CHALMERS

# Automatisk Animering

Examensarbete inom Högskoleingenjörsprogrammet i datateknik

Andreas Trakossas

Adam Grönberg

Institutionen för Data- och Informationsteknik

CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2014

**Automatisk Animering**

Adam Grönberg, Andreas Trakossas



Institutionen för data- och informationsteknik
Chalmers tekniska högskola
412 96 Göteborg
Tel: 031-772 1000
Fax: 031-772 3663

# Innehållsförteckning

## Sammanfattning

Denna rapport beskriver funktionen samt processen bakom ett verktyg för att automatiskt generera en 2d-animation. Applikationen tar in en sekvens av bilder tillsammans med en gemensam bakgrund till dessa. Sedan genereras de bilder som krävs för att skapa en animation där bildernas föremål rör sig mot föremål i nästa bild i sekvensen. Applikationen kan sedan omvandla dessa bilder till ett videoklipp.

## Abstract

This report describes the function and process behind an application thats been developed. The application takes in a sequence of images together with a common background. It then generates the images that are required to create an animation where the objects in a image moves toward the object in the next image in the sequence. The application can then convert these images to a video clip.

# Förord

Detta examensarbetet uppkom av en idé från Sakib Sistek och är sedan vidareutvecklad till den nuvarande applikationen av Andreas Trakossas och Adam Grönberg. Arbetet har utförts på Institutionen för data- och informationsteknik, Chalmers tekniska högskola, Göteborg och omfattar 15 högskolepoäng av de totala 180 högskolepoäng för dataingenjörer.

Vi vill tacka Sakib Sistek för tillfället att utföra detta examensarbete samt den hjälp och vägledning han bidragit med från start till slut.

Adam Grönberg och Andreas Trakossas 2014-05-29 Göteborg

## Centrala begrepp

**Application Programming Interface (API)** - Regeluppsättning för hur programvara kommunicerar.
**Bildsekvens** - En sekvens av bilder som är organiserade efter deras förekomst i den slutliga animationen.
**Delsekvens** - Bilderna mellan två stycken nyckelbilder. En del av hela bildsekvensen.
**Föremål** - Definieras som sammanhängde skillnaden mellan en nyckelbild och bakgrunden.
**Föremålskontur (FK)** - Ett föremåls ifyllda kontur, lagrad som en matris.
**Föremålsmatris** – Ett föremål separerat från dess backgrund, lagrad som en matris.
**Föremålssilhuett (FS)** - Ett föremåls silhuett, lagrad som en matris.
**GUI** - Grafiskt användargränssnitt.
**Icke Transparent Föremålsmatris (ITFM)** - En föremålsmatris utan den transparenta komponenten.
**ImageObject** - En Javaklass som innehåller information för att placera ut ett föremål på en mellanbild. Informationen består av en föremålsmatris, en ITFM, en mask och föremålets area.
**Mask** - En matris som innehåller formen av ett föremål, lagrad som en matris.
**Matcha** - I denna rapport står detta uttryck för att para ihop föremål.
**Mellanbild** - En bild som verktyget ska generera. Bilden ligger mellan två nyckelbilder.
**Minimum Bounding Rectangle (MBR)** - Minsta avgränsande rektangel runt ett föremål.
**Model, View, Controller (MVC)** - En struktur för att skilja det grafiska gränssnittet från den logiska implementationen.
**Nyckelbild** - En bild som applikationen har som indata.
**ObjectLocation** - En Javaklass som innehåller information för att kunna identifiera ett föremåls position i en nyckelbild. Informationen består av en MBR, en FS och en FK.
**Rotationspunkt** - Den punkt som ett föremål/bild roterar kring.
**Worker struktur** - En struktur för trådhantering. Består av trådar som man kan ge uppgifter till. En tråd benämns som en "worker" och dess uppgift "task".

# 1. Inledning

## 1.1 Bakgrund

Animering är en växande form av underhållning i dagens samhälle, det kan vara allt från korta klipp till stora multimiljon filmer. En gemensam faktor för animationsprocesser är att ett större antal bilder leder till en ökad kvalité. Beroende på kvalitet som eftersträvas och animationens typ kan antalet bilder per sekund som krävs varierar mellan 10 till 30.

## 1.2 Syfte

Syftet med detta arbetet är att implementera ett verktyg som kan underlätta processen för att skapa 2d-animationer genom att automatiskt generera mellanbilder.

## 1.3. Mål

Målet med projektet är att skapa ett animationsverktyg som kan ta in en sekvens av nyckelbilder som sedan kan används för att generera mellanbilder.

## 1.4. Precision av mål

För att precisera verktyget har vissa krav ställts:

- Nyckelbilderna ska innehåller ett antal föremål som kan röra sig mellan olika positioner i planet samt ändra rotation.
- Verktyget ska kunna para ihop föremål mellan två nyckelbilder.
- Verktyget ska vara användarvänligt samt generera animationer på acceptabel tid.
- Föremål försöker alltid ta kortaste vägen från tillståndet i första nyckelbilden till tillståndet i den andra nyckelbilden, detta gäller både rotation och rörelse i planet.

## 1.5 Avgränsningar

Vissa begränsningar har satts på projektet:

- Föremål kan enbart förflytta sig i planet samt rotera.
- Föremål som är 95% lika anses vara identiska.
- Bakgrunden för alla nyckelbilder måste vara identisk.
- Föremål måste vara större än 5% och mindre än 95% av nyckelbilden.
- Föremål får inte överlappa varandra.
- Föremål måste finnas helt inom en nyckelbild, eller inte alls.
- Föremål kan maximalt rotera 180 grader mellan två nyckelbilder.

# 2. Metod

Genomförandet av projektet kommer att ske i en iterativ process där en modulär struktur eftersträvas. En anpassad agil arbetsmetodik kommer att användas i utvecklingen, delarna består av:

- Dagliga korta möten.
- Parprogrammering.
- Minimal dokumentation.

En grov tidsplan har även tagits fram där vissa uppgifter har definierats. Se appendix B för ett gantt-schema.

Verktyget kommer att utvecklas i Java [1] med Eclipse [2] som utvecklingsmiljö. Valet av Java framför andra språk som har stöd för bildhantering, t.ex C++ och Python, är på grund av tidigare erfarenheter i Java. Några programmeringsbibliotek som kommer att användas är OpenCV [3] och Google Guava [4]. Applikationen kommer att utvecklas och testas i Windows-miljö, men ska i teorin stödjas av alla operativsystem som stödjer Java. För att synkronisera och versionshantera koden kommer Git [5] att användas. Java Swing [6] kommer att användas för att skapa ett GUI. Det var planerat att använda OpenCV för att omvandla bilder till videoklipp men på grund av en bugg har de inte stöd för det i Java. Istället kommer Java Media Frameworks (JMF) [7] att användas för att utföra konverteringen från bildsekvens till videoklipp.

Eftersom majoriteten av arbetet kommer att ske med hjälp av OpenCV kommer stora delar av informationen som behöves för att lösa de problem som uppstår att inhämtas från forum samt OpenCVs Application Programming Interface (API).

# 3. Teknisk bakgrund

OpenCV (Open Source Computer Vision) är ett bibliotek för att hantera och manipulera bilder. OpenCV är i grunden utvecklat med C/C++, men har även utvecklas till Java. OpenCV är optimerat för att effektivt lagra bilder under behandling. Detta är kritiskt för applikationen då flera bilder kommer att behandlas parallellt.

## 3.1 Generella OpenCV koncept

### 3.1.1 Matris

I OpenCV används matriser för att lagra bilder, där varje värde i matrisen är en pixel i bilden. En matris instansieras genom ett Mat-objekt som är ett rektangulärt schema av värden där ett visst radnummer tillsammans med ett kolumnnummer refererar till ett värde i matrisen. Vid instansieringen av Mat-objektet kan dess bredd och höjd specificeras.

