Visualisering av eyetrackingdata

Heatmapgenerator

JOHAN TIDÉN
och MATTEUS KLICH

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2011

Visualisering av eyetrackingdata

Heatmapgenerator

J O H A N T I D É N
o c h M A T T E U S K L I C H

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2011
Handledare på CSC var Mads Dam
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2011/tiden_johan_OCH_klich_matteus_K11003.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Abstract

Eyetracking is a technique developed to register a persons eye movements. The data collected from an eyetracking session is a set of points on a plane, on which the person has looked at. When large amounts of data have been collected, the simple dot representation of the data is not sufficient to recieve a good overview of the result. The need for a better data representation therefore arises.

This report presents a model for a heatmap generator. Each point is given radial mass, where the highest mass is in the center of the point. Each value is then translated into a color and an alpha value.

An implementation of the model has been made in C# and the results are described and discussed in the report.

Förord

Vi vill börja med att tacka EyeTrackShop för det samarbete vi har haft och för den inspiration vi har erhållit av dem. Den resulterande mjukvaran är nu en modul i deras mjukvarulösning.

Den teoretiska lösningen och mjukvaran påbörjades under sommaren 2010 i samarbete med EyeTrackShop. Där var vi själva ansvariga för uvecklingen men hade vissa krav på slutresultatet. EyeTrackShop var mest intresserade av slutresultatet, programmet i C#.

Själva formaliseringen av teorin, huvudmålet med rapporten, har vi själva helt stått för. Efter samarbetet med EyeTrackShop har utvecklingen av såväl teorin som mjukvaran fortsatt på eget ansvar, där kraven från EyeTrackShop har lättats. Både rapportskrivning och vidare utveckling har följt en scrumliknande metodik, där vi i början av varje arbetstillfälle diskuterat dagens mål.

Rapportskrivningen har skett genom att träffas och gemensamt diskutera och skriva. Delar av Kapitel 2 har Johan var mer ansvarig för, där hans erfarenhet från bl.a. kursen i bildbehandling kunde utnyttjas. Det är svårt att ange vem som är ansvarig för en specifik del av rapporten, eftersom vi haft ett nära samarbete när arbetet utförts.

Innehåll

0.1 Ordlista 1

1 Inledning **3**
1.1 Bakgrund 3
1.1.1 Eyetracking 3
1.1.2 Datavisualisering 4
1.1.3 Heatmap 5

2 Modell **7**
2.1 Datapunkter 7
2.2 Gammamatris 8
2.2.1 Gausskurva 8
2.2.2 Konvolution 9
2.3 Heatmap 10
2.3.1 Färgskala 10
2.3.2 Alfaskala 10

3 Resultat **13**
3.1 Cachad gammamatris 13
3.2 Snål konvolution 14
3.3 Cachad Colormap 15
3.4 Array vs. Matris 15
3.4.1 Prestandamätning av implementation 15

4 Diskussion **17**
4.1 Komplexitet 17
4.2 Gausskurva 18
4.3 Gammamatris 18
4.4 Användning av heltal 18
4.5 Övriga optimeringar 19
4.5.1 Parallelliserbarhet 19
4.5.2 Bildhantering 19
4.6 Slutsats 20

Litteraturförteckning **21**

Bilagor **22**

A **23**
A.1 Tabellvärden 23
A.1.1 Färgskala 23
A.1.2 Alfaskala 23

0.1 Ordlista

Alfa	Alfa inom bildhantering representerar hur ogenomskinlig en pixel är. Låg alfa betyder alltså att en pixel är genomskinlig.
ARGB	Alfa-RGB. Ett vanligt sätt att lagra färg och alfadata för en pixel.
Cache	Cachning, eller mellanlagring är inom datalogi generellt sett en metod att temporärt lagra data för senare bruk.
Cache-minne	Avser datorns inbyggda processor-cache som lagrar data som hämtats från RAM-minnet, som processorn kan läsa mycket snabbt.
Eyetracking	En process där man registrerar ögats fixeringar.
Filter	Inom bildbehandling avses en linjär operation som ska utföras på en bild.
Fixering	En fixering är en punkt ögat har stannat till vid.
Heatmap	Grafisk visualisering av data m.h.a. färger.
Hit-Ratio	Ett mått på hur väl cache-minnet utnyttjas.
Gammamatris	En matris som innehåller gammavärden.
Gammavärde	En representation av intensitet. Hög gamma betyder hög intensitet.
Gaussfilter	Ett filter som skapas av en gausskurva. Ger en naturlig utjämningseffekt.
RGB	Red Green Blue. Ett vanligt sätt att blanda färger för att beskriva en godtycklig färg.
Stimuli	Det en person ser framför sig under en eyetracking-session.

Kapitel 1

Inledning

Den här rapporten presenterar en modell för effektiv generering av heatmaps utifrån en mängd datapunkter.

Modellen implementeras i C#. Den implementeras även i Matlab, men endast för att visa att samma resultat uppnås oavsett programspråk. C# kommer vara helt i fokus eftersom man lättare kan bygga applikationer runt en implementerad modell.

Problemformuleringen för denna rapport kan sammanfattas med en kort frågeställning; Kan man specifiera en modell som är generell[1] men ändå tillräckligt specifik för att uppnå samma visuella resultat i olika programspråk? Med andra ord ska den presenterade modellen i rapporten kunna implementeras med olika programspråk, på sådant sätt att det visuella resultatet är det samma för alla implementationer.

