Silverlight i 3D, så arbejdes der også med to tråde, hvor den ene er UI tråden og den anden er compositions tråden, hvor alt renderingen sker. Designet af prototypen er lavet sådan, at på UI tråden bliver model og dertilhørende view objekter lavet og ændret løbende, og på compositions tråden sker der ikke ret meget andet, end at der læses view data og dette videregives til ens GPU.

Ved at bruge et design, der sørger for at compositions tråden ikke laver andet end at læse view data, opnås der en optimal præstation er prototypen.

Kapitel 5. Implementering

Da der her er tale om Silverlight 3D, så bliver view objekterne ikke lavet vha. XAML, som man normalt gør. I stedet angives de forskellige views vha. en masse positioner, og en værdi der henviser til, hvad positionen bliver ”mapped” sammen med, hvilket enten kan være en farve eller en UV værdi. I kapitel 3 blev der analyseret frem til nogle krav, som den nye segment side skal opfylde. Nogle af disse krav kræver, at der skal kunne vises dynamisk tekst på siden. Her er det vist, at dette kan opnås ved at bruge WriteableBitmaps som texture billeder.

Til sidst blev der også vist, hvordan UC: 5 fra design kapitlet er blevet implementeret. Dette er sket i metoden mouseToWorldCoordinate i Scene klassen. Denne sørger for tjekke, om brugeren af systemet har trykket på noget på siden.

Implementeringen af prototypen gør, at denne overholder de krav, der blev stillet i kapitel 3. Herunder at de forskellige segmenter bliver vist som platforme med søjler bagpå, der repræsenterer de forskellige antal, samt at brugeren kan rotere og bevæge sig på siden osv. Det at der også er fundet en måde at implementere UC 5, der omhandler at trykke på 3D objekter på siden gør, at man på den nye side, nemt og intuitivt kan ekspandere segmenterne, samt gå direkte til andre asp. sider i LicenceWatch's SAM system.

Kapitel 6. Test

Prototypen er blevet testet på 3 forskellige måde. En af de tre tests, var en unit test, hvor den ”ikke” grafiske funktionalitet blev testet. Denne unit test blev kørt sideløbende med implementeringen af funktionaliteten, og resultatet af denne er brugt som verificering er den ”ikke” grafiske funktionalitet. Til verificering af den grafiske funktionalitet er black-box tests blevet

brugt, hvor det altså er undersøgt, om der sker det forventede på siden. Til sidst er en lille brugertest blevet lavet, hvor test personerne var medarbejdere fra LicenseWatch. Denne test viste, at for personer, der kender den gamle segment side, er det nemt at gå over til den nye side med 3D.

Prototypens funktionalitet er altså blevet verificeret, og derudover har de forskellige tests også vist, at kravene, der blev opstillet i kapitel 3 og i de 6 use cases, opfyldes af prototypen.

Fremtidigt arbejde

I dette afsnit vil det først blive beskrevet, hvordan prototypen kunne videreudvikles, hvis det var muligt at arbejde et halvt år mere på den.
Derefter vil det blive beskrevet, hvilke planer LicenseWatch A/S har med segment siden, efter at de har prøvet prototypen, og set hvilke muligheder der er med 3D i Silverlight 5.

