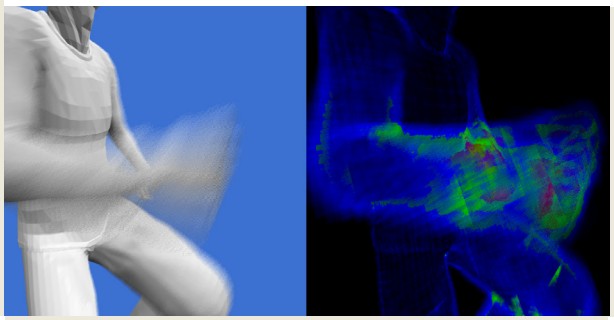
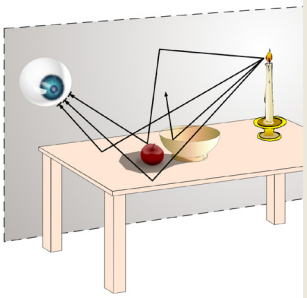
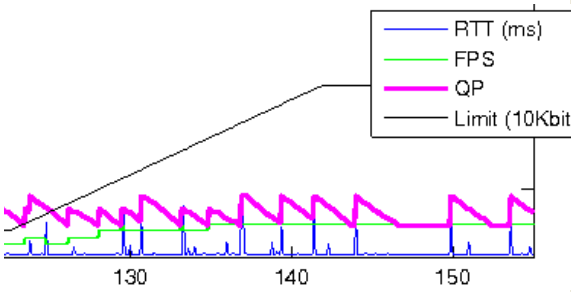
LUNDS UNIVERSITET

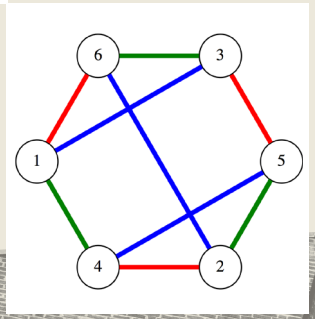
Examensarbete i datavetenskap 17 januari 2014 - program

LUNDS UNIVERSITET | LTH | DATAVETENSKAP









Program 17 januari 2014

Lokal: E:2116, E-huset, Ole römers väg 3, Lund

kl 09.15: On the performance of edge coloring algorithms for cubic graphs
Författare/Authors: Edvin Berglin
Handledare/Supervisor: Thore Husfeldt (LTH)
Examinator/Examiner: Jonas Skeppstedt (LTH)

kl 10.15: On-line Time Synchronization in a Log Analysis Tool
Författare/Authors: Therese Alenlöv
Handledare/Supervisor: Jörgen Bergström (SAAB)
Examinator/Examiner: Pierre Nugues (LTH)

kl 11.15: Motion blurred real-time ray tracing
Författare/Authors: Rasmus Persson
Handledare/Supervisor: Michael Doggett (LTH)
Examinator/Examiner: Tomas Akenine-Möller (LTH)

kl 12.15: Progressive Photon Mapping using Cloud Computing
Författare/Authors: Martin Jacobsson
Handledare/Supervisor: Rasmus Barringer (LTH)
Examinator/Examiner: Tomas Akenine-Möller (LTH)

kl 13.15: Detecting Unique Patterns in Human Head Rotation via Low Resolution Camera Sensor and Imaging Software
Note: bachelor of science/kandidatarbete (15hp)
Författare/Authors: Alexander Neckmar
Handledare/Supervisor: Paul Cronholm (Crunchfish AB)
Examinator/Examiner: Elin Anna Topp (LTH)

kl 14.15: Video conference communication on AXIS cameras
Författare/Authors: Robin Palmblad & Joachim Nelson
Handledare/Supervisor: Jesper Olavi (AXIS)
Examinator/Examiner: Mathias Haage (LTH)

kl 15.15: Performance- and Cost-efficient Cloud Architectures
Författare/Authors: Daniel Raniz Raneland
Handledare/Supervisor: Vadim Feldman (Flygprestanda AB)
Examinator/Examiner: Mathias Haage (LTH)

On the performance of edge coloring algorithms for cubic graphs

Författare/Authors: Edvin Berglin
Handledare/Supervisor: Thore Husfeldt (LTH)
Examinator/Examiner: Jonas Skeppstedt (LTH)

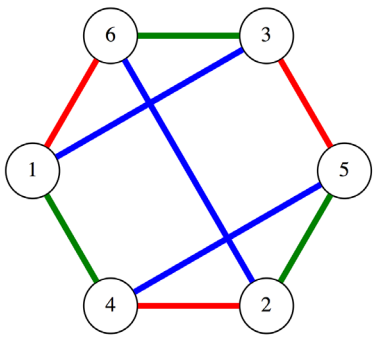
POPULAR SCIENCE SUMMARY (DRAFT)

Motivation

Imagine that you are the organizer of a large football tournament, with perhaps a few hundred competing teams. Most other tournaments begin with a standard round-robin stage, with groups of usually 4 different teams. Our tournament is a bit different in that there is only a single large group: everyone still plays against three other teams, but the teams you've played against do not necessarily face each other.