1.1 Bakgrund

Här ges en beskrivning av vad eyetracking är eftersom det var inom denna bransch modellen ursprungligen utformades. Däremot kan man tänka sig att den används inom andra områden där man vill visualisera många datapunkter.

1.1.1 Eyetracking

Forskningsområdet eyetracking började studeras redan på 1800-talet, genom att då manuellt göra direkta observationer [1]. Man märkte redan då att när man läser en text så sker det inte en "jämn" förflyttning av blicken, utan fokus stannar under korta stunder på platser med små hopp emellan. Det finns två typer av rörelser som intresserar forskare, ryckningar (eller sackader) och fixeringar [2]. Man säger att ögat rycker när den gör väldigt snabba förflyttningar, omkring 30 ms [3]. Under denna korta tid hinner ögat inte registrera det den tittar på. En fixering sker när dessa ryckningar tillfälligt stannar en viss tid. Den kortaste tiden för att klassifiera en fixering är definierat till någonstans mellan 100 ms till 300 ms, beroende på

[1] Med generell menas här att parametrar kan justeras för att uppnå olika visuella effekter.

källa [2][4][2]. Det är under dessa fixeringar som ögat kan registrera omgivningen. I fortsättningen av rapporten kommer dessa fixeringar att betraktas som datapunkter.

Det var inte förrän på 1970-talet som eyetracking som forskningsområde tog fart på riktigt, då man började använda filmkameror för att spela in ögonrörelserna. Man upptäckte att tekniken kunde användas i andra forskningsområden såsom psykologi och kognitiv lingvistik. På 1970-talet utvecklades metoder som genom att mäta ögonvitan[3] kunde ge en snabb mätning av horisontella ögonrörelser. Däremot var vertikala ögonrörelser svårare att registrera eftersom ögonvitan över och under iris oftast täcks av ögonlocken.

Moderna metoder använder infraröda ljuskällor för att belysa ögonen, som sedan fångas upp av speciella sensorer [3]. Med dessa metoder kan man i dagsläget registrera både horisontella och vertikala ögonrörelser med hög precision. I och med digitaliseringen av eyetracking-tekniken sparas den insamlade datan som koordinater, som anger var någonstans på en yta som testpersonen fäst blicken. En annan metod som börjar växa fram alltmer är användningen av webbkameror för att spela in ögonrörelser. Dessa inspelningar analyseras sedan av algoritmer som räknar ut var på en skärm man tittat.

Då tekniken de senaste åren blivit billigare har användningen vuxit rejält och inte endast inom forkningsvärlden. Tekniken uttnjyttjas också till fler områden än tidigare; inte bara för att registrera vad en person tittat på i syfte att granska det efteråt, utan även för att styra datorer för personer med funktionsnedsättningar.

Inom marknadsundersökningsbranschen används eyetracking mycket inom användbarhetstester, för att exempelvis avgöra kvaliteten av användargränssnitt på webbsidor. Den insamlade datan från en eyetracker kan användas för att bland annat räkna ut på vilka områden en person fäst blicken, och hur länge. Detta kompletteras ofta med uppföljningsfrågor, för att kunna se hur personens fixeringar på ett stimuli korrelerar med vad personen uppfattat och vad denne kommer ihåg.

1.1.2 Datavisualisering

Varje studie genererar mängder av data (fixeringspunkter), vilket gör att det finns ett behov av visualisering. Om man skulle representera varje fixering som en punkt på en yta, inser man snabbt att det skulle bli väldigt svårt att få en överblick över resultatet, även vid mindre studier.

Det finns självklart många olika sätt att visualisera rådata i punktform. Ett av dessa sätt är begreppet heatmap, vilket den här rapporten kommer avgränsa sig till.

[2]Tiden för en sackad och en fixering är inte väldefinierat, och kan därmed variera mellan olika rapporter.