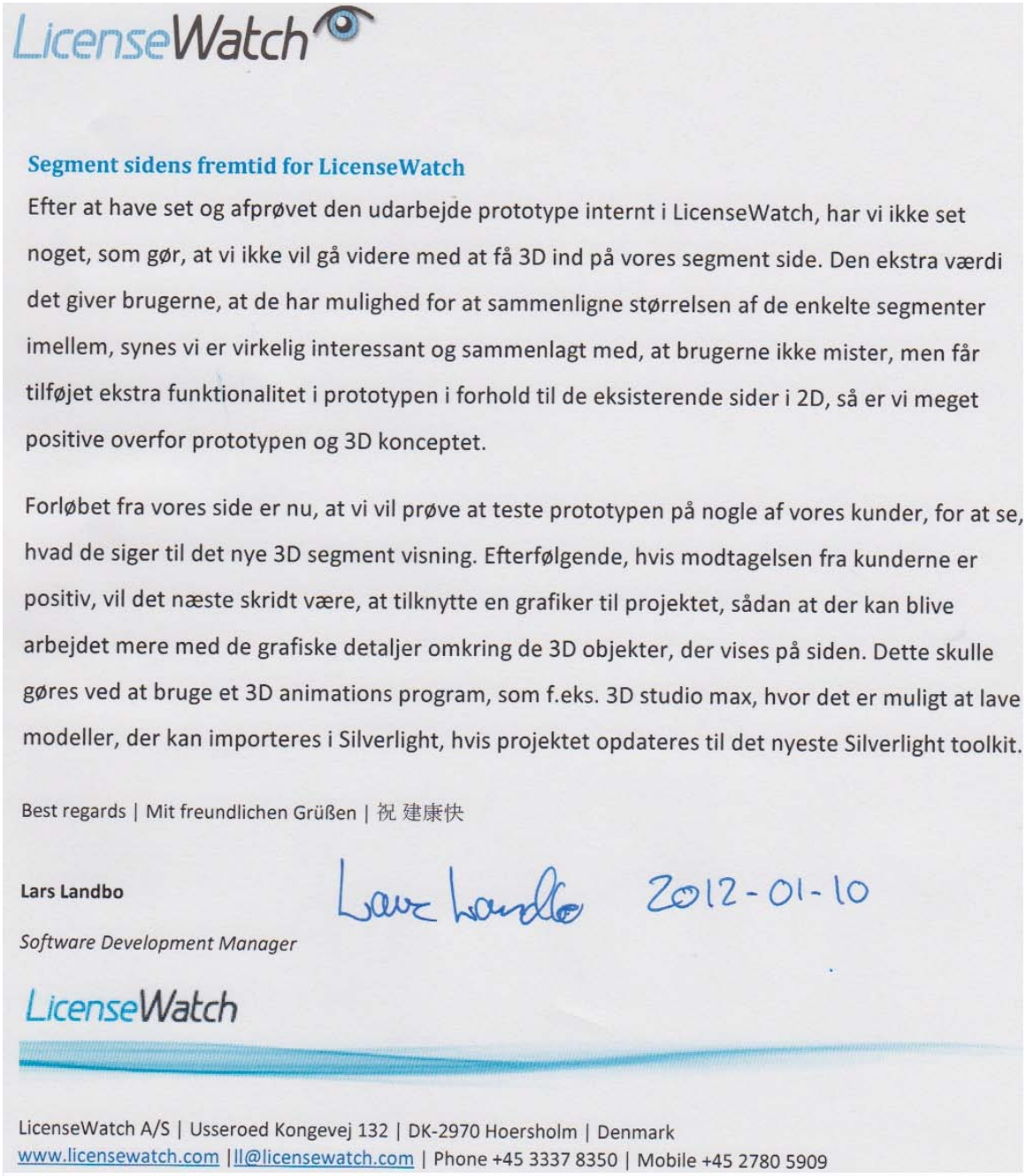
Videreudvikling

Var det muligt at arbejde videre med prototypen, så ville en af de første opgaver være, at få projektet up to date med det nyeste Silverlight toolkit. Med den nyeste version af Silverlihgt tookit er det muligt at oprette et decideret 3D projekt, hvor man får bedre mulighed for at arbejde med shaderne. Arbejdet med shaderne foregår med SilverlightEffect klassen. Her kan shaderne skrives som normalt i HLSL (High Level Shader Language), men igennem SilverlightEffect klassen tilgås, og derved kan der dynamisk gives flere input til shaderne, end det er muligt i det nuværende projekt. Dermed bliver det bl.a. muligt at opsætte lys i 3D scenen, som laves i shaderne. Udover det nye silverlight toolkit, er det dog også et krav at Microsoft xna game studio 4.0 er installeret. Dette gør dog også, at udover de nye muligheder med shaderne, så opnås der også mulighed for at importere forskellige 3D modeller, som kan laves i andre 3D modellerings programmer såsom 3D studio max. Herved kan der laves mere detaljerede og flotte views til ens segmenter osv.

Udover det rent tekniske med opgradering af projektets solution, så vil en spændende videreudvikling være, at prototypen formes til at fungere som et generelt framework, med henblik på at vise forskellige aspekter af en virksomhed i 3D. Ideen med at have en virksomhed opdelt i en struktur med forskellige afdelinger osv. der har forskellige emner tilknyttet sig, er ikke forbeholdt kunder af LicenseWatch A/S og deres SAM produkt. F.eks. kunne det tænkes, at en produktions-virksomhed som Danish Crown kunne ønske at få vist sine forskellige slagterier, og vha. af søjler

se, hvor meget svinekød eller oksekød disse forarbejder. Det kunne også være en virksomhed, der har lyst til at se, hvordan fordelingen af bestemte typer af medarbejder er fordelt på deres afdelinger. Derfor kan projektet arbejdes mod at blive et framework, hvor det er muligt, alt efter hvad den bestemte kunde har lyst til at se, at oploade passende billeder til søjlerne og ændre farve osv. sådan at det passer til den individuelle kunde.

Fremtidig arbejde for LicenseWatch



LicenseWatch

Segment sidens fremtid for LicenseWatch

Efter at have set og afprøvet den udarbejde prototype internt i LicenseWatch, har vi ikke set noget, som gør, at vi ikke vil gå videre med at få 3D ind på vores segment side. Den ekstra værdi det giver brugerne, at de har mulighed for at sammenligne størrelsen af de enkelte segmenter imellem, synes vi er virkelig interessant og sammenlagt med, at brugerne ikke mister, men får tilføjet ekstra funktionalitet i prototypen i forhold til de eksisterende sider i 2D, så er vi meget positive overfor prototypen og 3D konceptet.

Forløbet fra vores side er nu, at vi vil prøve at teste prototypen på nogle af vores kunder, for at se, hvad de siger til det nye 3D segment visning. Efterfølgende, hvis modtagelsen fra kunderne er positiv, vil det næste skridt være, at tilknytte en grafiker til projektet, sådan at der kan blive arbejdet mere med de grafiske detaljer omkring de 3D objekter, der vises på siden. Dette skulle gøres ved at bruge et 3D animations program, som f.eks. 3D studio max, hvor det er muligt at lave modeller, der kan importeres i Silverlight, hvis projektet opdateres til det nyeste Silverlight toolkit.

Best regards | Mit freundlichen Grüßen | 祝 建康快

Lars Landbo Lars Landbo 2012-01-10

Software Development Manager

LicenseWatch

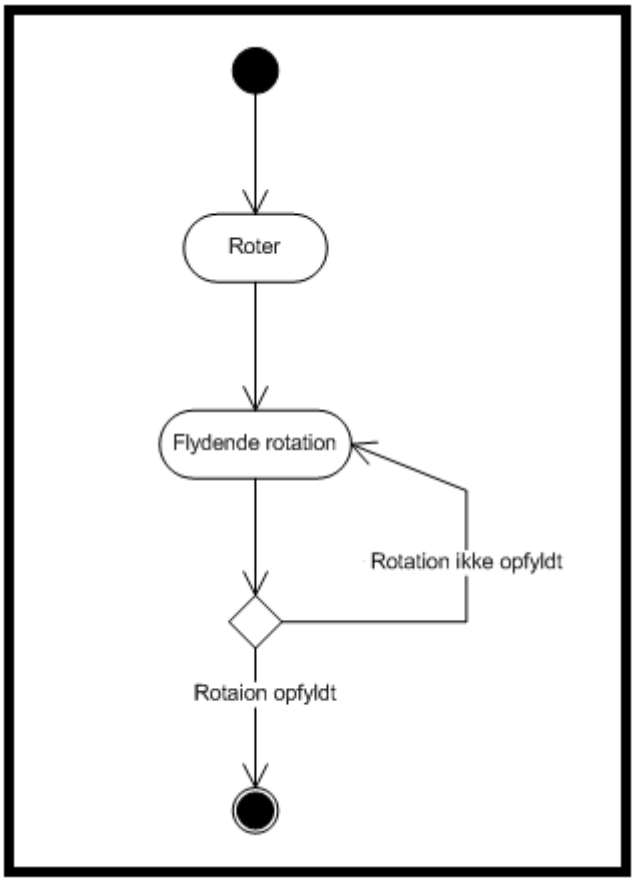
LicenseWatch A/S | Usseroed Kongevej 132 | DK-2970 Hoersholm | Denmark
www.licensewatch.com | ll@licensewatch.com | Phone +45 3337 8350 | Mobile +45 2780 5909

Figur 7.1 udtalelse fra LicenseWatch

Bilag

Bilag 1. Fully dressed use case.