Football is a physically demanding sport and players need their rest, so a team cannot play more than one game per day. But for you as the organizer, every day with games is a day where the football fields have to be booked and referees have to be paid. To minimize the cost for the organization, you must minimize the number of days with matches.

In mathematical terms, this problem is known as edge coloring. We can model the pseudo round-robin as a graph, where teams are represented as nodes and the "playing against" relationship is represented by an edge. Each node/team is then connected to three other nodes/teams. Our goal is to color the edges in a way that no node has two edges of the same color, using as few colors as possible. We can then associate the colors to the match schedule: "yellow" matches are played on Friday, "red" matches on Saturday, and so on.



Each node has three edges with different colors

The good news is that it can always be done with four colors. But ideally you would like to fit the entire round in only three days, and the bad news is that it is difficult to find out if that is possible. Edge coloring is *NP-complete*, which means that all known ways to solve it run in exponentially-growing time. That is to say, every time you add a team to the tournament, the time it takes to work out a new 3-day schedule is multiplied by some factor.

Method

We look at three algorithms for this problem. The multiplicative factor for each is known, so we know which one is faster when the number of teams is "large enough". But that number might be huge, greater than the number of football teams that actually exist, and it might take many years to calculate the answer. Our goal is to assess the time it takes when the number of teams is more manageable.

While not relevant to the example of football tournaments, one might also ask how many different ways there are to 3-color the edges. Two of the algorithms can answer this. In fact, one algorithm – incidentally the one with the lowest time multiplier – cannot find any 3-coloring without first counting all of them, and for that reason we call it CountColors. The other counting algorithm is EnumColors, which might be described as a refined trial-and-error search that can stop after finding one or continue to find them all. The final one we refer to as Kowalik after its author. It first confirms the existence of a 3-coloring before finding out what it looks like.

An important detail of CountColors is that its memory usage also grows exponentially. Even for relatively small problem instances, and with very large amounts of RAM, that program might expend all of the computer's memory and crash.

Findings

All three algorithms make some form of random decisions, which can dramatically influence the running time. They sometimes made the difference between finishing in less than a minute or in 2 hours. Using a simple technique the decisions become deterministic but still unintelligent: we can reproduce the same chain of choices, but not say why they are good or bad.

EnumColors and Kowalik were used for finding a single 3-coloring, and Kowalik was the clear victor. Single-find EnumColors could handle sizes up to 100 nodes in reasonable average time, while Kowalik could go just above 550. For the counting task, with 256 GB of RAM at its disposal, CountColors crashed about 25% of the time for graphs of 90 nodes, and about half the time for 100 nodes. But when it succeeded, it was markedly faster than counting EnumColors, which struggled with graphs of 76 nodes.

Lastly we looked to parallel processing to make our programs even faster. For EnumColors and Kowalik, specialized parallel software proved unhelpful. Instead, it was a better choice to simply run many concurrent programs with different decision chains, and wait for the quickest one to finish. This way, Kowalik and single-find EnumColors could cope with graphs twice as large as before. A parallel version of CountColors was up to 3 times faster, but at the cost of using about 40% extra memory so it did not let us work with larger graphs.

Tidssynkronisering i ett logganalysverktyg

Författare/Authors: Therese Alenlöv
Handledare/Supervisor: Jörgen Bergström (SAAB)
Examinator/Examiner: Pierre Nugues (LTH)

POPULÄRVETENSKAPLIG SAMMANFATTNING (UTKAST)

Introduktion

Att analysera loggar kan ofta vara ett tidskrävande arbete, ett arbete som dock underlättats av medarbetare på SAAB AB som skrivit ett nytt program kallat Logan, som istället för att vara ett klassiskt textbaserat logganalysverktyg utnyttjar människans förmåga att upptäcka grafiska mönster. Logan är skrivet så att en användare läser in en fil, som då visas som en matris av celler, där en cell innehåller värden som man sedan kan välja att markera och byta färg eller lysa upp, vilket leder till att alla celler med samma värde också byter färg eller lyses upp. Med hjälp av detta kan man då markera värden för att se hur ofta de förekommer, och snabbt få en översikt om det till exempel är så att ett värde alltid följs av ett annat precis innan en felkod, om det är det man letar efter.

