HÖGSKOLAN SKÖVDE

Institutionen för kommunikation och information
Examensarbete i datavetenskap 20p
C-nivå
Vårterminen 2007

Databaslagring av tredimensionella datorspelsobjekt i realtid

Tor Andersson

Tor Andersson DSUP04

Databaslagring av tredimensionella datorspelsobjekt i realtid

Examensrapport inlämnad av Tor Andersson till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information. Arbetet har handletts av Mikael Thieme.

2007-09-02

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Databaslagring av tredimensionella datorspelsobjekt i realtid

Tor Andersson

Sammanfattning

För att enbart kunna lagra ett tredimensionellt objekt i en databas krävs inte en väldigt optimerad databasdesign. Men beroende på användningsområdet av databasen kan det krävas stora optimeringar för just detta. I jämförelse med ett dataspel som har en framerate över 30 fps (uppdateringar per sekund) så är en databas väldigt långsam. Eftersom att databasen sett ur ett spels perspektiv tar lång tid på sig att svara när man frågar den något kommer detta att innebära stora problem när man försöker kombinera de två. Det finns dock saker man kan göra för att öka prestanda på en databas, exempelvis att optimera själva frågorna till databasen. Men framför allt kan man optimera själva databasdesignen. Självklart kan man utveckla spelet så att det inte nödvändigtvis behöver ha kontakt med databasen i varje uppdatering. Detta är något som denna rapport kommer att gå in på djupet med och då läggs större vikt på databassidan än på spelsidan.

Nyckelord: Databas, Optimering, Tredimensionella Objekt, Dataspel

Innehållsförteckning

1 Introduktion

De senaste åren har en genre i dataspelens värld fått nytt liv, då den har tagit steget in i grafiska tredimensionella världar från att ha varit textbaserade världar som endast funnits i spelarnas fantasi. Denna genre är MMORPG, ett exempel på just ett sådant spel är World Of Warcraft (2004).

Då steget in i den tredje dimensionen togs av denna genre innebar det stora förändringar i spelmekaniken. En av de största förändringarna är att spelen behöver lagra betydliga mängder mer data. Detta innebär att någon form av lagring för all den data måste utvecklas. Eftersom databaser är till för att på ett väl strukturerat sätt lagra stora mängder data, lämpar de sig väl att användas till detta. Eftersom databaser även är säkra vad gäller åtkomst till data som är lagrad i dem, innebär det att fusk kan förhindras genom att använda databaser. Det är dessutom möjligt att med hjälp av en databas spara ner tillräckligt mycket information om en spelvärld för att kunna återställa denna efter exempelvis en systemkrasch eller liknande. Är kopplingen mellan världen och databasen i realtid (vilket innebär att databasen hela tiden hålls uppdaterad enlig världen och vise versa) kan en administratör enkelt göra förändringar och påverka världen endast genom att ändra i databasen. Detta är något som hade varit betydligt krångligare utan kopplingen till databasen.

Det finns dock ett stort problem med dagens databaser vilket är att de är långsamma sett ur ett datorspels perspektiv. Skall en databas kunna användas till att lagra data från ett datorspel måste den även kunna tillhandahålla den data som lagras så att spelet utan begränsningar har tillgång till den. Det är just en sådan lösning som denna rapport undersöker.

2 Bakgrund

En databas är ett bra alternativ till att på ett säkert sätt lagra stora spelvärldar, något det finns indikationer på att de stora spelutvecklarna använder sig av och då främst Blizzard Entertainment World Of Warcraft.

Figur 1. *Bilden är hämtad ur World of Warcraft vilket är enligt www.mmogchart.org (2007) dagens största MMORPG och visar hur ett sådant spel kan se ut.*

Tyvärr är en databas en långsam applikation i jämförelse med ett datorspel. Detta innebär stora problem med att hålla databasen uppdaterad samt att se till att alla berörda användare får reda på uppdateringen. I ett spel är det viktigt att alla användare får reda på saker som händer i världen, om en klient inte får reda på en händelse tillräckligt snabbt kan det uppstå problem. I ett spel där en spelare kan hugga ner ett träd exempelvis, skulle det då ta en längre tid innan de andra spelarna blir varse om att trädet har blivit nerhugget kan problem uppstå. Dessa problem skulle kunna vara att andra spelare försöker hugga ner träd som egentligen inte finns. Detta innebär det att någon lösning till att använda sig av en databas till att lagra speldata även om den är långsamma. Med hjälp av att optimering av databasen kan dess hastighet ökas betydligt. Dock inte tillräckligt mycket för att databasen skall kunna användas i ett spel men det är en början.

Enligt Quax, P. Jehaes, T. Jorissen, P. Lamotte, W. (2003) används databaser redan till att lagra speldata. Dock avslöjas inte hur detta kan åstadkommas. Att information om hur problem i sådana applikationer undanhålls kan till vis del förstås då antagligen företagen inte vill låta andra ta del av deras arbete.

Det finns mycket som indikeringar på att databaser används i spel, men inte hur problem som uppstår kan lösas. Detta anses vara något som gör just detta problem intressant att jobba med.

2.1 MMORPG

MMORPG är en förkortning på Massive Multiplayer Online Role Playing Game. Som namnet antyder är ett MMORPG en speciell typ av online baserat rollspel. Ett MMORPG känns igen på att det är många spelare i en och samma värld, samt att

världen är persistent det vill säga att världen alltid finns kvar och förändras även om det inte för tillfället är någon spelare inloggad i spelet.

Nedan följer två bilder hämtade ur MMORPG spelet World Of Warcraft och visar exempel på hur ett sådant spel kan se ut. Den första bilden visar ett antal olika karaktärer som interagerar med varandra. Den andra bilden är tagen då några av dessa karaktärer är ute på någon form av uppdrag.

Den första av dessa två bilder (Figur 2) visar exempel på interaktion mellan olika spelare. Interaktionen mellan spelare i MMORPG är viktig för att driva spelet framåt. Den kan vara som i detta fall att planera inför ett uppdrag, men kan också vara att utbyta information om platser i spelet. Interaktionen kan även gå ut på att köpa eller sälja föremål mellan de olika spelarna. Genom interaktionen mellan spelare kan många av de uppdrag eller äventyr i spelet skapas, då denna göra att spelare kan samarbetar mot gemensamma mål.

Figur 2. *Bilden är hämtad ur spelet World of Warcraft och visar hur det kan se ut då spelare i ett MMORPG interagerar med varandra.*

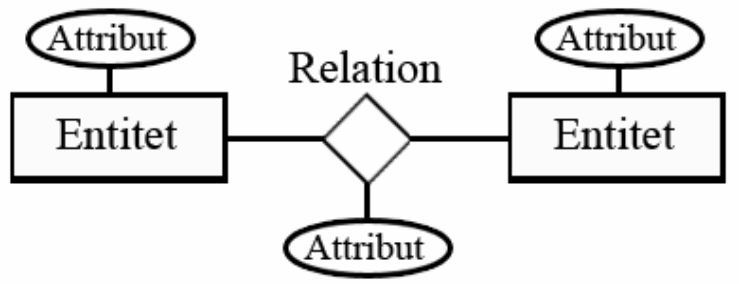
Bilden här nedan är den andra av de två som beskrevs tidigare i detta kapitel. Denna bild visar ett exempel på uppdrag i ett MMORPG. Uppdraget som just denna bild visar är en strid mellan olika karaktärer i spelet. Dock behöver inte nödvändigtvis uppdragen i ett MMORPG gå ut på att strida med andra karaktärer. De kan vara så enkla som att på något vis skaffa mat till sin karaktär, eller samla resurser som kan användas till att skapa något föremål. Detta är dock bara exempel på vad för uppdrag som karaktärerna i ett MMORPG kan utsättas för.

Figur 3. *Bilden är hämtad ur spelet World of Warcraft och visar hur det kan se ut då spelare i ett MMORPG är ut på någon form av uppdrag.*

Ett MMORPG är ofta ett stort spel som består av många olika delar som samarbetar på olika sätt. För att skapa en uppfattning om hur ett sådant spels olika delar fungerar och hänger samman är det lämpligt att använda sig av någon form av grafiskt representation, exempelvis ett ER-diagram.

2.2 ER-Diagram

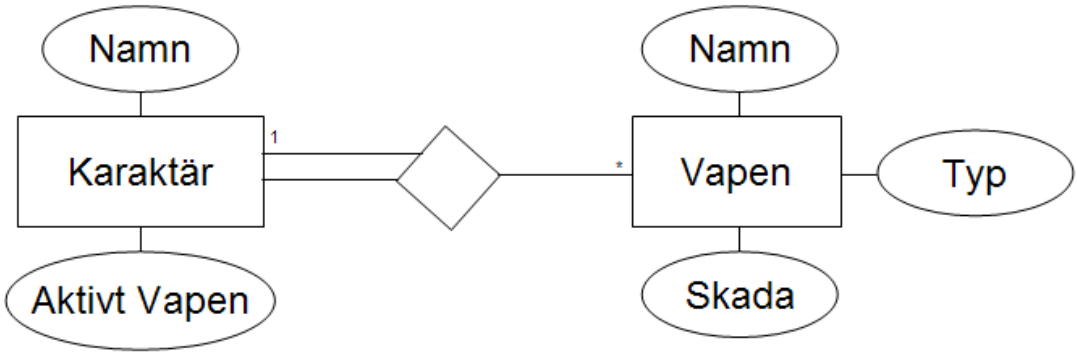
ER är en förkortning av Entity-Relationship vilket är precis vad ett sådant diagram beskriver, det vill säga hur olika delar av exempelvis en databas är kopplade till varandra. Dock kan ER-diagram användas för att beskriva mycket annat än databaser, tillexempel en fabriks produktionslinje. Tänk dig en fabrik som skall tillverkar en bilmotor vilken består av ett visst antal delar vilka tillverkas på olika ställen i fabriken. Ett ER-diagram kan då beskriva hur fabrikens olika delar hänger ihop endast genom att använda sig av information om de olika motordelarna som tillverkas i olika delar a v fabriken.