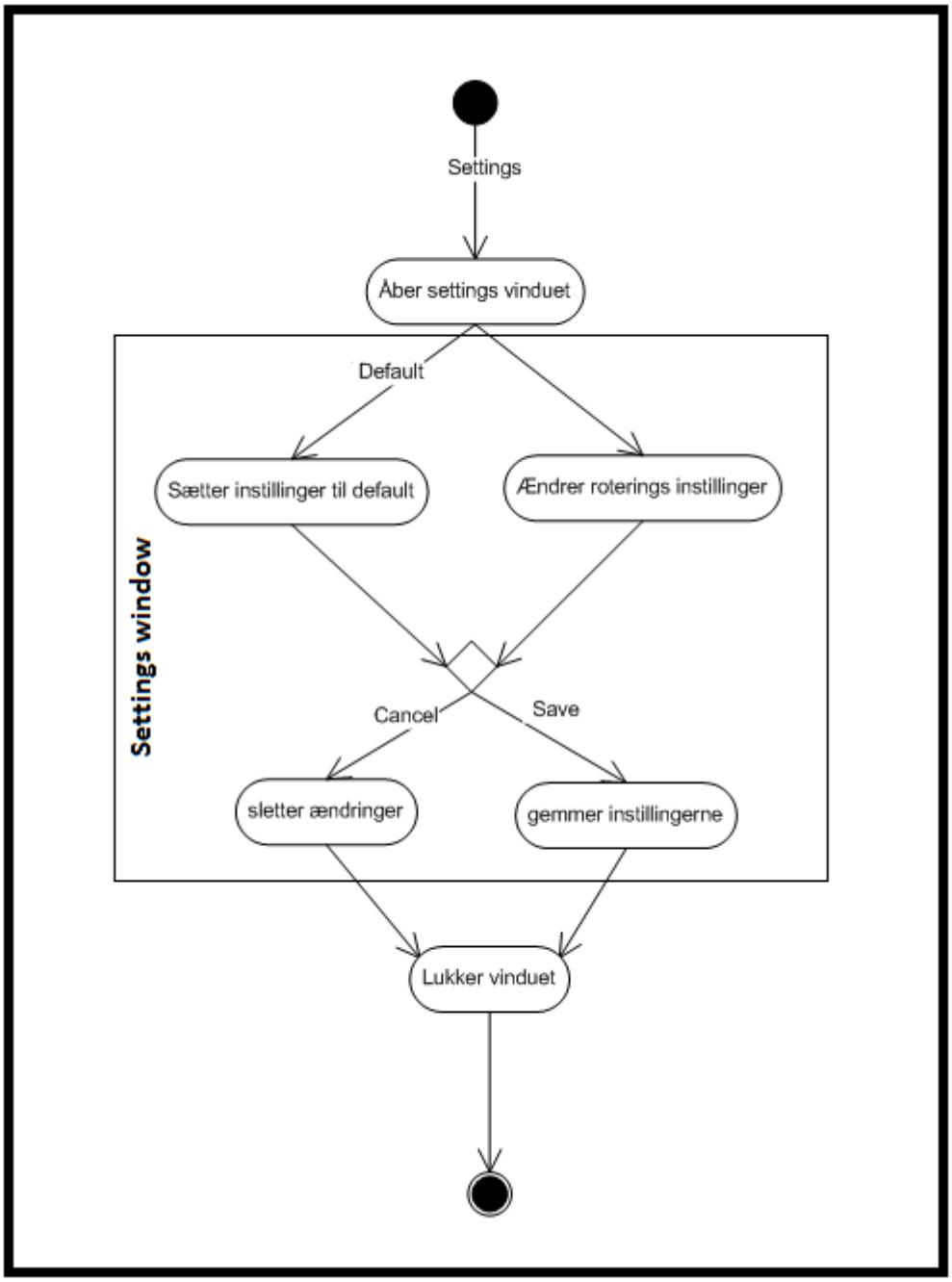
ID:	UC 1
Titel	Roter billedet
Beskrivelse	Brugeren af systemet roter billedet sådan, at de bagvedliggende søjler bliver synlige for brugeren
Primær aktør	LicenseWatch bruger
"Precondition"	Brugeren er inde på segment siden, og data fra database er klar. Samtidig ser man platformene forfra.
"Postcondition"	Billedet er vendt sådan, at søjlerne bliver vist, ud fra de roterings indstillinger der er lavet.
Hoved succes scenarie	1. Brugeren trykker på en roter knap 2. Systemet laver en flydende rotation af billedet 3. Systemet stopper med at roterer, når den har roteret sådan at indstillingerne omkring rotation er opfyldt.
Udvidelser	I denne use case er der ingen udvidelser, da hoved succes scenariet er det eneste mulige scenarie.
Hyppighed af brug	Brugeren vil ofte rotere billedet, men det er dog ikke hver gang, at brugeren har brug for at se antallet og dermed søjlerne.
Prioritet	5