För att göra Logan så användbar som möjligt är den konstruerad så att all inläsning av filer och formatering av celler sker genom en plug-in. Det är nu framtaget en plug-in som klarar av att läsa in mer än en fil åt gången, för att kunna analysera loggfiler upptagna under samma körning men av olika datorer. Det är därmed önskvärt att denna plug-in också kan tidssynkronisera dessa loggfiler så att användaren kan se i vilken ordning olika events har inträffat, där ett event motsvaras av en cell. Anledningen till att det krävs en tidssynkroniseringsalgoritm är att de olika datorerna har varsin intern klocka, och dessa klockor kan man inte lita på att de går lika. Därför introduceras en modell för varje intern klocka sådan att en klockas interna tid beroendes på den verkliga tiden (*t*) kan uttryckas genom $C(T)=b*t+a+R(t)$, där $R(t)$ är ett brus med medelvärde noll.

Linjär regression

Då det inte finns någon tillgång till verklig tid, uttrycks istället ett samband mellan lokal tid och global tid, där en godtycklig fil är utvald att ha global tid. Detta fungerar då det är ordningen mellan cellerna som är viktig, och inte de verkliga tiderna. Sambandet kan med hjälp av omskrivning av ovan nämnda formel uttryckas som $t=E*C(t)+D+S(t)$. Med hjälp av linjär regression och så kallade globala event kan nu parametrarna E och D skattas, där ett globalt event innebär ett event som loggats av fler än en dator. Dessa kan utnyttjas eftersom de måste ha samma globala tid, och med linjär regression kan vi då skatta parametrarna E och D.

Algoritmen

Den linjära regressionen kan ske med olika mycket material som grund, vilket har undersökts noga. I en första variant av algoritmen valdes en fil ut som global, och denna jämfördes sedan med alla andra filer, en och en för att skatta dessa filers parametrar. Denna metod blir väldigt känslig för val av fil med global tid, då det kan bli så att denna fil innehåller ytterst få globala event, och den linjära regressionen kräver åtminstone två punkter för att kunna skatta parametrarna. Utvecklingar behövde alltså ske av denna metod, varför nästa steg var att fortfarande ta ut en fil som anses vara global, och jämföra denna med en annan fil och skatta dennas parametrar. Nästa fil kan därmed jämföras med båda dessa filer som redan konverterats till global tid, för att få fler punkter vilket gör den linjära regressionen säkrare, då den blir mindre bruskänslig ju mer material den baseras på. Då detta förbättrar skattningarna för filerna som jämförs med fler än bara en fil, så finns dock fortfarande filer som alltså använder väldigt lite material för sina skattningar. För att kunna förbättra skattningarna även för dessa infördes en tredje version av algoritmen, som inte nöjer sig med att gå igenom filerna en gång, utan fortsätter att skatta filernas parametrar i en loop, som bryts först när parametrarna stabiliserats, det vill säga när två skattningar av samma parametrar inte skiljer sig mer än en bestämd tröskelnivå.