OpenCV skräphanterar aldrig matriser. Istället har Mat-objektet en metod vid namn *release()* som frigör minnet som matrisen upptar och måste kontinuerligt användas för att undgå minnesläckor [8].

### 3.1.2 Pixel

En pixel i OpenCV består av ett antal komponenter. Varje komponent i pixeln kallas kanal och beskriver pixeln.

OpenCV lagrar pixlar genom flera olika lagringsstrukturer, dessa beskrivs genom formen *<bit-depth>{U|S|F}C<number_of_channels>.* Här är *<bit-depth>* antalet bitar, *{U|S|F}* datatypen och *C<number_of_channels>* antalet kanaler. De olika datatyperna {U|S|F} är unsigned int, signed int respektive float. I projektet kommer endast 8U pixlar användas men antalet kanaler kan variera. Detta medför att varje kanal består av ett värde mellan 0 till 255.

För att beskriva en färgad pixel i OpenCV används RGB-systemet [9], det vill säga genom att

kombinera röda, gröna och blåa färger kan övriga färger skapas. Varje färgkomponent representeras av en kanal. För att göra en pixel transparent används ytterligare en kanal där värdet beskriver hur transparent en pixel är. Lägsta värdet kanalen kan representera betyder 100% genomskinlig medans det högsta betyder fullt synligt.

För hämta en specifik pixel används Mat-objektets metod *get()*. Metoden har en position i matrisen som inparameter och returnerar den specifika pixelns kanaler som en array av typen double. Notera att inte en referens till pixeln returneras utan en kopia av pixeln som fås som en double array.

För att lagra en pixel i en matris används Mat-objektets metod *put()*. Metodens inparametrar består av en position i matrisen samt den array som ska lagras.

De olika strukturerna som används:

- **8UC1** används för att lagra pixlar med enbart en kanal. Värdet i kanalen betraktas som gråskaligt där 0 är vitt och svart är 255.
- **8UC3** används då vissa metoder i OpenCV inte stödjer den transparenta komponenten. Detta leder till varje pixel enbart innehåller tre kanaler vilket tillsammans används för att lagra RBG-komponenten.
- **8UC4** används för att behandla transparenta pixlar. De tre första kanalerna består av RGB-komponenten och den sista kanalen innehåller den transparenta komponenten.

Vid instansieringen av en matris kan dess datatyp samt ett gemensamt startvärde för alla pixlar bestämmas.

### 3.1.3 Rektangel

För att avgränsa viktiga delar i en matris använder OpenCV sig av klassen Rect. Denna klassen innehåller en x-och en y-koordinat, en höjd samt en bredd. Denna klass kan användas för att skapa submatriser av en matris. Dessa matriser som skapas är ett eget Mat-objekt men dess pixlar refererar till original matrisen.

## 3.2 OpenCVs uppbyggnad

OpenCV-biblioteket är uppbyggt i en modulär struktur [10], vilket betyder att OpenCV består av flera mindre moduler.

### 3.2.1 Core

Core-modulen hanterar datalagring i form av flerdimensionella matriser. I projektet sker enbart animering av 2d-bilder, detta medför att enbart 2d-matriser användas.

### 3.2.2 Imgproc

Imgproc-modulen används främst till att manipulera matriser. De metoder som kommer att användas i projektet är:

- ***warpAffine():*** Roterar en matris kring dess rotationspunkt efter angivet gradtal.
- ***findContours():*** Returnerar en lista med konturer från en matris. En kontur är en lista med punkter, där varje punkt är en position i matrisen. En kontur skapas av intensitetsskillnader mellan sammanhängande pixlar.
- ***boundingRect():*** Tar in en lista med punkter och skapar ett Rect-objekt utifrån de högsta och lägsta x- och y- koordinaterna i listan.
- ***threshold():*** Identifierar färgskillnader i en matris.
- ***matchTemplate():*** Jämför två matriser genom att dra den minsta matrisen över den största. Den bästa matchningen returneras som en procentsiffra.

Se appendix A för länkar till metodernas API.

### 3.2.3 Highgui

Highgui-modulen innehåller metoder:

- ***imread():*** Används för att läsa in bilder till matriser i valbart antal kanaler och i valbart bildformat.

- ***imwrite():*** Används för att lagrar matriser i valbart bildformat.

Se appendix A för länkar till metodernas API.

Formaten som stöds av dessa metod är:

- Windows bitmaps - *.bmp, *.dib
- JPEG files - *.jpeg, *.jpg, *.jpe
- JPEG 2000 files - *.jp2
- Portable Network Graphics - *.png
- Portable image format - *.pbm, *.pgm, *.ppm
- Sun rasters - *.sr, *.ras
- TIFF files - *.tiff, *.tif

## 3.3 Google Guava

För att köra trådar används Google biblioteket Guava. Guava innehåller en förbättrad version av Javas implementation av futures [11], det vill säga trådar som kan returnera värden. Dessa Guava-klasser implementerar interfacet *Callable<T>*. Genom att dekorera en trådpool av workers med hjälp av metoden *listeningDecorator()* får trådarna ett Listenable interface. Detta resulterar i trådar som direkt meddelar när deras task är utfört och på så sätt kan resultatet användas så snabbt som möjligt och tråden kan utföra nästa task.

## 3.4 JpegImagesToMovie

För att omvandla bilder till ett animationsklipp används den färdiga klassen JpegImagesToMovie [12]. Klassen använder sig av JMF och har metoden *makeMovie().* Metoden genererar ett animationsklipp utifrån en bildsekvens där varje bild har formatet jpg.

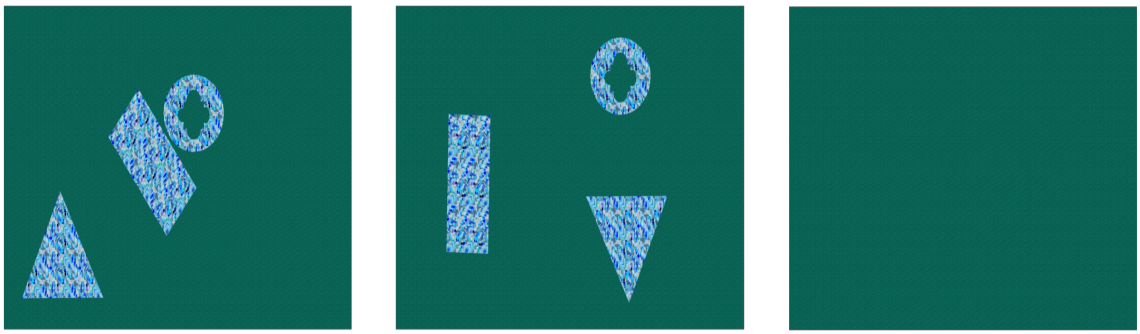
## 3.5 MVC

MVC-modellen [13] består av tre komponenter: Model, View och Controller. Strukturen bygger på att Model behandlar och lagrar data i programmet och är helt oberoende av de övriga två

komponenterna. View är den grafiska representationen av den data lagrad i model och Controller hanterar användarinteraktioner. Detta gör att Model separeras från det grafiska gränssnittet. Controller och View är även oberoende av varandra vilket resulterar i modulär kod där Controller och View fritt kan bytas utan påverka Model.

# 4. Genomförande

Verktyget ska kunna ta in ett obegränsat antal nyckelbilder (hårdvarans tillgängliga minne som gräns) samt en gemensam bakgrundsbild. Verktyget skapar sedan ett antal mellanbilder utifrån nyckelbilderna och bakgrunden. Antalet mellanbilder som ska genereras för varje delsekvens kan specificeras av användaren. Alla genererade mellanbilder tillsammans med nyckelbilderna kan sedan omvandlas till ett videoklipp (se appendix C för en komplett bildsekvens). Underkapitel 4.3 till 4.7 beskriver processen för att generera mellanbilderna till två nyckelbilder (se figur 4.1).