Figur 8.1 Use case UC 1 aktivitets diagram

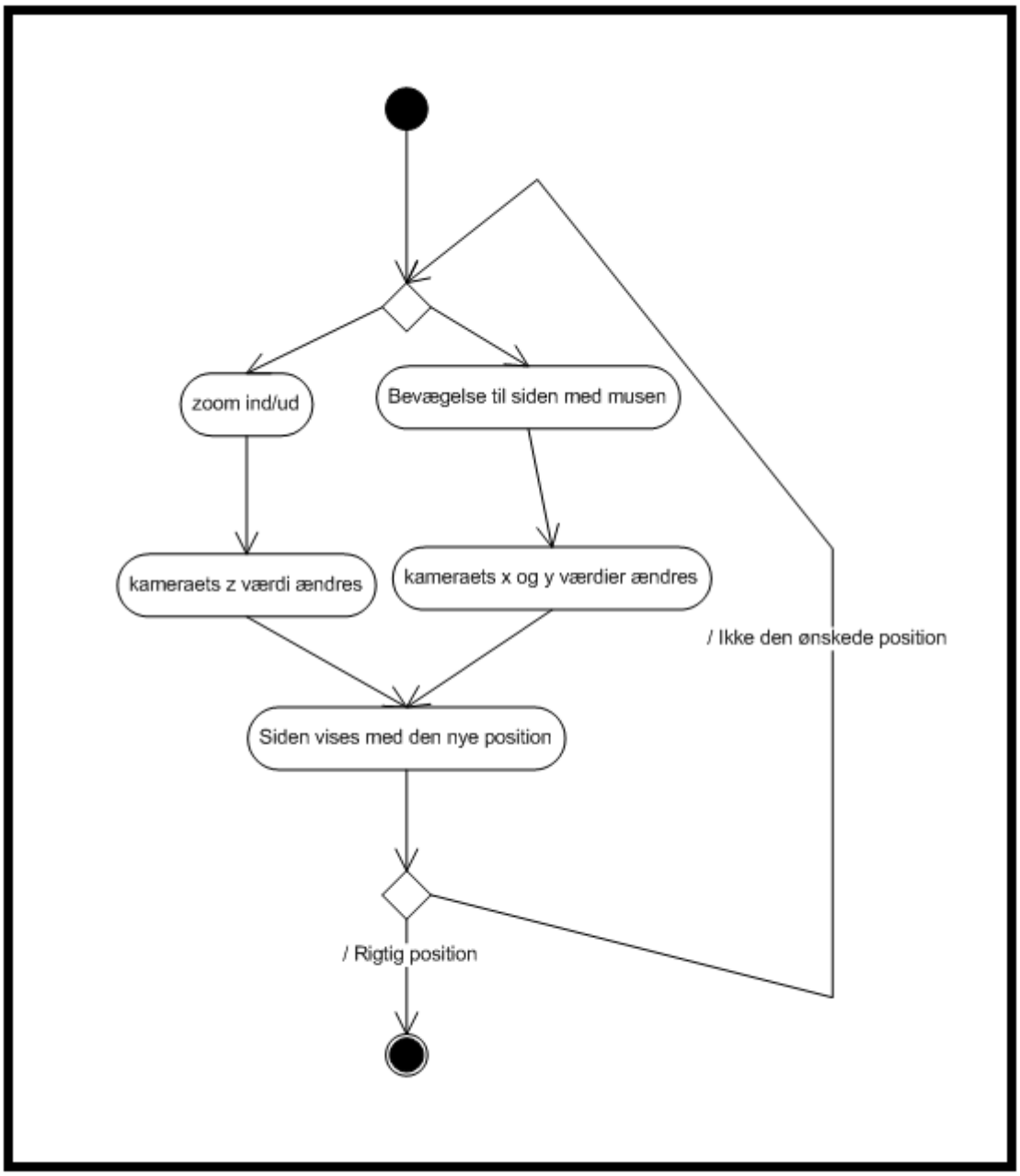
ID:	UC 2
Titel	Ændre indstillinger for rotation
Beskrivelse	Brugeren af systemet åbner "settings" vinduet og ændrer på indstillingerne for rotationen.
Primær aktør	LicenseWatch bruger
"Precondition"	Brugeren er inde på segment siden og data fra database er klar.
"Postcondition"	Roterings indstillingerne er gemte, og kan nu bruges når der roteres.
Hoved succes scenarie	1. Brugeren trykker på en "settings" knap 2. Systemet viser et nyt vindue, hvor roterings indstillingerne vises 3. Brugeren ændrer på indstillingerne, for meget der skal roteres omkring de forskellige akser 4. Brugeren trykker på "Save" knappen 5. Systemet gemmer de nye indstillinger så de kan blive brugt når der roteres. 6. Systemet lukker vinduet og man er igen i hovedvinduet
Udvidelser	3a. Brugeren ændrer ikke på alle indstillingerne, men kun en eller to af dem. 3b. Brugeren trykker på "Default" knappen, og indstillingerne sættes til at være lig default indstillingerne, sådan at brugen nemt kommer tilbage til start udgangspunktet. 4a. Brugeren trykker på "Cancel" knappen, og indstillingerne sættes til at være lig dem, der var gemte da vinduet blev åbnet. 6a. Er billedet i hoved vinduet allerede roteret, inden der blev lavet nye roterings indstillinger, så roteres billedet med det

	samme, så det passer med de nye indstillinger.
Hyppighed af brug	Det er sjældent, at denne use case vil blive brugt, da default værdierne for rotationen ofte er passende for brugeren.
Prioritet	2



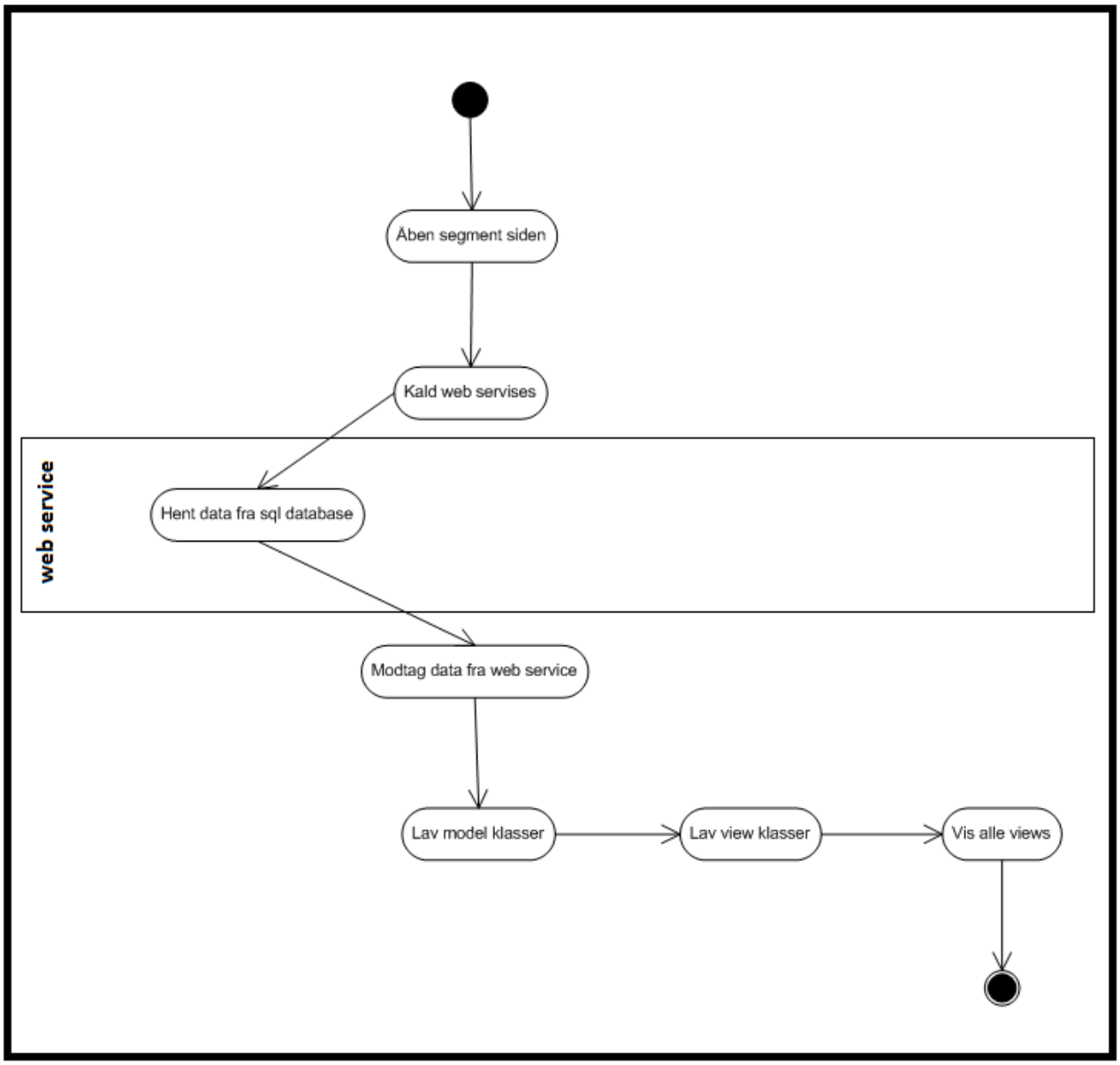
Figur 8.2 Use case UC 2 aktivitets diagram

