HÖGSKOLAN SKÖVDE

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

Effektiv uppdatering av datorspel med hjälp av filskillnader

Björn Andersson

Effektiv uppdatering av datorspel med hjälp av filskillnader

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

2007-06-01

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: _____

Effektiv uppdatering av datorspel med hjälp av filskillnader

Björn Andersson

Sammanfattning

Detta examensarbete undersöker om och hur man med hjälp av filskillnader kan effektivisera uppdateringar av datorspel. Flertalet olika tester med deltaenkodning har utförts, detta på olika versioner av filer från kommersiella och aktuella spel. Bland annat har försöken med deltaenkodning jämförts med de officiella speluppdateringarna, och med vanlig komprimering av de aktuella filerna. Resultaten visar att användning av deltaenkodning på de filer som ändrats mellan spelens versioner kan vara mycket effektivare och ta mindre plats än de officiella speluppdateringarna.

Nyckelord: Deltaenkodning, Patch, Diff, Programuppdateringar, Speluppdateringar, Xdelta, Komprimering, Speldata.

Innehållsförteckning

1 Introduktion 1

2 Bakgrund 2

2.1 Speluppbyggnad 2

2.1.1 Spelmotorer 2

2.1.2 Data och filtyper i spel 3

2.2 Speluppdatering 3

2.3 Komprimering 3

2.3.1 Förstörande komprimering 3

2.3.2 Icke-förstörande komprimering 4

2.4 Deltaenkodning 4

2.4.1 Användningsområden 5

2.4.2 Diff/patch 5

2.4.3 Algoritmer, tekniker och bibliotek för deltaenkodning 7

3 Problembeskrivning 9

3.1 Delmål 9

4 Metod 11

4.1 Metod för delmål 1: Identifikation och klassifikation av relevanta filtyper 11

4.2 Metod för delmål 2: Inhämtning av data från spel 11

4.3 Metod för delmål 3: Val av deltaenkodningsteknik 11

4.4 Metod för delmål 4: Implementation av testsystem 12

4.5 Metod för delmål 5: Utvärdering och analys av tester 12

4.5.1 Testutrustning 12

5 Resultat 13

5.1 Resultat för delmål 1: Identifikation och klassifikation av relevanta filtyper 13

5.1.1 Bilder och texturer 13

5.1.2 Modeller, geometri, animation och andra filer för 3d-data 13

5.1.3 Ljud och musik 13

5.1.4 Script- och konfigurationsfiler 14

5.1.5 Programfiler 14

5.1.6 Videofiler 14

5.1.7 Filarkiv 14

5.2 Resultat för delmål 2: Inhämtning av data från spel 14

5.3 Resultat för delmål 3: Val av deltaenkodningsteknik 15
5.4 Resultat för delmål 4: Implementation av testsystem 17
5.5 Resultat för delmål 5: Utvärdering och analys av tester 17
5.5.1 Urval av filer 17
5.5.2 Uppdateringarnas omfattning 18
5.5.3 Deltaenkodningens komprimeringseffektivitet 19
5.5.4 Tidsåtgång för komprimering och dekomprimering med hjälp av deltaenkodning 22
5.5.5 Undersökning av de officiella speluppdateringarnas storlek 23
6 Slutsats 26
6.1 Diskussion kring resultatet 26
6.2 Framtida arbete 26
Referenser 27

1 Introduktion

Dagens dator- och tv-spel blir alltmer avancerade. På 80-talet var det något som en eller ett par personer kunde utveckla på relativt kort tid utan att ens behöva ha någon budget. Nu – ett par decennier senare – så är det inte ovanligt med arbetslag på hundratals personer, budgetar på många miljoner dollar, och utvecklingstider på flera år.

I och med att spelen blir alltmer komplexa har mer och mer fel kunnat smyga sig in efter det att spelen släpps i butiker. Detta har skapat ett behov av att ”lappa ihop” (eng. patch) eller uppdatera spelen, vilket vanligtvis görs genom att det läggs ut nedladdningsbara uppdateringsfiler på tillverkarens hemsida.

För spelare som spelar PC-spel har detta varit mer eller mindre vardag de senaste åren. Konsoller och deras spel har varit relativt skonade från uppdateringar då dessa spel oftare haft en högre kvalitet från början, men eftersom de senaste generationerna av konsoller har stöd för Internet-anslutning har även uppdateringar börjat dyka upp så smått där.

Speluppdateringar innebär ofta en stor underhållskostnad för den stora mängd bandbredd som krävs (Shaikh, Sahu, Rosu, Shea & Saha 2006). Utöver behovet av att minska underhållskostnaden finns även behovet av att reducera tiden speluppdateringarna tar att överföra. Båda dessa behov kan genomföras genom att minska storleken på den data som speluppdateringarna består av.

För onlinespel ökar behovet av mindre speluppdateringar i och med att nedladdningarna av dem koncentreras till då nya versioner släpps, eftersom spelarna oftast inte kan spela när de har gamla versioner (Chambers, C., Wu-chang, F., 2005).

Givetvis kan man använda sig av vanliga komprimeringsalgoritmer för att minska mängden data till speluppdateringarna och deras överföringar, men varför skicka med sådan data man redan har, när man behöver uppdatera? Även om många filer blir ändrade i samband med en uppdatering så behöver det inte vara hela filen som blir ändrad, snarare tvärsom.

Deltaenkodning är en teknik som sedan 1970-talet (Wagner & Fisher, 1973) utvecklats för att i datahantering beskriva skillnaden mellan två filer eller datamängder. Genom att använda sig av en deltaenkodningsalgoritm kan man enbart skicka över den del av datan som ska ändras. Detta kan spara enormt med plats, vilket leder till att man sparar in på både bandbreddsbehov och tid.

Det här arbetet skall undersöka deltaenkodning i syfte att minska storleken på de data- och programuppdateringar som används till spel. Genom att använda sig av redan existerande deltaenkodningsverktyg i ett egetkomponerat testsystem kan data i olika versioner från relevanta och aktuella spel generera statistik som kan ge svar på huruvida deltaenkodning är värt att använda för detta syfte.

Kapitel 2 beskriver bakgrunden kring speluppdateringar och deltaenkodning. Kapitel 3 ställer upp problembeskrivningen för rapporten, och kapitel 4 beskriver metoden för lösningen av problemet. I kapitel 5 beskrivs själva resultatet, och i kapitel 6 sammanfattas lärdomarna från resultatet och funderingar på ett eventuellt framtida arbete inom området beskrivs.

2 Bakgrund

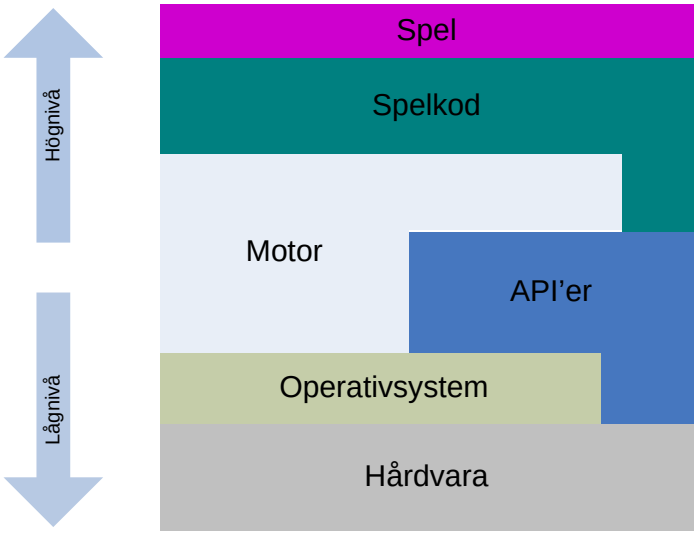
Det här kapitlet tar upp relevant information och bakomliggande tekniker kring områdena för deltakomprimering och speluppdateringar.

2.1 Speluppbyggnad

Ett datorspel är – till skillnad från vad många tror – en komplex sammansättning av grafik, ljud och en mängd programinstruktioner.

2.1.1 Spelmotorer

En spelmotor är ett allmänt begrepp för att beskriva en eller flera mjukvarukomponenter som utgör själva kärnan för spel. Exempelvis så finns det i spelmotorer ofta stöd för sådana grundliga saker som att dekoda ljudfiler eller rita ut bilder (Lewis & Jacobson, 2002). Det är inte ovanligt att spelmotorerna används till mer än ett spel (Herwig & Paar, 2002).



Figur 1: Ett exempel på hur en spelmotor kan vara relaterad till andra programmeringskomponenter i ett system.

Det finns en mängd utvecklade motorer – både kommersiella och icke-kommersiella. Bland de mer kända kommersiella spelmotorerna finns bland annat (Wikipedia, 2007):

- Doom 3 Engine – Är en vidareutveckling av id Softwares Quake 3-motor. Kända spel innefattar bland annat: Prey, Quake 4, och Enemy Territory: Quake Wars.
- Gamebryo – Är känd för sin flexibilitet, och har använts av en mängd olika typer av spel, däribland Civilization IV, Empire Earth III, Entropia Universe och The Elder Scrolls IV: Oblivion (Emergent game technologies, 2007).
- Source – Dark Messiah of Might and Magic, Half Life 2 (Valve, 2007).
- Unreal Engine – Är nu uppe i sin tredje upplaga, som bland annat har använts till kända spel såsom Gears of War, Halo Wars och Unreal Tournament 3 (Epic Games, 2007).

Flera av de ovan nämnda spelen och spelmotorerna är s.k. multi-platform, dvs. de finns till flera spelplattformar (t.ex. Windows, Playstation 3, Nintendo Wii).