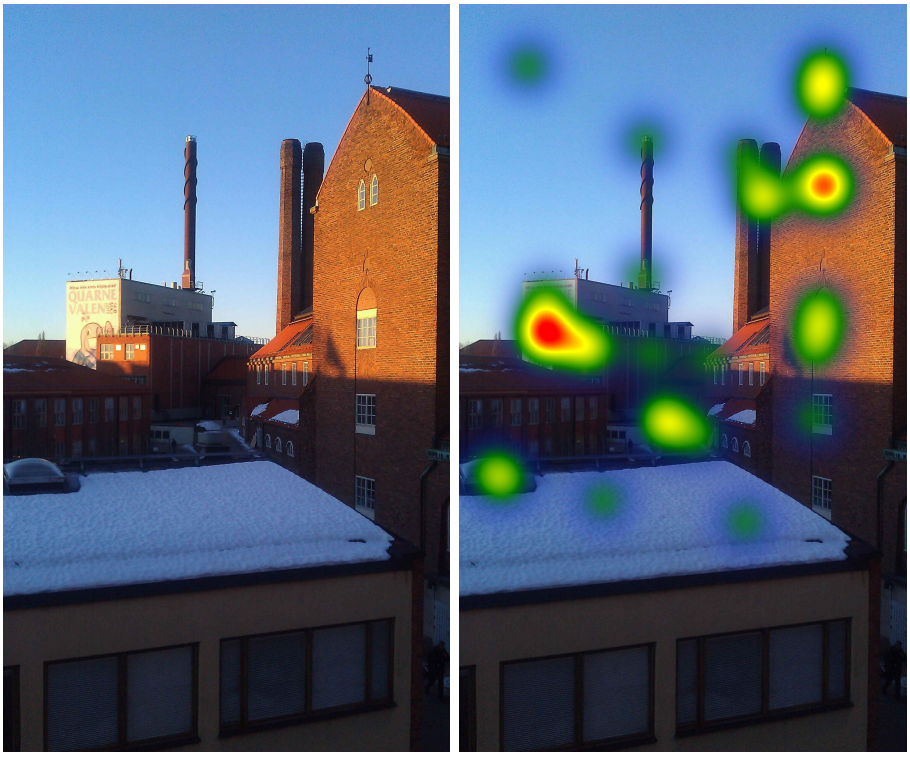
[3]Ögats vita senhinna

1.1.3 Heatmap

En heatmap är ett sätt att grafiskt representera data på en bild genom att använda sig av en färgskala. Vanligtvis är dessa delvis transparenta, flerfärgade lager där områden med många fixeringspunkter får en varmare färg, och de områden med färre fixeringspunkter får kyligare färg [5]. Rött uppfattas som varmt, och blått som kallt. Syftet med en sådan heatmap är att enkelt kunna avgöra vilka områden på ett stimuli som fångar mest uppmärksamhet.

En heatmap kan jämföras med en karta där höjdskillnader i landskapet är utritade med höjdkurvor. Platser med hög höjd får ett högt värde och därmed skulle den på en heatmap få en varm färg. I takt med att höjden minskar och går mot noll, följer färgen med enligt färgskalan och blir tillslut helt transperent.

Olika heatmap-lösningar kan ha en visuellt märkbar skillnad, beroende på vilken metod man använder för att ge massa till datapunkterna (vilket diskuteras senare), och även hur man väljer att färglägga. En heatmap kan således ha olika utseende men det som de har gemensamt med varandra är att varje värde kan omvandlas till en viss färg på en färgskala.



(a) Originalbild (b) Heatmap

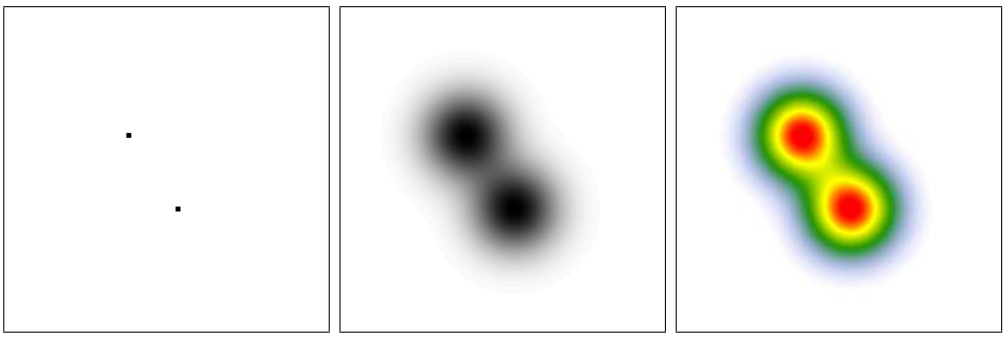
Figur 1.1. Till höger en bild på en heatmap skapad av den beskrivna implementationen i C#. Originalbilden till vänster.

Kapitel 2

Modell

Här beskrivs modellen för att skapa heatmaps. Detta skrivs ur ett bildbehandlingsperpspektiv eftersom större delen av teknikerna som används går bra att beskriva inom terminologin. I det här kapitlet beskrivs lösningen i teorin snarare än i praktiken.

En heatmap kan skapas i tre steg. Man börjar med ett antal datapunkter 2.1(a), lägger till en massa runt dessa punkter och skapar en så kallad gammamatris 2.1(b). Till sist använder man en färgskala för att ”färglägga” denna och får som resultat en heatmap 2.1(c). Kommande sektioner beskriver dessa steg mer i detalj.



(a) Datapunkter (b) Gammamatris (c) Heatmap

Figur 2.1. Översikt över processen att skapa heatmaps. Man börjar med ett antal datapunkter, ger dem massa, och till slut färglägger man.

2.1 Datapunkter

En fixering lagras av systemet som en punkt, med en x- och en y-koordinat. Ritar man ut detta digitalt på en bild, motsvarar detta en pixel på bilden. Man skapar här en matris (eller bild om man så vill) med ettor och nollor.

2.2 Gammamatris

Om man bara färglägger dessa punkter kommer man inte att få en tydligare visuell bild av eyetrackingdatat. Punkterna måste ges massa. Detta kan göras på olika sätt [6]. Det vanligaste är att man skapar en radiell funktion där mitten innehar störst intensitet (eftersom det var där originalfixeringen skedde), och amplituden bestäms av en avtagande funktion. Valet av funktion diskuteras i 2.2.1 nedan.

Ett sätt att spara resultatet från detta steg är att skapa en matris som innehåller heltal. Varje element i matrisen motsvarar en pixel hos en given bild där heatmap-bilden ska ritas ut. Genom att använda denna datastruktur kan man enkelt lägga till nya datapunkter genom att alltid summera resultatet från den nya punkten, med den sparade matrisen (se mer om detta i kapitel 3).