ID:	UC 4
Titel	Bevægelse af kameraet.
Beskrivelse	Da det er forskelligt hvor mange segmenter der er på siden, og det er forskelligt om brugerne vil se alle eller fokusere på enkelte segmenter, skal det være muligt at bevæge kameraet.
Primær aktør	LicenseWatch bruger
"Precondition"	Brugeren er inde på segment siden, og data fra databasen er klar.
"Postcondition"	Brugeren har flyttet kameraet enten ved at zoome eller bevæge sig til siden eller op eller ned.
Hoved succes scenarie	1. Systemet viser en zoom slider et sted på siden. 2. Brugeren rykker med slideren og zoomer derved enten ind eller ud. 3. Systemet viser siden med den nye kamera position og brugeren har en følelse af at der er blevet zoomet.
Udvidelser	2.a Brugeren af siden, ønsker ikke at zoome, men at bevæge sig til en af siderne. Hertil flyttes musen til den ønskede side samtidig med at venste museknap holdes i bund. 3.a Systemet viser nu siden med den nye kamera position, og brugeren har en følelse af at der er blevet flyttet til en af siderne.
Hyppighed af brug	Stort set altid. Det er sjældent at start position er optimal for brugerne efter at der f.eks. efter der er ekspanderet.
Prioritet	5



Figur 8.3 Uuse case UC 4 aktivitets diagram

ID:	UC 5
Titel	Hent data
Beskrivelse	Systemet henter data fra en database, og Silverlight delen af systemet får disse data vha. web services.
Primær aktør	<<System>> Database
"Precondition"	Brugeren har valgt at åbne segment siden.
"Postcondition"	Segment siden bliver vist for brugeren, og data bliver vist som platforme og søjler.
Hoved succes scenarie	1. Systemet åbner segment siden 2. Silverlight delen af system laver med det samme web service kald. 3. Systemet henter data fra databasen, og sender det tilbage til Silverlight delen. 4. Silverlight delen af systemet modtager data fra web service kaldene. 5. Der laves model klasser ud fra hentet data. 6. Ud fra model klasserne laves views. 7. Views(søjler, platformer osv.) vises i scenen.
Udvidelser	Er der ingen segment data, er siden blank
Hyppighed af brug	Altid
Prioritet	5



Figur 8.4 Uuse case UC 5 aktivitets diagram

Bilag 2 vertex positioner

Vertex positioner for ExpandButton

Kigges der på ExpandButton view, så er vertex positionerne for dette view meget simple, se figur 9.1, da dette er en simpel kvadratisk firkant, med side længde på to. Positionerne laves ud fra, hvor ExpandButton skal placeres i forhold til platformens nederrste venstre hjørne på bagsiden af det platform view, som expand knappen placeres ved siden af.

```
//Forside
Vector3 expandLeftBottomFront = new Vector3(-3.5f, -1.0f, 1.0f);
Vector3 expandLeftTopFront = new Vector3(-3.5f, 1.0f, 1.0f);
Vector3 expandRightBottomFront = new Vector3(-1.5f, -1.0f, 1.0f);
Vector3 expandRightTopFront = new Vector3(-1.5f, 1.0f, 1.0f);
//Bagside
Vector3 expandLeftBottomBack = new Vector3(- 3.5f,  - 1.0f, -1.0f);
Vector3 expandLeftTopBack = new Vector3(- 3.5f,  1.0f, -1.0f);
Vector3 expandRightBottomBack = new Vector3(- 1.5f,  - 1.0f, -1.0f);
Vector3 expandRightTopBack = new Vector3( - 1.5f,  1.0f, -1.0f);
```

Figur 9.1ExpandButton positioner

På overstående figur ses det, at der for x værdierne bliver brugt værdierne 1.5 og 3. På denne måde opnås en bredde på 2 som ønsket. At det lige er 1.5 og 3.5 er ikke så vigtigt, da disse værdier kun henviser til, hvor langt væk fra platformen denne skal placeres. Det vigtigste er, at der er en forskel på 2 mellem de to værdier. Da ExpandButton også skal have en højde og dybe på 2, så er der også en forskel på 2 mellem de værdier der bliver brugt der.

Vertex positioner for ChildLine

```
Vector3 startPoint = new Vector3(startXY, 0);
Vector3 breakPoint= new Vector3(startXY.X, endXY.Y, 0);
Vector3 endPoint = new Vector3(endXY.X, endXY.Y, 0);
```

Figur 9.2 ChildLine positioner

Som forklaring til overstående billede, henvises til ChildLine beskrivelsen i kapitel 5 under opbygning af 3D views. Det skal her nævnes at startXY og endXY kommer fra de 2 platform views, som linjen skal tegnes imellem. Et Platform view har to properties, kaldet ChildLinePosition og ParentLinePosition, og det er disse to, som bliver brugt, hvor ChildLinePosition fra platform

view'et, der repræsenterer parent segmentet, bliver brugt som startXY, og ParentLinePosition fra platform view'et, der repræsenterer undersegmentet, bliver brugt som endXY.

Da dette view skal være en linje, så skal graphicDevice enheden have at vide, at den ikke skal tegne trekant primitiver ud fra data i vertexbufferen, og derfor får enheden at vide i dens DrawPrimitives metode, at primitive type er PrimitiveType.LineList.