2.1.2 Data och filtyper i spel

I spel förekommer en stor mängd med olika typer av filer, som har till uppgift att hålla spelets all data och kod. Exempel på data som ett spel lagrar i sina filer är 3d-geometri, animationer, skriptkod, bilder och ljud. Ofta tar spelutvecklarna och spelmotortillverkarna fram egna varianter och specifikationer av dessa filer, men i en del fall har vissa filer blivit till mer eller mindre en standard. Detta gäller framförallt ljud och bilder, som används flitigt även utanför spelindustrin.

2.2 Speluppdatering

Speluppdateringar finns i en mängd olika varianter, men det vanligaste är ett exekverbart program som helt enkelt uppdaterar spelets filer till nya versioner. Vissa typer av speluppdateringar – inkrementella speluppdateringar – är begränsade till att bara uppdatera mellan vissa specifika versioner av sitt spel, vilket kräver att man laddar hem rätt utgåva av uppdateringen, men även kan minska uppdateringens storlek signifikant.

Speluppdateringar består inte bara av uppdateringar av spelets kod, utan även till stor del uppdateringar av övrig data – såsom bilder, 3d-geometri, ljud, och filmer. Den här typen av data tar på grund av dess natur mycket mer plats. Kombinerar man det med en större mängd olika uppdateringar, samt med viljan från spelarna att uppdatera direkt en ny uppdatering har kommit ut, så blir det mycket stora krav på överföringskapaciteten för de servrar och tjänster som gör speluppdateringarna tillgängliga.

Shaikh, et. al. (2006) påvisar att det finns stora bandbreddskostnader associerade med speluppdateringar, och således är det av stor vikt att datan i uppdateringarna minskas. En metod för att göra detta är att använda sig av komprimering på den data speluppdateringen består av.

2.3 Komprimering

Med komprimering innebär att en mängd data formas om så att utrymmet datan tar minskar. Detta är användbart dels när man minska det lagringsutrymmet datan tar, och dels när man minska tidsåtgången som krävs när man överför datan på en begränsad bandbredd.

2.3.1 Förstörande komprimering

Förstörande komprimering (eng. lossy compression) är en viss typ av komprimering som ”förvränger” originaldatan för att komprimeringsgraden ska kunna bli högre. Detta används framförallt i mediafiler såsom bildfiler (jpg), videofiler (mpg, wmv) och ljudfiler (mp3, ogg). Dessa filtyper passar bra till den här typen av komprimering eftersom man med fördel kan ta bort eller simplifiera sådant som människans uppfattningsförmåga ändå inte lägger märke till (se Figur 2 respektive Figur 3).

Figur 2: En bild i originalformat. I bitmappsformat tar den 38 kilobyte.

Figur 3: Samma bild som tidigare fast hårt komprimerad med den förstörande komprimeringsalgoritmen JPEG. Även om kvalitén är märkbart sämre är det inga problem att utgöra vad bilden föreställer. I detta format tar den 27 kilobyte, vilket är 11 kilobyte mindre än den föregående.

2.3.2 Icke-förstörande komprimering

Icke-förstörande komprimering (eng. lossless compression) är motsatsen till förstörande komprimering. Filer som använder denna typ av komprimering är betydligt större än motsvarande filer som använder sig av förstörande komprimering, eftersom inga data får kastas bort.

Kortfattat kan man säga att icke-förstörande komprimering används för sådant som skall tolkas av datorn, såsom programfiler eller dynamiska programbibliotek. Detta till skillnad från exempelvis bilder, som skall tolkas av det mänskliga ögat.

2.4 Deltaenkodning

Med *delta encoding* eller *diff* menas att man sparar skillnaden mellan två filer. Detta är användbart när man har två olika versioner av samma fil, där den egentliga skillnaden är minimal. Kopplingen mellan deltaenkodning och komprimering är rätt självklar från ett användarperspektiv, då deltaenkodning kan ses som en slags komprimering. Därutav kallas deltaenkodning även ibland *deltakompression*.

Om det finns en textsträng i två olika versioner:

```
"Statsministern anlände sent på måndagskvällen i Stockholm, samtidigt som den stora fotbollsmatchen avgjordes."
```

```
"Statsministern anlände sent på tisdagskvällen i Stockholm, samtidigt som den stora fotbollsmatchen avgjordes."
```

Skillnaden mellan dem är minimal - "måndagskvällen" har bytts ut mot "tisdagskvällen" i den andra versionen av textsträngen. Istället för att lagra hela den andra versionen kan man istället bara lagra beskrivningen av skillnaden mellan dem. I

exemplet med de två textsträngarna ovan blir beskrivningen på skillnaden från den första till den andra kunna bli:

”Från tecken 31 till och med 33 skall texten bytas ut mot ’tis’”.

Lagrar man detta som mycket mer kompakta instruktioner skulle det exempelvis kunna bli:

```
C31-33:”tis”
```

Den ovanstående raden är märkbart mindre än att skriva ut hela den andra versionen, och det är självklart även mer lättöverskådligt med att få skillnaderna beskrivna.

2.4.1 Användningsområden

De flesta program som använder sig av deltaenkodning gör det för att minska mängden data som skall lagras eller skickas (Suel & Memon, 2003).

Utvecklingen av deltaenkodning har starka kopplingar till versionshantering (Suel & Memon, 2003), som sedan länge utnyttjat tekniken och helt klart är ett utav de mer vanliga användningsområdena. Med begreppet versionshantering avses att man kontrollerar multipla versioner av samma fil eller datamängd, så att man kan återskapa tidigare versioner av dem, eller spåra ändringar mellan dem.

När flera versioner av filer ska lagras är det oftast bara små skillnader mellan dem, vilket gör användandet av deltaenkodning ypperligt. Exempel på vanliga versionshanteringssystem är Concurrent versions system (CVS), Revision control system (RCS) och Subversion (SVN).

Så kallade diff-verktyg är vanliga i kombination med versionshanteringssystem, där de används för att jämföra olika versioner av samma fil. Dessa verktyg underlättar lösningen av eventuella konflikter som uppstår när man försöker ersätta olika versioner av filer med varandra. Det finns diff-verktyg till en stor mängd av filtyper, men vanligast är ändå att de är gjorde för att jämföra vanliga textdokument. Detta görs oftast med de två versionerna sida vid sida, och ofta med färgkodade rader. Diff-verktygen brukar oftast följa med versionshanteringssystemen.

Ett annat vanligt användningsområde för deltaenkodning är för att minska ner storleken på programuppdateringar som ges ut för mjukvara. Om en användare redan har en gammal version av en mjukvara denne vill uppdatera, då är det onödigt att i uppdateringspatchen skicka med sådan data som redan finns i den gamla versionen. Genom att använda sig deltaenkodning så kan uppdateringar oftast bli små i relation till hela programmet.

Deltaenkodning kan även med fördel användas i lagrings- eller överföringskritiska system. Det förekommer att man hittar den här användningen av deltaenkodning i exempelvis proxy-servrar för http, som i samarbete med http-servern minskar överföringskraven på densamma.

MacDonald (2000) presenterar filsystemet XDFS (XDelta File System) som är ett filsystem baserat på tekniker i Xdelta-systemet. XDFS liknar till stor del ett versionshanteringssystem i sin design och med sitt stöd för liknande operationer, men är ändock inte ett sådant.

2.4.2 Diff/patch

Diff respektive patch är två kommandoradsverktyg som har sin bakgrund inom UNIX-världen. Verktyget diff används för att skapa en fil som beskriver skillnaden mellan

två andra filer. Verktyget patch används för att modifiera en av filerna till den andra med den av diff skapade skillnadsfilen.

Diff skiljer sig från andra snarlika verktyg då den i standardutförandet jämför rad-för-rad istället för tecken-för-tecken (Hunt, Vo & Tichy, 1996). Detta är mycket mer tidseffektivt då den slipper jämföra alla möjliga kombinationer av tecken.

Nedan följer exempel på en fil i två olika versioner, och därefter resultatet som kommandot *diff* genererar.

Version 1 av test.h:

```
#ifndef __FILE__DATASTREAM_H
#define __FILE__DATASTREAM_H

namespace rf
{
        namespace fs
        {
                class File;

                class DataStream
                {
                public:
                        DataStream();
                        ~DataStream();

                        void Open(File& file);
                        void Close();
                        void Read(char* data, unsigned long size);
                        void Seek(unsigned long position);

                        unsigned long Tell() const;
                        unsigned long Size() const;
                        bool IsOpen() const;

                protected:
                        unsigned long m_position;
                        File* m_file;
                };

        }
}

#endif
```

Version 2 av test.h:

```
#ifndef __DATASTREAM_H
#define __DATASTREAM_H

namespace rf
{
        namespace fs
        {
                class File;

                //!  \brief A fstream-like class used to read from the filesystem.
                class DataStream
                {
                public:
                        DataStream();
                        ~DataStream();

                        //! \brief Opens a datastream.
                        //! \param file A File reference.
                        //! \return A character pointer.
                        void Open(File& file);
                        void Close();
                        void Read(char* data, unsigned long size);
                        void Seek(unsigned long position);

                        //! \brief Returns the current position in this stream.
                        //! \return The current position.
                        unsigned long Tell() const;
                        unsigned long Size() const;
                        bool IsOpen() const;

                protected:
                        unsigned long m_position;
                        File* m_file;

                };

        }
```

```
}

#endif
```

Resultatet genererat av kommandot *diff*:

```
1,2c1,2
< #ifndef __FILE__DATASTREAM_H
< #define __FILE__DATASTREAM_H
---
> #ifndef __DATASTREAM_H
> #define __DATASTREAM_H
9a10
>               //!  \brief A fstream-like class used to read from the filesystem.
15a17,19
>                       //! \brief Opens a datastream.
>                       //! \param file A File reference.
>                       //! \return A character pointer.
20a25,26
>                       //! \brief Returns the current position in this stream.
>                       //! \return The current position.
27a34
>
```

*Diff*s resultatformat men kan skilja sig mycket beroende på version och inställningar, men vanligtvis blir resultatet i formatet enligt ovan. Första raden beskriver en ändring i filen, som i detta fall beskriver att rad 1 till rad 2 i originalversionen av filen har ändrats till samma rader i den nya versionen av filen. Vi kan se att det är just en förändring som beskrivs eftersom bokstaven c är där (*changed*). På raderna efter beskrivs först hur det stycket såg ut i originalform, och efter skiljeraden (”---”) beskrivs hur stycket ser ut i den nya versionen av filen.