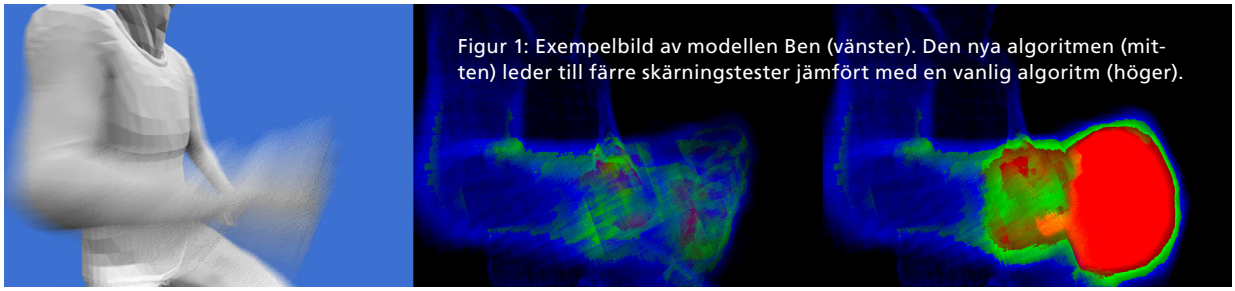
Utvärdering

Det visade sig att det var svårt att trots stora skillnader i implementeringen av algoritmen upptäcka några skillnader i deras respektive prestation. Det stod dock klart att den första versionen var sämst, och då vi inte kunde se några försämringar för den tredje versionen valdes denna ut som den bästa, då den i teorin borde kunna reducera bruset bäst. Det som utmärkte sig var att ordningen på filerna som synkroniserades spelade viss roll för resultatet, men i de tester som gjordes bevarades samma ordning av cellerna, även om tiderna skiljde sig olika mycket mellan eventen. Trots att huvudidén är att de globala eventen har samma globala tid så har de efter omvandling till global tid sällan just detta, vilket beror på brus, vilket gör metoden väldigt bruskänslig. Eftersom ordningen spelar roll och de globala eventen nästan alltid misslyckas med att få samma globala tid är metoden inte perfekt, däremot lyckas de mellanliggande eventen nästan alltid hamna i rätt ordning i de tester som utförts, vilket måste ses som ett lyckande.

Strålföljning i realtid med rörelseoskärpa

Författare/Authors: Rasmus Persson
Handledare/Supervisor: Michael Doggett (LTH)
Examinator/Examiner: Tomas Akenine-Möller (LTH)

POPULÄRVETENSKAPLIG SAMMANFATTNING (UTKAST)



Figur 1: Exempelbild av modellen Ben (vänster). Den nya algoritmen (mitten) leder till färre skärningstester jämfört med en vanlig algoritm (höger).

Introduktion

Datorgrafik har utvecklats mycket de senaste decennierna och i takt med att beräkningskraften ökar så ökar kraven på fotorealism. Inom filmindustrin används strålföljning för datorgenererade effekter, vilket ger fotorealistiska bilder men kräver mycket tid. Datorgenerering av bilder används också inom 3D-spel, där bilder genereras vid visningstillfället, i realtid. Detta examensarbete har fokuserat på en visuel effekt, rörelseoskärpa, och har undersökt möjligheterna att utveckla en strålföljare för realtidsrendering som hanterar rörelseoskärpa.

Ray tracing

Strålföljning kan förklaras som simulering av ljusstrålar. Genom att simulera ljusstrålarna som träffar kameralinsen kan man räkna ut var de kommer ifrån och vilken färg det blir på var och en av pixlarna i bilden. Det gör att man får väldigt realistiska bilder där skuggor, reflektioner och refraktioner blir verklighetstrogna. Eftersom det handlar om väldigt många ljusstrålar, vilka kan reflekteras eller brytas oändligt många gånger innan de träffar kameran blir det väldigt beräkningstungt. Genom att införa vissa begränsningar kan bilderna renderas på skälig tid.

Motion blur

Rörelseoskärpa uppstår då objekt eller kamera rör sig under tiden som kamerans slutare är öppen. På grund av detta uppfattas suddighet i bilder som rörelse, och spelfilm kan använda 24 bilder per sekund och ändå upplevas mjukt och utan hack. Om man inte hade haft rörelseoskärpa på datoranimerade objekt i film skulle dessa synas tydligt och upplevas felplacerade. Rörelseoskärpan är alltså en viktig visuell effekt som ger information om rörelse i bilden.

Accelererande datastruktur

Vid strålföljning måste varje ljusstråle testas mot varje triangel för att se vilken triangel den träffar. Detta är väldigt tidsödande och för att snabba upp dessa test används en accelererande datastruktur. Denna datastruktur organiserar ofta trianglarna i en trädstruktur och reducerar antalet tester som krävs. I detta projekt har en BVH (Bounding Volume Hierarchy) använts, vilken organiserar trianglarna i avgränsande volymer rekursivt.

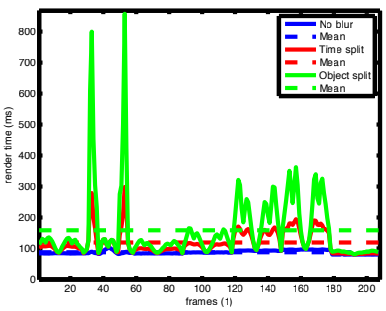
Utförande

För att få rörelseoskärpa i en strålföljare måste ljusstrålarna kopplas till en specifik tid i intervallet då kamerans slutare är öppen. Rörelseoskärpan simuleras genom att följa flera strålar för varje pixel och tilldela var och en av dessa en slumpmässig tid. När färgen för varje stråle blandas för att bestämma färgen på pixeln blir bilden suddig om olika strålar har träffat olika trianglar. I projektet har en fungerande strålföljare anpassats för att hantera olika tider för strålarna. Vid generering av ljusstrålarna tilldelades varje stråle en tid i intervallet [0, 1]. Datastrukturen anpassades för att hantera trianglarna i två tidpunkter (t = 0, t = 1) samt kompletterades med delning i tid utöver delning i rummet som normalt används för att dela in trianglarna i en hierarki. Datastrukturen byggdes innan strålföljningen kördes. För följningsmomentet förändrades koden så att den tar hänsyn till tidsintervallet på noder, trianglar och strålar samtidigt som skärningstest utförs i rummet. För varje triangel som ska testas så interpoleras positionen för triangeln fram utifrån tiden som strålen har. Därefter utförs ett skärningstest mellan triangel och stråle.