Bilag 3 UV "mapping" med VertexPositionTexture

```
//Forside
vertices[54] = new VertexPositionTexture(midBottomLeftFront, midBottomRightUV);
vertices[55] = new VertexPositionTexture(bottomLeftFront, bottomRightUV);
vertices[56] = new VertexPositionTexture(midBottomRightFront, midBottomLeftUV);
vertices[57] = new VertexPositionTexture(midBottomRightFront, midBottomLeftUV);
vertices[58] = new VertexPositionTexture(bottomLeftFront, bottomRightUV);
vertices[59] = new VertexPositionTexture(bottomRightFront, bottomLeftUV);

vertices[60] = new VertexPositionTexture(midTopLeftFront, midTopRightUV);
vertices[61] = new VertexPositionTexture(midBottomLeftFront, midBottomRightUV);
vertices[62] = new VertexPositionTexture(midTopRightFront, midTopLeftUV);
vertices[63] = new VertexPositionTexture(midTopRightFront, midTopLeftUV);
vertices[64] = new VertexPositionTexture(midBottomLeftFront, midBottomRightUV);
vertices[65] = new VertexPositionTexture(midBottomRightFront, midBottomLeftUV);

vertices[66] = new VertexPositionTexture(topMidLeftFront, topMidRightUV);
vertices[67] = new VertexPositionTexture(midTopLeftFront, midTopRightUV);
vertices[68] = new VertexPositionTexture(topMidRightFront, topMidLeftUV);
vertices[69] = new VertexPositionTexture(topMidRightFront, topMidLeftUV);
vertices[70] = new VertexPositionTexture(midTopLeftFront, midTopRightUV);
vertices[71] = new VertexPositionTexture(midTopRightFront, midTopLeftUV);

vertices[72] = new VertexPositionTexture(topLeftFront, topRightUV);
vertices[73] = new VertexPositionTexture(topMidLeftFront, topMidRightUV);
vertices[74] = new VertexPositionTexture(topRightFront, topLeftUV);
vertices[75] = new VertexPositionTexture(topRightFront, topLeftUV);
vertices[76] = new VertexPositionTexture(topMidLeftFront, topMidRightUV);
vertices[77] = new VertexPositionTexture(topMidRightFront, topMidLeftUV);
```

Figur 10.1 UV mapning for søjlens forside.

Figur 10.1 viser, hvordan uv "mapping" foregår vist med koden til søjlens forside, hvor den texture der bliver brugt, er det specifikke søjle billede. Her kan man se, at siden består af fire firkanter, som hver repræsenteres med to retvinklede trekanter, som hver skal angives med tre VertexPositionTexture objekter. Rækkefølgen hvorpå de bliver lagt i vertecies array'et er meget vigtig, og trekanternes tre VertexPositionTexture objekter skal indsættes efter hinanden. Havde det været en VertexPositionColor, så skulle dennes konstuktør have en Color værdi i stedet for en Vector2 UV koordinat.

Bilag 4 Oprettelse af Pillar view

```
//---------søjler når segmentet ikke er ekspanderet ----------------------------------------------------
Dictionary<string, int> sum = new Dictionary<string, int>();
sum.Add("Users", 0);
sum.Add("Workstations", 0);
sum.Add("Assets", 0);
sum.Add("Licenses", 0);
sum = getSumCounts(segment, sum);
int userSumCount = sum["Users"];
if (!(userSumCount < 1))
{
    Pillar pillarUser = new Pillar(segmentPosition, 0, ((float)userSumCount / (float)totalUserCount) * 20, 2, userSumCount);
    pillarUser.SegmentID = segment.SegmentID;
    IviewInterface viewPillarUser = pillarUser;
    tempViewObjects.Add(viewPillarUser);
}
```

Figur 11.1 Oprettese af bruger Pillar(søjle) view

På overstående billede ses, hvordan et Pillar(søjle) view bliver lavet. Dette tilfælde viser oprettelsen af en søjle, der skal vise ”user” antallet for et segment, der ikke er ekspanderet. Først oprettes en tom Dictionary liste, og i den indsættes så fire int værdier med værdien 0 og de får hver en string nøgle, så man kan se hvilket antal de forskellige værdier hører til. Derefter kaldes getSumCounts metoden, og den får den segmentModel instans der kigges på, samt sum listen med som parametre. Når denne rekursive metode er færdig, er Dictionary listen sum, så opdateret, sådan at den nu indeholder summerne af segmentets og dens undersegmenter antal.

Hvis summen af brugere overstiger 0, så laves der så et Pillar view. Denne får først positionen af segmentet, og derefter offset værdien der viser, hvor søjlen placeres bag på segmentet, i dette tilfælde i bunden. De efterfølgende værdier i Pillar konstruktøren er højden, der bliver udregnet som en procent af 20, en type værdi, der viser at det er en user søjle(1->workstation, 2->user,3->assets, 4->license), og derfor skal user billedet bruges som texture og til sidst den sum som skal skrives på bunden af søjlen.

Når Pillar view'et er oprettet puttes den til sidst i tempViewObejcts listen, hvor alle view objekterne bliver gemt, inden de kopieres over i ViewObjekt listen.

Bilag 5 Shader filer

```
// transformation matrix provided by the application
float4x4 WorldViewProj : register(c0);

// vertex input to the shader
struct VertexData
{
  float3 Position : POSITION;
  float2 UV : TEXCOORD;
};

// vertex shader output passed through to geometry
// processing and a pixel shader
struct VertexShaderOutput
{
  float4 Position : POSITION;
  float2 UV : TEXCOORD;
};

// main shader function
VertexShaderOutput main(VertexData vertex)
{
  VertexShaderOutput output;

  // apply standard transformation for rendering
  output.Position = mul(float4(vertex.Position,1), WorldViewProj);

  // pass the UV coordinates through to the next stage
  output.UV = vertex.UV;

  return output;
}
```

Figur 12.1

```
texture texture1 : register(t0);
sampler sampler1 = sampler_state
{
    texture = <texture1>;
};

// output from the vertex shader
struct VertexShaderOutput
{
  float4 Position : POSITION;
  float2 UV : TEXCOORD;
};

// main shader function
float4 main(VertexShaderOutput vertex) : COLOR
{
  return tex2D(sampler1, vertex.UV).rgba;
}
```

Figur 12.2

Figur 12.1 og 12.2 viser en vertex shader og pixel shader, der kan bruges sammen med en texture. De er lavet i HLSL (High Level Shader Language). Shaderne der bruges sammen med en farve i dette projekt, er stort set magen til. Forskellen er at i stedet for en UV koordinat, så skal vertex shaderen have en Color, og det er så denne color der bliver output i pixel shaderen.