Figur 4. *Bilden beskriver de olika delarna av ett ER-diagram.*

Som namnet indikerar använder sig ER-diagram primärt av två saker, entiteter samt relationerna mellan dessa. Dock har entiteterna oftast ett antal attribut av olika slag samt att även relationerna kan ha attribut. Ett attribut är något som entiteten består av eller innehåller. I exemplet med motorfabriken skulle en av entiteterna tillexempel kunna tillverka kolvar till motorn, då skulle ett attribut kunna vara diametern på kolven eller någon annan information om kolven. Relationerna kan se ut som i bilden

nedan, denna relation säger att en karaktär kan ha flera vapen men ett vapen kan bara finnas hos en karaktär. Vad relationen säger kan utläsas av om den har som i detta fall dubbla sträck till en av entiteterna, samt vad som står skrivit där relationen möter entiteten. Siffran ett vilken står skriven ovanför relationen där den möter entiteten Karaktär indikerar på att ett vapen endast kan finnas hos en karaktär. Det samma gäller för stjärnan vid entiteten Vapen, dock säger denna att en Karaktär kan ha flera vapen.



Figur 5. *Ett exempel på ER-diagram som det skulle kunna se ut då det används i ett datorspel.*

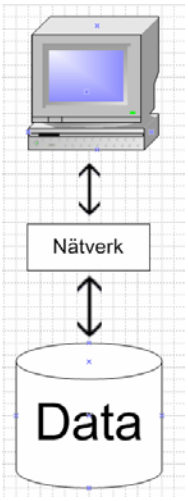
Bilden ovan visar ett exempel på hur ett ER-diagram kan användas för att beskriva förhållandet mellan olika delar i ett datorspel. Just detta exempel beskriver hur en karaktär hänger ihop med de olika vapnen som finns i ett datorspel. Diagrammet beskriver att en karaktär kan ha ett namn samt ett aktivt vapen, ett vapen kan ha ett namn, en skada samt en typ. Diagrammet beskriver även att en karaktär kan ha många vapen men ett vapen kan endast finnas på en karaktär.

2.3 Databas

Då en databas skall utvecklas är det lämpligt att först modellera den med hjälp av någon form av grafiskt verktyg. Detta för att många oklarheter samt brister lättare kan upptäckas i ett tidigt skede och på så sätt undvikas. ER-diagram är ett exempel på sådant verktyg, vilket lämpar sig bra till detta då det kan utformas så det beskriver alla tabeller vilka databasen kommer att innehålla samt hur relationerna mellan dessa ser ut och fungerar.

Enligt Nationalencyklopedin (1990) är en databas ”mängd av data, ordnade i ett eller flera dataregister, som är tillräckligt för ett visst ändamål eller för ett visst databehandlingssystem”.

Bilden nedan visar en representation över en dator som är kopplad till en databas genom någon form av nätverk. Nätverket kan ha/bestå av olika komponenter beroende på vad databasen skall användas till. Det kan vara så enkelt som att databasen ligger på datorns egen hårddisk i det fallet utgörs nätverket av själva datorn och databashanteraren. Nätverket kan också utgöras av kopplingen mellan datorn i fråga och en server som innehåller databasen. I detta fall kan flera olika datorer kopplas till samma databas så att alla kommer åt och kan dra nytta av data som är lagrad i den.



Figur 6. *Bilden representerar en dator som är kopplad till en databas genom någon form av nätverk.*

Rent tekniskt sett är en databas ett antal tabeller som på något sätt är relaterade till varandra. Tabellerna består av ett antal rader vilka är indelade i olika kolumner med bestämd datatyp.

Figur 5 visar ett exempel på ett ER-diagramm vilket används för att beskriva en databas grafisk. Då en databas skall implementeras måste ER-diagrammet först översättas till de tabeller som utgör själva databasen. Just detta exempel kommer att ge upphov till tre sådana tabeller. Den första kommer att utgöras av entiteten Karaktär och ha kolumnerna Namn samt Aktivt Vapen. Den andra utgörs av entiteten Vapen och denna kommer få kolumnerna Namn, Typ samt Skada. Den tredje tabellen som fås av detta exempel är en tabell vilken innehåller relationen mellan de två andra tabellerna. Det lämpligaste att välja att ha i kolumnerna i denna tabell är namnet på karaktären samt namnet på det vapnet som för tillfället är aktivt.

Karaktär

Namn	Aktivt Vapen
Tor	Mjölner
Oden	Gungner

Karaktär Vapen Relation

Karaktärs Namn	Vapen Namn
Tor	Mjölner
Oden	Gungner

Vapen

Namn	Typ	Skada
Mjölner	Hammare	999999
Gungner	Spjut	999999
Blåtunga	Svärd	512

Figur 7. *Bilden visar ett exempel hur de tre tabellerna skulle se ut och hur de är sammankopplade. Namnen på karaktärerna samt vapnen är helt tagen ur luften och har ingenting med funktionen på tabellerna att göra.*

Det exempel har som beskrivs ovan gett upphov till tre tabeller vilka kommer att innehålla många rader, vilket kommer detta att begränsa hastigheten på databasen. Eftersom denna databasdesign kommer att ge upphov till en onödigt invecklad och långsam databas bör den optimeras om den skall användas i ett spel.

2.4 Databasoptimering

Då användning av databaser i spel kräver att databasen har snabb åtkomst är optimering en viktig punkt att titta på.

Databashanteraren optimerar de frågorna som ställs till viss del, detta är något som sker automatiskt. Det är dock möjligt att påverka denna optimering ytterligare genom att ställa frågorna på specifika sätt. I boken Datalager (1997) beskrivs hur optimeringa av frågorna sker samt vad som kan göras för att påverka detta. Det som databasadministratören kan göra de största optimeringarna på är dock själva strukturen i databasdesignen. Sådan optimering går till stor del ut på att de tabeller som utgör databasen omarbetas. Det är då med utgångs punkt att eliminera antalet rader tabellen kommer att innehålla så långt som möjligt, något som kan göras genom att byta plats på data så att tabellen istället består av fler kolumner. Även de relationer som binder samman de olika tabellerna kan ha stor inverkan på databasens tidsåtgång, då dessa gör att en sökning i databasen går genom flera tabeller.

2.5 BLOB

De flesta vanliga datatyperna så som int (heltal), double (flyttal), varchar[] (char med en bestämd längd), med flera kan användas i en databas. Eftersom dessa datatyper har en specifik förutbestämd mängd minnesallokering, samt att de har specifika användningsområden lämpar de sig inte till att lagra stora mängder data. För att effektivt lagra stora mängder data är det lämpligt att omvandla data till binär form då detta minskar minnesåtgången.

Då det gäller att lagra stora mängder data i en databas finns det två olika datatyper som lämpar sig. Dessa är TEXT och BLOB, de är i stort sätt identiska med skillnaden att TEXT används för att lagra just text och BLOB används för lagring av binärdata.

BLOB är en förkortning av Binary Large Object och är en speciell datatyp som används i databaser, se DuBois (2000). Som namnet indikerar lagrar en BLOB data binärt vilket innebär nästan vad som helst kan lagras i dem. Det finns 4 stycken olika typer av BLOB, dessa är TINYBLOB, BLOB, MEDIUMBLOB och LONGBLOB. Den enda skillnaden mellan dem är hur mycket data de kan lagra förutom det är de identiska. Mängden data de kan lagra varierar ordentligt mellan de olika typerna vilket visas i tabellen nedan (MySql, 2007).

Datatyp	Lagringskapacitet
TINYBLOB	256 byte
BLOB	65535 byte
MEDIUMBLOB	16 Megabyte
LONGBLOB	4 Gigabyte

Figur 8. *De olika BLOBtyperna med respektive lagringskapacitet.*

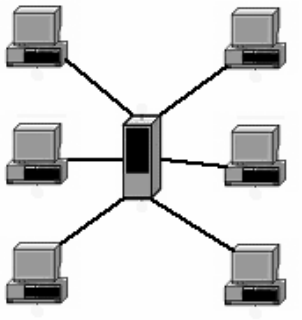
BLOB är även skiftlägeskänslig (vilket innebär att den känner skillnad på stora och små bokstäver) vilket ger ännu större variation på den data som kan lagras. Vanliga användningsområden är exempelvis lagra filmer, ljud, bilder, stora text filer med mera. Att de har så stor frihet med vad och hur mycket som kan lagras i dem innebär stora möjligheter att använda dem i dataspelssammanhang.

2.6 Binäromvandling

Då användandet av BLOB kräver att data som skall lagras är i binär form, måste all data först omvandlas. Att bara omvandla data till binärdata är inget större problem dock finns det specialtecken vilka har olika betydelse när en dator läser dem i binär form. Ett exempel på sådant tecken är NULL (\0) vilket indikerar slutet på en textsträng då en dator läser det. Detta innebär att all data som efterföljer ett NULL-tecken kommer gå förlorad eftersom datorn tror att textsträngen var slut. Beroende på vilken databashanterare som används är det lite skillnad i vilka tecken som har betydelse och hur problemet kan undkommas. I detta projekt kommer databashanteraren MySql (*MySQL* 2007) användas. Tecknen just denna hanterare har problem med är \0, \', \", \b, \n, \r, \t, \z, \\, \, \% samt _, enklaste sättet att lösa problemet i just denna hanterare är att lägga till ett \ framför alla de tecken som har en speciell betydelse. På så vis luras datorn till att läsa dessa tecken som om de varit helt vanliga tecken utan någon som helst speciell betydelse.