På samma sätt fortsätter det. Istället för förändring (c som i *changed*) som första delen beskrev, så kan även tillägg finnas med (a som i *added*) samt borttagningar (d som i *deleted*).

2.4.3 Algoritmer, tekniker och bibliotek för deltaenkodning

Även om det på senare år har dykt upp fler och fler typer av deltaenkodningsmetoder finns det fortfarande inte ett särskilt stort urval. I Trendafilov, Memon och Suel (2002) nämns bland annat Zdelta, Vdelta och Xdelta som tre vanliga verktyg för deltaenkodning. Alla tre använder olika metoder för att ta fram skillnaderna mellan filerna och för att hitta den optimala kombinationen av kopieringsinstruktioner. Dessa algoritmer är effektiva för att komprimera data, även gentemot vanliga komprimeringsalgoritmer och komprimeringsprogram, såsom gzip. Utöver dessa finns andra bibliotek, tekniker och verktyg, såsom bsdiff till UNIX, bdiff (Hunt, et al., 1996) eller det kommersiella och mer uppdateringsinriktade RTPatch.

Zdelta och Xdelta är bland de mer intressanta verktygen, eftersom båda har en längre historia av utveckling, använder sig av öppen källkod och är lättanvända – eftersom de finns tillgängliga som bibliotek för flera plattformar.

Zdelta är ett delta-komprimeringsbibliotek byggt ovanpå komprimeringsbiblioteket zlib (http://www.zlib.net). Biblioteket använder sig av en variation av LZ77 (Lempel-Ziv 1977), som är en icke-förstörande datakomprimeringsalgoritm.

Xdelta är ett verktyg som har funnits i flera olika generationer, nu senast har den kommit ut i tredje upplagan (version 3.0q). Xdelta använder sig av två separata licenser; GPL respektive en egen kommersiell licens.

Xdelta använder sig av VCDIFF som är ett standardiserat dataformat för deltaenkodning och komprimering (Korn, Macdonald, Mogul & Vo, 2002). Detta gör att de skillnadsfiler Xdelta genererar blir kompatibla med andra deltaenkodningsverktyg för binärdata.

Tekniken bakom dessa verktyg och bibliotek fungerar i teorin på liknande sätt som för kommandona diff och patch (se ovan), men istället för att hantera textfiler så finns i de flesta biblioteken fullt stöd för binära filer.

3 Problembeskrivning

Det här arbetet inriktar sig på hur man kan effektivisera uppdateringar till spel och snarlika applikationer genom att minska på mängden data som uppdateringen är beroende av, med hjälp av filskillnader. Problemet är intressant just för att uppdateringar till spel blir allt vanligare samtidigt som spelets datafiler blir allt större.

Genom att undersöka filtyper, speldata samt deltaenkodningstekniker kan vi få fram resultat över huruvida deltaenkodning duger för detta ändamål, och vilka metoder som ska användas för att få resultaten så pass effektiva som möjligt.

Arbetsprocessen delas in i fem delmål, som är tänkta att utföras i turordning, även om en viss överlappning kan komma att ske. Dessa är: Identifikation och klassifikation av relevanta filtyper, inhämtning av data från spel, val av deltakodningsteknik, implementation av testsystem, samt utvärdering och analys av tester.

3.1 Delmål

- **Delmål 1: Identifikation och klassifikation av relevanta filtyper**

  Ett vanligt spel bygger vanligtvis på en mängd olika filer som innehåller allt spelet kan behöva: 3d-modeller, ljud, bilder, banor och inte minst själva programmet. Att känna till filernas egenskaper, deras uppbyggnad och hur de strukturellt formar om sig vid modifikation är en nödvändighet om man måste få ned deras storlek.

  Här är det även viktigt att ha kännedom kring arkiv som ofta används i spel, eftersom dess filer ofta är av olika format och således kräver olika metoder vid uppdatering.

  Klassificeringen av filtyperna kommer dels att göras efter deras ändamål, men även efter hur de förändras rent datamässigt i olika versioner.

- **Delmål 2: Inhämtning av data från spel.**

  För att undersöka dessa tekniker vi är ute efter krävs realistisk data från spel. Denna data ska bestå av flertalet olika typer av filer, som i sin tur ska finnas i minst två olika versioner.

- **Delmål 3: Val av deltaenkodningsteknik**

  Även om idén bakom delta-enkodning alltid framställs som densamma, så finns det många sätt att implementera och lösa den på. Detta steg skall utvärdera de tillgängliga algoritmer och bibliotek för deltaenkodning för att sedan välja det mest lämpliga för vårat ändamål. Komprimeringseffektivitet samt användbarhet är de två främsta kriterierna när deltaenkodningstekniken kommer att väljas.

- **Delmål 4: Implementation av testsystem**

  För att gå vidare med problemet krävs en djupare testning av de olika filtyperna och teknikerna som finns tillgängliga. För detta behöver ett testsystem implementeras som med hjälp av den valda deltaenkodningstekniken jämför och testar den inhämtade speldatan, för att sedan generera statistik över effektivitet.

- **Delmål 5: Utvärdering och analys av tester**

Resultaten som testsystemet ger skall nu undersökas. Hur pass effektivt är det att använda deltaenkodning på de olika filerna? Vad för typ av data tjänar mest på användningen av denna teknik, och är tekniken ens värd att använda?

4 Metod

För att lösa problemet delas det upp i delmålen som presenterades i problembeskrivningen, och valet av metoder kommer att skilja sig mellan varje delmål. Metoderna är specifika för respektive delmål, eftersom problemen kring de olika delmålen skiljer sig markant från varandra.

4.1 Metod för delmål 1: Identifikation och klassifikation av relevanta filtyper

Metoden för det här delmålet är främst olika varianter av analyser. Litteraturanalys framstår som en självklar kandidat här, då främst av artiklar, men även av standardiserade fildefinitioner samt av programtekniska beskrivningar, som kan ge en inblick i hur filtyperna används och hur troligt det är att de ändras.

Att använda sig av implementation som metod för detta delmål vore möjligt; då i syfte att hitta mönster i filtyper och mellan olika versioner av dem. Detta kräver dock omfattande kunskap kring respektive filtyps uppbyggnad och således är implementation inte en lämplig metod för detta delmål.

Här är det viktigt att mängden filtyper begränsas på något sätt, för att uppdelningen av filtyperna ska vara användbar. Skulle det ändå dyka upp en större mängd filtyper som behöver klassificeras bör man använda sig av hierarkisk gruppering.

4.2 Metod för delmål 2: Inhämtning av data från spel

För att lösa detta delmål i problembeskrivningen är det viktigt att valet av spel uppfyller vissa krav:

- **Aktuellt:** Spelet måste ha släppts de senaste åren för att filtyperna ska kunna vara någorlunda aktuella.
- **Relevanta filtyper:** Spelet måste innehålla någon av de filtyper som har klassificerats i det föregående delmålet (eller åtminstone varianter av dem).
- **Flera olika versioner:** De aktuella filtyperna måste finnas i minst två olika versioner av samma fil (detta framgår tydligt i problembeskrivningen).

I största möjliga mån skall det väljas flera olika spel, för att inte resultaten skall bli alltför unilaterala.

Några alternativ för val av metod i det här delmålet kunde ej uppbringas, på grund av dess smala problemspecifikation. Följaktligen utförs denna metod enligt ovan.

4.3 Metod för delmål 3: Val av deltaenkodningsteknik

Valet av deltaenkodningsteknik styrs framförallt utifrån två kriterier: komprimeringseffektivitet samt användbarhet. Komprimeringseffektivitet innefattar både tidsåtgång för komprimeringsoperationerna och storlek på den deltakomprimerade datan. För att undersöka dessa två kriterier kan dels implementation med efterföljande analys användas, men även litteraturanalys.

Implementationen ska med hjälp av olika deltaenkodningstekniker utföra operationer på vanliga filer, för att sedan kunna jämföra resultaten från dem. Implementationen ska även lyfta fram deltaenkodningsteknikernas olika inställningsmöjligheter och extrafunktioner så att även dessa kan jämföras.

Litteraturanalysen är med här för att med säkerhet fastställa att resultaten från implemenationen inte kan vara helt felaktiga. I exempelvis Hunt, et al. (1996) och Trendafilov, et al. (2002) jämförs flera bibliotek och verktyg för deltaenkodning grundligt med varandra.

4.4 Metod för delmål 4: Implementation av testsystem

Testsystemet ska implementeras på ett sådant sätt att den bygger uppdateringsinformation och skillnadsfiler mellan olika versioner av en filsamling med hjälp av deltaenkodning. Utöver detta skall man även kunna använda sig av vanlig komprimering istället för deltaenkodning, detta för att kunna jämföra deltaenkodningens relativa effektivitet.

Testsystemet ska även implementeras så att man automatiskt får ut resultat över storleksskillnader mellan filer i de olika versionerna. Resultat över filernas respektive storlekar i olika versioner skall man också få ut av testsystemet, likaså tiden det tar att använda deltakomprimeringsbibliotekets operationer. Även resultat över eventuella operationer utförda av vanlig komprimering skall gå att få ut.