Billag 6 Unit test

Hentning af data fra sql server

```
[TestMethod]
[Asynchronous]
public void testNumberOfSegmentsFromSQL()
{
    bool done = false;
    DataFromSql.Instance.AsyncCallbackCompleted += (() => done = true);
    EnqueueCallback(() => DataFromSql.Instance.FetchDataFromSql());
    EnqueueConditional(() => done);
    EnqueueCallback(() =>
    {
       //result checking
       String error = "";
       bool result = true;
       if (DataFromSql.Instance.AllModelsList.Count != 92)
       {
           result = false;
           error += "\n The number of segments should be 92. The Count was: "
                    + DataFromSql.Instance.AllModelsList.Count;
       };

       if (DataFromSql.Instance.AllModelsList[3].SegmentID != 3)
       {
           result = false;
           error += "\n The segment id is not correct for segment with id=3, the test showed that is was: "
                    + DataFromSql.Instance.AllModelsList[3].SegmentID;
       }
       //osv.
       Assert.IsTrue(result, error);

    });
    EnqueueTestComplete();
}
```

Figur 13.1 metode til at teste asynkrone kald

På overstående figur ses, hvordan det i test klassen skal skrives, hvis der ønskes at teste asynkrone kald. Først oprettes en bool variabel "done". Inde i DataFromSql klassen, er der lavet en AsyncCallbackComplete eventhandler, og dennes callback sættes til at sætte done variablen til true vha. lambda expression. Efterfølgende kaldes den metode, der starter de asynkrone webservice kald, og når disse er færdige, bliver AsyncCallbackCompleted fyret afsted og done bliver så sat til true. EnqueueConditional metoden betyder, at testen skal vente, indtil done er sat til true, og derfor venter testen indtil de asynkrone kald er færdige. I den efterfølgende EnqueueCallback kan resultaterne blive testet op mod det forventede resultat, og i bunden laves så metodens assertion. På figuren er der kun vist, hvordan der tjekkes for om det samlede antal af SegmentModel instanser er rigtigt, og om SegmentID er rigtigt for en enkel instans i stikprøven. De

resterende SegmentID, ParentID, SegmentName og Niveau tjek er udeladt på billedet pga. overskuelighed, men i den rigtige test, er de placeret efter linjen "//osv.".

Omdannelse af model data til view objekter

```
[TestMethod]
public void TestgetSumMethod()
{
    DataFromSql data = DataFromSql.Instance;

    Creation of SegmentModel

    Random adding SegmentModels to SegmentChildren list

    Dictionary<string, int> sum = new Dictionary<string, int>();
    sum.Add("Users", 0);
    sum.Add("Workstations", 0);
    sum.Add("Assets", 0);
    sum.Add("Licenses", 0);
    sum = data.getSumCounts(parentSegment, sum);
    bool userCountCorrect = false; bool workstationCountCorrect = false;
    bool assetCountCorrect = false; bool licenseCountCorrect = false;

    if (sum["Users"] == 1032) { userCountCorrect = true; }
    if (sum["Workstations"] == 922) { workstationCountCorrect = true; }
    if (sum["Assets"] == 896) { assetCountCorrect = true; }
    if (sum["Licenses"] == 2394) { licenseCountCorrect = true; }

    Assert.IsTrue(
        userCountCorrect && workstationCountCorrect && assetCountCorrect && licenseCountCorrect,
        "\n user sum is :" + userCountCorrect + "\n workstation sum is :" + workstationCountCorrect +
        "\n asset sum is :" + assetCountCorrect + "\n license sum is :" + licenseCountCorrect
        );
}
```

Figur 13.2

På overstående billede ses, hvordan getSumCounts metoden bliver testet. I de minimerede regioner "Creation af SegmentModel" og "Random adding SegmentModels to ChildrenList", sker der ikke andet end at der bliver oprettet en masse SegmentModel objekter og der bliver givet værdier til de forskellige count properties. Derudover bliver objekterne random puttet i hinandens SegmentChildren lister.

View og Projection matricerne

```
Scene scene = new Scene();
[TestMethod]
public void TestChangeCamera()
{
    scene.State.MoveXVal = 10;
    scene.State.MoveYVal = 20;
    scene.State.ZoomVal = 5;
    Matrix sceneView = scene.changeCameraView();

    Vector3 testPosition = new Vector3(44.0f, 43.0f, 125.0f);
    Vector3 testTarget = new Vector3(44.0f, 43.0f, 0.0f);
    Matrix testView = Matrix.CreateLookAt(testPosition, testTarget, Vector3.Up);

    Assert.AreEqual<Matrix>(testView, sceneView,"the view matrix is not correct after the changeCameraView method");
}
```

Figur 13.3

På billedet ses, hvordan der testes for, om den matrix, der bliver returneret af changeCameraView i scene klassen er korrekt. Matricen testes imod den matrix, der kommer ved direkte brug af Matrix.CreateLookAt. og det skal her siges, at værdierne i testPosition og testTarget kommer ved at lægge 10, 20 og 5 til start værdierne for kameraets position (34,23,120) og target (34,23,0). Til sidst i test metoden køres der så en assertion på, om de to matricer er ens.

Testen TestChangeProjection der tester projection matricen foregår på samme måde. Her er det bare værdien AspectRatio i State klassen, der giver forholdet mellem højde og bredde på skærmen, som skal bruges. Denne værdi bruges af changeProjection metoden, der returner det nye viewing fustrum i form af en projection matrix. Igen testes der for, om denne matrix er lig den matrix der fås, ved direkte brug af Matrix.CreatePerspectiveFieldOfView metoden, sammen med den ønskede aspect ratio.

Billag 7 Brugertest

Brugertest

Test del 1. Her er det kun svarene der er vigtige

Spørgsmål	Svar
1. Hvor mange licenser hører specifikt til "Göteborg" segmentet?	
2. Hvor mange segmenter har "Norge" segmentet som top(parent) segment?	
3. Har segmentet "Salg" nogle brugere specifikt tilknyttet sig? *Stien: "England>London>Salg"*	Ja/nej
4. Hvilket segmentID har segmentet "IT"? SegmentID kan ses på segmentets summary side. *Stien: "Danmark>Fyn>IT"*	
5. Hvad er navnet/navene på det/de segment/segmenter der ligger under *"Norge>Oslo>IT"*	
6. Uden at få givet den præcise sti til segmentet, hvad er da antallet af computere, der tilhører "Salg" segmentet, som er et undersegment til "New York" segmentet? Antallet der ledes efter, er det samlede antal for "Salg" segmentet og dens undersegmenter.	
7. Samme spørgsmål som før, men her bedes om det antal der specifikt hører til "Salg" segmentet.	
8. Hvad er segmentID'et for segmentet "Offentlig"? og denne gang må summary siden ikke bruges. (se brugervejledning) *Stien: "Danmark>Sjælland>Marketing>Offentlig"*	
9. Hvad er dybden af segment strukturen for "Norge " segmentet og dens undersegmenter?	
10. Når ingen af segmenterne er ekspanderet, hvilket segment har så næst flest licenser?	