Hädanefter kallas denna matris för gammamatris, eller kort Γ i ekvationerna, och värdena som sparas i denna kallas för gammavärden. Dessa kan ses som någon form av ”höjd” eller ”värme” vilket förklaras mer noggrant nedan.

2.2.1 Gausskurva

Formeln för en generell symmetrisk gausskurva i 2D är:

$$f(x,y) = Ae^{-\left(\frac{(x-x_0)^2}{2\sigma^2}+\frac{(y-y_0)^2}{2\sigma^2}\right)} \tag{2.1}$$

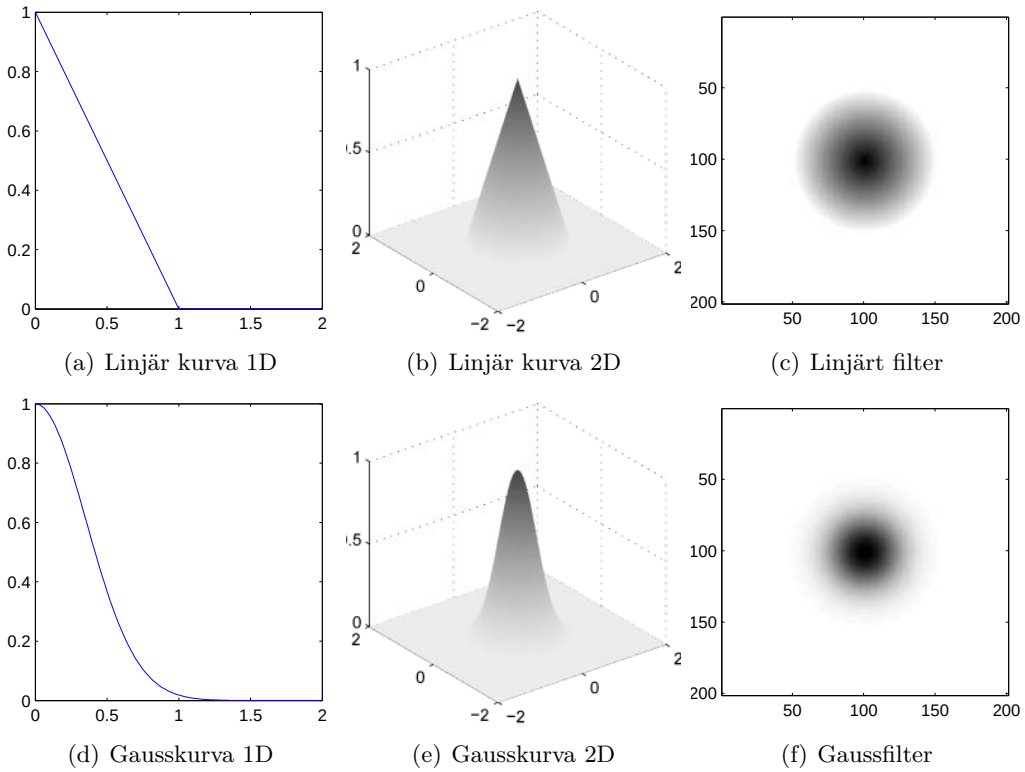
där A är amplituden, σ är standardavvikelsen och x_0 och y_0 är koordinaterna för centrum. A och σ måste bestämmas innan programmet startas. Detta tas upp i kapitel 4.4. x_0 och y_0 bestäms av fixationspunkterna. Det är väl värt att notera att funktionen endast behandlar en punkt. Det krävs alltså att man itererar igenom alla x och y om man vill räkna ut hela ytan.

Gausskurvan är ett bra val i det här fallet på grund av dess inbyggda egenskaper. När man summerar två gausskurvor ovanpå varandra skapar de en väldigt naturlig och intuitiv övergång. Detta kan beskrivas mer noggrant inom signalbehandling där konvolution med en gausskurva precis svarar mot att applicera ett lågpassfilter, som har en utjämnande effekt [7]. De mer ingående tekniska detaljerna för detta kommer inte vidare beskrivas i den här rapporten[1].

Funktionen bildar en avtagande kurva som går från maximala intensiteten A ner till 0. Man måste man välja ett tillräckligt högt värde för A, (Se 4.4) sådan att övergången från A till 0 kan minska på ett ”naturligt” sätt med gausskurvan. Eftersom gammamatrisen endast lagrar heltal måste det finnas tillräckligt många nivåer på intensiteten för att kvantiseringen[2] inte ska bli synlig för användaren [7].

[1] Den som är intresserad rekommenderas att läsa [7]

[2] Kvantisering diskuteras i detalj i 4.4.



(a) Linjär kurva 1D (b) Linjär kurva 2D (c) Linjärt filter

(d) Gausskurva 1D (e) Gausskurva 2D (f) Gaussfilter

Figur 2.2. Skillnader mellan två olika filter. Till vänster ses kurvorna i genomskäring. I mitten ses samma kurvor fast med två rymdkoordinater. Till höger ses de resulterande filtren. Gaussfiltret 2.2(f) är det som används i den beskrivna lösningen.