Figur 2: Renderingstider plottade för test utan rörelseoskärpa (blå), rörelseoskärpa med ny algoritm (röd) och rörelseoskärpa med gammal algoritm (grön).

Slutsats

Resultatet av att dela trianglarna i scenen inte bara i rumsdimensionerna utan även i tids-dimensionen är att de avgränsande volymerna blir mindre. Detta leder till att färre skärningstester måste utföras vilket förbättrar prestandan vid mycket rörelse. I figur 2 syns hur renderingstiderna förbättras i partier där det är mycket rörelse. Eftersom datastrukturen inte byggs i realtid under körning så är det svårt att säga vad prestandan hade blivit om så var fallet. Det står ändå klart att prestandan i en strålföljare med rörelseoskärpa förbättras med denna algoritm.



Att generera realistisk datorgrafik med tusen datorer

Författare/Authors: Martin Jacobsson
Handledare/Supervisor: Rasmus Barringer (LTH)
Examinator/Examiner: Tomas Akenine-Möller (LTH)

POPULÄRVETENSKAPLIG SAMMANFATTNING (UTKAST)

För många för ordet datorgrafik tanken till datorspel. I datorspel är det viktigt att grafiken framställs snabbt och känns följsam. Men i andra fall är det istället grafikens realism som är det viktigaste. Man kan då acceptera att det tar längre tid att framställa den. Enligt utsago skall t.ex. 25% av bilderna i IKEAs senaste katalog genererats av en dator. En svårighet när man vill ta fram verklighetstrogen grafik är att simulera ljusets utbredning tillräckligt noggrant och att göra detta tillräckligt snabbt.

Indirekt ljus

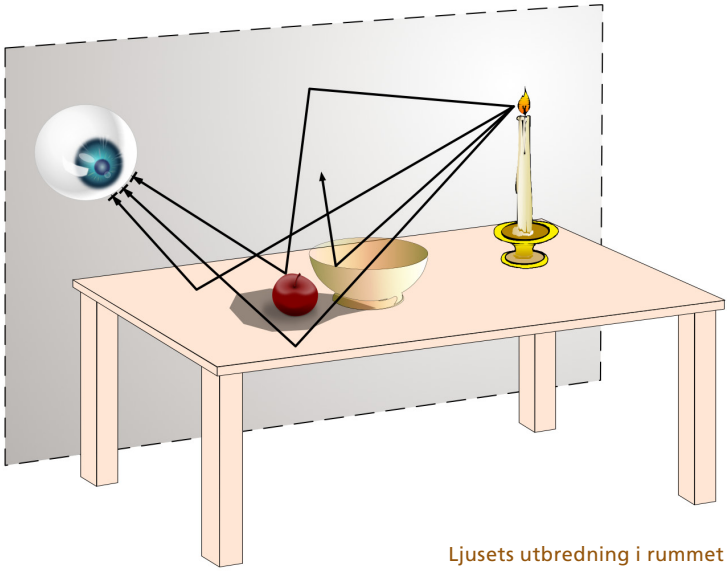
För att förstå svårigheten kan man föreställa sig ett rum utan fönster eller dörrar. I rummet finns ett bord och på bordet står ett levande ljus och en fruktskål. Skålen kastar en skugga och i denna vilar ett äpple. Man får nu föreställa sig ljusets utbredning som strålar i alla riktningar med ljuslågan som utgångspunkt. Skuggan uppstår när skålen stoppar strålarna på väg mot äpplet. Men om skålen hindrar ljuset från att nå äpplet, hur kan vi då uppfatta det med våra ögon?

Jo, när en stråle når en yta absorberas en del av ljuset. Denna absorption är grunden till vad vi upplever som färg och intensitet. Den del av ljuset som inte absorberas reflekteras ut i rummet. Ljuset kan på så sätt nå andra delar av rummet och reflekteras igen. Genom upprepande reflektion i väggar, tak och andra saker i rummet - ja till och med fruktskålen - kan ljuset slutligen nå fram till äpplet. Äpplet reflekterar sedan i sin tur en del av ljuset in i våra ögon. Det är det stora antalet vägar ljuset kan ta innan det når våra ögon som gör det svårt att simulera ljuset i en dator.

