Flash2HTML5 converter

xfl to HTML5-canvas objects

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg

Datateknik

Examensarbete:

Zoran Pribojevic

LTH Ingenjörshögskolan vid Campus Helsingborg

Lunds universitet

Box 882

251 08 Helsingborg

LTH School of Engineering

Lund University

Box 882

SE-251 08 Helsingborg

Sweden

Tryckt i Sverige

Media-Tryck

Biblioteksdirektionen

Lunds universitet

Lund 2012

Sammanfattning

En implementation har skapats för att konvertera ett av Adobe Flash filformat till HTML5. De två viktigaste delproblemen var att välja ett lämpligt programmeringsspråk och att hitta dokumentation av xfl-formatet.

Javascript valdes för implementationen. Brist på dokumentation, tidsbrist och den enorma omfattningen av arbetet var tre skäl till att produkten inte kunde färdigställas och att alla valda bilder inte ritades ut på ett korrekt sätt. De flesta bilder som testades blev dock korrekt konverterade.

Adobes program Flash tappar mer och mer mark till HTML5. Apple tillåter inte Flash på sina mest populära produkter och var även en av de drivande krafterna bakom HTML5. En produkt som implementerar en konvertering av Flash format till HTML5 kommer sannolikt att behövas i framtiden. Eftersom det redan finns flera implementationer som konverterar äldre versioner av Flash valdes det i detta arbete att konvertera Flash nyaste format xfl. En bild i xfl-formatet ritas i Flash CS5 och skall sedan ritas i en webbläsare med HTML5-canvasobjekt.

Rapporten innehåller information om xfl:s uppbyggnad, olika noder som behöver tolkas för konvertering samt hur koordinathanteringen hos xfl fungerar. Rapporten informerar om hur en framtida produkt som konverterar xfl-format skall implementeras

Nyckelord:

HTML5, flash, xfl, Javascript, canvas, XML, converter, Adobe, Apple, W3C

Abstract

An implementation to convert one of Adobes Flash file-format (xfl) to HTML5 has been created. The problems that need to be solved for completion of this converter are. What programming language should be used that is suitable for the application and locating some form of documentation of the file-format xfl.

Javascript was chosen for implementation, but a usable documentation of xfl was not to be found. The lack of documentation, the lack of time and the huge amount of code to be processed were three of the main reasons for why the product could not be finished and all pictures tested not correctly converted.

Adobes application Flash is losing more and more ground to HTML5. Apple does not allow Flash on their most popular products and were one of the most persuasive parts towards the creation and development of HTML5. A product that implements a conversion of Flash formats to HTML5 is likely to be needed in the near future. Because it already exist a few products that converts Flash older file-formats it was chosen in this thesis to convert the newest Flash file-format xfl. A picture of the xfl-format is drawn in Flash CS5 and is later drawn on a website with HTML5-canvas objects.

This thesis contains information about the how xfl operates, different nodes that need interpret for conversion and how xfl manages its coordinate system. This thesis could give valuable information of how a future product which converts xfl file-format should be implemented.

Keywords: HTML5, flash, xfl, Javascript, canvas, XML, converter, Adobe, Apple, W3C

Förord

Arbetet har utförts på uppdrag av Jadestone och har även utförts på Jadestones arbetsplats där ett skrivbord, dator och de anställdas expertis varit tillgänglig.

Jadestone har varit en trevlig arbetsplats med många kunniga, hjälpsamma och roliga anställda. Jag har trivts bra hos Jadestone och känner att jag tagit stor lärdom av att jobba på arbetsplatsen.

Jag vill tacka Jadestone för att jag fått arbeta hos dem och speciellt tacka min handledare Markus Högberg för dennes råd och tips vid implementering av konverteraren.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

I detta examensarbete implementeras ett verktyg som konverterar Flash-filer till HTML5. Flash har under en lång tid varit den mest användbara produkten för visning av bilder och animationer på internet. HTML5 är den senaste HTML-versionen och kan användas för att visa animationer och bilder på nätet. Flera stora företag har gått från att använda sig av Flash till att använda HTML5.