2.2.2 Konvolution

Konvolution är en matematisk operation där man kombinerar två funktioner för att skapa en tredje. Varje punkt i den resulterande funktionen räknas ut som en kombinerad summa på följande sätt:

$$f(x', y') = g * h = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} g(x', y') \cdot h(x, y) \tag{2.2}$$

där g och h är två matriser och $f(x', y')$ är en punkt i den resulterande matrisen. Här definieras $*$-operatorn som konvolution mellan två mängder.

I det här rapporten konvolverar vi punkterna tillsammans med gaussfiltret[3], enligt ekvation 2.2. Detta resulterar i en mjuk bild (se Figur 2.1(b)). Intuitivt kan man se det som att varje punkt får en liten massa som bidrar till den nya bilden, vilket är ett önskvärt resultat.

[3]Notera att bildfiltrering är en tillämpning av konvolution. Vi använder oss av samma aspekter som bildfiltrering men handskas inte nödvändigtvis endast med bilder.

Det är värt att nämna att datapunkter som är nära varandra summerar sina respektive värden genom konvolutionen, och gammavärdena mellan dessa blir därmed större än värdena hade blivit om de hade varit ensamma. Man kan så att säga se gammavärdet som ett mått på hur tätt punkter ligger inom ett område.

2.3 Heatmap

Nästa steg, innan allt ritas ut är att omvandla gammavärdena till färger. Detta görs genom att först tillsätta ett godtyckligt antal färger i en skala och sedan applicera färgskalan för varje gammavärde i matrisen.

För varje gammavärde används faktiskt två skalor: en färgskala och en tillhörande alfaskala. På så sätt kan färg- och genomskinlighet justeras separat.

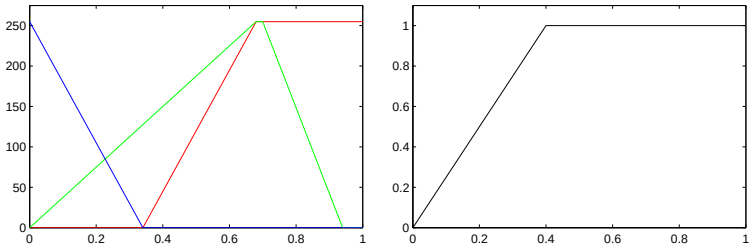
2.3.1 Färgskala

För att ge heatmapen det klassiska utseendet som återfinns i t.ex. värmekameror väljs en passande färgskala där färgerna går från varmt till kallt. Låga gammavärden representeras av blått, som övergår till grönt, gult och slutligen rött som representerar det största gammavärdet. Skalan byggs upp så att det maximala återfunna gammavärdet i matrisen alltid motsvarar högsta värdet på skalan. Man dividerar helt enkelt varje gammavärde med det maximala värdet och får istället en skala mellan 0 och 1, där 1 är den maximala intensiteten. Se bilaga A.1.1 för exakt vilka värden som används här.

2.3.2 Alfaskala

Man brukar inom bildbehandling ge bilder fyra olika värden. De tre första är RGB - röd, grön och blå, och den sista är alfa, som bestämmer hur genomskinlig en given pixel ska vara. Om alfavärdet är noll är pixeln helt genomskinlig och om alfavärdet är 255 är pixeln helt ogenomskinlig. För att få en mjuk övergång mellan bakgrunden och själva heatmaplagret, används en linjär funktion som går från 0 till 255 för låga gammavärden (se 2.3(b)).

Se bilaga A.1.2 för exakt vilka värden som används här.



(a) Färgvärden

(b) Alfavärde

(c) Resulterande färgskala

(d) Resulterande färgskala med applicerad alfaskala

Figur 2.3. Linjär interpolation skapar färgskalan och alfaskalan. Alfaskalan läggs på färgskalan i efterhand. I bild 2.3(d) används svart som bakgrundsfärg men vanligtvis finns en fördefinierad bakgrundsbild här.

Kapitel 3

Resultat

Resultatet bekrivs som en samling av tekniska hinder och lösningar som uppstått under själva implementationsfasen. Undersektionerna tar upp varsin lösning mer detaljerat.

Redan under implementationsfasen togs ett beslut om att profilering ska utföras löpande för att finna flaskhalsar. Detta för att minimera tiden som krävs för generera en bild.

Vid profilering märktes tydligt att konvolution var det som använde mest processortid, vilket ledde till en hel del beslut om optimeringar för detta. Man behöver t.ex. inte konvolvera partier som får summan 0 om man i förväg kan räkna ut vilka dessa är. Flitig användning av temporära data används för att slippa starta om hela processen från början vid varje ändring av indata.

Ett stort designbeslut var att lagra heltal (istället för flyttal) i gammamatrisen för att tillåta en del optimeringar, vilka förklaras mer i detalj nedan.

3.1 Cachad gammamatris

Vissa lösningar för att skapa heatmaps inkluderar funktionen att man kan lägga till punkter efter hand. Man kan alltså inte anta att alla datapunkter finns tillgängliga från början. Vi antar istället att bara en ny datapunkt finns att tillgå vid varje tillfälle. Detta är mer generellt eftersom en lista av punkter alltid kan göras om till en sekvens, och kan således lösas med samma metod.

För att slippa räkna om allt från början varje gång en datapunkt tillkommer, kan man temporärt spara gammamatrisen och endast addera den nya datan till matrisen. Detta kan vi göra utan att förlora precision eftersom vi lagrar heltal. Vi använder oss således av mer minne men sparar väldigt mycket tid för varje ändring.