*a)* *b)* *c)*

*Figur 4.1. a) och b) är nyckelbilderna som används i genomförandet. c) är dess gemensamma bakgrund.*

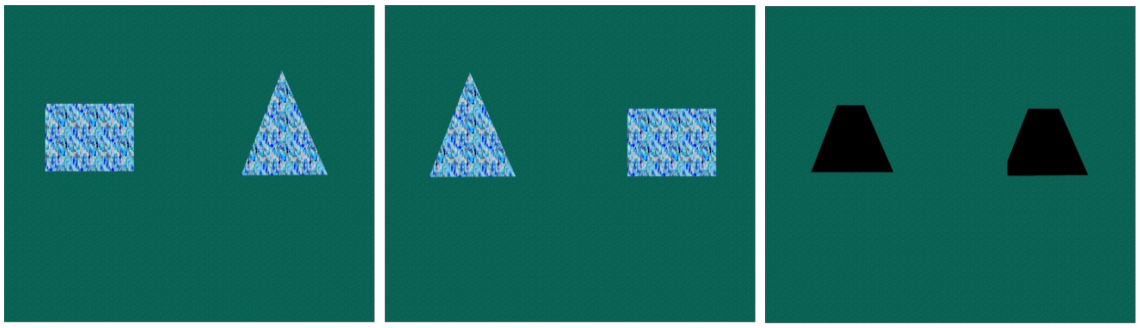
## 4.1 Trådhantering

För att optimera trådhanteringen i verktyget används en trådpool av Executors, vilket är Java's implementation av worker-strukturen. Eftersom Java är långsam på att starta upp nya trådar är det effektivt att använda sig av en trådpool [14]. Det startas lika många workers som hårdvaran som kör verktyget har kärnor. Detta gör att programmet använder nästintill 100% av datorns kapacitet under de mer krävande processerna och uppstarten av en ny task blir minimal.

## 4.2 Inhämta bilderna

För att läsa in nyckelbilderna och bakgrunden till verktyget används metoden *imread().* Då verktyget ska hantera transparenta bilder läses de in i en 8UC4-matris. I de fall då den transparenta komponenten saknas skapas en extra kanal för att göra programmet konsekvent. Dessa fall uppstår när bilder läses in i format som ej stödjer den transparenta komponenten. Den transparenta kanalen som skapas får värdet 255 i alla pixlar vilket resulterar i att bilden är helt ogenomskinlig.

Från början var det planerat att bakgrunden skulle genereras utifrån de nyckelbilder som användaren gav som indata. Dock kan mellanbilderna endast genereras om hela bakgrunden är tillgänglig, det vill säga att det inte finns några okända ytor i bakgrunden. Om föremålen tas bort från nyckelbilderna i figur 4.2 kommer nyckelbilderna att ha gemensamma okända områden (se figur 4.2.c), detta gör att det inte går att generera den kompletta bakgrunden. För att verktyget ska fungera vid alla situationer tas bakgrunden därför in som indata.



*a)* *b)* *c)*

*Figur 4.2 Om man genererar backgrunden med hjälp av nyckelbilderna a) och b) skapas de okända områdena i c).*

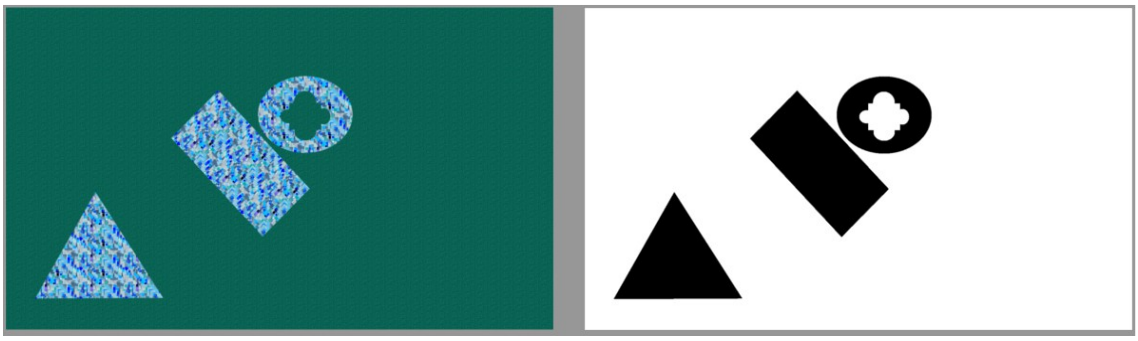
För att göra inläsningen så optimal som möjligt är klassen för inläsning gjord som en task och körs asynkront. Nyckelbilderna och bakgrunden som läses in måste även vara kompatibla, det vill säga ha samma bildstorlek och bildformat.

## 4.3 Identifiera föremålen

För att identifiera ett föremål skapas en ObjectLocation-instans. Denna instans kommer att innehålla föremålets position i nyckelbilden och föremålets form. Instansens komponenter skapas i tre steg vilket beskrivs i följande underkapitel.

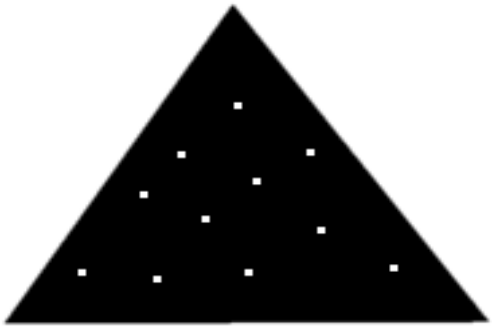
### 4.3.1 Generera silhuett

Första steget består av att generera en silhuett för varje nyckelbild. En nyckelbilds silhuett lagras som en 8UC1-matris och genereras genom att jämföra pixlar som har samma position i bakgrunden och nyckelbilden. Har de båda pixlarna samma värde betraktas det som pixeln tillhör bakgrunden och sparas då som en vit pixel i silhuetten. Har pixlarna olika värden betraktas det som att pixeln tillhör ett föremål och lagras som en svart pixel i silhuetten (se figur 4.3).



*Figur 4.3 En nyckelbild och dess silhuett.*

Eftersom ett föremål betraktas som skillnaden mellan nyckelbilden och bakgrunden uppstår ett problem när föremål i nyckelbilden och bakgrunden har samma pixelvärde. Problemet består av brus, där bruset kan vara enstaka vita pixlar (se figur 4.4). För att motverka bruset kontrolleras varje pixel i silhuetten. Om det förekommer en vit pixel omringad av svarta pixlar kommer den vita pixeln att betraktas som brus och ändras till en svart pixel.

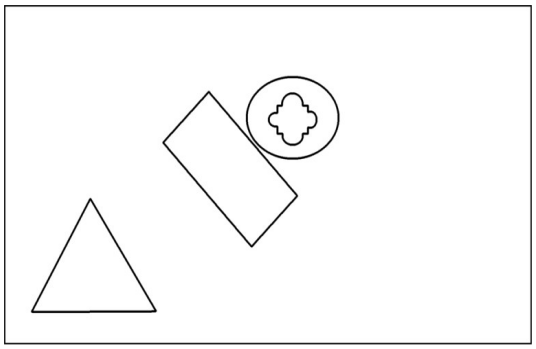


*Figur 4.4 Visar hur bruset kan se ut i en silhuett.*

### 4.3.2 Lokalisera potentiella föremål

Det andra steget i processen är att identifiera alla föremål i silhuetterna. För att identifiera ett föremål generas dess Minimum Bounding Rect (MBR) vilket görs i två steg:

Först används metoden *threshold()* vilket jämför intensiteten mellan färger i en silhuett och returnerar en 8UC1-matris där intensitetsskillnaderna är markerade med svarta pixlar. När detta görs på silhuetten i figur 4.3 skapas *intensitetsbilden* i figur 4.5.