1.1.1 Behov av produkten

På grund av att HTML5 är den nya tekniken som kan komma att ta över efter Flash så är det bra att ha en applikation som snabbt och effektivt kan konvertera Flash-filer till HTML5. Det finns väldigt få verktyg att använda sig av för att konvertera vektorgrafik till något Javascript-vänligt format.

Youtube har börjat testa att visa videoklipp med HTML5 istället för Adobe Flash. Man kan välja att använda testversionen av HTML5 men den är fortfarande under utvecklingsstadiet och är inte fullt funktionell än och är således ännu begränsad.

Apples produkter tillåter inte Flash på sina iPhones, iPods och iPads och varför detta är så förklarar Apples numera avlidne grundare Steve Jobs i ett brev på Apples hemsida [2]. I brevet framställs HTML5 som ett på flera punkter bättre alternativ till Flash.

Google släppte nyligen en betaversion av sin webbläsare Chrome för Android som är googles operativsystem för mobiler och surfplattor. Det saknas stöd för Flash i denna applikation.

1.1.2 Liknande arbeten

Det finns några applikationer som konverterar olika Flash-format till HTML men inga av dessa konverterar xfl till HTML5 med Javascript.

Window7 Flash to HTML5 converter släpptes 2011-08-04 och konverterar ett av Flash filformat (SWF) till flera filformat inklusive HTML5-animationer [15]. Det är oklart exakt hur detta funkar men den är inte likadan som mitt arbete.

Google har den 28 juni 2011 släppt Swiffy som är ett verktyg som konverterar Flash SWF filer till HTML5 [18]. Swiffy är fortfarande inte helt färdig och en mängd arbete återstår. Den konverterar i nuläget inte xfl-format.

1.1.3 Om Jadestone

Jadestone grundades 2002 och är ett företag beläget i Stockholm som specialiserar sig på att utveckla online-spel. De har ett fyrtiotal anställda. Deras spel spelas av människor i flera Europeiska länder. Jadestone samarbetar med flera kända företag som bwin, Betfair, Nokia, Aftonbladet, Unibet och många fler.

Arbetet är utfört på uppdrag av Jadestone AB och arbetet har även utförts på Jadestones arbetsplats.

1.2 Syfte

Syftet med detta examensarbete är att skapa ett verktyg för att konvertera Adobe Flash filformat (xfl) till en kombination av HTML5-canvasobjekt och Javascript. Som beskrivet i kapitel 1.1 har fler stora företag börjat använda sig av HTML5 till istället för Flash. HTML5 kan som tidigare nämnts komma att ta över och göra Flash överflödigt.

1.3 Problemformulering

Huvudproblemet i detta arbete är att konvertera en bild ritad i Flash till att ritas i en webbläsare med HTML5-canvasobjekt.

Följande delproblem behöver lösas för att kunna lösa huvudproblemet.

- Vilket programmeringsspråk som bör användas för att implementera konverteraren. Jadestone hade två alternativ, Java och Javascript. Ett av

dessa bör väljas beroende på vilket av dem som är mest effektiv och passar in på arbetet som skall utföras.

- Finns det dokumentation över xfl som beskriver hur koden är uppbyggd eller vilka funktioner som används när objekt ritas i Flash xfl-format? En dokumentation över xfl:s olika funktioner och uppbyggnad är nödvändig för effektiv implementering av en konverterare.

1.4 Avgränsningar

I arbetet ingick inte att konvertera animationer från xfl till HTML5. Bilderna som skulle konverteras var av enklare slag.

Att konvertera xfl-filer till HTML5 är en enorm uppgift då en xfl-fil innehåller en stor mängd noder som skall bearbetas av implementationen.

2 Teknisk bakgrund

2.1 World Wide Web Consortium (W3C)

W3C är ett internationellt samarbete för att utveckla webbstandarder för hur webbaserat innehåll skall skapas och tolkas[19]. Bland de 357 registrerade medlemmarna finns Apple, Microsoft, Ericsson och de flesta stora företag i branschen.

2.2 HTML5

HTML

HTML står för Hypertext Markup Language och är ett märkspråk som används för att publicera webbsidor.