2.7 Klient-Server Applikation

Klient-Server applikationer är en speciell typ av applikation, eller egentligen två olika applikationer. Klienten och servern är konstruerade så att de kan kommunicera med varandra vanligtvis över ett nätverk men de kan i vissa fall även finnas på en samma dator. En klient är vanligtvis ett gränssnitt mot användaren exempelvis ett spel eller ett chatfönster. En server har oftast inget direkt användargränssnitt (men kan ha ett användargränssnitt för att administrera servern) utan sköter endast kommunikationen mellan de olika klienterna. Server applikationen kan även sköta vissa arbeten för att underlätta för klienten exempelvis rendera bilder eller utföra krävande beräkningar. Som Gunnarsson (1998) skriver ”Den ena datorn för befälet och begär att den andra datorn skall utföra en tjänst” det är då klienten som begär och servern som utför. En server kan ha flera klienter kopplade till den men det är ovanligt att en klient kopplad till flera servrar.



Figur 9. *Bilden ovan illustrerar en server i mitten och 6 stycken klienter som är kopplade till servern över någon typ av nätverk.*

Då en klient-server applikation fungerar just så att klienterna begär tjänster av servern lämpar sig en sådan lösning bra till att använda sig av tillsammans med en databas.

Med en server kopplad till en databas kan information sparas på ett väl strukturerat och säkert sätt. Då det bara är servern som är kopplad till databasen och klienterna ber servern om informationen de är intresserade av kan många problem undkommas. Sådana problem kan vara problem med att hålla databasen uppdaterad. Även fusk då databasen används av ett spel kan förhindras genom att klienterna inte får direkt tillgång till databasen.

3 Problem

Problemet som detta projekt går ut på är i grund och botten att eftersom en databas är en långsam applikation jämfört med ett datorspel undersöka om det över huvud taget är möjligt att, i realtid lagra datorspelsdata med hjälp av en databas.

Detta problem anses vara intressant eftersom databaser kan lagra stora mängder data på ett väl strukturerat sätt. Databaser är dessutom säkra vad gäller åtkomst till data som är lagrad i den, något som exempelvis kan förhindra fusk i ett datorspel.

För att skapa struktur i projektet samt på ett lämpligt sätt dela upp arbetet har problemet delats in i mindre delmål. En annan anledning till att problemet har valts att delas upp delmål är att dessa kan itereras för att nå en så optimal lösning som möjligt. De olika delmål som valts är 1: Utformning av en lösning till problemet, 2: Implementering och 3: Utvärdering. Dock kommer den största vikten av arbetet att ligga på delmål 1 och då främst på att optimera den databasdesign som kontinuerligt kommer att arbetas fram under delmålet. Målet med de tre delmålen är att kunna svara på om och i så fall hur väl den lösning som tagits fram under arbetet klarar av att lösa problemet.

3.1 Delmål 1: Utformning av en lösning till problemet

Då problemet ligger i att svara på om det är möjligt att använda en databas till att spara speldata, har det i detta delmål valts begränsa problemet till att spara enstaka tredimensionella objekt i en databas.

Problemet består i grunden av flera delar vilka kommer att behöva samarbeta, samt enskilt måste fungera för att helheten skall fungera. Därför har även detta delmål valts att delas upp enligt dessa delar vilka är Objekt, Databasdesign, Konvertering av data och Databas ←→Applikation.

Något som är viktigt att tänka på i detta delmål är att lösningen som utformas endast skall vara en teoretisk möjlighet, inte nödvändigtvis en fungerande lösning till problemet.

Objekt. Då applikationen skall komma att representera de objekt som finns sparade i databasen med en tredimensionell grafisk bild, måste något sätt att representera dessa objekt i datorns minne utformas.

Databasdesign. För att på ett så optimalt sätt som möjligt skapa de tabeller som utgör själva databasen bör en databasdesign utarbetas. Denna design bör optimeras så långt som möjligt då den är till stor del avgörande för hur snabbt databasen kommer att arbeta. Eftersom databasdesignen just är avgörande för hastigheten på databasen är detta steg ett av de viktigare i arbetet. Därför bör största delen av arbetet ligga på att göra databasdesignen så optimalt som möjligt.

Konvertering av data. Databasen och applikationen kommer att spara data om objekt på olika sätt. Data om dem kommer att utbytas kontinuerligt under drift av applikationen. Detta innebär att något sätt att växla mellan de olika sätten att spara data bör utvecklas.

Databas ←→ Applikation. Databasen och applikationen behöver kommunicera med varandra och på så vis utbyta data. Då databasen och applikationen inte talar samma språk måste något sätt att översätta mellan dem utvecklas.

3.2 Delmål 2:Implementering

Tanken med det här målet är att implementera den lösning som tagits fram under föregående delmål. Det är under detta delmål som bekymmer med lösningen visa sig vilket leder till att den förmodligen kommer att behöva omarbetas. Målet med implementeringen är att testa lösningen som framtagits genom det föregående steget. Dels för att undersöka huruvida lösningen faktiskt fungerar och om inte ta reda på varför. På detta sätt kan olika brister i lösningen hittas och elimineras, och på så vis successivt utveckla en fungerande lösning.

3.3 Delmål 3: Utvärdering

I grunden är tanken med utvärderingen att svara på, om lösningen som utarbetats under detta projekt över huvud taget klarar av att i realtid lagra speldata med hjälp av en databas. Visar det sig att så är fallet kommer olika tester att utformas för att testa hur bra lösningen fungerar, samt att undersöka vad som kan göras för att förbättra resultatet från dessa tester.

4 Metod

Det finns flera olika sätt att genomföra ett sådant här projekt och minst lika många sätt att hitta lösningar på de problem som uppkommer. Här kommer olika metoder och tillvägagångssätt att gås igenom. Även de val av metoder som gjorts under detta projekt kommer tas upp i detta kapitel.

För att nå de olika delmålen kan flertalet olika tekniker eller tillvägagångssätt användas. Beroende på vilken teknik eller vilket tillvägagångssätt som valts att använda sig av kan resultatet variera betydligt. Dessa val är därför kritiska då det gäller att utforma en lösning till ett problem av arkitektur.

4.1 Metod för Delmål 1: Utformning av en lösning till problemet

I problembeskrivningen specificeras att problemet består av ett antal olika delar, detta innebär att även lösningen kommer att beså av dessa delar. Att problemet är uppdelat i olika delar vilka måste lösas har inneburit att även löningen har blivit uppdelad i olika delar. Dock har liknande metoder kunnat användas för alla de olika delarna.

Litteraturanalys. Ett tillvägagångssätt som kan användas för att införskaffa sig information om hur problemet kan lösas är genom att studera litteratur. Den litteratur som studeras kan till exempel vara böcker i ämnet, vetenskapliga arbeten eller artiklar. Genom att studera olika tillvägagångssätt och snarlika problem kan en bild skapas över hur lösningen till det problem man själv står inför.

Intervjuer. En annan metod som kan användas för att införskaffa den information som krävs för att lösa problemet är att intervjuer. Det rör sig då om att intervjua personer som besitter kunskap inom området. Genom att ställa frågor till dessa personer kan även här en bild skapas.

Under detta projekt har en blandning av de två metoderna som beskrivs här ovan använts, vilket innebär att båda litterära studier samt att intervjuer har genomförts. För att på så vis införskaffa den information som behövts för att utforma löningen på problemet. Intervjuer som genomförts har varit diskussioner lärare på Högskolan i Skövde. Det skulle dock lika gärna ha kunna ha varit genom e-mail kontakt till någon professor inom området.

4.2 Metod för Delmål 2: Implementering

Implementering är ett sätt att testa den eller de lösningar som utformats i föregående delmål. För att göra detta finns flertalet olika tillvägagångssätt och tekniker som kan utnyttjas. Till att börja med beror det helt på vad det är som skall implementeras, i det här fallet är det ett datorprogram. Då det gäller att implementera just ett datorprogram finns det ett antal viktiga val, listan nedan tar upp de viktigaste.

- *Programmeringsspråk.* Det finns flera olika programmeringsspråk som kan användas för att implementera den lösning som framtagits till problemet i detta projekt. Istället för att här räkna upp alla de olika språk som finna att tillgå, konstateras det istället helt enkelt att *C++* (Stroustrup. B., 1999) kommer att användas. Detta val har gjorts med anledning av min personliga kunskap inom området.
- *API.* Då det kommer till val av API finns det i sammanhanget två kandidater, vilka är *OpenGL* och *Direct3D*. Dessa API används båda till grafisk rendering med den största skillnaden att OpenGL är plattforms oberoende

medan Direct3D är bunden till *Microsoft Windows*. Val av API har gjorts i detta fall på samma grunder som valet av programmeringsspråk. Det API som använts under detta projekt är OpenGL.

- *Designmönster*. Med designmönster menas i detta fall vilket programmerings sätt som lämpar sig alltså *Procedurell* eller *Objektorienterad programmering*. Eftersom lösningen är uppdelad i tydliga delar som kommer att samarbeta, är det lämpligaste valet i detta projekt att använda sig av *Objektorienterad programmering.*
- *Databashanterare*. Då det kommer till val av databashanterare finns det flertalet olika att välja ibland. Exempel på sådana är Oracle (2007), MySql (DuBois, P., 2000) och Microsoft SQL Server (2007). Efter litterära studier på dessa valdes MySql anledningen till detta val var även här den samma som valet av API och Programmeringsspråk.