Denna optimering ger oss även möjligheten att justera färg- och alfaskalorna utan att behöva räkna om hela gammamatrisen, vilket är tidsmässigt kritiskt ur användarens perspektiv eftersom det troligvis sker många små justeringar under körtid.

Eftersom konvolution är en summa (och därmed associativ) så går det utmärkt att bryta ut termer: Låt P vara mängden med alla datapunkter: $P = (p_1, p_2, \ldots, p_n)$ och $P' = P \backslash p_n = (p_1, p_2, \ldots, p_{n-1})$. P' är alltså de punkter vi redan lagt till i gamla gammamatrisen Γ'. Från ekvation 2.2:

$$\begin{aligned}\Gamma(x', y') &= g * f \\ &= \sum_{(x,y)\in P} \sum g(x - x', y - y') \cdot u(x', y')\end{aligned}$$

Bryt ut $p_n = (x_n, y_n)$ från summan.

$$= \sum_{(x,y)\in P'} \sum [g(x - x') \cdot u(x')] + g(x_n - x', y_n - y') \cdot u(x', y')$$

Låt oss kalla den nya summan för $g' * f$ och den utbrutna termen för δ

$$\begin{aligned}&= g' * f + \delta \\ &= \Gamma' + \delta\end{aligned} \tag{3.1}$$

Vi kan nu utnyttja att vi redan har Γ' sparad (då detta är gammamatrisen), och endast summera δ-delen (som är konvolutionen av den nya punkten). Notera att detta endast avser en pixel och måste alltså appliceras på varje pixel i nya bilden.

3.2 Snål konvolution

Denna optimering följer från optimeringen ovan, 3.1. Eftersom man bara adderar en enda gausskurva (δ i ekvation 3.1) varje gång, kan man begränsa vilken yta som behöver uppdateras till den nya gammamatrisen.

Man behöver inte uppdatera hela matrisen. En gausskurva avtar snabbt men får alltid en lång tunn svans (se Figur 2.2(d)). Däremot blir svansen så liten att den inte skulle ge något synligt resultat. Därför kan man efter en viss tröskel välja att inte räkna ut värdet, utan att istället bara hoppa över alla de gammavärden som är utanför tröskelvärdet. Det är viktigt att se till att denna tröskel är tillräckligt stor eftersom ett för litet värde påverkar resultatet. Det är en god idé att låta tröskelvärdet bero på hur stor standardavvikelsen är eftersom den direkt bestämmer gausskurvan. I den här lösningen valdes $|x_0 - x| \leq 6 \cdot \sigma$ och $|y_0 - y| \leq 6 \cdot \sigma$[1]. Begränsningen sätts alltså på bildkoordinaterna. Det kan till exempel (i typfallet) resultera i en uppdatering av storlek 200x200 pixlar istället för 1024x768 pixlar - en markant skillnad i antal pixlar.

[1] σavser här standardavvikelsen för gausskurvan.

3.3 Cachad Colormap

Det är en ganska kostsam process att räkna ut färgmatchingen från gammamatrisen till heatmapbilden. Om man vill undvika att använda direkta minnespekare (så kallad unsafe code i *C#*), så måste man använda Color-objekt[2]. Man förstår ganska snabbt att det är en dålig idé att skapa ett Color-objekt för varje pixel i hela bilden. Detta har avhjälpts lite genom att utnyttja det faktum att gausskurvorna är cirkulära och därmed symmetriska; många av gråskalevärdena kommer att vara identiska. Många av gammavärdena kommer också att vara 0 eftersom det är mycket troligt att hela bilden inte kommer att ha datapunkter (tack vare 3.2). Implementationen är baserad på en Dictionary (HashMap) som man söker igenom. Om värdet (färgen) finns i Dictionaryn så returneras värdet direkt och endast om värdet inte redan finns så räknas det ut och sparas.

3.4 Array vs. Matris

Författarna vill här rekommendera läsaren att läsa igenom ordlistan för att förstå skillnaden mellan cache och cache-minne, för att inte blanda ihop begreppen. Den här sektionen handlar om processorns cache-minne.

Notera att denna optimering är väldigt programspråksspecifik, läsaren uppmanas att själv avgöra om denna kan tillämpas på andra språk. Den här lösningen avser C#.

För den diskuterade lösningen är en 1D-vektor avsevärt mycket snabbare än en 2D-vektor. Detta gäller främst på grund av att man använda cachen optimalt när man itererar genom en lång vektor. I en 2D-vektor hamnar alla elementen *inom en rad* precis efter varandra i minnet, men raderna i sig kan hamna på helt olika ställen. I en 1D-vektor hamnar hela innehållet på samma ställe i minnet. Detta leder till att man kan gå igenom listan element för element utan att hoppa runt i minnet. Det är speciellt effektivt eftersom cache-minnet automatiskt kommer få en högre hit-ratio tack vare bättre minneslokalitet[8].