HTML5

HTML5 är en framtida officiell webbstandard från World Wide Web Consortium (W3C). År 2004 började man arbeta med att utveckla HTML5 och de flesta utvecklarna var från Apple, Mozilla och Opera[6].

HTML5 är den senaste versionen av HTML och är en kommande standard som ger en möjlighet att visa ljud, bild och rörliga animationer utan externa program som måste laddas ner som exempelvis Flash. HTML5 utvecklas fortfarande och är ännu inte en färdig produkt.

HTML5 Canvas

Canvas är en yta som skapas med HTML5. När den skapas anges värden för längd och bredd. Inuti denna yta kan sedan bilder eller animationer visas samt ritas. HTML5 klassen `getContext` innehåller funktioner för att rita i canvasytan.

2.3 Adobe Flash

Ett datorprogram för att att skapa animationer, spel och bilder i två dimensioner[20]. Adobe Flash Player är ett insticksprogram som behövs för att visa Flash grafiken i en webbläsare. De flesta rörliga bilder man ser när man surfar på en hemsida använder sig av Flash applikationer. Actionscript är skriptspråket som används vid programmering inom Flash och är baserat på Ecmascript.

2.3.1 xfl

Abobe Flash:s senaste filformat som lagrar data i XML-format. Föregångaren till xfl är fla och används fortfarande. Formatet som syns i olika webbläsare är swf och är en komprimerad version av fla. Det är i fla-filerna data och kod finns tillgängligt.

2.4 Javascript

Javascript skapades av Netscape och introducerades 1995. Brendan Eich designade skriptspråket som är influerat av C, Java och Python med flera och det har influerat ActionScript, CoffeScript, JScript för att nämna några.

Javascript fungerar i alla de stora webbläsarna som Internet Explorer, Firefox, Chrome, Opera och Safari.

Javascript är ett skriptspråk (interpreterande) som är prototyp-baserat och dynamiskt typat. Med dynamiskt typat menas att man i förväg inte bestämmer vilken typ en variabel är. Javascript är det mest vanliga skriptspråket på internet och fungerar med alla stora webbläsare. Koden skrivs inuti HTML-koden och kompileras inte, koden exekveras direkt i webbläsaren[7].

Java är ett programmeringsspråk som kan skapa program och Javascript är ett skriptspråk som kan användas i HTML-kod. En annan viktig detalj är att Java använder sig av klasser medan det i Javascript inte finns klasser, istället skapas klassliknande funktioner som används som en klass i Java (kap 4.2.1).

Kodexempel

```
<html>
<body >
<script type="text/javascript">

   document.write("Hello, World!");

</script>
</body>
</html>
```

Vad gör Javascript

- Javascript ger HTML-designers ett programmeringsverktyg
- Javascript kan exekveras vid en händelse som inträffar på en webbsida
- Javascript kan läsa och ändra innehåll i ett HTML element
- Javascript använder sig liksom HTML av taggar

2.4.1 jQuery

jQuery är ett Javascript-bibliotek innehållande många väldigt bra funktioner man kan använda sig av när man skall skriva Javascript. Bibliotekets funktioner förenklar användandet av Javascript.

2.4.2 EaselJS

EaselJS är ett Javascript-bibliotek som förenklar arbete med HTML5-canvasobjekt. Den innehåller en mängd nyttiga funktioner som kan användas för att rita upp bilder och objekt i en canvasyta.

2.5 XML (Extensible Markup Language)

XML är utvecklat av W3C och har varit en rekommendation sedan den 10 februari 1998. XML är en universell standard som skapades för att överföra, lagra och strukturera alla former av data. XML är alltså inte designad för att visa data utan för att lagra data. Formatet xfl lagras med XML.

Lagring

XML är ett verktyg för att lagra data som är helt oberoende av mjuk- och hårdvara vilket gör det enklare att använda då lagrad data kan delas av olika applikationsformer.

Överföring

XML är det vanligaste verktyget för överföring av data mellan alla typer av applikationer[8]. Att sända data över icke kompatibla system på internet är väldigt tidskrävande om man inte använder XML.