Givetvis finns det flera val som är direkt avgörande för hur lyckad lösningen blir, men det är dessa som är de mest naturliga. Samt de är de första och mest kritiska valen då de lägger grunden för implementeringen.

4.3 Metod för Delmål 3: Utvärdering

Utvärderingen av arbetet med ett projekt av detta slak kan göras på flera sätt och kan visa många olika saker, men den kan även vara så enkel som att endast besvara frågan *Fungerar lösningen?*

För att testa en lösnings funktionalitet kan olika testfall utvecklas utifrån kriterier baserade i problemet. Dessa kan sedan appliceras på den färdiga lösningen för att testa hur väl den klarar de olika testen sett ur kriteriernas perspektiv.

Utvärderingen kan även gå ut på att hitta fel i lösningen genom olika tester. Det finns två vanliga typer av sådana test White Box och Black Box. Eftersom endast lösningens funktionalitet att lösa problemet och inte att fungera fel fritt är intressant i detta projekt har dessa tester inte använts för att testa den lösning som utvecklats. Därför är det inte heller intressant att här gå in på hur de fungerar.

Skulle lösningen som framtagits under detta projekt klara av att lösa problemet så som det är specificerat i kapitlet om just problemet kommer ett antal test att utvecklas. Dessa tester skall vara av den typ vilken endast testar hur väl lösningen presterar i olika fall.

4.4 Maskinvara

Applikationen använder sig av två stycken olika datorer, en dator vilken sköter om själva applikationen. Samt en dator vilken innehåller databasen, i detta fall en server cirka 40 mil bort från den dator som kör applikationen. Kopplingen mellan de två datorerna i detta fall är Internet.

Applikationsdatorn

- Operativsystem: Microsoft Windows XP Professional Service Pack 2
- CPU: Intel Pentium Dual Core (2.8 GHz)
- RAM: 2.0 GB
- Grafikkort: ATI Radeon X1600

Databas servern

- Operativsystem: Linux, Slackware 11
- Databashanterare: MySql version 5.0.24
- CPU: Intel Pentium 3 (800 MHz)
- RAM: 128 MB

5 Resultat

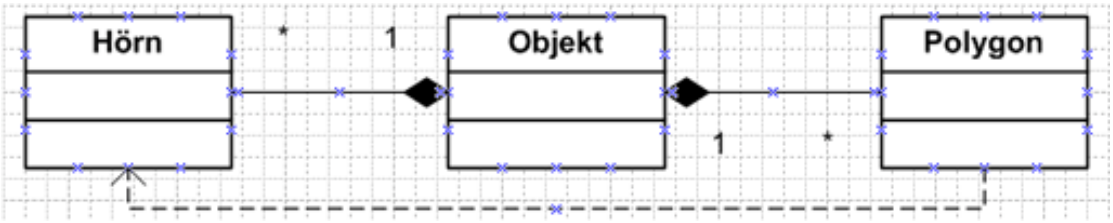
Detta kapitel beskriver arbetet med att utveckla, implementera samt testa en möjlig lösning till problemet med att använda sig av långsamma databaser för att lagra speldata i realtid.

5.1 Resultat för Delmål 1: Utformning av en lösning till problemet

Här beskrivs arbetet med att utforma en potentiell lösning till problemet utifrån specifikationerna i problembeskrivningen.

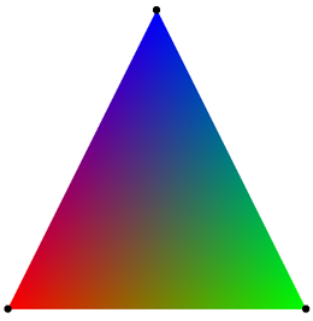
5.1.1 Objekt

För att bygga ett objekt i en tredimensionell värld behövs två saker, dels ett antal hörn (eng. vertexes). Samt något som beskriver hur de olika hörnen hänger ihop för att de skall kunna utgöra de polygoner (eng. faces) som själva objektet byggs upp av. Här nedan följer ett UML-diagram över hur detta problem är löst i den applikation som utvecklats under arbetet med denna rapport.



Figur 10. *UML-diagram över klasserna som bygger upp ett objekt.*

Som diagrammet ovan visar utgörs ett Objekt inte bara av klassen Objekt utan de två klasserna Polygon och Hörn krävas för att kunna skapa ett objekt. Diagrammet säger även att ett Objekt består av en eller flera Polygoner och ett eller flera Hörn, samt att klassen Polygon känner till Hörn klassen. Detta innebär att ett Objekt kan ha en eller flera Polygoner som det byggs upp av ett förutbestämt antal Hörn. I just detta fall har en polygon tre stycken hörn den utgörs av så att den bildar en triangel. Figuren nedan visar ett exempel på hur en polygon kan se ut. Triangeln representerar själva polygonen och de svarta prickarna representerar hörnen.

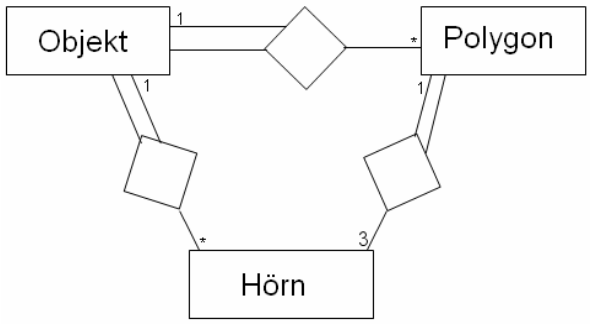


Figur 11. *En polygon som utgörs av tre stycken hörn.*

Hörnen finns specificerade enligt Hörn klassen och på så vis hör allt samman. Detta upplägg fungerar bra för applikationen som håller all information i datorns RAM-minne. För att lagra samma information i databasen är detta upplägg inte effektivt då detta hade inneburit att databasen inte kunnat optimeras. Istället används ett annorlunda upplägg i databasen vilket är optimerat för att den skall fungera så effektivt som möjligt.

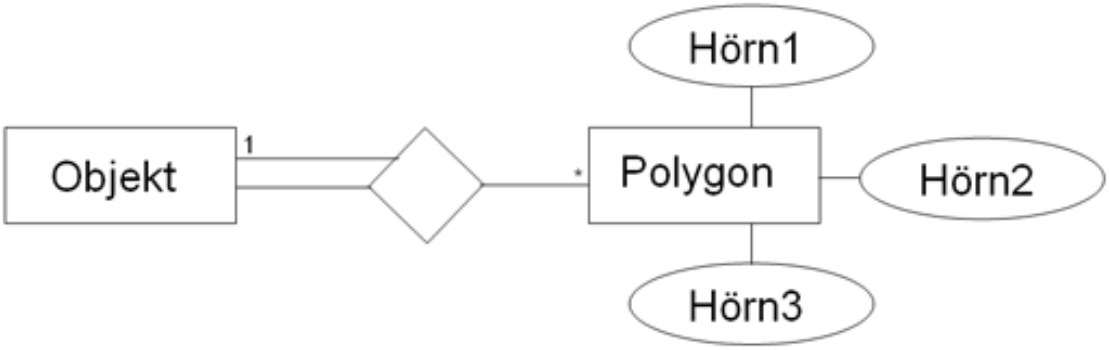
5.1.2 Databasdesignen

Då databasdesignen har stor inverkan på hur effektiv databasen blir är det viktigt i just ett sådant här projekt att denna är så optimal som möjligt. En databas lagrar data på en hårddisk och inte som applikationen som beskrevs i förra kapitlet vilken lagrar data i RAM-minnet. Data som är lagrad i RAM-minnet har datorn näst intill direkt åtkoms till, för att läsa data från en hårddisk krävs en viss tid. Detta innebär att lösningen som fungerade bra till applikationen, inte alls är bra att använda sig av då det gäller att utforma databasen. Då syftet med detta steg i arbetet är att utforma en tids effektiv databas. Skall databasen ens kunna komma i närheten av att klara av uppgiften måste denna lösning omarbetas samt optimeras. Figuren nedan är ett ER-diagram över hur databasen skulle ha sett ut om den använt sig av samma lösning som applikationen.



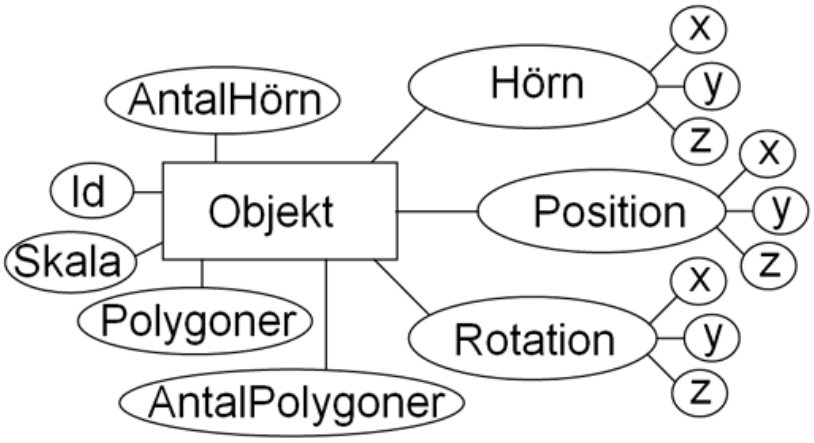
Figur 12. *O-optimerad databas design för ett objekt. Det finns inga attribut med i diagrammet då dessa är ointressanta i detta skede.*

Då varje entitet i ER-diagrammet resulterar i en tabell då implementeringen av databasen äger rum kommer denna lösning att ge upphov till tre tabeller. De tre tabellerna är Objekt, Polygon samt Hörn, varje objekt i databasen kommer att bestå av ett visst antal polygoner vilka består av 3 stycken hörn. Detta innebär både tabellen Hörn och tabellen polygon kommer att innehålla många rader med få kolumner något som inte är speciellt optimalt. Varje hörn som används för att skapa objektet kommer att representeras som en rad i tabellen Hörn detta medför att denna tabell kommer att få en rad för varje hörn. Eftersom optimeringen av en databas går ut på att undvika att ha många rader och istället ha flera kolumner är detta något som måste omarbetas för att optimera databasen. Även om en polygon består av tre hörn så kan samma hörn användas för att skapa flera polygoner, detta innebär att även tabellen Polygon kommer att innehålla många rader. Så alltså är även tabellen Polygon något som måste omarbetas för att optimera databasen.