Indatan för testsystemet skall helt enkelt vara flera olika versioner av samma spel, där varje enskild versionsinstans ligger i en separat katalog.

4.5 Metod för delmål 5: Utvärdering och analys av tester

I det här delmålet skall de resultaten från testsystemet i föregående delmål analyseras och utvärderas. Genom att jämföra de olika resultaten i tabeller och diagram kan man snabbt få en överblick över tids- och komprimeringseffektivitet.

4.5.1 Testutrustning

Testerna kommer att genomföras på en dator med följande konfiguration:

- Processor: AMD X2 4400+ (2x2.2 GHz, 2x1 MiB L2-cacheminne)
- Arbetsminne: 2 GiB
- Chipset: NVIDIA nForce4 Ultra
- Aktuell hårddisk: Maxtor DiamondMax 10 250GB 7200RPM S-ATA/150 16MB cache
- Operativsystem: Microsoft Windows XP Professional Service Pack 2

5 Resultat

Det här avsnittet beskriver hur de olika de framställda problemen i problembeskrivningen löstes.

5.1 Resultat för delmål 1: Identifikation och klassifikation av relevanta filtyper

I dagens spel kan man hitta en enorm mängd med olika filtyper. Men givetvis har varje filtyp en uppgift, och således blir det faktiskt ganska lätt att kategorisera dem.

Att kategorisera filtyper kan vara av stor vikt, framförallt om det behövs olika tekniker och tillvägagångssätt för att få ner storleken på dem. Dessutom kan det finnas vissa filtyper som inte ens är värda att försöka komprimera, på grund av deras uppbyggnad eller möjligtvis på grund av deras uppdateringsmönster – vissa filer kan tänkas få hela sitt innehåll utbytt.

5.1.1 Bilder och texturer

PNG (Portable Network Graphics) är ett vanligt förekommande bildformat, som med icke-förstörande komprimering effektivt minskar storleken på bilderna utan att försämra deras kvalitet.

TGA är ett format som är snarlikt PNG, och också väldigt vanligt i spel.

DDS (DirectDraw Surface) är ett texturformat som tar steget längre än att bara lagra en bild. Formatet har bland annat stöd för samma typ av komprimering som grafikkort vanligtvis stöder, så fildatan mer eller mindre kan lyftas över direkt till grafikminnet.

JPEG är ett mycket vanligt bildformat som utnyttjar sig av förstörande komprimering, vilket gör att filstorleken ibland kan bli mycket liten, jämfört med andra bildformat. JPEG används sällan i spel, just pga. dess kvalitetsförstörande struktur.

Andra förekommande bildformat är bland annat BMP (bitmap), och RAW.

5.1.2 Modeller, geometri, animation och andra filer för 3d-data

Det är inte lätt att gruppera filtyper inom detta område. Filerna behöver inte alls ha något med varandra att göra; de kan ha hand om allt från en 3d-modells animationsmönster till en stor bana i ett spel. Ändock har de gemensamt att de lagrar data som på något sätt är relaterad till tredimensionell information.

En annan anledning till varför filer inom det här området är så överblickade är att de ofta skiljer sig från spel till spel. Även om två spel använder sig av samma spelmotor kan filtypen ändå vara annorlunda, t.ex. pga. en annan version av spelmotorn.

5.1.3 Ljud och musik

För att lagra ljuddata finns inte fullt lika många filtyper att välja mellan som för andra kategorier. Ogg Vorbis är ett på senare år populärt format, som till skillnad från det välkända Mp3-formatet inte kräver licens vid användning. Vanligast är nog Wave-formatet, som vanligtvis lagrar sin data helt okomprimerat men har en lättanvänd struktur.

5.1.4 Script- och konfigurationsfiler

I takt med att spelen får högre och högre krav på sig att vara data-drivna ökar även behovet av lätthanterliga och lättmodifierade filtyper. Xml har blivit populärt inom hela datorvärlden de senaste åren – inte minst i web-relaterade sammanhang. Även inom spelrelaterade områden har den lyckats bryta mark, och används i vissa spel till allt tänkbart.

Scripting har också blivit mer populärt i datorspelssammanhang, framförallt med hjälp av språken Lua och Python.

Det script-filer, xml-dokument, konfigurationsfiler och motsvarande textfiler har gemensamt är att de så gott som alltid är i ASCII-format, och använder en begränsad uppsättning tecken.

5.1.5 Programfiler

Själva programkoden som spelets applikation består av finns på Microsoft Windows-system vanligtvis i EXE-filer (exekverbara program) samt i DLL-filer (dynamiskt länkade bibliotek).

5.1.6 Videofiler

Videofiler är inget ovanligt i spel, särskilt för ändamål såsom exempelvis introfilmer. Ofta använder sig spelen av filformat som är vanliga för operativsystemet, till exempel AVI (*Audio Video Interleaved*) eller WMV (*Windows Media Video*). Självklart finns fler än dessa, men dessa filer skiljer sig internt så pass mycket då de använder sig av olika rad olika komprimeringsmetoder – trots att de kan ha samma filändelse.

BIK (*Bink video*) är ett videofilformat speciellt gjort för spel utvecklat av *RAD Game Tools* och har använts av över 3500 olika speltitlar (RAD Game Tools, 2007). Stöd för alla moderna spel-plattformar finns.

5.1.7 Filarkiv

Med filarkiv menas filer som sammanfogar flera filer till en enda fil. I spel används filarkiv av flera anledningar:

- För att dölja filerna från spelaren, så att denne inte kan gå in i filen och titta eller ändra för att fuska.
- För att sammanfoga alla småfiler och ”snygga upp” strukturen.
- För att kunna komprimera filerna.

Filarkivstyper finns det en uppsjö utav, men i grund och botten brukar de flesta likna varandra. Det är dock inte ovanligt att vissa spel använder sig utav exempelvis Zip-arkiv för att lagra sina filer, ibland även om de inte använder sig av den filändelsen.

Pak, pk2, pk3 och pk4 är arkivfilstyper som kan dyka upp i en del spel. Dessa är egentligen vanliga zip-arkiv som helt enkelt har andra namn, och används i motorer som baseras på de olika Quake-motorerna.

5.2 Resultat för delmål 2: Inhämtning av data från spel

Följande spel – med följande versioner – valdes:

- Civilization IV: 1.00, 1.09, 1.52, 1.61.

- Quake 4: 1.0, 1.2, 1.3.
- The Elder Scrolls IV: Oblivion: 1.0, 1.1, 1.2.
- Tom Clancy's Rainbow Six: Vegas: 1.00, 1.01, 1.02, 1.03, 1.04.

Samtliga spel uppfyllde de tidigare ställda kraven:

- **Aktuellt:** Civilization IV släpptes 2005, och Quake 4 likaså. The Elder Scrolls IV: Oblivion släpptes 2006, och Tom Clancy's Rainbow Six: Vegas släpptes 2006. Både Civilization IV och Quake 4 använder sig av Gamebryo. Quake 4 använder sig av Doom 3-motorn. Tom Clancy's Rainbow Six: Vegas använder Unreal Engine 3.0.
- **Relevanta filtyper:** Alla fyra spelen använder stora spelmotorer, och därför bör det inte i större utsträckning vara något problem med udda och irrelevanta filtyper.
- **Flera olika versioner:** De valda spelen hade mellan tre till fem olika versioner tillgängliga via uppdateringar (originalversionerna inräknade).

Spelen installerades, och uppdaterades i tur och ordning. Innan varje uppdatering togs en kopia på hela spelets filstruktur och flyttades undan, så att man i slutändan hade en fullständig samling av filer för varje version, för respektive spel.

De fyra spelen skiljer sig inte bara åt när det gäller typer av filer, utan även när det gäller i vilken skala filupplägget påverkas i samband med uppdateringarna. Som exempel så är den första uppdateringen till The Elder Scrolls IV: Oblivion på bara 1,56 MiB (1 638 668 bytes), medan en av uppdateringarna till Quake 4 är på nästan 235 MiB (246 227 764 bytes).

Mängden spel och versioner som detta delmål resulterade i var medvetet stor, detta för att försäkra tillkomsten av användbar statistik till delmål 5.

5.3 Resultat för delmål 3: Val av deltaenkodningsteknik

För att välja deltaenkodningsteknik fick först en gallring av de tillgängliga genomföras genom att sätta ett grundkrav: deltaenkodningstekniken skall gå att använda i implementationen av testsystemet. Detta krav innebär i stort sett att den måste kunna hantera binära filer, samt att den måste på kunna integreras i testsystemet så att det inte innebär några större problem vid implementationen av det.

Utifrån ovanstående krav finns endast Xdelta och Zdelta som relevanta val, då de flesta andra bara har stöd för deltaenkodning av textfiler, och/eller bara finns som kommandoradsverktyg.

Ett av de två viktigaste kriterierna framställdes vara komprimeringseffektivitet. I Trendafilov, Memon och Suel (2002) jämförs effektiviteten mellan bland annat Xdelta och Zdelta, och Zdelta framstår i flera av testerna som tämligen mycket mer effektivare än Xdelta. Dock har den undersökningen några år på nacken, då det är en tidigare generation av Xdelta som testas, och således är den – likt många andra tester – helt värdelös i det här avseendet.

För att få fram respektive verktygs effektivitet kan man helt enkelt testa dem på olika filer. Detta genomfördes genom att från resultatet för delmål 2 ta några enstaka filer och köra dem både genom Xdelta och genom Zdelta. I det här fallet valdes filer av olika typer och storlekar från spelet Tom Clancy's Rainbow Six: Vegas mellan

versionerna 1.00 och 1.01, samt från Civilization IV mellan versionerna 1.00 och 1.09.