En 2D-vektor tar emot två heltal för att hämta ut ett element, till skillnad från en 1D-vektor som endast tar emot ett heltal. För att simulera en 2D-vektor med en 1D-vektor måste de två heltal som tas emot göras om till ett unikt heltal i 1D-vektorn såsom $f[y][x] \rightarrow f[y \cdot width + x]$

3.4.1 Prestandamätning av implementation

Den modell som presenterats i rapporten implementerades i programspråket C#, tillsammans med de optimeringar som beskrivits i detta kapitel. Programmet har testats på en stationär dator med en 3,4 GHz Intel i7-2600K processor, 8 GB RAM, ett nVIDIA GeForce GTX 580 grafikkort med 1536 MB grafikminne, samt Windows 7 64-bitars operativsystem. Programmet har ett gränssnitt som möjliggör det

[2]Inbyggd datastruktur i *C#* som används för att representera en färg.

för användaren att lägga till datapunkter genom att använda datormusen. Genomsnittstiden för att lägga till och rita ut en punkt beräknades genom att lägga till 10 datapunkter och räkna ut medelvärdet av detta, som då blev genomsnittstiden. Den uppmätta genomsnittstiden för att lägga till en datapunkt blev ca 467 ms. Tiderna kan variera beroende på innehållet i enlighet med 3.2 Snål konvolution.

Kapitel 4

Diskussion

Modellen är så välspecifierad att man utan problem kan uppnå samma visuella resultat[1] i både C# och Matlab. Detta bevisar däremot inte att man uppnår samma resultat oavsett programmeringsspråk, men det är en stark indikation.

Komplexitet och körtid kommer inte diskuteras för implementationen i Matlab, då syftet med den endast var att verifiera det visuella resultatet.

4.1 Komplexitet

Efter analys av implementationen fås tidskomplexiteten:

$$O(s^2 + s^2 + w \cdot h + w \cdot h) = O(s^2 + w \cdot h) \tag{4.1}$$

där s^2 är antal pixlar i gaussfiltret, w är bredden på bilden och h är höjden på bilden.

Komplexiteten beror alltså på bildens storlek och på storleken av gaussfiltret (vilken till viss del är proportionell till bildstorleken då visualiseringen bör skalas upp för större bilder). Observera här att komplexiteten avser tilläggning av en punkt. För fler punkter behöver hela algoritmen köras på nytt. I diskussionen tar vi upp valet att rita ut på en gång gentemot att först lagra alla rådatapunkter för att sedan rita allt samtidigt.

Detta är nära relaterat till hur generell konvolution beter sig. *Total* konvolution har $O(s^2 \cdot w \cdot h)$ i komplexitet då varje punkt i ena bilden ska kombineras med varje punkt i andra bilden. Nu väljs istället bara fixeringspunkterna ut som värdefulla och resten betraktas som 0, vilket gör att konvolutionen får komplexiteten $O(s^2 \cdot n)$, där n är antalet fixeringspunkter. Notera att $n = 1$ då vi lägger till en punkt i taget, vilket ger $O(s^2)$ i den slutgiltiga komplexiteten (se 4.1).

$O(w \cdot h)$ i ekvation 4.1 kommer från att man räknar ut färgvärden samt ritar ut dessa för varje pixel i bilden.

[1] Med samma visuella resultat menas att avvikelser kan förekomma, men är små nog att en människa inte kan uppfatta detta visuellt.

Valet att rita hela bilden för varje ny datapunkt betyder att prestanda går till spillo. Om man redan i förväg har tillgång till alla datapunkter, kan man ytterligare optimera för detta genom att inte starta om hela algoritmen för varje punkt.

4.2 Gausskurva

Valet att använda en gausskurva för att lägga till massa visade sig vara ett bra val eftersom dess avtagande svans fungerar väldigt bra vid addition av flera närliggande gausskurvor. Deras sammanlagda massa upplevs fortfarande som naturlig och mjuk (och man ser fortfarande var orginalfixeringen skedde), vilket inte är fallet hos t.ex. en linjärt avtagande kurva.

4.3 Gammamatris

Att mellanlagra data såsom det beskrivs i 2.2 Gammamatris hade många fördelar. Processen blir väldigt tydlig eftersom man reducerar problemet till bildbehandling och gör den mindre abstrakt. Det ställer dock lite högre krav på utvecklaren eftersom denne måste förstå grunderna i bildbehandling.

Gammamatrisen är också mycket lättare att debugga eftersom det är mycket enklare att förstå data som heltal jämfört med om man bara hade en färdig heatmap som består av färgdata(ARGB).

4.4 Användning av heltal

Att använda sig av heltal i gammamatrisen och gaussfiltret införde ett behov av att justera amplituden för gaussfiltret för att överkomma kvantiseringsproblem [7] i gammamatrisen. Om parametern hade för lågt värde blev resultatet att värden avrundades med alldeles för stora trösklar vilket fick synliga effekter (se Figur 4.1(b)). Högre amplitud på gaussfiltret ledde till fler kvantiseringsnivåer, vilket minimerade de synliga effekterna (se Figur 4.1(a)).

Det som bestämmer vilket värde man väljer för amplituden hos gausskurvan, är antalet tillgängliga nivåer, d.v.s. storleken på den datatyp som lagrar heltalen, dividerat med antalet möjliga datapunkter. Detta samband är en smärtgräns för hur hög intensiteten får vara för varje datapunkts gammarepresentation. Idén bakom den är att programmet måste klara av att lagra alla punkter inom gränserna för datatypen. Det största värdet på intensitet som går att få är om man sätter samtliga datapunkter på exakt samma plats i bilden. Detta är ett värsta fall, men måste tas i beräkning.