Figur 13. *En första optimering av databasdesignen för ett objekt. Endast ett fåtal attribut är med i figuren då de andra är ointressanta i detta skede.*

Figuren ovan visar ett steg i optimeringsarbetet med databas designen. I detta skede har entiteten Hörn tagits bort och istället ersatts med 3 stycken attribut i entiteten Polygon. Men denna design blir det betydligt färre rader i tabellerna men även de två beroendena till Hörn entiteten har försvunnit. Detta är ett steg i rätt riktning men fortfarande kommer tabellen Polygon att innehålla många rader. Eftersom ett Objekt byggs upp av flera Polygoner så är i slutändan även denna design bristfällig och behöver omarbetas.



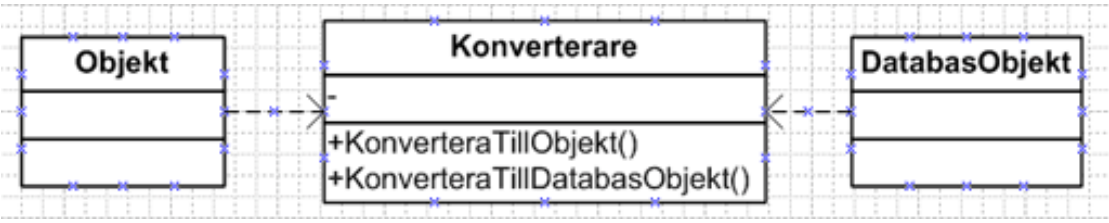
Figur 14. *Den slutgiltiga databasdesignen så optimerad som det går att göra den.*

Databasdesign i bilen ovan är den slutgiltiga designen som tagits fram under detta projekt. Med hänsyn av vad databasen kommer att användas till kan denna databasdesign optimeras ytterligare, dock är det en väldigt optimerad databasdesign. Med denna lösning kommer varje objekt att motsvaras av endast en rad i databasen, något som håller nere antalet rader i tabellen. Vilket är något som eftersträvas då arbete med att optimera databasdesignen fortlöper.

5.1.3 Konvertering av data

Eftersom databasen och applikationen lagrar data på olika sätt samt att de kommer att utbyta data mellan dem, så behöver något sätt att konvertera data på utvecklas. Lämpligast är att vänta tills både databasdesignen och applikationsdesignen är klara så att all information om hur de olika designerna lagrar data finns tillgänglig.

Då all informationen om de olika designerna är tillgänglig kan arbetet med att utveckla konverteraren påbörjas. Lösningen på detta problem vilken har använts under detta projekt beskrivs med hjälp av UML-diagrammet nedan.



Figur 15. *UML-Diagram för hur konverteringen mellan Objekt och DatabasObjekt hänger ihop.*

Det som föll sig naturligast här var att skapa klassen DatabasObjekt i applikationen. Denna klass lagrar data precis likadant som databasen fast i direkt i minnet på datorn. Det enda som behövs nu är något sätt att växla data mellan de två klasserna Objekt och DatabasObjekt. Vilket kan lösas genom att skapa funktioner som tar data från den ena klassen och steg för steg omvandlar den så som den andra klassen lagrar den.

5.1.4 Databas ←→ Applikation

För att alla de olika delarna av lösningen skall kunna samarbeta krävs en hel del jobb. Det största problemet är att datatypen BLOB lagrar data binärt. Det innebär att när nya objekt skall läggas till i databasen måste den data som skall lagras i en BLOB omvandlas. Eftersom stor del av den data som skall lagras består av koordinater på olika hörn innebär det att siffran 0 förekommer ofta. Detta innebär stora problem då 0 omvandlat till binära tecken innebär NULL vilket tolkas som att det är slutet av datasträngen. Även tecknen \z, \n, \r, \\ samt \ har betydelse då de läses vilket innebär att även dessa tecken måste göras något åt. En möjlighet för att undkomma detta problem är att lägga till ett \ tecken framför alla dessa tecken när de omvandlas till binärdata. Detta gör att de läses som de tecken de är och inte vad de har för betydelse.

5.2 Resultat för Delmål 2: Implementeringen

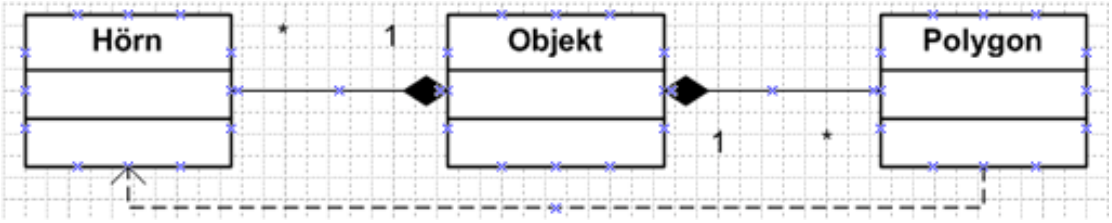
För att kunna testa lösningen som tagits fram under delmål1: Utformning av en lösning till problemet, är det lämpligt att implementera denna lösning. Detta är något som kan ske på flera olika sätt och med ett antal olika tillvägagångssätt. Val av dessa samt vad som finns tillgängligt beskrivs i kapitlet Metod.

5.2.1 Objekt

Bilden nedan visar det UML-diagrammet som tagits fram för att representera lösningen till att implementera ett Objekt vilken tagits fram i det förra delmålet.

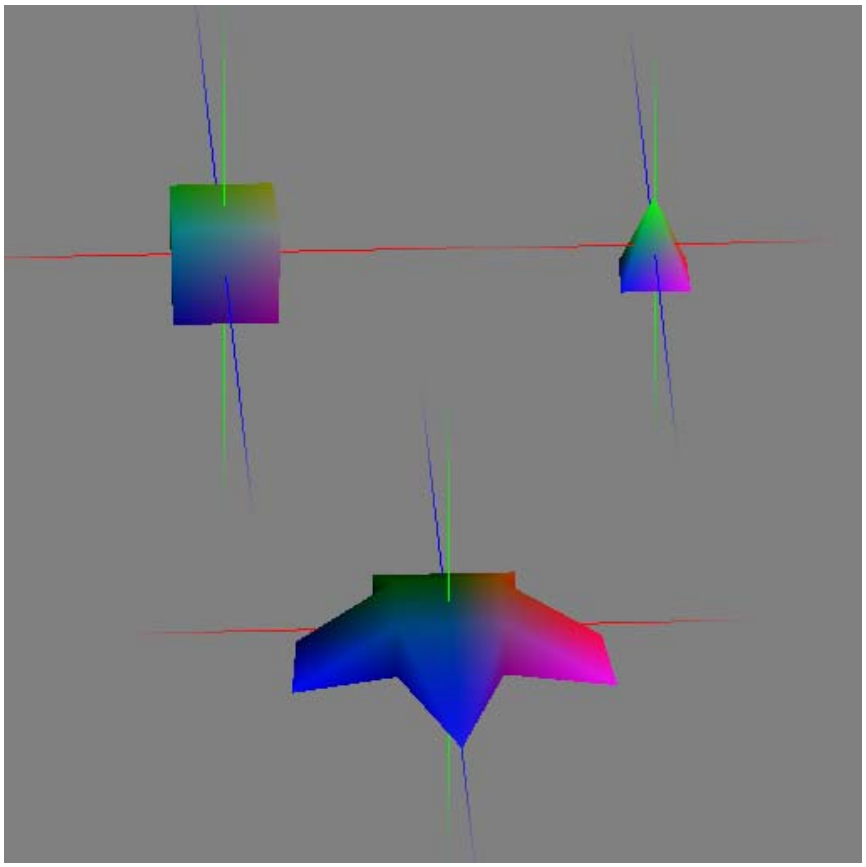
Denna lösning ger upphov till de tre klasserna Objekt, Polygon samt Hörn. Ett hörn som i grunden är en position i en tredimensionell rymd innehåller attributen X-koordinat, Y-koordinat samt Z- koordinat. Med hjälp av dessa tre koordinater positionen i rymden beskrivas då det finns en utgångspunkt (origo).

Klassen Polygon innehåller tre stycken viktiga attribut även den, dessa är de tre hörnen vilka utgör polygonens hörnpunkter och på så vis utgör polygonens kanter. Ett hörn i en Polygon utgörs av ett hörn i klassen Hörn. På så vis hänger de två klasserna Hörn och Polygon ihop. Det klassen Objekt gör är att den innehåller en lista på alla de hörn som finns tillgängliga för att bygga upp objektet. Klassen innehåller även en lista på de polygoner som utgör Objektet, och på så vis hänger alla de tre klasserna ihop.