Filtyp	*Storlek (första versionen)*	*Storlek (andra versionen)*	*Resultat: Xdelta*	*Resultat: Zdelta*
Bik	5 221 908	1 452 336	1 391 317	1 384 913
Exe (1)	33 071 104	33 300 480	9 013 366	13 932 949
Exe (2)	11 691 000	11 575 944	7 725 912	9 599 203
Py	83 463	83 923	161	133
Xml	708 832	708 833	113	177

Tabell 1: Resultat från tester av Xdelta respektive Zdelta med utvalda filer från Tom Clancy's Rainbow Six: Vegas mellan versionerna 1.00 och 1.01, samt från Civilization IV mellan versionerna 1.00 och 1.09. Valet av filerna gjordes så att flera olika storlekskategorier och filtyper kunde testas. Värdena är angivna i bytes.

Tabell 1 visar att skillnaden inte är särskilt stor på de mindre filerna – men på de större filerna är ändå Xdelta långt mer effektiv.

Utöver komprimeringseffektivitet så var även användbarhet ett av de två viktigaste kriterierna för valet av deltaenkodningsteknik. Användbarhet är självklart något väldigt subjektivt, men om man syftar på användbarhet utifrån implementation av vårat testsystem så kommer innebörden av det fram rätt tydligt. Bland annat är följande punkter tecken på god användbarhet i den här kontexten:

- Deltaenkodningstekniken ska helst gå att använda både som kommandoradsverktyg och programmeringsbibliotek.
- Deltaenkodningsteknikens olika val ska vara lättåtkomliga, lätta att ändra, och inte vara svårförståeliga. Med val innebär i detta fall exempelvis nivå av komprimering och aktivering av sekundär komprimering.

Till skillnad från Zdelta så finns Xdelta både som kommandoradsverktyg och programmeringsbibliotek, vilket gör Xdelta betydligt mer lättanvänt. Xdelta har dessutom fler val och inställningsmöjligheter än Zdelta, såsom olika typer av sekundär komprimering.

Dessutom har Xdelta har stöd för VCDIFF-formatet, något som kan räknas som en markant fördel när dekomprimering av den deltaenkodade filen ska genomföras, eftersom man ej behöver Xdelta för detta.

Som en sista – men egentligen viktigast – punkt har även de olika teknikerna undersökts utifrån hur aktuella de är. Zdelta kom senast ut med en version 2004-02-23 (version 2.1), medan Xdelta senast kom ut med en version 2007-03-25 (version 3.0q). Granskar man dessutom de olika hemsidorna kan man se att Xdeltas hemsida under perioden 2006 till 2007 varit aktiv, till skillnad från Zdeltas hemsida som inte haft en uppdatering efter den senaste versionen, dvs. 2004-02-23.

Följaktligen är Xdelta ett bättre val än Zdelta för integrering med testsystemet.

För att göra Xdelta än mer effektivt i testsystemet kommer den högsta nivån av komprimeringsnivå att användas, samt sekundär komprimering. Används Xdelta som kommandoradsverktyg blir kommandoradsparametrarna för detta *-9* respektive *–S djw*.

5.4 Resultat för delmål 4: Implementation av testsystem

Testsystemet byggdes upp utefter den valda metoden och implementerades med hjälp av programspråket C++ i Microsoft Visual Studio 2003 .NET i Microsoft Windows XP.

Testsystemet implementerades på sådant sätt att det inte var beroende av ett specifikt deltaenkodnings-bibliotek utan kunde lätt byta – förutsatt att deltaenkodnings-biblioteket uppfyllde de tidigare ställda kraven. Dock utnyttjades inte den möjligheten då ett specifikt deltaenkodnings-bibliotek hade valts tidigare.

För att resultaten skulle bli lättåtkomliga skrev varje operation till en logg-fil. I denna logg-filen lagras för varje versionsändring filnamn med sökväg, originalstorlek, ny storlek, samt storlek på skillnadsfilen. Dessutom lagras även tiden för skapandet av skillnadsfilen samt tiden för återställandet av densamma.

Den lagrade tiden kommer vara helt beroende av datorns prestanda, men eftersom samma datorutrustning skall användas vid samtliga tester är detta inget problem.

5.5 Resultat för delmål 5: Utvärdering och analys av tester

5.5.1 Urval av filer

I var och ett av de valda spelen finns en större mängd filer. För att underlätta överblicken senare har enbart en mindre mängd filer valts ut från de spel vars filer skall undersökas senare (se Tabell 2 respektive Tabell 3). Valet av filer för varje spel är gjorda så att så många filtyper som möjligt är med, för att intressanta jämförelser ska kunna göras.

Id	*Sökväg + filnamn*	*Filtyp*
Bik-1	Assets/Art/Movies/Wonders/RockAndRoll.bik	Videofil
Dll-1	Assets/CvGameCoreDLL.dll	Programfil
Py-1	Assets/Python/Screens/CvDebugInfoScreen.py	Script/text
Py-2	Assets/Python/pyHelper/Unit.py	Script/text
Xml-1	Assets/XML/Civilizations/CIV4LeaderHeadInfos.xml	Script/text
Xml-2	Assets/XML/Text/CIV4GameText_Misc1.xml	Script/text
Thm-1	Resource/Themes/Civ4/Civ4Theme_Window.thm	Script/text
Fx-1	Shaders/FXO/Civ4Leaderheadshader.fx	Script/text
Exe-1	Civilization4.exe	Programfil

Tabell 2: Valda filer från Civilization IV.

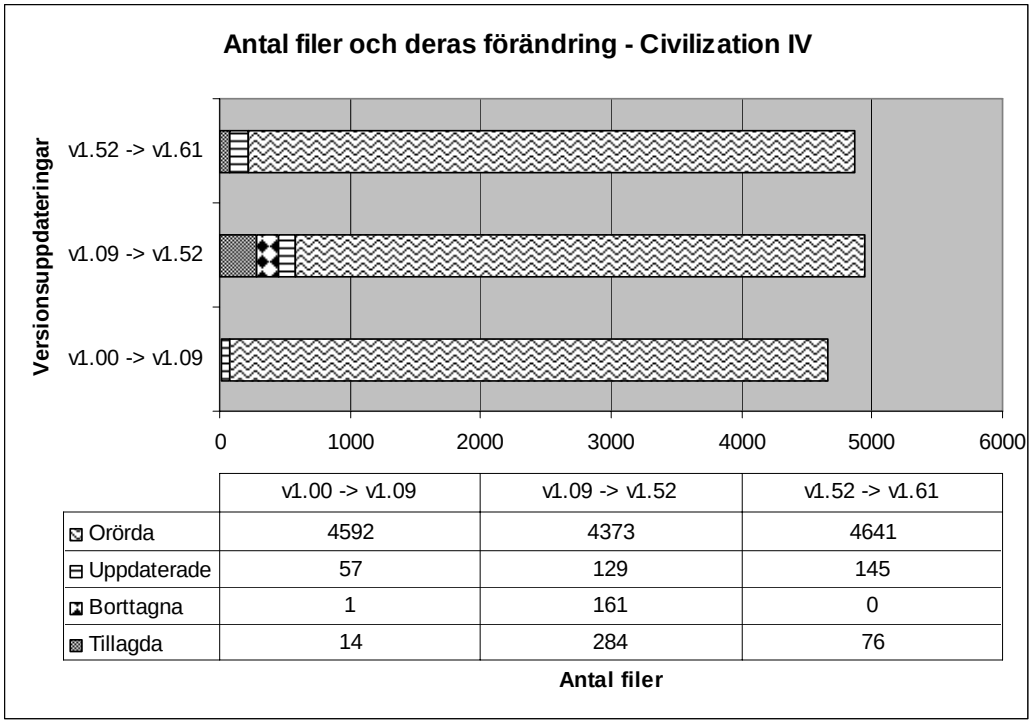
Id	*Sökväg + filnamn*	*Filtyp*
Dll-1	MayaImportx86.dll	Programfil
Exe-1	Quake4.exe	Programfil
Exe-2	Quake4Ded.exe	Programfil
Dll-2	Toolsx86.dll	Programfil
Dll-3	libcurl.dll	Programfil
Pk4-1	q4base/game000.pk4	Filarkiv
Pk4-2	q4base/game100.pk4	Filarkiv
Pk4-3	q4base/game200.pk4	Filarkiv

Tabell 3: Valda filer från Quake 4.

5.5.2 Uppdateringarnas omfattning

Vad som uppdateras skiljer sig avsevärd mellan olika spel. Det kan vara allt från enbart kritiska buggfixar till överflödigt extramaterial som läggs med i spelens uppdateringar.