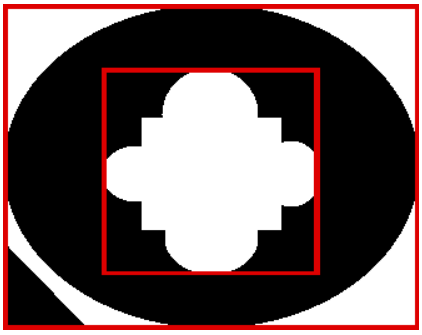


*Figur 4.5 Intensitetsskillnaderna från silhuetten på figur 4.3. Notera att ramen är markerad som en intensitetsskillnad.*

I andra steget generas alla konturer av alla föremål genom att använda metoden *findContours()* på *intensitetsbilden*. Ett Rect-objekt skapas sedan för varje kontur genom att använda metoden *boundingRect(),* detta objekt är MBR för föremålet.

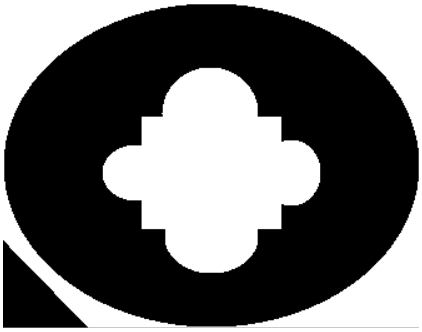
### 4.3.3 Kontroll av MBR:s

Då avgränsningarna för verktyget inte betraktar vissa MBR:s som föremål görs en kontroll av alla MBR:s som genererades. MBR:s som har en storlek som är mindre än 5 procent av nyckelbilden ignoreras. MBR:s som är större än 95 procent av nyckelbildens storlek kommer också att ignoreras eftersom *findContours()* betraktar bildramen som en kontur. MBR:s som befinner sig inuti andra MBR:s ignoreras eftersom föremål inte får befinna sig inuti andra föremål (se figur 4.6).



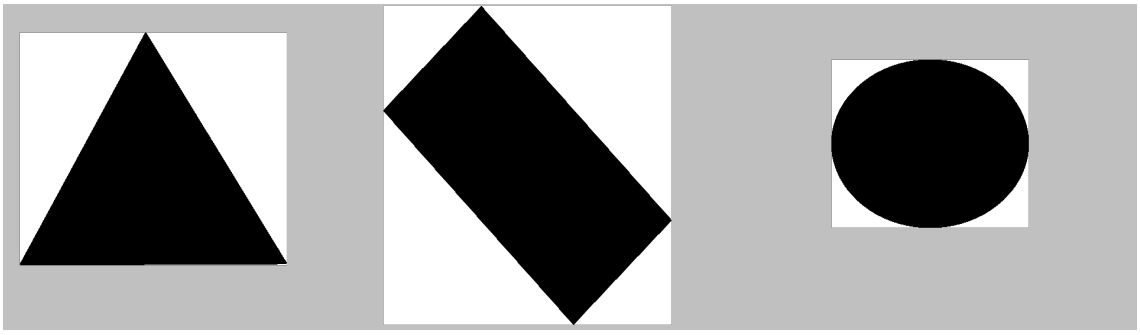
*Figur 4.6 Visar hur inre MBR:s kan betraktas som separata föremål av metoden findContours().*

När en MBR har ignorerats raderas den kontur som MBR:en genererades ifrån. När processen är klar används alla godkända MBR:s för att skapa föremålssilhuetter (FS) från silhuetten av nyckelbilden (se figur 4.7).



*Figur 4.7 FS av cirkeln från silhuetten i figur 4.3.*

Det som kan noteras ifrån figur 4.7 är att delar av andra FS:er kan förekomma i en FS. För att lösa detta problem genereras även en 8UC1-matris där varje kontur är utritad och sedan ifylld med svarta pixlar, denna matris får namnet föremålskontur (FK). FK:ens bakgrund är vit vilket resulterar i att den innehåller formen av föremålet (se figur 4.8).

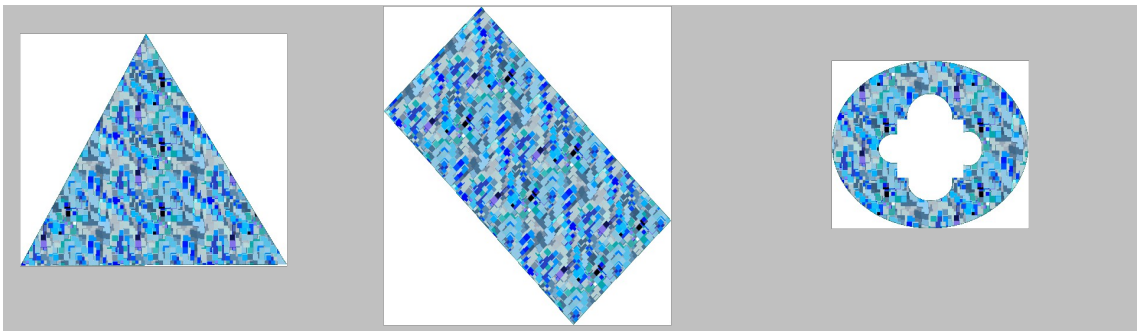


*Figur 4.8* FK:s *extraherade av silhuetten i figur 4.3.*

Ett föremåls FS, FK och MBR sparas sedan tillsammans i objektet ObjectLocation.

## 4.4 Extrahera föremål från bakgrund

Med hjälp av föremålens ObjectLocation-instans är det möjligt att skilja föremålen ifrån dess bakgrund (se figur 4.9).

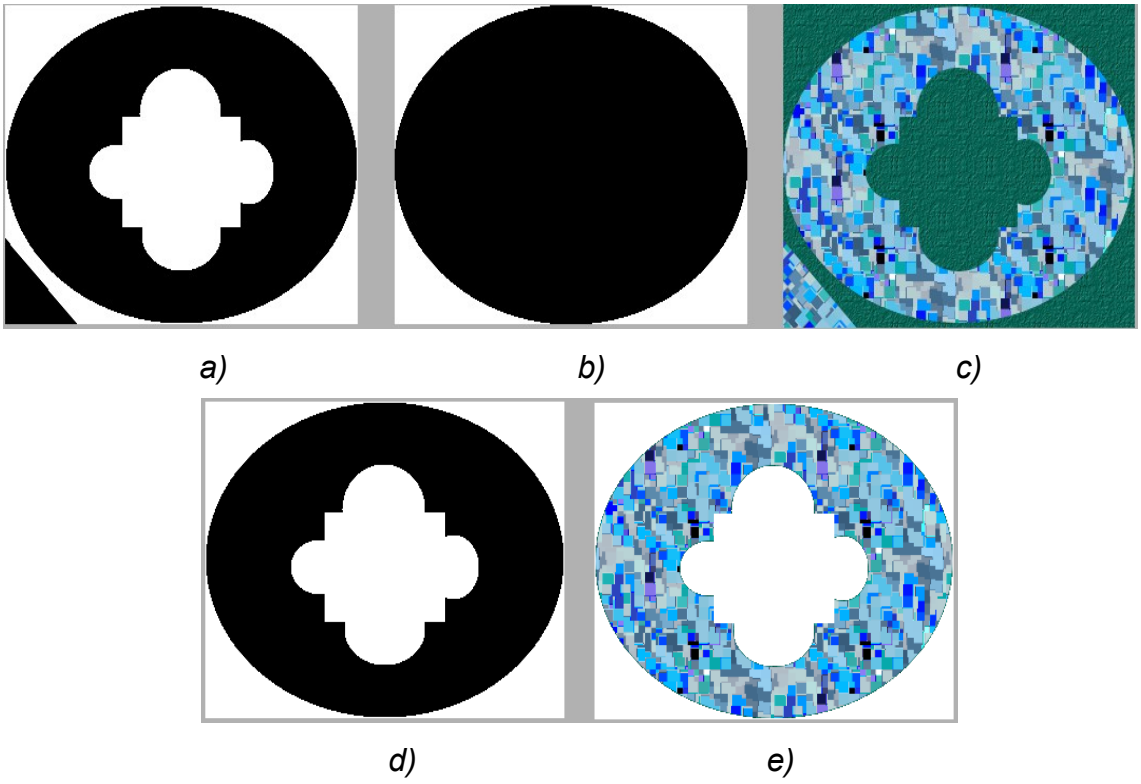


*Figur 4.9 Föremålen från nyckelbilden i figur 4.3 extraherade utan bakgrund.*

Först instansieras en 8UC4-matris med vita pixlar. Denna matris ska efter hela processen innehålla föremålet separerat från nyckelbildens bakgrund, denna namnges föremålsmatris.