Figur 16. *UML-diagram över hur de olika delarna av ett objekt fungerar och hänger ihop.*

Då implementeringen av alla tre klasserna var klar kunde olika former definieras enligt specifikationerna och läsas in i minnet. Då ett objekt är inläst i minnet kan det renderas med hjälp av en enkel renderingsfunktion. Bilden nedan visar tre stycken objekt som är uppbyggda så som UML-diagrammet ovan visar och renderade med hjälp av renderingsfunktionen.

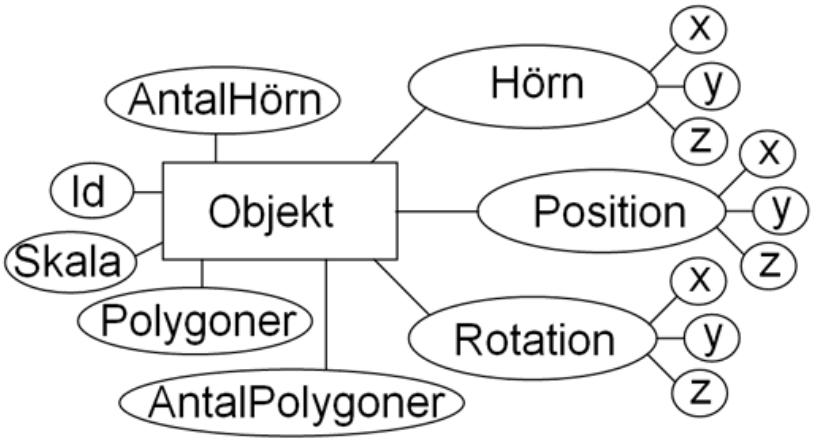


Figur 17. *Enkla former uppbyggda enligt Objekt Klassen och renderade med hjälp av OpenGl*

5.2.2 Databasdesign

För att databasdesignen skall kunna användas till att implementera databasen behöver den översättas till tabeller och relationer. Det är i detta skede som kopplingen mellan själva designen och den faktiska implementeringen klargörs.

Bilen nedan visar databasdesignen som utvecklats och använts under detta projekt.



Figur 18. *Den slutgiltiga databasdesignen som har itererats fram under det föregående delmålet.*

Just denna databasdesign består endast av en tabell och innehåller ingen relationen till någon annan tabell. Nedan visas hur koden som används för att implementera tabellen i databasen. Något som bör läggas märke till är att Id är satt som UNIQUE detta innebär att det endast kan finnas ett objekt som har ett specifikt id. Anledningen till

detta är att det är ett enkelt sätt att hålla isär de olika objekten och enkelt komma åt det specifika objektet man är intresserad av.

```
CREATE TABLE Objekt (
        Id INT UNIQUE,
        Skala DOUBLE,
        PositionX DOUBLE,
        PositionY DOUBLE,
        PositionZ DOUBLE,
        RotationX DOUBLE,
        RotationY DOUBLE,
        RotationZ DOUBLE,
        AntalHörn INT,
        HörnX BLOB,
        HörnY BLOB,
        HörnZ BLOB,
        AntalPolygoner INT,
        Polygoner BLOB);
```

Figur 19. *Query till databasen vilken används för att skapa tabellen Objekt.*

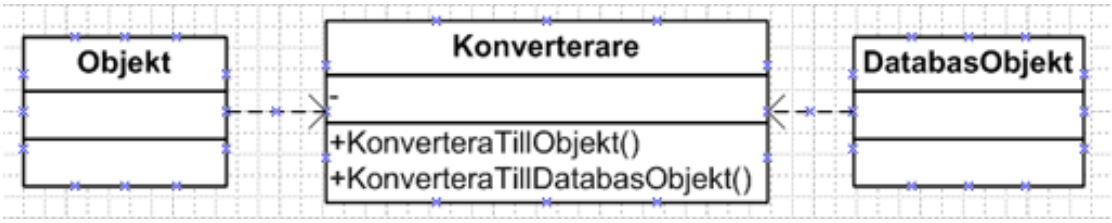
För att lättare få grepp om hur tabellen ser ut så följer här en grafiskrepresentation med beskrivningar för varje kolumn för hur tabellen kan se ut då den skulle ritas ut.

Id	Antal Hörn	Skala	Antal Polygoner	Polygoner	Hörn X	Hörn Y
Id nummer för att kunna komma åt det specifika objektet	Hur många hörn objektet innehåller	Vilken skala objektet har för tillfället	Hur många polygoner objektet innehåller	Binärdata innehållande alla objektets polygoner	Binärdata innehållande objektets alla x positioner på alla objektets hörn	Binärdata innehållande objektets alla y positioner på alla objektets hörn

Hörn Z	Xposition	Yposition	Xposition	Xrotation	Yrotation	Zrotation
Binärdata innehållande objektets alla z positioner på alla objektets hörn	Objektets position i världen	Objektets position i världen	Objektets position i världen	Objektets rotation i världen	Objektets rotation i världen	Objektets rotation i världen

Figur 20. *Bilden beskriver hur tabellen objekt kommer att se u och fungerat då den implementerats i databasen.*

5.2.3 Konvertering av data



Figur 21. *UML-Diagram för hur konverteringen mellan Objekt och DatabasObjekt hänger ihop.*

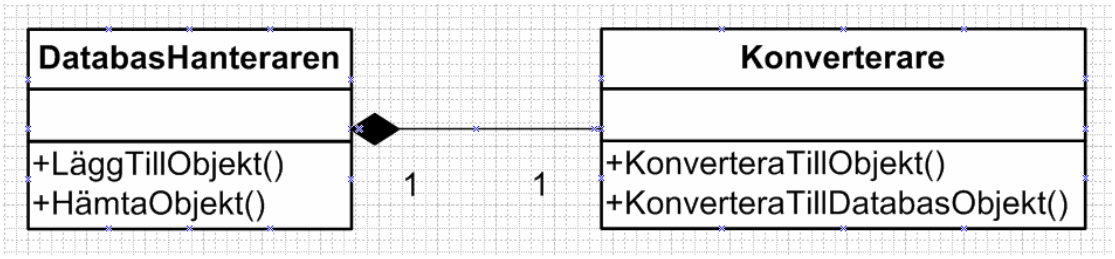
Som UML-diagrammet visar har Konverteraren två funktioner, en för att konvertera ett DatabasObjekt till ett Objekt, samt en funktion för att konvertera ett Objekt till ett DatabasObjekt. KonverteraTillObjekt() funktionen fungerar så att då den anropas kan en pekare till ett DatabasObjekt skickas med. Funktionen tar sedan all data från det DatabasObjekt som skickades med och omvandlar den så att den lagras så som data lagras i ett Objekt. Funktionen KonverteraTillDatabasObjekt fungerar på samma vis med undantaget att den utför omvandlingen av data från Objekts sätt att lagra den till DatabasObjekts sätt att lagra den. Med hjälp av dessa två funktioner kan enkelt växling mellan Objekt och DatabasObjekt ske. Med hjälp av detta underlättas interagerandet mellan applikationen och databasen.

5.2.4 Databas ←→ Applikation

Kopplingen mellan databasen och applikationen utgörs med hjälp av klassen DatabasHanteraren. Denna klass har främst två funktioner som är intressanta, en för att lägga till objekt i databasen (LäggTillObjekt) samt en för att hämta ut informationen om dessa objekt ur databasen (HämtaObjekt).

Med hjälp av modelleringsverktyget Maya (2007) skapas ett objekt vilket kan exporteras som Mayas eget objekt format (.obj). För att kunna använda detta format har en funktion utformats, vilken kan läsa en sådan fil och utifrån den skapa ett objekt så som det är specificerat i detta projekt. Då objektet finns lagrat i minnet kan funktionen LäggTillObjekt anropas, vilken i sin tur anropar konverteraren som omvandlar objektet till ett DatabasObjekt. Detta kan sedan enkelt lagras i databasen med hjälp av en fråga (eng. Query).

Men en fråga till databasen kan ett objekt hämtas ut ur databasen. Svaret som erhålls från databasen innehåller all data om objektet, så som DatabasObjekt är uppbyggt för att spara data. Detta innebär att den data som erhålls från databasen enkelt kan läggas in i ett DatabasObjekt. Då DatabasObjektet innehåller all data kan det med hjälp av konverteraren omvandlas till ett Objekt vilket kan användas av applikationen för att rendera en tredimensionell grafiskrepresentation av Objektet.



Figur 22. *Något förenklat UML-diagram över klasserna DatabasHanterare och Konverterare samt relationerna mellan dessa.*

Förutom detta används också en extern kommandotolk för att utföra enklare operationer på objekten i databasen. Detta kan vara att ändra positioner och rotationer eller att ändra skalan på objekten.