I Civilization IV används inga filarkiv, vilket märks tydligt på antalet filer (se Figur 4). Något som också är värt att notera är att det inte är särskilt stor del av filerna som faktiskt uppdateras eller på något annat sätt ändras i samband med uppdateringarna.



	v1.00 -> v1.09	v1.09 -> v1.52	v1.52 -> v1.61
Orörda	4592	4373	4641
Uppdaterade	57	129	145
Borttagna	1	161	0
Tillagda	14	284	76

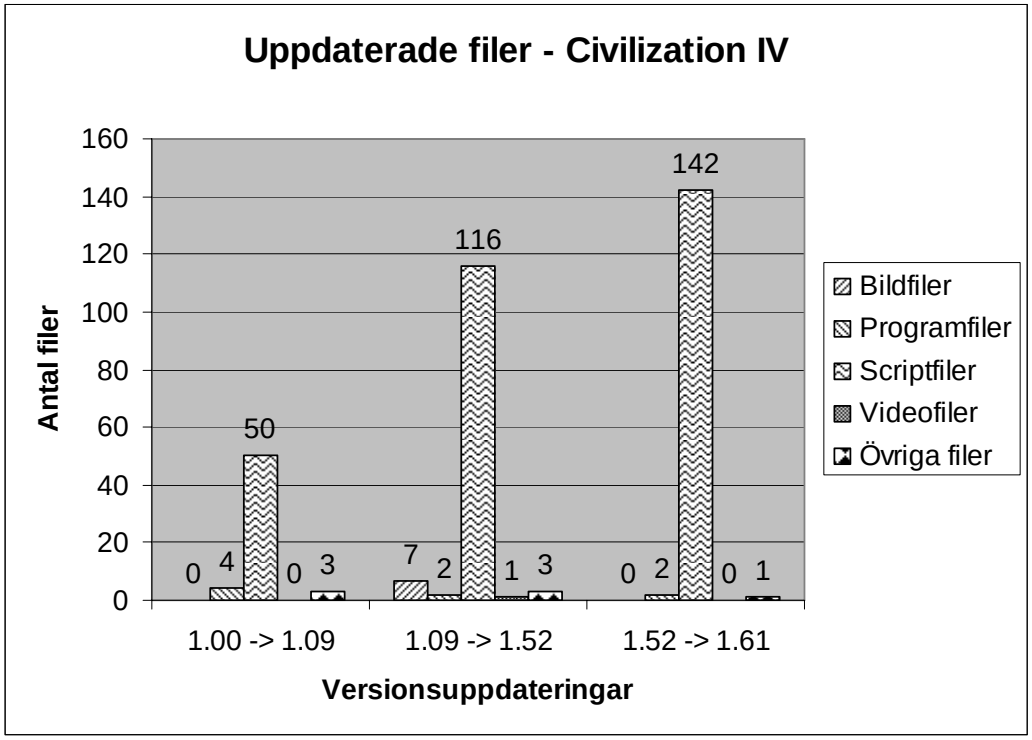
Figur 4: Jämförelse mellan olika filers ändringar i de olika uppdateringarna i Civilization IV.

Till skillnad från Civilization IV använder exempelvis Quake 4 filarkiv, och detta resulterar i betydligt mindre antal filer som hanteras. Vad som inte märks är hur filerna inuti arkiven ändras.

Vad är det då för filtyper som vanligtvis ändras? I Civilization IV (se Figur 5) är det överhängande scriptfiler som uppdateras. Utöver scriptfilerna så ändras även de centrala programfilerna vid varje uppdatering. Till skillnad från exempelvis bildfiler

så är oftast småfel i scriptfiler och programfiler av mer kritisk karaktär. Fel i bildfiler hindrar sällan spelet från att fungera, och upptäcks dessutom lättare innan första versionen släppts.

Scriptfiler står dock sällan för den majoriteten av storleken, då de ofta är relativt små. I fallet med Civilization IV så är (i de två första versionsuppdateringarna) summan av scriptfilernas storlek fortfarande mindre än summan av de övriga filernas storlek.



Figur 5: Fördelningen av de uppdaterade filerna i Civilization IV.

Av ovanstående statistik kan man dra slutsatsen att det vanligtvis är programfiler och scriptfiler som uppdateras. Mediafilerna – bilder, ljud, 3d-data och video – uppdateras tämligen sällan vid vanliga versionsuppdateringar.

5.5.3 Deltaenkodningens komprimeringseffektivitet

Testsystemet konstruerades på så vis att det med hjälp av det valda deltaenkodningsbiblioteket skapar skillnadsfiler för varje fil som har uppdaterats. Filer som tagits bort, lagts till eller är orörda mellan de olika versionerna rörs inte av deltaenkodningsbiblioteket.

I Civilization IV var resultatet av att deltaenkoda de uppdaterade filerna märkbart effektivt. Framförallt script-filerna hanterades bra, som i vissa fall hade skillnadsfiler som tog mindre än 1 % av destinationsfilen (se Tabell 4).

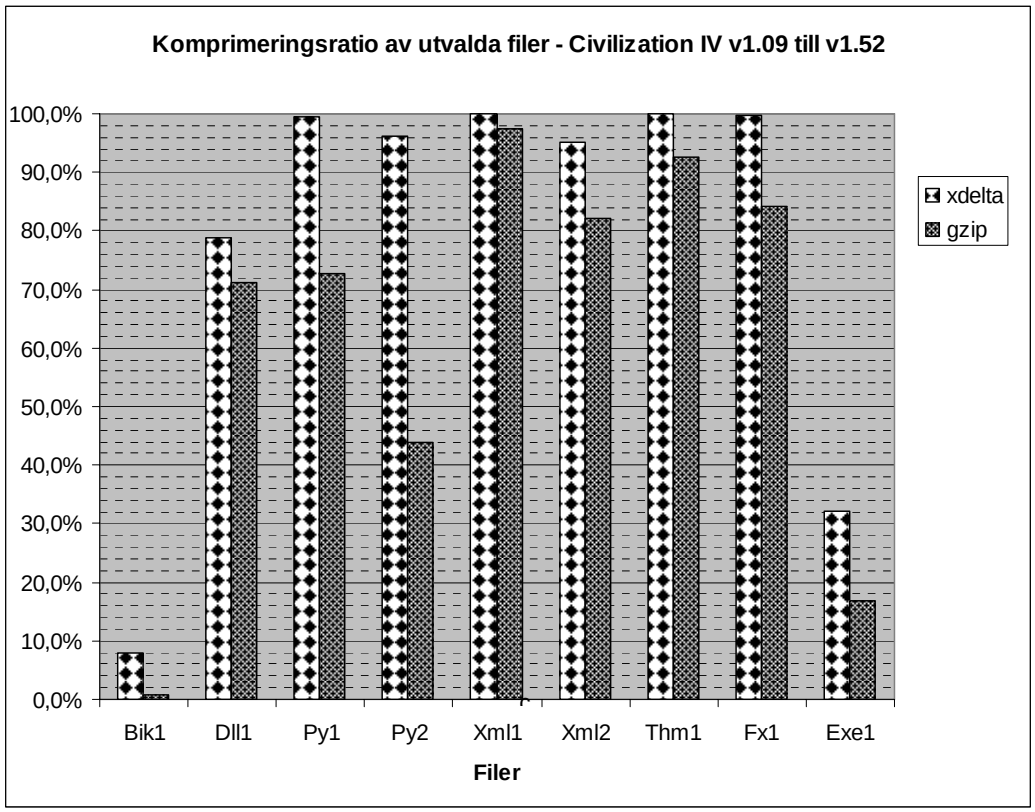
Fil	*Storlek v1.09*	*Storlek v1.52*	*Storlek skillnadsfil v1.09 till v1.52*	*Komprimerings-ratio*
Bik1	9 388 376	10 450 368	9 612 144	**8,0 %**
Dll1	3 579 904	3 588 096	763 060	**78,7 %**
Py1	15 202	15 128	96	**99,4 %**
Py2	1 280	1 281	49	**96,2 %**
Xml1	529 769	529 750	467	**99,9 %**
Xml2	317 605	498 387	23 779	**95,2 %**
Thm1	131 089	131 007	107	**99,9 %**
Fx1	41 564	41 564	137	**99,7 %**
Exe1	11 575 944	11 657 862	7 908 484	**32,2 %**

Tabell 4: En liten del av den mängden filer som var uppdaterade mellan version 1.09 och 1.52 i Civilization IV. Storlekarna anges i bytes. Kolumnen komprimeringsratio visar hur stor skillnadsfilen är i förhållande till den versionen skillnadsfilen uppdaterar till.

Även om de genererade skillnadsfilerna tar väldigt mycket mindre än deras respektive destinationsfiler så är det inte en helt rättvis jämförelse, eftersom det inte är särskilt troligt att filerna skulle lämnas okomprimerade av de ansvariga om man lade till dem i ett uppdateringspaket. Istället är det troligt att de i hög grad skulle bli komprimerade.

För att undersöka hur vanlig komprimering står sig mot komprimering med hjälp av deltaenkodning fick filerna i Tabell 4 även mätas komprimerade. För komprimeringen användes här Gzip med inställningar för bästa komprimering.

Figur 6 visar hur resultatet i form av komprimeringsratio i jämförelse med den komprimeringsration när deltaenkodning istället användes. På samtliga filer är deltaenkodning mer effektivt än vanlig komprimering.



Figur 6: Komprimeringsratio för både vanlig komprimering och komprimering med deltaenkodning i Civilization IV mellan version 1.09 och 1.52. Endast ett mindre urval av filerna är med.

Filnamn	*Storlek v1.2*	*Storlek v1.3*	*Storlek skillnadsfil v1.2 till v1.3*
Dll-1	839 680	839 680	**536**
Exe-1	4 947 968	4 952 064	**555 465**
Exe-2	4 853 760	4 857 856	**511 275**
Dll-2	3 416 064	3 416 064	**6 165**
Dll-3	237 568	237 568	**128**
Pk4-1	1 688 687	1 706 451	**1 703 375**
Pk4-2	2 280 777	2 325 248	**2 293 666**
Pk4-3	5 749 025	5 818 299	**5 759 751**

Tabell 5: Tabell över storleken på de valda filerna som blivit uppdaterade mellan version 1.2 och 1.3 i Quake 4. Värdena är angivna i bytes.

Även om deltaenkodning generellt sett är effektivt, så finns det lägen då tekniken inte gör särskilt mycket nytta. I Tabell 5 kan man se storleken på tre varianter av respektive fil: version 1.2, version 1.3, samt skillnadsfilen mellan de två versionerna. Till skillnad från de flesta filerna i tabellen så är skillnadsfilerna för arkivfilerna (Pk4) ungefär lika stora som ursprungsfilerna.

Huruvida det var hela arkivfilen som bytts ut i samband med verionsuppdateringen, eller om det var deltaenkodningsbiblioteket som inte var effektivt nog för den typen av filarkiv framgår inte. Detta kan dock undersökas genom att faktiskt ta ur filerna ur filarkiven och behandla var och en av dem som separata filer.