Även en 8UC1-matris instansieras med vita pixlar. När processen är klar ska denna matris innehålla en mask för att markera föremålets form. Sist skapas en submatris av nyckelbilden (se figur 4.10.c) med hjälp av föremålets MBR från ObjectLocation.

När matriserna är skapade sker en jämförelse mellan FK:s och FS:s pixlar genom en AND operand. Detta medför att om båda pixelvärderna är svarta ska föremålsmatrisen innehålla pixelvärdet från motsvarande position från submatrisen medans masken får en svart pixel (se figur 4.10).




*Figur 4.10 a) är FS, b) är FK och c) är submatrisen av nyckelbilden. d) är masken som är resultatet av en AND operand av FS och FK. e) är föremålet separerat från bakgrunden.*

Samtidigt som detta sker räknas antalet pixlar som föremålet består av, detta värde betraktas som *arean* av föremålet. *Arean* kommer att användas för att jämföra föremåls likhet.

Då vissa metoder senare i programmet inte klarar 8UC4-matriser skapas även en 8UC3-matris där den transparenta delen av föremålsmatris tas bort. Denna matris namnges icke transparent föremålsmatris (ITFM).

När hela processen är klar sparas masken, ITFM, föremålsmatrisen och *arean* i ett objekt som kallas ImageObject.

Hela denna processen sker asynkront i en task för varje ObjectLocation-instans.

## 4.5 Matcha föremål

För att matcha föremål mellan nyckelbilder används resultatet som sparades i ImageObject. Matchningen sker genom att para ihop den första nyckelbildens föremål med den andra nyckelbildens föremål. Avståndet och rotationen mellan matchande föremål sparas. Matchningen av föremål sker i tre steg vilket beskrivs i följande underkapitel. För att underlätta beskrivningen i följande underkapitel namnges den första nyckelbilden till bild1 och den andra nyckelbilden till bild2. Varje föremål som verktyget ska matcha körs asynkront i en task då de är oberoende av varandra.

### 4.5.1 Hitta potentiella matchningar

Först ska alla potentiella matchningar mellan bild1 och bild2 identifieras. Detta sker genom att jämföra alla föremål i bild1 med alla föremålen i bild2. Jämförelsen mellan två föremål sker i följande steg.

Det första steget är en kontroll av de två föremålens *area*, om föremålens *area* avviker på mer än 5 procent ifrån varandra anses det att inte kunna matcha och processen avbryts.

I nästa steg jämförs de båda föremålen med metoden *matchTemplate().* Då *matchTemplate()* enbart kan hantera tre kanaler används ITFM från föremålens ImageObject. Jämförelsen av föremålen sker först utan rotation och sedan med rotation där *rotationssteget* mellan varje

jämförelse är 3 grader. Föremålet från bild1 roteras med metoden *warpAffine()* mellan 3 grader och 357 grader. Om resultatet från *matchTemplate()* är 95% eller högre sparas föremålen som en potentiell matchning tillsammans med *rotationsvärdet* och avståndet mellan föremålen. R*otationssteget* valdes så att en 95% matchning mellan två föremål kunde fås även fast rotationen mellan föremålen inte var exakt.

Ifall en matchning hittas vid ett visst *rotationssteg* avbryts inte processen eftersom en bättre matchning än 95% kan hittas vid ett annat *rotationssteg*, ifall detta sker förkastas den föregående matchningen och den nya sparas. Processen avslutas dock om en matchning mellan två föremål är större eller lika med 99,9%. Denna procentsiffra skulle kunna ändras för att optimera processen.

Notera att flera föremål i bild1 och bild2 kan matcha och då måste alla dessa föremål sparas som potentiella matchningar. Alla matchningar sparas i en lista som är sorterad med det kortaste avståndet mellan två matchade föremålen först.

### 4.5.2 Kontroll av matchningar

Då flera föremål från bild1 kan matcha till samma föremål i bild2, eller omvänt, används en algoritm som parar ihop de matchande föremålen. Algoritmen parar först ihop de matchande föremålen som har det kortaste avståndet mellan dem. När två föremål har parats ihop tas de elementen där de två föremålen förekommer bort från listan med matchningar. Denna process upprepas tills listan är tom.

Denna process görs för att användaren ska kunna förutse hur verktyget kommer att arbeta. Denna algoritm är även designad så att den är enkel att byta ut om användaren skulle vilja ha en annan funktionalitet.

### 4.5.3 Kalibrering av rotation

Ett mer exakt *rotationsvärde* beräknas när alla matchningar har bestämts. Kalibreringen sker

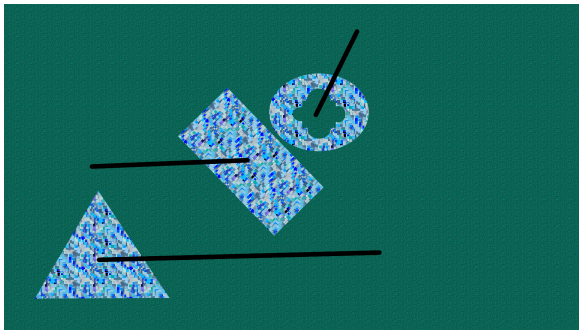
återigen med metoden *matchTemplate()* fast nu med ett *rotationssteg* på 0.5 grader. Eftersom den föregående rotationen mellan föremålen är sparad, kontrolleras enbart gradtal som ligger +-3 grader av det föregående *rotationsvärdet*. Felmarginalen kan inte vara större än det *rotationssteg* som användes när matchningen hittades. Genom att beräkna det exakta *rotationsvärdet* efter att föremålen har parats ihop minskar processtiden drastiskt i de fall då föremål har flera matchningar.

## 4.6 Bestämma föremålens rörelse

Resultatet av matchningen används för att beräkna varje position och rotation ett föremål ska ha i den delsekvens som ska genereras.

Då föremål alltid ska röra sig det kortaste avståndet blir rotationen negativ om ett föremåls *rotationsvärde* är över 180 grader. Denna beräkning görs genom att subtrahera 360 från *rotationsvärdet* som beräknades vid matchningen av föremålen. Detta medför att ett föremål kan rotera max 180 grader mellan två nyckelbilder. Rotationen som beräknas görs utifrån föremålets rotationspunkt.

Då verktyget ska generera ett bestämt antal mellanbilder för delsekvensen och alla rörelser som skall ske är beräknade, består detta steg enbart av att dividera alla givna avstånd med antalet bilder i delsekvensen. Avståndet kan sedan sekventiellt adderas med föremålets utgångspunkt. Varje delvärde av denna addition resulterar i en lista av punkter och rotationer. Denna lista refereras som *rörelsesekvensen* för ett givet föremål (se figur 4.11).