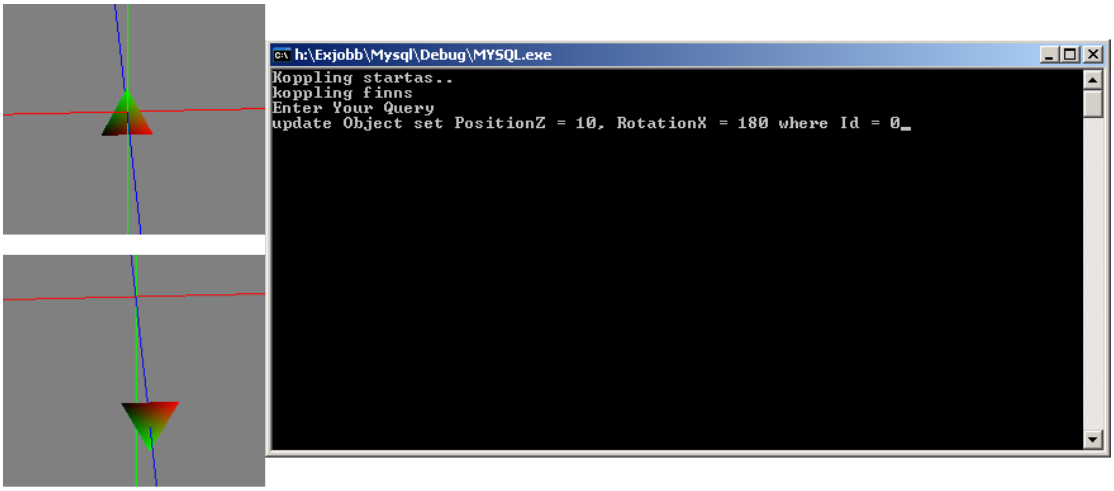
5.3 Resultat för Delmål 3: Utvärdering

Då lösningen som utarbetats och implementerats faktiskt klarar av att lösa problemet, är det intressant att ta reda på hur bra den klarar av uppgiften. Detta kan göras med hjälp av olika tester, vilka undersöker lösningens funktionalitet utifrån någon form av kriterier. Eftersom uppdateringshastigheten är kritisk i datorspel har det här valts att ha denna som mått för hur bra lösningen klarar de olika tester som framtagits.

5.3.1 Applikationen

Själva applikationen består av två delar. Den ena är en tredimensionell grafisk representation av de objekt som sparats i databasen och den andra delen av applikationen utgörs av en enkel kommandotolk. Kommandotolken används för att skriva frågor (eng. Querys) till databasen, på så vis är det möjligt att interagera med objekten. Interaktionen innebär att position samt rotationen på ett objekt kan ändras.

Bilden nedan visar ett exempel på hur applikationen ser ut, till vänster den grafiska representationen av ett tredimensionellt objekt som är lagrat i databasen. Till höger i bilden visad kommandotolken som används är att interagera med objekten i databasen.

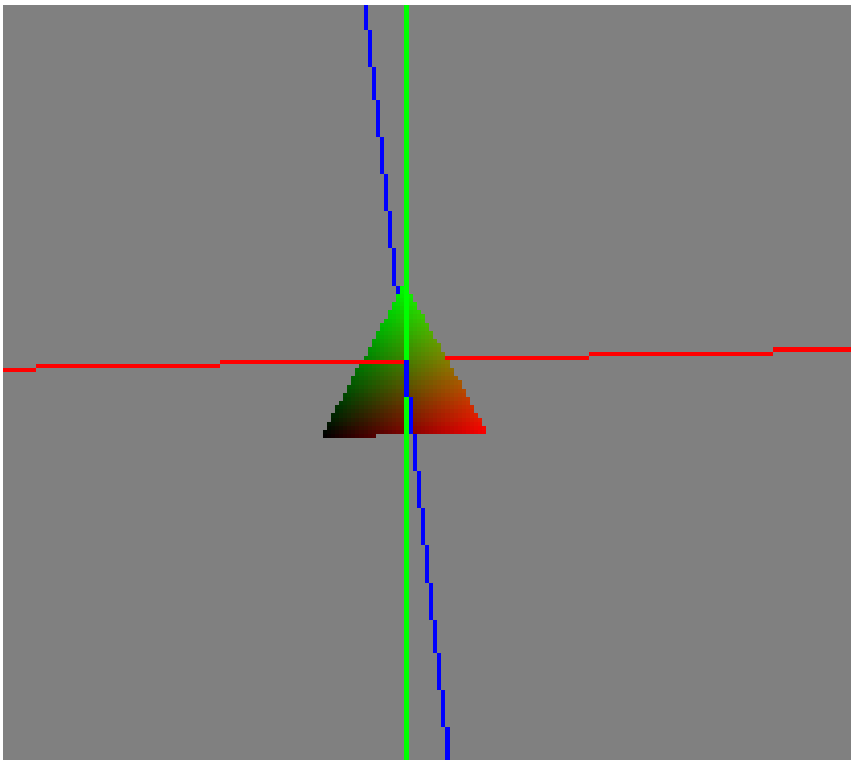


Figur 23. *Bilden visar ett exempel på hur det kan se ut då kommandotolken används för att ändra position samt rotation på ett objekt som är sparat i databasen.*

5.3.2 Testfall

Som beskrevs i tidigare kapitel så är det olika datorer som innehåller databasen och applikationen samt att de är sammankopplade över Internet. Detta är något som inverkar på resultatet eftersom det ofta förekommer olika störningar på Internet vilket medför att överföringshastigheten kan variera från fall till fall. Mått av latensen (eng. *Latency*, tiden det tar för ett paket att skickas från en dator till en annan dator och tillbaka igen) mellan de olika datorerna har mätts vid ett flertal tillfällen. Medelvärdet har inte varierat mycket mellan de olika tillfällena, och vid en längre test har medelvärdet uppmätts till 58 millisekunder.

Testfall 1, En polygon. Med endast en polygon i objektet samt att den hämtas varje uppdatering fås en uppdateringshastighet på 75 uppdateringar per sekund. Detta kan anses vara ointressant då det är väldigt få tredimensionella objekt som endast består av en polygon. Det visar dock att lösningen fungerar, men skall den användas i ett större projekt bör lösningen bearbetas ytterligare för att på så vis nå bättre resultat.

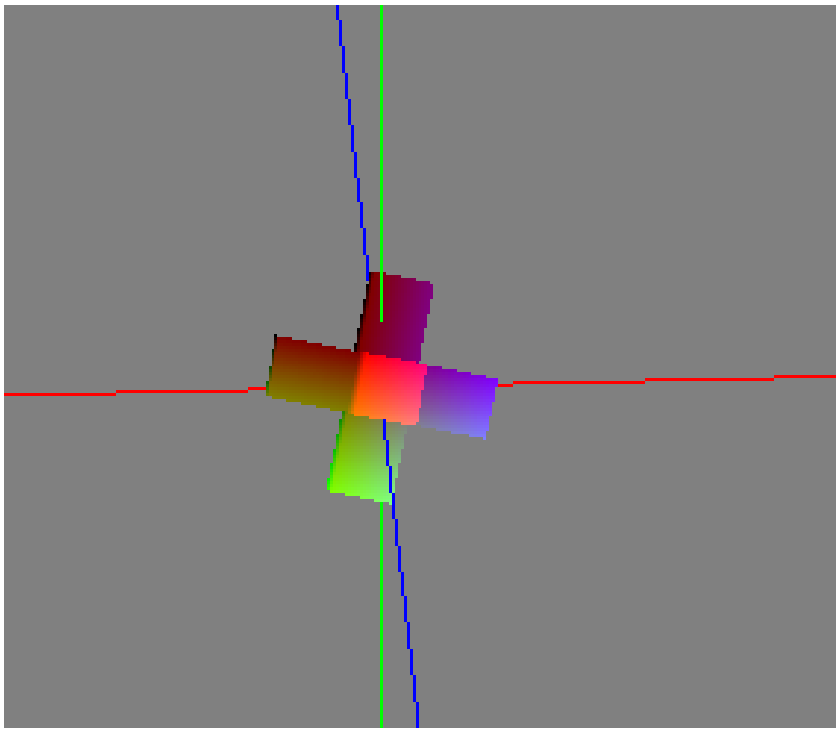


Figur 24. *En polygon som är lagrad i och hämtas ur databasen.*

Testfall 2, Flera polygoner. Ytterligare testfall genomfördes, tillexempel genom att lagra objekt bestående av flera polygoner samt att hämta ut samma objekt flera gånger under samma uppdatering.

Då en ensam polygon hämtades 10 gånger under samma uppdatering vilket kan jämföras med att hämta ut 10 objekt bestående av endast en polygon blev resultatet 10 uppdateringar per sekund. Detta innebär att skulle en sådan här applikation användas i ett spel och dess databas behöva innehålla mer än 10 enkla objekt så skulle spelet uppdateras 10 gånger i sekunden vilket får till följd att spelet skulle se ryckigt och hakigt ut.

Testfall 3, Objekt bestående av flertalet polygoner. Då tester med objekt bestående av flera polygoner genomfördes blev resultatet liknande alltså att uppdateringshastigheten sjönk mer och mer ju flera polygoner objektet bestod av. Exempelvis ett objekt bestående av 60 polygoner uppmättes en uppdateringshastighet på 45 uppdateringar per sekund.