Om man kollar på vad som händer med filerna inuti filarkiven, så ser man att det inte handlar om något utbyte av filer, utan precis som med de tidigare filerna fungerar nu deltaenkodningen acceptabelt. Här är det dock viktigt att man känner till att pk4-filerna är komprimerade, vilket syns på en större fil inuti filarkivet Pk4-3 (*game.so.bundle/Contents/MacOS/game.so*) som okomprimerad når en storlek på strax under 17 MiB, när hela Pk4-3 den ingår i är på knappt 6 MiB.

Tabell 6 visar summan av innehållet i respektive filarkiv för version 1.2, version 1.3, samt summan av skillnadsfilerna för versionerna däremellan. Jämför man med skillnadsfilerna från filarkiven i Tabell 5 ser man att storleken på skillnadsfilerna nu är omkring 50 % mindre. Deltaenkodningen blev alltså betydligt mer effektiv när den slapp hantera den här typen av filarkiv.

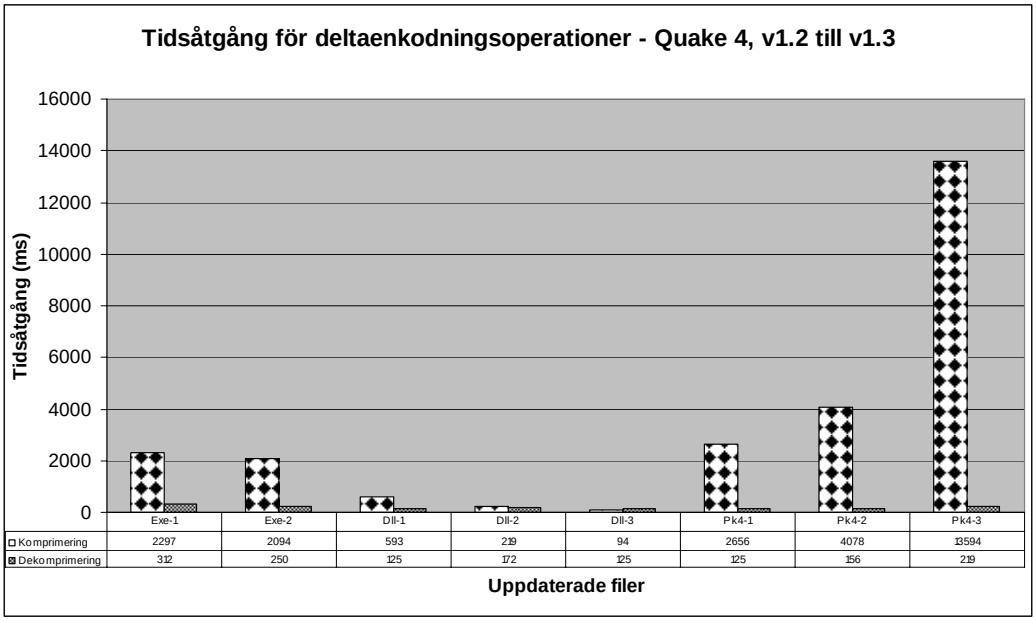
Ursprungsarkiv	*Summa storlek v1.2*	*Summa storlek v1.3*	*Summa storlek skillnadsfil*
Pk4-1	3 911 864	3 956 739	762 149
Pk4-2	5 717 578	5 832 958	1 436 474
Pk4-3	16 885 873	17 075 310	2 649 526

Tabell 6: Filarkiven i Quake 4 (version 1.2 och 1.3) med storlekssumman av respektive filarkivs innehåll.

5.5.4 Tidsåtgång för komprimering och dekomprimering med hjälp av deltaenkodning

En intressant men också ofta bortglömd aspekt i området kring uppdateringar är hur lång tid själva operationerna tar att genomföra. En av motiveringarna till att använda deltaenkodning vid uppdateringar är ju just att det ska ta mindre tid vid överföringar, och således bör det sannolikt vara av stor vikt att deltaenkodningens operationer tar minimalt med tid.

Här bör man inte glömma att tidskravet på dekomprimering är betydligt högre än tidskravet på komprimering. Detta eftersom komprimeringen bara kommer att köras en gång, medan dekomprimeringen kommer att köras en gång för varje användare som uppdaterar. Ur marknadssynvinkel är det av stor vikt att man inte fördröjer eller irriterar användarna – kunderna.



Figur 7: Tidsåtgång för deltaenkodningsoperationer (komprimering och dekomprimering) på de uppdaterade filerna mellan version 1.2 och 1.3 av Quake 4. Tidsåtgången är angiven i millisekunder.

Figur 7 åskådliggör hur tidsskillnaden starkt skiljer sig mellan komprimering och dekomprimering. Generellt sett så tar dekomprimeringen mellan en femtedel och en femtiondedel av tiden som komprimeringen tar. Dock finns det undantag till detta, då små filer har ungefär likadan komprimeringstid som dekomprimeringstid – i vissa fall kan dekomprimeringstiden vara längre!

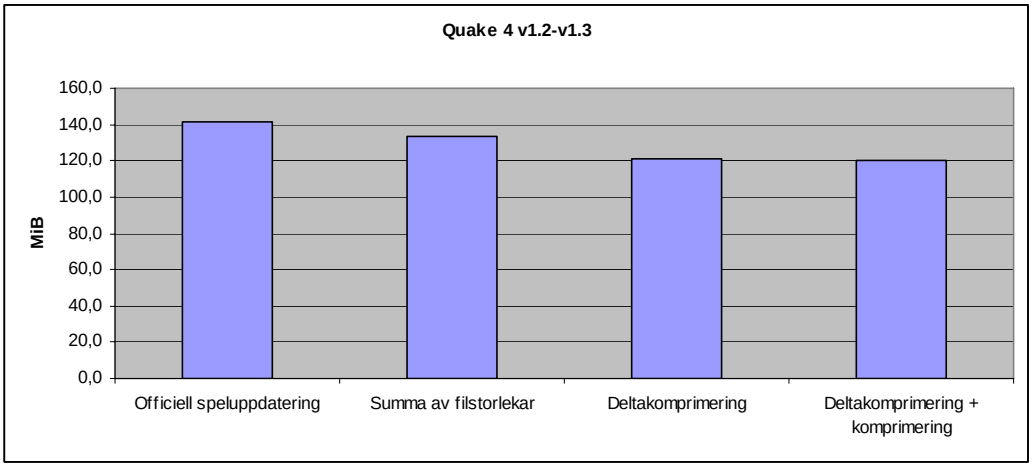
5.5.5 Undersökning av de officiella speluppdateringarnas storlek

För att göra ovanstående resultat än mer intressant kan man jämföra med de speluppdateringar som faktiskt finns till spel. Vad har de för egenskaper? Vad för data kan de tänkas innehålla och hur är den strukturerad?

En majoritet av de speluppdateringar som släpps är inte inkrementella – dvs. de fungerar bara mellan två specifika versioner. Av de valda spelen gick det bara att hitta inkrementella speluppdateringar till Quake 4. Dock hade spelet även icke-inkrementella speluppdateringar utöver de inkrementella.

För Quake 4 finns en inkrementell speluppdatering som uppdaterar från version 1.2 till version 1.3. Den är i EXE-format och tar ca 141 MiB (147 902 828 bytes).

Om man tar summan av filstorleken för filerna som i den patchen både läggs till och uppdateras blir det ca 138 MiB (140 274 113 bytes), alltså mindre än patchen! Om man nu effektiviserar filerna genom att använda deltaenkodning på de filer som blivit uppdaterade sjunker man ner i strax över 121 MiB (126 950 000 bytes). Genom att även ta och komprimera de tillagda filerna når man till 120 MiB (126 296 932 bytes), vilket är nästan 85 % av storleken på den riktiga speluppdateringsfilen!



Figur 8: Storleken på den officiella speluppdateringen för Quake 4 mellan version 1.2 och 1.3 gentemot egna sammanställningar av filerna i samma uppdatering.

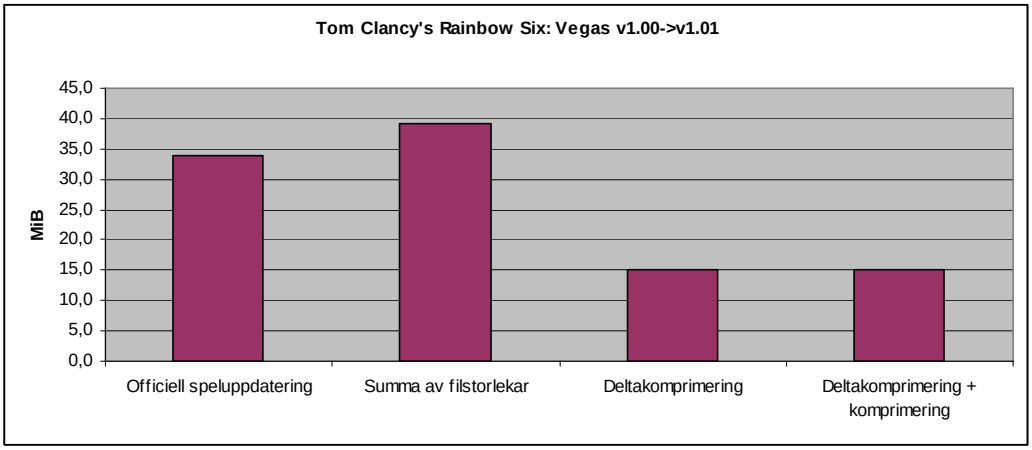
Här kan flertalet intressanta fenomen noteras. Först och främst är komprimeringen av de tillagda filerna (i sista steget ovan) inte särskilt stor nytta då den största tillagda filen – en arkivfil (.pk4) på ca 111 MiB – redan använder sig av komprimering internt.