Metoder

Grundläggande metoder för att simulera ljusets utbredning i en dator togs fram för flera decennier sedan. Metoderna var exakta men långsamma. Det kunde ta en dator flera veckor att framställa en bild.

Men datorerna utvecklades och blev snabbare. Forskarna började också arbeta på mer effektiva metoder som snabbare kunde utföra simuleringarna - särskilt med de nya datorerna. På 90-talet uppfanns en metod som kom att kallas "Photon Mapping" och fick stort genomslag. Ett problem med denna metod var att den ibland krävde mycket datorminne för att nå hög kvalitet på resultatet.



Ljusets utbredning i rummet

En vidareutveckling presenterades 2008 och fick namnet "Progressive Photon Mapping". Epitetet "progressive" fick den nya metoden eftersom man kunde fortsätta simuleringen från en redan framställd bild för att förbättra kvaliteten. Man kunde på detta sätt nå högre kvalitet utan att det krävdes mycket mer minne.

Men trots att man med den nya metoden kunde simulera ljus med relativt lite minne var man fortfarande begränsad av datorns beräkningskapacitet. För att nå resultat snabbare skulle det krävas att man använde flera datorer samtidigt för en och samma simulering.

Beräkning i molnet

Under senare halvan av 2000-talet såg man ett teknikskifte. Plötsligt kunde man enkelt hyra datorer på internet och betala för dem per timme. Istället för att själv inhandla en hel uppsättning datorer kan man alltså idag köra simuleringen i det s.k. "molnet".

För en komplicerad simulering kan man också, istället för att hyra en dator och göra bildberäkningen på 1000 timmar, hyra 1000 datorer och utföra den på en enda timme. Och detta för samma kostnad!

Ett villkor för att detta skall fungera är att man först anpassar simuleringsmetoden så att man kan dela upp beräkningarna på flera datorer i molnet samtidigt. Detta är inte alltid enkelt. Särskild hänsyn måste t.ex. tas till hur effektivt man kan skicka information mellan datorerna som utför beräkningen. Detta eftersom en dators beräkning kan bero på sådant som en annan dator redan räknat ut.

I detta arbete har vi anpassat metoden "Progressive Photon Mapping" och uppvisar det resulterande beräkningssystemet i "molnet". Vi visar så hur man effektivt kan utföra komplicerade ljussimuleringar på kort tid.

Detecting Unique Patterns in Human Head Rotation via Low Resolution Camera Sensor and Imaging Software

Författare/Authors: Alexander Neckmar
Handledare/Supervisor: Paul Cronholm (Crunchfish AB)
Examinator/Examiner: Elin Anna Topp (LTH)

POPULÄRVETENSKAPLIG SAMMANFATTNING (UTKAST)

Introduction

This paper is a study of ways to recognize horizontal rotational head movement using existing software capable of pinpointing face position in images from video streams. The software makes use of the embedded camera in laptops and mobile devices. The hopes of this study is to identify a unique pattern in rotation of the human head that could then be used to produce left and right commands, using once head, to various devices with access to cameras. For instance, these commands could then become useful in situations where traversal of information from a devises is desirable but where both hands are otherwise occupied.

Architecture

The already existing imaging software provides x and y coordinates referring to the center of the face and the width of the face for each frame in the image stream. In my solution, this data is sent to a state manager that filters out consistent and consecutive sequences of data, consistent in that they all share a common change in the x direction. The state manager adds the data to a vector which is then sent to an algorithm to be analyzed whenever the change in direction of the data has come to an end.

Algorithms

Three different methods were devised for making the distinction. The first method generates a theoretical sequence of a head rotation movement based on the data from the recorded vector. It makes the assumption that the initial angle of the face, the start of the sequence, is perpendicular to the screen. It also assumes that the velocity of the rotation is constant. The end angle of the rotation is either predetermined as a condition, or approximated using the length of the sequence and the average face width. After generating this sequence, the average deviation between this sequence and the recorded sequence is calculated and used as a measurement of how much the actual sequence resembles rotation.

The second method is based on calculating the initial acceleration in x of the sequence, which tends to be greater when the sequence is produced by rotation. This was estimated to take place in the first 62.5 milliseconds of the movement. The frame rate required to collect enough data for the calculations was estimated to 50 frames per second, and the precision required would have to be high enough to detect a change in x representing at least 0.71 percent of the face width.