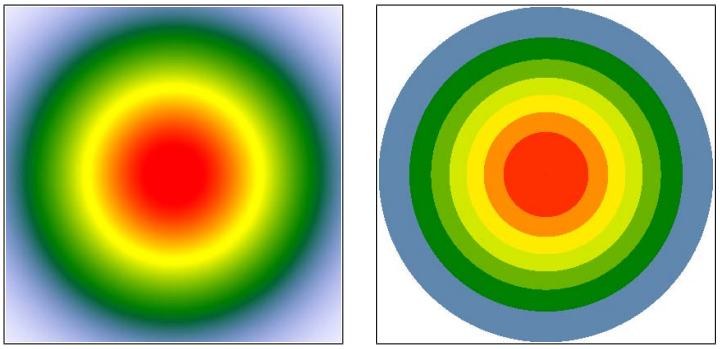
Författarna anser att en maxgräns på 100000 punkter är väldigt högt tak, då detta skulle ta extremt lång tid att beräkna. I den beskrivna lösningen används 32 bitars heltal vilket ger 4294967296 nivåer. Utifrån detta får vi att maximala tillåtna

intensiteten:

$$\frac{2^{32}}{100000} \approx 42949 \approx 40000$$

Författarna kunde inte empiriskt märka av någon visuell kvantisering på så få som 200 nivåer. Figur 4.1(b) har 8 nivåer.

En bättre lösning hade varit att använda flyttal för lagringen så att så mycket precision bevaras så långt som möjligt. Detta ger ett bättre visuellt resultat och tar bort behovet av att justera amplituden på gaussfiltret. Tyvärr måste man i sådana fall räkna med att inte kunna använda alla optimeringar som presenterats i kapitel 3. Flyttalsoperationer är dessutom allmänt mer långsamma än heltalsoperationer. Här handlar det alltså om en avvägning mellan prestanda och bildkvalitet.



(a) Högupplöst (b) Kvantiserad

Figur 4.1. Effekterna av kvantisering på gammanivån.

4.5 Övriga optimeringar

4.5.1 Parallelliserbarhet

Det finns områden där beräkningar är oberoende av andra värden som beräknas samtidigt, vilket gör det möjligt att beräknina dessa parallellt. Exempel på sådana områden: hitta maxvärde i matrisen, omvandla värde i matrisen till färg samt rita ut pixlar på bild.

4.5.2 Bildhantering

Redigering av bitmapbilder i C# är väldigt tidskrävande. Varje pixeluppdatering kräver först att hela bitmapbilden låses, sedan får pixeln uppdateras, och först därefter kan låset på bitmapen släppas. Att hantera låset är en tidskrävande operation när flera pixlar behöver uppdateras. En lösning till detta problem vore att implementera denna bildhantering i exempelvis C++[2], där man har större kontroll

[2]Kod skrivet i C++ kan enkelt inkluderas i C# projekt.

över minnet. Man skulle då kunna läsa in hela bilden i minnet och uppdatera varje pixel utan lås. Detta kan även med fördel kombineras med den ovan beskrivna optimeringen om parallelisering.

4.6 Slutsats

Modellen har vuxit fram genom experimenterande av olika lösningar. En del beslut kan därför vara påverkade av själva implementationen och behovet av optimeringar redan från grunden.

Det finns definitivt mycket kvar att förbättra. Författarna vill påpeka att valet av heltalsmatriser kan ha varit ett misstag. Det införde mer tekniska problem till en relativt liten prestandavinst.

Litteraturförteckning

[1] E. Huey, *The Psychology and Pedagogy of Reading*, ch. 2, pp. 11–19.

[2] H. Y. Tsang, M. Tory, and C. Swindells, "Visualizing sequential fixation patterns," tech. rep., University of Victoria, 2010.

[3] M. Penzo, "Introduction to eyetracking: Seeing through your users' eyes," December 2005. `http://www.uxmatters.com/mt/archives/2005/12/introduction-to-eyetracking-seeing-through-your-users-eyes.php`.

[4] R. Ramloll, C. Trepaginer, M. Sebrechts, and J. Beedasy, "Gaze data visualization tools: opportunities and challenges," Tech. Rep. IEEE 10.1109/IV.2004.1320141, National Rehabilitation Hospital, Catholic University of America Department of Psychology and University of Mauritius Faculty of Engineering.

[5] P. Blignaut, "Visual span and other parameters for the generation of heatmaps," tech. rep., Department of Computer Science and Informatics, University of the Free State, South Africa, 2010.

[6] O. Špakov and D. Miniotas, "Visualization of eye gaze data using heat maps," Tech. Rep. ISSN 1392-1215, Department of Computer Science, University of Tampere and Department of Electronic Systems, Vilnius Gediminas Technical University, 2007.

[7] R. C. Gonzalez and R. E. Woods, *Digital Image Processing 2ed.*, ch. 2, pp. 52–54. Prentice Hall.

[8] D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*, ch. 5, p. 454. Morgan Kaufman, 2009.

Bilaga A

A.1 Tabellvärden

A.1.1 Färgskala

p*	färg
0	Blå
0.34	Grön
0.68	Gul
0.7	Gul
0.94	Röd
1	Röd

A.1.2 Alfaskala

p*	alfavärde
0	0
0.4	255
1	255

* p = gammavärde / maxgamma, där maxgamma här avser det maximala värdet i gammamatrisen, vid division blir det största värdet således 1.