*Figur 4.11 De svarta linjerna är de olika föremålens rörelsesekvens.*

Notera att föremål i första nyckelbilden inte måste matcha till något annat föremål i andra nyckelbilden. *Rörelsesekvensen* för föremål där detta sker kommer då att bli den kortaste vägen ut ifrån dess nyckelbild med en rotation på 0 grader. Samma process sker för föremål i den andra nyckelbilden fast där betraktas det som att föremålet ska träda in i nyckelbilden.

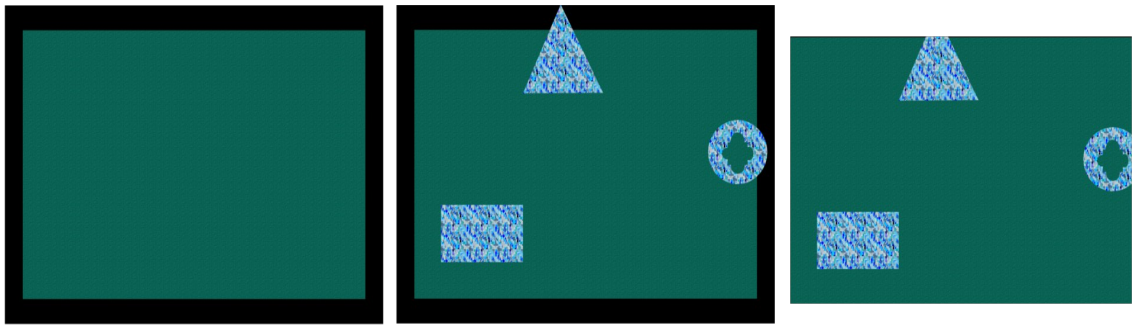
Detta steg görs asynkront i tasks för varje föremål i varje nyckelbild eftersom de är oberoende av övriga uträkningar.

## 4.7 Generera mellanbilderna

Efter att alla föremålens *rörelsesekvenser* är beräknad kan mellanbilderna genereras.

Först instansieras en 8UC4-matris med svarta pixlar. Matrisen är större än bakgrunden för att kringgå problemet som uppstår när pixlar placeras utanför en matris. Bakgrunden placeras i mitten av matrisen så att den är centrerad (se figur 4.12.a). Det svarta området i 8UC4-matrisen är proportionellt mot de största avståndet som ett föremål ligger utanför bakgrundsmatrisen. Detta leder till att de delar av föremålet istället placeras i det svarta området (se figur 4.12.b). Föremålet samt den mask som skapades roteras sedan med metoden *warpAffine()* efter *rörelsesekvensens* rotationsvärde. Sedan stegas den roterade masken igenom. Om pixeln i masken är svart sparas pixeln av den roterade föremålet på den positionen som definierats i *rörelsesekvensen*. Detta resulterar i att enbart föremålets pixlar placeras i den nya 8UC4-matrisen och ej dess vita bakgrund.

När alla föremålen i den mellanbild som genereras har gått igenom denna process är mellanbilden klar och det svarta området runt bilden kapas bort (se figur 4.12.c). Mellanbilden sparas sedan med hjälp av metoden *imwrite()*. Varje mellanbild som ska generas körs asynkront i tasks.



*a)* *b)* *c)*

*Figur 4.12 Beskriver processen av att placera föremålen på den slutgiltiga mellanbilden.*

## 4.8 Omvandla mellanbilderna till videoklipp

När alla mellanbilder har genererats är sista steget att skapa animationsklippet vilket görs med klassen JpegImagesToMovie. Metoden *makeMovie()* skapar animationsklippet genom att ta in storleken på bilderna, antalet bilder per sekund, en lista med bilderna i bildsekvensen, destination och videoformatet som klippet ska sparas i.

## 4.9 Programstruktur

För att strukturera verktyget används en anpassad version av MVC-modellen. Implementationen av denna struktur bygger på tre klasser: Model, View och Controller. Model innehåller den logiska implementationen av programmet. View bygger på att användarevent är kopplade till ActionListeners som är definierade i Controller. Detta resulterar i att Controller anropas vid användarevent i GUI:t som sedan utför metoder i Model som returnerar svar till View som kan visas för användaren.

### 4.9.1 Model

De metoder från Model som Controller kan använda är följande:

- ***createNewAnimationProcess()*:** Initierar Model genom att starta trådpoolen som används och sätter antalet bilder per sekund som programmet ska använda. Här sker även en kompatibilitetskontroll mellan nyckelbilderna och backgrunden.

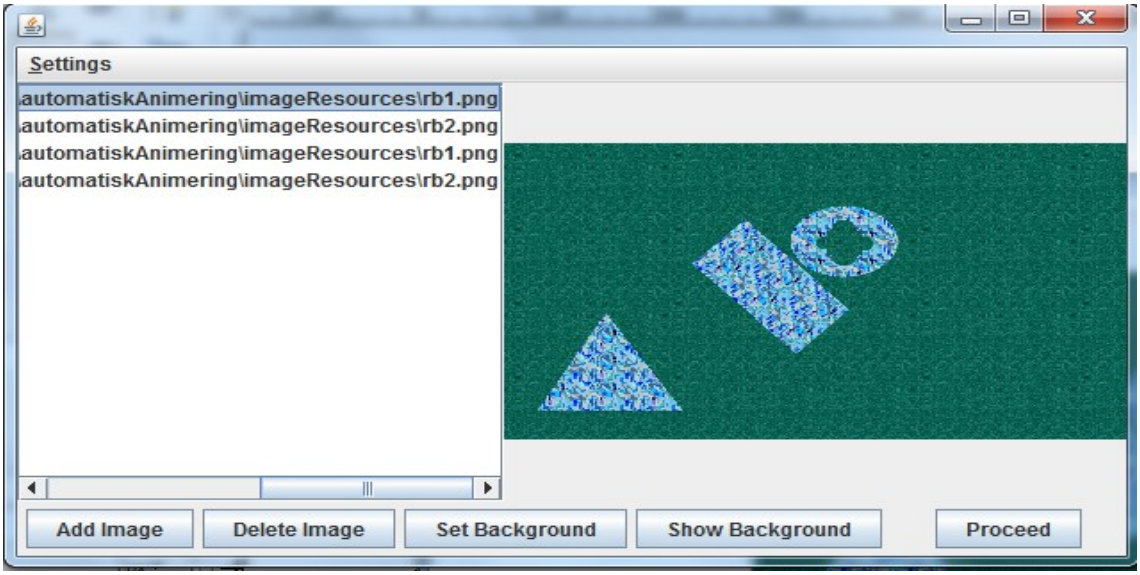
- ***getImageObjects()*:** Tar in en lista med sökvägar till nyckelbilder och returnerar en lista med nyckelbildernas ImageObjects. Alla ImageObjects sparas även i Model.
- ***generateImages()*:** Genererar mellanbilderna med hjälp av alla ImageObjects sparade i Model.
- ***generateVideo()*:** Genererar ett videoklipp utifrån alla mellanbilder och nyckelbilderna.
- ***reset()*:** Avslutar animationsprocessen genom att stänga ner trådpoolen. Metoden frigör även alla matriser som används i verktyget genom metoden *release()*.

### 4.9.2 GUI

View- och Controller- klassen är applikationens GUI. GUI:t består av ett fönster som är uppbyggt av en JFrame som kan befinna sig i två olika vyer, bildinläsningsvyn och sekvensvyn. Båda vyerna implementerar en abstrakt klass som innehåller alla gemensamma GUI-komponenter. Genom att enbart byta vy är det enkelt och modulärt att byta de innehållet som verktyget visar för användaren.

#### 4.9.2.1 Bildinläsningsvyn

Bildinläsningsvyn (se figur 4.13) är den vyn som visas när verktyget startas.




*Figur 4.13 Bildinläsningsvyn.*

Menyn högst upp till vänster med namn "Settings" innehåller en flik för att ändra inställningar. Klickar användaren på denna öppnas ett nytt fönster i vilket det går att ange antal bilder per sekund som videoklippet ska använda. Det går även bestämma ett globalt antal bilder som varje delsekvens ska generera.