The third method is based on the previous method. One of the problems with the previous method is the difficulty in calculating the acceleration. The third method gets around this by instead calculating the average speed of the gesture. Due to the initial acceleration in x of the gestures produced by rotation tending to be higher than those produced by normal translation, sequences with a smaller total change in x, where the initial acceleration has greater impact, should be faster when produced by rotation. The following formula was devised: , where is the error in precision. Thirty tests using quick rotation and thirty tests using quick translation all with a similar change in x and at the same distance from the camera where made and yielded the following 95 percent confidence intervals; 1.8203 +/- 0.4488 for rotation and 1.1377 +/- 0.2183 for translation.

Conclusion

The first method is the only method of the three capable of detecting rotation at any velocity as long as that velocity is constant. However, the assumption that the velocity is constant is not true, that is, the method should take this into account. In order to do that, a deeper study of rotational head patterns would be required so that the model could be properly compensated. Also, the method demands high accuracy and precision, something that is difficult with low resolution images.

The second and third methods are limited to detect quick head rotations. The second method would have a faster response time since it only deals with the initial part of the gesture, but as with the first method, it has high demands on accuracy and precision.

The last method is the one that worked best, the tests that were made showed a strong distinction between rotation and translation at a 95 percent confidence level.

Video conference communication on AXIS cameras

Författare/Authors: Robin Palmblad & Joachim Nelson
Handledare/Supervisor: Jesper Olavi (AXIS)
Examinator/Examiner: Mathias Haage (LTH)

POPULAR SCIENCE SUMMARY (DRAFT)

Introduction

Video conference calls is today a widely spread method of communication and is used both in professional or personal environment. There are many popular applications for video conferencing available on the market, such as iChat and Skype. These types of applications are not stand alone and thus requires separate hardware, such as a computer, to be able to use. In our work we have examined the possibilities of implementing a video conference solution directly into an Axis network camera and thus eliminating the need for additional hardware.

An important issue for video conference systems is to be able to handle networks that are slow or varies a lot in available bandwidth. In this work we have examined how to continuously regulate the quality in order to produce a video stream of as high quality as the network can handle.

Method

This thesis was carried out at Axis communication. Axis contributed with a setup consisting of two Axis network cameras. The first task was to implement a way for these cameras to send a video to each other and also to receive and process the video streams. With this initial solution working, the video streams needed a way to be regulated somehow for when the network couldn't handle the load. This required the sending side to get feedback from the receiver telling it how good the network is. The thesis examines how this may be done and also what actions the sender can take to regulate the video stream based on this feedback.

The sender needed an algorithm to know when and how much it should regulate the video stream. In this thesis we have developed and compared two such algorithms in aspects of user experience, video quality and time it takes to transfer the frames (latency).

Quality maintenance

The video streams were sent over the network in H.264 video format. This is a very well known video compression format for video streams. It has the functionality to create lossy video, which means that not all information is maintained in the compressed video. This results in a smaller video size at the cost of lower video quality. This has been utilized in the thesis in order to lower the bitrate. We have also implemented a way to regulate the numbers of frame per second (FPS) being sent out for this purpose.

So now that we knew how to lower the bitrate, the question was when? To determine this we made use of Real-time Transport Control Protocol (RTCP) packages. These packages are sent from the receiver to the sender and contains information the sender can use to figure out the quality of the network. The thesis mostly focuses on latency and the percentage of packages lost over the network.

By examining the feedback we receive in the RTCP packages it was possible to determine if the video quality or FPS should be regulated. In the thesis we developed and compared two algorithms. Both algorithms mainly focuses on the latency to determine whether the video quality or FPS should be increased, maintained or decreased. The difference is how the algorithms handle quality and FPS regulation. The first, naive solution, simply lowers or increases the quality when needed. When the video quality is at maximum/minimum value, the FPS is decreased/increased if possible. A test run of the naive algorithm is shown in figure 1. A more advanced solution takes into account that H.264 generates a higher bitrate when there is much movement in front of the camera. It predicts what quality is most suitable based on how much movement there was in previous frames. This algorithm is able to maintain a certain bitrate by constantly regulating the video quality and FPS.

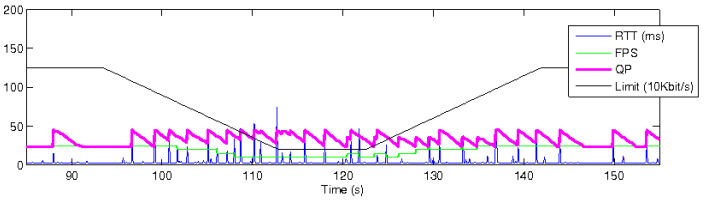


Figure 1: A test run of the naive algorithm.

Result

The implementation resulted in a fully working video conference system with a few issues such as occasional screen tearing in the video. Due to issues with the H.264 format, the video stream has proven to be sensitive to package loss. This makes it difficult to use in a real world situation where there is much package loss.