Den stora frågan är dock varför den tar den officiella patchen så mycket mer? Självklart innehåller den betydligt mer än bara fildatan. Exempel på sådant som den kan tänkas innehålla är: programmet som sådant, listor med sökvägar, checksums och filstorlekar. Säkerligen finns det betydligt mer som används, men inget som är tillräckligt stort för att förklara varför speluppdateringen är så pass stor som den faktiskt är. Här används med stor sannolikhet inte någon deltaenkodning överhuvudtaget, vilket bevisligen hade minskat storleken på uppdateringsfilen – och även den tid det skulle ta för spelarna att ladda hem den.

Den ovan nämnda speluppdateringen är dock inte ett bra exempel då den innehåller en stor förkomprimerad arkivfil. I brist på andra inkrementella speluppdateringar kan en vanlig uppdatering väljas. I exempelvis Tom Clancy's Rainbow Six: Vegas finns en uppdatering som uppdaterar till version 1.01. Eftersom spelets har släppts i version 1.00 och det inte finns några versioner mellan 1.00 och 1.01 kan man förutsätta att den specifika speluppdateringen i praktiken fungerar som en inkrementell sådan.

Speluppdateringen som är i EXE-format är nästan 34 MiB stor (35 543 800 bytes). Om man går tillväga på samma sätt som tidigare och tar summan av filstorlekarna för de filer som ändrats eller uppdaterats blir det 39 MiB (40 985 225 bytes). Här kan man tydligt se att den officiella speluppdateringen använder sig av komprimering.

Tar man nu och använder sig av deltaenkodning på de filer som ändras i samband med det versionsbyte den specifika speluppdateringen innebär blir kommer man ner i 15 MiB (15 863 441 bytes). Nyttjar man dessutom komprimering på de tillagda filerna blir det även här ca 15 MiB (15 831 788 bytes). Här tjänar man mängder av utrymme tack vare deltaenkodningen, men när det gäller den vanliga komprimeringen sparar man knappt in något då de tillagda filerna bara är fem stycken, och dessutom är förhållandevis små.



Figur 9: Storleken på den officiella speluppdateringen för Tom Clancy's Rainbow Six: Vegas mellan version 1.00 och 1.01 gentemot egna sammanställningar av filerna i samma uppdatering.

Utifrån testerna med de två ovanstående speluppdateringarna uppstår två frågor: Varför används inte deltaenkodning, och varför är det så ovanligt med inkrementella speluppdateringar?

De två frågorna går egentligen in i varandra, eftersom deltaenkodning kräver att man använder inkrementella speluppdateringar. Och varför använder man inte inkrementella speluppdateringar? En anledning till varför är troligtvis att de faktiskt är odugliga om man inte har en specifik version av spelet man ska uppdatera. Dessutom krävs det att de redan existerande filerna inte är modifierade.

Icke-inkrementella speluppdateringar erbjuder till skillnad från deras inkrementella motsvarigheter redundans – i det här fallet att man kan uppdatera från valfri version – vilket många gånger krävs med tanke på att användarna inte alltid gör *rätt* när de ska uppdatera. Om en användare ska uppdatera sitt spel med en inkrementell speluppdatering men först råkar ladda ner uppdateringen för fel version, så har den storleks- och tidsvinsten som deltaenkodningen gav ändå försvunnit.

6 Slutsats

Resultaten från testsystemet visar tydligt att deltaenkodning är effektivt för att minska storleken på de filer som uppdateras mellan de testade spelens olika versioner. Om man skulle göra egna motsvarigheter på de officiella speluppdateringar som var med i testet skulle den totala storleken i vissa fall mer än halveras tack vare deltaenkodningen.

6.1 Diskussion kring resultatet

De resultaten det föregående kapitlet tog fram visar som sagt att deltaenkodning är användbart för att få ner storleken på filer i speluppdateringar, men att det medför kravet på att speluppdateringarna måste vara inkrementella – alltså att de bara kan uppdatera en specifik version av spelet.

Problemet med att uppdateringarna måste vara inkrementella för att använda sig av deltaenkodning innebär inte bara att spelet måste vara i en specifik version – om någon har modifierat någon av de filer som ska uppdateras kommer uppdateringen att misslyckas! Det bästa valet i denna situation vore helt enkelt att spelutvecklarna gör både inkrementella och icke-inkrementella patchar tillgängliga, och låter spelaren välja utefter behov.

Även filtyper och deras förekomst i speluppdateringar undersöktes, och här kunde man se att det var framförallt script- och programfiler som uppdaterades. Detta var ju givetvis inte oväntat, då felaktigheter i andra filer inte lika ofta skapar programfel. Man kunde även se att vissa filtyper var mer lämpade för deltaenkodning än andra. Detta gällde framförallt textfiler, men även filer som ej hade någon intern komprimering.

Filarkiven som testades av testsystemet gav inte något vidare resultat på deltaenkodningsoperationerna, men man ska ändock inte räkna med dessa som omöjliga på grund av detta. Utifall att man skulle ha använt deltaenkodning för att skapa riktiga uppdateringar måste man se till att deltaenkodningen görs på filerna *inuti* filarkiven.

6.2 Framtida arbete

Sett utifrån ett väldigt kort perspektiv skulle ett framtida arbete kunna bestå av ytterligare tester och analyser. Det skulle även vara intressant att lyfta in många fler spel vars uppdateringar och deras innehåll kan testas med deltaenkodning, för helt enkelt få en klarare bild över hur olika sorters data blir hanterad.

En annan utveckling av detta arbete skulle kunna bestå i att utveckla ett komplett online-uppdateringssystem, i syfte att helt enkelt eliminera de problem som uppstår i och med att kravet på att uppdateringarna måste vara inkrementella. Om spelaren kör ett uppdateringprogram kan det helt enkelt ta reda på vilka versioner de lokala filerna är i, och sedan ladda ner exakt rätt skillnadsfil för att uppdatera till senaste versionen.

Man skulle även kunna tänka sig att ett sådant system skulle kunna utvecklas ytterligare genom att transparent väva in det i själva spelapplikationen. Om spelet själv kan ladda ner uppdateringarna kan man göra så att spelaren knappt märker detta när han eller hon kör spelet, och således har hela uppdateringsprocessen gått till att bli mer eller mindre osynlig. Detta medför att spelaren inte längre behöver bry sig om eventuella uppdateringar – spelet sköter det helt på egen hand.

Referenser

Bink Video (2007) [Datorprogram] Rad Game Tools http://www.radgametools.com/bnkmain.htm

Chambers, C. & Wu-chang, F. (2005) Patch scheduling for on-line games. *NetGames 2005.*

Civilzation IV (2005) [Datorprogram] Firaxis games, 2K games.

Game engine (2007) *Wikipedia*. Tillgänglig på Internet: http://en.wikipedia.org/wiki/Game_engine [Hämtad 2007.03.26]

Gamebryo Engine (2007) Emergent game technologies. http://www.emergent.net/index.php/homepage/products-and-services/gamebryo/gamebryo-engine

Gzip (Version: 1.2.4) (2007) [Datorprogram] http://www.gzip.org/

Herwig, A. & Paar, P. (2002) Game engines: tools for landscape visualization and planning? *Trends in GIS and virtualization in environmental planning and design* (s. 162–171). Wichmann Verlag.

Hunt, J. J., Vo, K. & Tichy, W. F. (1996) An empirical study of delta algorithms. I: I. Sommerville (red.), *Software Configuration Management: ICSE'96 SCM 6 Workshop, Berlin, Germany, March 25-26, 1996 - Selected Papers (Lecture Notes in Computer Science)* (s. 49-66). London: Springer-Verlag.

Korn, D., Macdonald, J., Mogul, J. & Vo, K. (2002) RFC3284: The VCDIFF Generic Differencing and Compression Data Format.

Lewis, M. & Jacobson, J. (2002) Game engines in scientific research. *Communications of the ACM, 45*, 27-31.

MacDonald, J. P. (2000) *File System Support for Delta Compression*, University of California. Tillgänglig på Internet: http://www.xmailserver.org/xdfs.pdf [Hämtad 2007.03.27].

Quake 4 (2005) [Datorprogram] Raven software, Id software, Activision.

RTPatch (2007) [Datorprogram] Pocketsoft. http://www.pocketsoft.com/

Shaikh, A., Sahu, S., Rosu, MC., Shea, M. & Saha, D,. (2006) On demand platform for online games. *IBM systems journal, 45*.

Source Engine (2007) Valve software. http://www.valvesoftware.com/sourcelicense/default.htm

Suel, T. &, Memon, N. (2003) Algorithms for delta compression and remote file synchronization. I: K. Sayood (red.), *Lossless compression handbook* (s. 269-287). Academic Press.

The Elder Scrolls IV: Oblivion (2006) [Datorprogram] Bethesda Softworks, 2K Games.

Tom Clancy's Rainbow Six: Vegas (2006) [Datorprogram] Ubisoft Montreal, Ubisoft.

Trendafilov, D., Memon, N. & Suel, T. (2002). zdelta: An Efficient Delta Compression Tool. *TR-CIS-2002-02*. Polytechnic University.

Unreal technology (2007) Epic Games. http://www.unrealtechnology.com/

Wagner, R. A. & Fisher, M. J (1973) The string-to-string correction problem. *Journal of ACM, 21*(1), 168-173.

Xdelta (Version: 3.0q) (2007) [Datorprogram] Tillgängligt på Internet: http://xdelta.org/

Zdelta (Version: 2.1) (2004) [Datorprogram] Tillgängligt på Internet: http://cis.poly.edu/zdelta/