Listan till vänster innehåller sökvägar till de nyckelbilder som användaren har importerat. Om användaren klickar på något av elementen i listan kommer bilden att visas i bildramen till höger. Det går även att förflytta elementen i listan genom att klicka på dem och dra till en annan position.

“Add Image”-knappen öppnar ett nytt fönster där användaren kan importera nyckelbilder till listan med sökvägar.

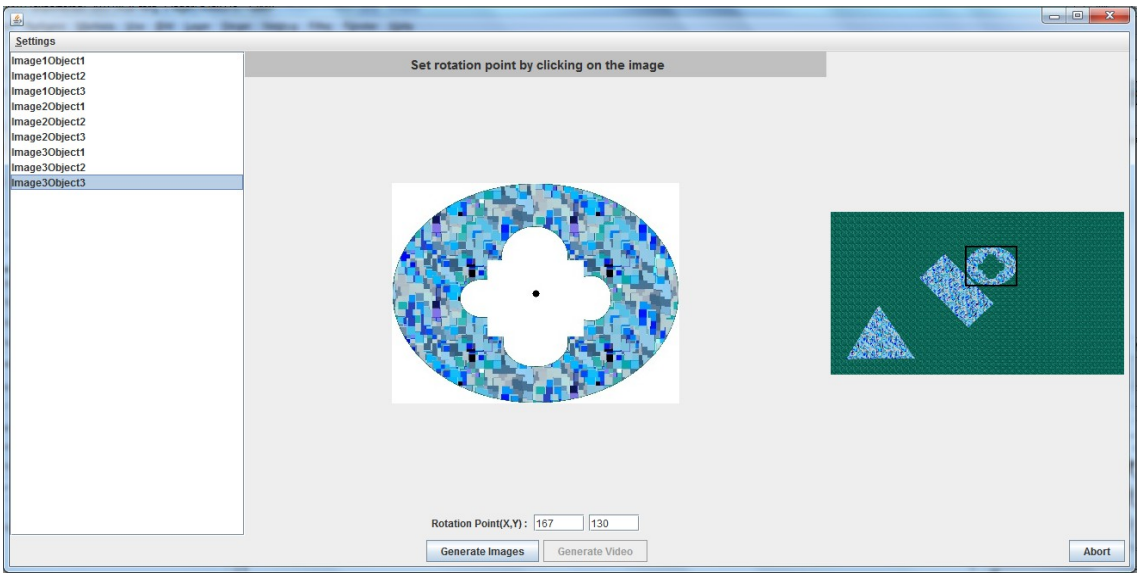
“Delete Image”-knappen tar bort det markerade elementet ifrån listan.

“Set Background”-knappen öppnar ett fönster där användaren kan importera filen med bakgrunden som verktyget ska använda sig av. Användaren kan sedan använda “Show Background”-knappen för att visa bakgrunden i bildramen.

När “Proceed”-knappen aktiveras börjar animationsprocessen genom att anropa metoden *createNewAnimationProcess()* följt av *getImageObjects()*. När alla ImageObjects har genererats byter GUI:t vy till sekvensvyn.

#### 4.9.2.2 Sekvensvyn

Sekvensvyn är den vyn som visas när alla föremål har lokaliserats i nyckelbilderna (Se figur 4.14).




*Figur 4.14 Sekvensvyn.*

Menyn vid namn ”Settings” i sekvensvyn öppnar ett fönster i vilket användaren kan ange antalet bilder som varje delsekvens ska generera.

Listan i sekvensvyn går inte att redigera. Varje element i denna lista är ett ImageObject som genererades med metoden *getImageObjects().* När ett element i listan markeras visas två nya GUI-komponenter. GUI-komponenten i mitten visar föremålet som är markerat i listan. Föremålets rotationspunkt är utritad som en svart prick. Klickar användaren på föremålet förflyttas rotationspunken till där användaren klickade. Användaren kan också manuellt skriva in rotationspunkten i de två fälten under föremålet. GUI-komponenten till höger innehåller nyckelbilden som det markerade ImageObject genererades ifrån. I nyckelbilden är även föremålet markerat med en svart MBR för att förtydliga vilket föremål som behandlas.

”Generate Images”-knappen kallar på metoden *generateImages()* för att generera mellanbilderna.

“Generate Video”-knappen skapar videoklippet genom att anropa metoden *generateVideo()*.

“Abort”-knappen nollställer allt i verktyget genom att kalla på metoden *reset()* och byter vy till bildinläsningsvyn.

# 5. Resultat

Ett verktyg har utvecklas som kan underlätta processen för att skapa 2d-animeringar. Verktyget tar in en sekvens av nyckelbilder som kan användas för att generera ett antal mellanbilder. Från nyckelbilderna tillsammans de genererade mellanbilder kan verktyget skapa ett 2d-animationsklispp.

Resultatet av denna process följer de preciserade målen som definierades för projektet samt de avgränsningar som sattes på verktyget (se appendix C för slutresultatet av en bildsekvens).

För att göra verktyget användarvänligt har ett GUI implementerats. Detta GUI gör att användaren kan sätta de inställningar som krävs för att skapa animationen. Inställningarna involverar:

- Specificera antal bilder per sekund för det animerade klippet.
- Specificera antal bilder för varje delsekvens.
- Specificera varje föremåls rotationspunkt.

GUI:t har även en viss input-kontroll där bildernas kompatibilitet i from av bildstorlek samt bildformat kontrolleras.

Verktyget kan i slutändan betraktas som en fungerande prototyp som kan vidareutvecklas för att öka dess användningsområden.

# 6. Slutsats

## 6.1 Kritisk diskussion

Målet med projektet var att implementera ett verktyg som kan hjälpa animerare att skapa animationer. Användningsområdet för verktyget är dock diskutabelt. Verktyget kan endast skapa 2d-animationer där föremålen alltid har konstant form och storlek. Föremålen kan endast förflytta sig i planet samt rotera. Eftersom bakgrunden i nyckelbilderna även måste vara konstant kan animationen ej byta scen. Dessa begränsningar sätter en gräns på användningsområdena för verktyget.

En potentiell användning av verktyget är att animera rörelser av föremål som armar och ben där kroppen används som bakgrund. Animering av denna typ skulle kunna användas inom spelutveckling för 2d-animering av sprites.

Under utvecklingen har stor tyngd lagts på optimering av verktygets resurshantering. Detta har medfört en effektiv minneshantering samt låg processtid. Dessa optimeringar tillsammans med effektiv trådhantering har dragit ner processtiden för verktyget. Processtiden för att generera mellanbilderna var ytterst viktig, målet med verktyget var att med acceptabel tid generera extra mellanbilder till de existerande nyckelbilder. Hur mellanbilderna genereras är dock delvis ineffektiv. I den nuvarande implementeringen skapas en större bild som sedan bakgrunden placeras i. Detta är en suboptimal lösning då den svarta ytan är proportionell mot hur mycket föremålet placerades utanför mellanbilden. Detta leder till att genereringen av bilderna blir mer minneskrävande.

Det ursprungliga tidsplaneringen (se i appendix B) skiljer sig från den verkliga tidsuppdelningen som skedde. I slutändan blev det en mer kontinuerlig och iterativ process som var uppdelad i två steg. Först implementationen av logiken sedan implementationen av GUI:t. Alla avvikelser i arbetet kan sammanfattas som brist av tidigare erfarenhet inom området.

## 6.3. Miljö aspekter

En dators strömförbrukning är värd att notera. Jämför man denna applikation som kan skapa tusentals bilder på några minuter med en animerare som sitter och gör dessa bilder för hand ser man en enorm tidsskillnad och därigenom påverkas den totala strömförbrukning som krävs för att skapa animationen.