The video conference system were tested in a testing environment that simulated different network scenarios. It was shown that the naive algorithm was able to maintain a latency of below 150 ms when there are sudden changes in available bandwidth. The more advanced algorithm couldn't be as agile to change the quality and thus had much more latency. The naive solution was worse at utilizing the available bandwidth and never utilized more than 50% of the available bandwidth while the more advanced algorithm could use almost 100% of available bandwidth.

All in all, the naive algorithm is usable for video conferences but there is much room for improvement. The more advanced algorithm needs to react faster to get lower latency in order to be usable.

Prestanda- och kostnadsoptimerade molnarkitekturer

Författare/Authors: Daniel Raniz Raneland
Handledare/Supervisor: Vadim Feldman (Flygprestanda AB)
Examinator/Examiner: Mathias Haage (LTH)

POPULÄRVETENSKAPLIG SAMMANFATTNING (UTKAST)

Att flytta sina servrar från källaren till molnet blir mer och mer populärt, man behöver inte längre oroa sig för att köpa in och underhålla fysiska datorer och man kan enkelt utöka sin kapacitet utan att behöva vänta på en budbil. När Flygprestanda AB flyttade sin serverlösning från kundens källare till Amazons EC2 öppnades möjligheter för att effektivisera resursanvändningen samtidigt som en del tidigare okända prestandaflaskhalsar uppvisade sig.

Flygprestanda AB har tidigare erbjudit sin klient-/serverlösning, FOCS, till kunder som ett installationsprogram de kan köra på sina egna servrar och klientdatorer men efter att flera mindre företag som inte har möjlighet att drifta sina egna servrar uttryckt önskemål om att köpa en komplett lösning med serverhårdvara inkluderad bestämdes det att en centraliserad lösning skulle sättas upp på Amazon Web Services där Flygprestanda har full kontroll över både (virtuell) hårdvara och mjukvara. Samtidigt ska även den traditionella modellen fungera för kunder som har möjlighet att drifta sina egna servrar.

Jämfört med en isolerad lösning som körs på plats hos kunden erbjuder en centraliserad lösning som körs på Amazons EC2 möjligheter dela gemensamma resurser mellan flera olika FOCS servrar för att uppnå en högre effektivitet. Värt att notera här är att FOCS är utvecklat på ett sätt som inte tillåter att en serverinstans servar flera olika kunder. Däremot är en del av de tjänster som FOCS utnyttjar för beräkningar helt tillståndslösa och kan därmed brytas ut och användas som delade resurser.

Att dela resurser mellan flera olika servrar öppnar möjligheter för att antingen spara pengar genom att använda totalt färre resurser eller genom att erbjuda bättre och snabbare service för samma kostnad. Genom att använda så kallad autoskalning så kan man kombinera dessa och matcha den beräkningskapacitet som finns tillgänglig med den som krävs och därmed se till att man inte betalar för mer beräkningskapacitet än vad som behövs.

När klienten testkördes mot en nyuppsatt FOCS server på Amazon EC2 upptäcktes en del prestandaproblem som efter undersökning med hjälp av instrumentering av källkoden visade sig bero på både den längre fördröjningen mellan klient och server för en server som befinner sig i ett datacenter på Irland gentemot en server på samma lokala nätverk samt på grund av den nätverksbaserade ”hårddisk” som valt att användas på molnservern på grund av att det gav enklare administration av servern.

Dessa prestandaproblem löstes genom att minska antalet anrop mellan klient och server genom att slå ihop flera mindre anrop till ett större och genom att lagra alla I/O-intensiva resurser på det lagringsutrymme som finns tillgängligt lokalt på varje virtuell server i EC2, detta innebar dock en längre uppstartstid för servern eftersom dessa resurser måste kopieras över nätverket innan de är färdiga att användas.

I slutändan uppnåddes en centraliserad servermiljö körandes på EC2 med flera beräkningsintensiva resurser delade mellan flera servrar. Någon autoskalning sattes däremot aldrig upp eftersom kapacitetskraven på de delade resurserna inte var tillräckligt stora för att kräva mer än en server per resurs.

Lunds Universitet | LTH | Datavetenskap • Produktion: Jonas Wisbrant 2014-01-17 v.1 • Tryck: E-husets vaktmästeri

LUNDS UNIVERSITET
Lunds Tekniska Högskola

cs.lth.se/examensarbete

LUNDS UNIVERSITET
Lunds Tekniska Högskola
Datavetenskap
Box 118, 221 00 LUND
Tel 046-222 00 00
cs.lth.se