Test del 2. Her er det både svarene og antal handlinger der kigges på.

	Spørgsmål	Svar
1.	Hvor mange segmenter i England har ingen brugere, computere, licenser og aktiver?	
2.	Hvor mange søjler har segmentet "Salg" når denne er ekspanderet? *Stien: "Sverige>Stockholm>Salg"*	
3.	Hvis der kun er roteret ca. 180 grader omkring y aksen, hvad er da rækkefølgen af søjlerne? rækkefølgen skal nævnes fra toppen af skærmen og ned.Det er nok at nævne rækkefølgen for et segment	
4.	Få siden tilbage til det samme udgangspunkt som når siden lige er åbnet.(ønskes ingen svar)	Skal ikke bruge svar i denne
5.	hvad er SegmentID'et for segmentet IT ? *Stien: "Danmark>Jylland>IT"*	
6.	Hvor mange brugere har segmentet Privat? *Stien: "Danmark>Fyn>Salg>Privat"?*	
7.	Hvilket segment af Jyllands undersegmenter: Salg, Marketing, IT og Produktion, har flest brugere?	

Billag 8 Brugervejledning

Segment page

For at ekspandere et segment, trykkes på expand knappen der viser et "+" og for at minimere et segment trykkes "-".

For at gå til summary siden for et segment, så trykkes på segmentet med musen. På summary siden kan SegmentID'et bla. ses.

For at gå til users, workstations, assets eller licenses siderne for et specefikt segment, så trykkes på dertilhørende søjler. På siderne kan SegmentID'et bl.a. også ses.

Et ikke ekspanderet segment viser antallet for den selv og alle dens undersegmenter, men er segmentet ekspanderet, så viser segmentet kun de antal der specifikt hører til den.

For at bevæge sig til siderne, og op og ned, holdes venstre museknap i bund, og der flyttes med musen.

For at zoome ind og ud bruges zoom slideren øverst i venstre hjørne.

"Default" knappen nulstiller både zoom og bevægelser på siden, sådan at man ender i startposition

"Settings" knappen bruges til at åbne en settings side, hvor indstillingerne for rotering kan ændres.

Er der ikke roteret på siden, så vises "Roter" knappen, og trykkes der på denne, så roter segmenterne i forhold til indstillingerne. Er der roteret, så vises "Normal" knappen. Trykkes der på den, så nulstilles roteringen og segmenterne bliver vist forfra.

Settings page

Denne side står for roterings indstillingerne. Som default bliver der roteret 90 grader om x og z akserne.

Til at ændre roterings værdierne bruges de 3 slidere.

"Default" knappen sætter værdierne til at være de 90 grader om x og z akserne.

"Save" knappen sætter slider værdierne til at være de nye roterings værdier og lukker vinduet. Når der efterfølgende trykkes "roter", så bruges de nye værdier.

"Cancel" knappen sætter værdierne til at være lig dem, som var gemt da vinduet åbnede.

Litteratur liste

[1] " UVMapping.png", Wikipidia, http://en.wikipedia.org/wiki/UV_mapping (September 2011)

[2] " UVMapping.png", Wikipidia, http://en.wikipedia.org/wiki/File:UVMapping.png (September 2011)

[3] Var Der Byl, Leigh, "Understanding Mapping Coodinates", http://teaching3d.com/resources/articles/uv_mapping_theory.pdf (September 2011)

[4] "Shader" Wikipidia, http://en.wikipedia.org/wiki/Shader (September 2011)

[5] "World, View and Projection Matrix Unveiled", rovertokoci, http://robertokoci.com/world-view-projection-matrix-unveiled/ (Oktober 2011)

[6] "Spaces and Matrix in XNA", Toymaker, http://www.toymaker.info/Games/XNA/html/xna_matrix.html (September 2011)

[7] " 3-D Graphics Overview", Microsoft, http://msdn.microsoft.com/en-us/library/gg197424(v=XNAGameStudio.35).aspx (September 2011)

[8] Johnson, David," Silverlight 3D : Tri-Winning (Part 1)", Clarity Consulting, http://blogs.claritycon.com/blog/tag/silverlight-5/ (oktober 2011)

[9] " Alice in Mirrorland – Silverlight 5 Beta and XNA", onterrasystems, http://blogs.onterrasystems.com/gisblog/?p=700 (September 2011)

[10] "Walkthrough: Creating and Animating a 3-D Textured Cube in Silverlight",Microsoft, http://msdn.microsoft.com/en-us/library/gg197425(v=XNAGameStudio.35).aspx (August 2011)

[11] "3-D Graphics Overview", Microsoft, http://msdn.microsoft.com/en-us/library/gg197424(v=XNAGameStudio.35).aspx (September 2011)

[12] "Colored (Non-Textured) 3D Cube Sample (Silverlight)", Microsoft , http://code.msdn.microsoft.com/Colored-Non-Textured-3D-c0e3cda6 (September 2011)

[13] "How to: Detect Whether a User Clicked a 3D Object", Microsoft, http://msdn.microsoft.com/en-us/library/bb203905(v=xnagamestudio.10).aspx (November 2011)

[14] "XNA Collision Detection for 3D models – Part 1", Sharky's blog, http://sharky.bluecog.co.nz/?p=108 (November 2011)

[15] Angel, Edward: *OpenGL: A PRIMER,* New York, Addison-Wesley,2002.

[16] Angel, Edward: *Interactive computer graphics: A Top-Down Approach Using OpenGl, New York*, third Edition, Addison-Wesley,2003.

[17] L. Norman, Kent: *Cyperpsychology: An introduction to Human-Computer interaction,*Cambrigde, Cambrigde universit press, 2008

[18] Dispatcher.BeginInvoke Method, Microsoft, http://msdn.microsoft.com/en-us/library/cc190259(v=vs.95).aspx (November 2011)

[19] "Lambda Expressions", Microsoft, http://msdn.microsoft.com/en-us/library/bb397687.aspx (November 2011)