## 6.4 Vidareutveckling

En funktionalitet som kan implementeras till verktyget är föremål som kan ha leder.
Med leder menas att föremål ska kunna böja sig där leder är utsatta. Detta skulle resulterat i mer naturliga rörelser av till exempel armar, där en led skulle kunna placeras vid armbågen. Detta skulle kunna ske på liknande sätt som användaren sätter ut rotationspunkter.

En annan potentiell funktionalitet är att verktyget ska kunna hantera föremål som befinner sig inuti andra föremål. Med detta menas att till exempel en fyrkant skulle kunna ha en cirkel i sig som kan röra sig individuellt inom fyrkanten samtidigt som fyrkanten rör sig.

GUI:t kan utvecklas både i dess funktionalitet samt design. I nuvarande GUI kan man ej specificera vart mellanbilder och videoklipp lagras eller dess filformat utan detta är hårdkodat.

## 6.5 Reflektion

I den nuvarande applikationen ligger stort fokus på att para ihop föremål mellan nyckelbilder. Om projektet skulle gjorts från början med vår nuvarande kunskap hade fokusen istället lagts på att låta användaren specificera vilka föremål som matchar. Detta hade lett till att verktyget blivit väldigt annorlunda där en intressant aspekt av att manipulera föremål hade blivit i fokus. Detta hade medfört att föremål inte behöver vara exakt lika varandra. Istället hade verktyget försökt manipulera föremålen för att matcha de föremål som användaren redan definierat som lika.

# 7. Referenser

[1] Java's websida (3 juni 2014) [online] Available:

http://www.java.com

[2] Eclipse's informationssida (3 juni 2014) [online] Available:

http://www.eclipse.org/eclipse/

[3] OpenCV's websida (3 juni 2014) [online] Available:
http://opencv.org/

[4] Google Guava websida (20 maj 2014) [online] Available:
https://code.google.com/p/guava-libraries/

[5] Git (17 juni 2014) [online] Available:
http://sv.wikipedia.org/wiki/Git

[6] Java swing resurssamling (14 maj 2014) [online] Available:
http://docs.oracle.com/javase/7/docs/technotes/guides/swing/index.html

[7] JMF informationssida (3 juni 2014) [online] Available:
http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html

[8] Memory management and reference counting (3 juni 2014) [online] Available:
http://docs.opencv.org/doc/user_guide/ug_mat.html

[9] RGB Color Codes Chart (3 maj 2014) [online] Available:
http://www.rapidtables.com/web/color/RGB_Color.htm

[10] OpenCVs moduler (6 juni 2014) [online] Available:

http://docs.opencv.org/modules/core/doc/intro.html

[11] ListenableFutureExplained (3 juni 2014) [online] Available:
http://code.google.com/p/guava-libraries/wiki/ListenableFutureExplained

[12] JpegImagesToMovie (18 juni 2014) [online] Available:
https://code.google.com/p/jpegs2movie/source/browse/JPEGs2MOVie/src/movie/maker/vilius/kraujutis/lt/JpegImagesToMovie.java?r=11

[13] Fred Swartz. (24 maj 2014) Model-View-Controller (MVC) Structure [online] Available:
http://leepoint.net/notes-java/GUI/structure/40mvc.html

[14] Java trådspolar (18 juni 2014) [online] Available:
http://docs.oracle.com/javase/tutorial/essential/concurrency/pools.html

# Appendix A JavaDoc:

**imread():**

http://docs.opencv.org/java/org/opencv/highgui/Highgui.html#imread%28java.lang.String,%20int%29

**imwrite():**

http://docs.opencv.org/java/org/opencv/highgui/Highgui.html#imwrite%28java.lang.String,%20org.opencv.core.Mat%29

**findContours():**

http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours

**warpAffine():**

http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html?highlight=warpaffine#warpaffine

**threshhold():**

http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#threshold

**matchTemplate():**

http://docs.opencv.org/modules/imgproc/doc/object_detection.html#matchtemplate

**boundingRect():**

http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=boundingrect#boundingrect

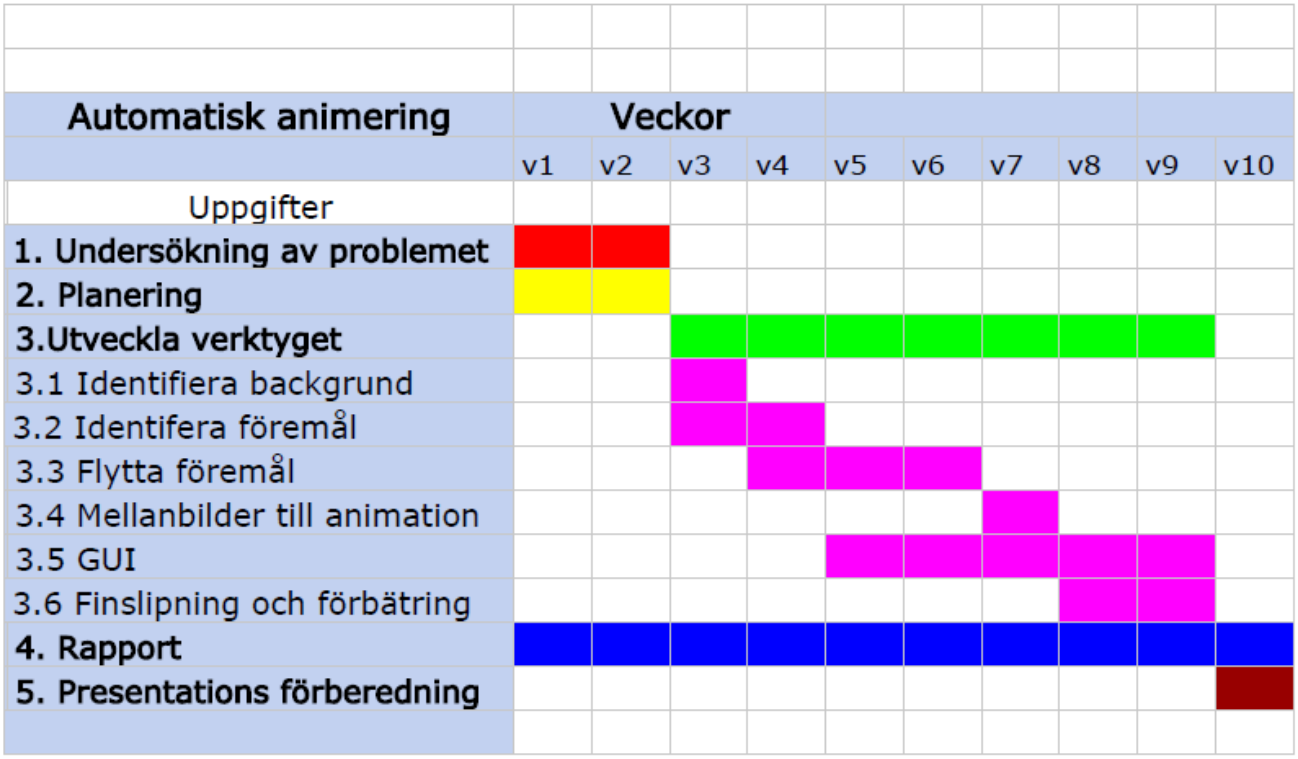
**get():**

http://docs.opencv.org/java/2.4.2/org/opencv/core/Mat.html#get%28int,%20int%29

**put():**

http://docs.opencv.org/java/2.4.2/org/opencv/core/Mat.html#put%28int,%20int,%20double...%29

# Appendix B Tidsplan:



## Appendix C Genererad bildsekvens:

Varje rad med bilder är en delsekvens, där första bilden i varje sekvens är en nyckelbild. Resterande i sekvensen är mellanbilder som applikationen har genererat. Notera att sekvenserna har ett varierande antal bilder.