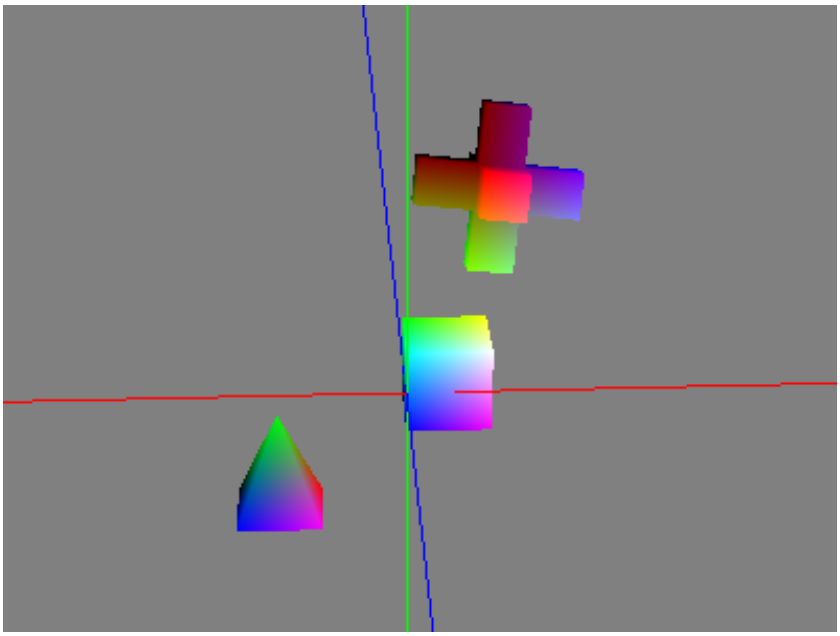


Figur 25. *Ett objekt bestående av flera polygoner hämtat ur databasen.*

Detta resultat indikerar på att såväl antal objekt som antal polygoner har stor inverkan på uppdateringshastigheten. Med tanke på att 60 är förhållandevis lite polygoner tillexempel en Town Hall från Warcraft III (2002) består av 800 polygoner innebär dessa tester att skall en databas användas i ett spel måste lösningen utökas med både trådning samt någon form av funktion för att ta reda på förändringar istället för att hämta informationen i varje uppdatering.

Figur 26. *Bilden visar en Town Hall hämtad ur datorspelet Warcraft III: Regin of Chaos.*

Testfall 4, Flera objekt. Det sista testfall som utvecklats är att hämta flera objekt bestående av olika antal polygoner ur databasen. Även då detta test utfördes mättes uppdateringshastigheten vilken indikerade på att ju mer data som hämtas vid ett och samma tillfälle har inverkan på uppdateringshastigheten. Resultatet som kan utläsas ur detta test är ju mer data som hämtas ur databasen desto långsammare blir uppdateringshastighet på applikationen.



Figur 27. *Flera objekt hämtade från databasen.*

Med tanke på att den enda information som sparas om hur objektet ser ut är dess hörn samt sidor är dessa resultat intressanta. Då mängden information som sparats i databasen har inverkan på hastigheten skulle den påverkas betydligt mer om all information om ett objekt skulle sparas ner. Figur 26 visar en bild på en Town Hall ur spelet Warcraft III, som det syns på den bilden krävs det en hel del mer information för att ett objekt skall kunna avvänds i ett spel. Exempelvis de texturkoordinater som används för att lägga på bilder på objektets sidor så att det ser ut att likna en borg. Detta är bara ett exempel på vad mer som behöver sparas för att ett spel skulle kunna använda sig av lösningen som tagits fram under detta projekt.

6 Slutsats

Detta projekt har gått ut på är att utforma och testa en lösning till att lagra speldata med hjälp av en databas samt att hämta ut den i realtid. Den lösning som utformats är långt ifrån optimal och besitter goda möjligheter att utvecklas mer. Dock var inte tanken detta projekt att hitta en optimal lösning, ej heller att utöka lösningen som framtagits till perfektion. Istället har detta projekt har gått ut på att utforma samt testa ett möjligt sätt att lösa problemet på. Den lösning som har framtagits under projektet innehåller flertalet brister, och för att denna lösning skulle kunna användas till något annat än att bevisa just att den löser problemet behöver den utökas. Dock har den visat att möjligheten att lösa detta problem kan göras genom en liknande applikation.

Under arbetets gång har problemet delats upp i tre mindre delmål, detta för att få en enkel struktur i arbetet. De delmål som valts är 1: *Utformning av en lösning till problemet*, 2: *Implementering* samt 3: *Utvärdering.* Delmål 1 har även valts att delas upp i mindre delar vilka enskilt måste fungera för att hela lösningen skall fungera. Dessa delar är *Objekt, Databasdesign, Konvertering av data och Applikation* ←→ *Databas.* För att avgöra om och hur bra de olika delarna fungerar har de använts i nästkommande delmål, det har varit en iterativ process. Vilket innebär att då något problem uppstått under implementering har arbetet gått tillbaka till att utforma lösningen för att undkomma detta problem. Då lösningen fungerade har valet gjorts att utforma och genomföra ett antal test för att avgöra hur bra lösningen fungerar. Dessa tester har testat själva nätverksdelen i databasen och visar att mycket arbete återstår om denna lösning skall kunna används av ett dataspel. Dock ligger detta arbete inte på själva databasen och har därför valts att lämnas till framtida arbete.

6.1 Diskussion

Resultaten från de tester som beskrivs i föregående kapitel visar att lösningen inte bara löser problemet utan även hur väl den löser det. Dessa tester påvisar något som befarats under hela projektet, att lösningen trots all optimering inte är tillräcklig för att kunna användas i ett spel. Då det inte var tanken med projektet att finna den perfekta lösningen utan endast undersöka möjligheterna har medvetet vissa arkitekturer valts att utelämnas, exempelvis trådning vilket kunnat förbättra lösningens resultat betydligt.

Eftersom grunden i projektet är att testa om det finns någon som hälst möjlighet att lagra speldata i en databas, har val att begränsat problemet gjorts, detta för till att lägga vikten på att testa just möjligheten. Skulle detta arbete kunna användas av ett spel behöver lösningen utökas betydligt. Exempelvis har endast den nödvändigaste informationen om de objekt som lagrats i databasen valts att ha med i lösningen. Exempel på sådan information är Texturkoordinater, Normaler (vertex eller polygon) och Grupper av polygoner.

Resultatet i detta projekt säger dock att det faktiskt är möjligt att använda en databas till att spara data i en spelapplikation. För ett mindre och enklare datorspel har inte detta resultat någon direkt inverkan. Däremot ett spel som just World Of Warcraft skulle kunna ha stor nytta av dessa resultat. Att det faktiskt går att använda en databas för att lagra speldata och hålla den uppdaterad i realtid innebär att all data om en spelvärld kan lagras och enkelt återskapas efter en systemkrasch. Det innebär också att många uppdateringar till spel lägga till direkt i databasen istället för att enskilda spelare blir tvungna att uppdatera sina datorspel.

6.2 Framtida arbete

Under arbetets gång har flertalet idéer uppkommit vilka är möjligheter till att öka löningens prestanda. Då dessa inte har varit direkt avgörande för att testa lösningens funktionalitet har valet att inte implementera dem gjorts.

Att uppdateringshastigheten på applikationen är direkt beroende på svarstiden från databasen är något som konstaterats under arbetets gång. Detta beror på att applikationen stannar upp och väntar tills svar från databasen mottagits efter att en fråga ställts. Detta problem ligger just i nätverksdelen av databasen och skulle kunna avhjälpas på olika sätt.

Att tråda applikationen är en lösning på detta problem. Då skulle applikationen istället för att bli blockerad till svar fås från databasen kunna köra vidare till dess att svar erhålls. En annan idé är att använda sig av någon funktion som istället tar reda på förändringar i databasen, så att istället för att applikationen konstant frågar databasen efter data som finns lagrad i den så skickas förändringen till applikationen då den skett.

Givetvis finns det mycket mer som kan göras för att förbättra lösningen och även som måste göras för att den skall kunna användas i ett datorspel. Dock är dessa två uppdateringar något som hade gjort stor inverkan på resultatet från de tester som utförts på lösningen i detta projekt.

Skulle arbetet med denna lösning fortlöpa för att kunna använda denna teknik i ett dataspel behövs mycket av arbetet göras om. Denna lösning har framtagits för att bevisa att det är möjligt att använda en databas till att lagra datorspelsdata. Då en databasdesign tas fram för ett enda ändamål är denna det första som måste omarbetas. Med databasdesignen följer mycket av de andra delarna i lösningen automatiskt då dessa är framtagna för just denna databasdesign.

Då det gället att integrera en databas med ett dataspel finns det väldigt många saker att ta hänsyn till.

Referenser

DuBois, P. (2000) *MySql*. Indianapolis, IN: New Riders

Elmasri, R. & Navathe, S.B. (2007) *Fundamentals Of Database Systems*. Pearson Higher Education

Gunnarsson, G. (1998) *TCP/IP-handboken*. Pagina AB

Maya (2007) *Wikipedia.* Tillgänglig på Internet: http://sv.wikipedia.org/wiki/Maya%28datorprogram%29 [Hämtad 2007.07.13]

Microsoft SQL Server (2000) *Wikipedia.* Tillgänglig på Internet: http://sv.wikipedia.org/wiki/Microsoft_SQL_Server [Hämtad 2007.05.22]

MMOG Active Subscription (2007) Tillgänglig på Internet: http://www.mmogchart.com [Hämtad 2007 03 14]

MySQL (2007) MySQL AB Tillgänglig på Internet: http://www.MySql.org [Hämtad 2007.04.24]

Nationalencyklopedin. (1990) Band 4. Bra Böcker AB

Oracle (2007) *Wikipedia.* Tillgänglig på Internet: http://sv.wikipedia.org/wiki/Oracle [Hämtad 2007.05.22]

Quax, P. Jehaes, T. Jorissen, P. Lamotte, W. (2003) *A Multi-User Framework Supporting Video-Based Avatars*. ACM Press

Söderström. P. (1997) *"Data Warehouse"-Datalager: Verksamhet, Metod, Teknik.* Dataföreningen i Sverige DA förlags AB

Stroustrup. B. (1999) *The C++ Programming Language.* Addison-Wesley Publishing Company

Warcraft III: Regin of Chaos (2002) [datorprogram] Blizzard Entertainment

World of Warcraft (2004) [datorprogram] Blizzard Entertainment, Vivendi Universal