XML och HTML

HTML är skapad för att publicera (visa upp) data medan XML är skapad för att lagra data. XML skall inte ersätta HTML utan komplettera HTML. XML liknar HTML och använder taggar.

Exempel på XML-kod

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

3 Metod

3.1 Arbetsmetod

3.1.1 Iterativ metod

Iterativ metod används för att under processens gång testa olika bildobjekt och skriva koden anpassat efter testerna.

Motivering av metodval

Arbetet är uppbyggt på ett sådant sätt att man testar sig fram vid konvertering av olika bildobjekt. Detta medför att koden som skrivs hela tiden testas och ändras tills den slutligen fungerar som tänkt. Om en bild ritad i Flash blir konverterad och ritad i webbläsaren och de båda bilderna ser likadana ut kan nästa bildobjekt testas för konvertering, dock utan att utgå från att koden helt och hållet fungerar som planerat. Ytterligare ändring i koden kan behövas under vidare testning av andra bildobjekt.

3.2 Informationsinhämtning och inlärning

3.2.1 Javascript

Informationskällor

Information om Javascript hämtades från en tutorial [7]. En annan sida där information inhämtades från var coffeescript.org [12]. En erkänd Javascript-expert vid namn Douglas Crockford föreläser om Javascript och dessa föreläsningar [14] finns på Youtube.

Inlärning

Eftersom informationen om bilder i xfl-formatet finns i XML-filer är det viktigt att kunna tolka XML-filer.

De olika klasserna JS String och JS Array används ofta i arbetet, det är därför viktigt att känna till vilka funktioner som finns i dessa klasser och hur de används. Hur HTML5-kod integreras med Javascript är en annan viktig sak att kunna samt hur man skapar och använder egentillverkade klasser och funktioner.

jQuery

Vilka funktioner det finns i JQuery och var man kan hitta dokumentation om dem är viktigt att veta [10].

EaselJS

Vilka funktioner som finns i Eease1JS och var man hittar dokumentation om dem är viktigt att veta. Dessutom måste man känna till de klasser som används för att rita bilder på en canvasyta [11].

3.2.2 Xfl

Informationskällor

Information om xfl var väldigt svårt att finna då det inte finns någon dokumentation eller api om hur xfl fungerar. Det lilla som hittades var från olika forum som stackoverflows forum [5] samt även Adobes hjälpsida [13] som beskriver matriser.

Inlärning

Det mesta av det som behövdes som grund för arbetet med konvertering av xfl hittades inte. Det finns inte någon tillgänglig dokumentation eller api över funktioner som finns tillgängliga inom xfl. Inte heller någon beskrivning av xfl som är användbar för arbetet finns att tillgå. På de få forum som finns, kan man få information om hur koordinatfunktionerna ser ut, vilka kommandon som används för att flytta markören utan att rita, rita linjer samt hur bezierkurvor ritas.

Den största erfarenheten om konvertering av xfl-formatet erhölls genom tester av olika bilder och studier av XML-filen som beskrev dessa bilder.

3.2.3 HTML5 (canvas)

Informationskällor

Hos w3schools finns en bra och lärorik tutorial om HTML5 [6] även EaselJS innehåller bra information som behövs för arbete med HTML5-canvas [11].

Inlärning

HTML5 använder sig av olika taggar och attribut vid implementering. Referenser samt förklaringar till dessa finns tillgängliga på en tutorial i w3schools webbsida. Det finns även möjlighet att pröva på att skriva egen HTML-kod.

De bilder som skall konverteras ritas ut i en canvasyta, därmed bör kunskap om arbete med canvasobjekt samt de olika ritfunktionerna ur canvasklassen `getContext2d` inhämtas. Det är även nödvändigt att kunna skapa en canvasrita och att kunna rita på den.

4 Analys

4.1 Källkritik

All information som använts vid kodning har inhämtats ur de olika referenserna och testats i samband med att koden skrivits och bildobjekt ritats. Alla funktioner som använts har således testats och fungerar som de påstås fungera. Referenserna nedan finns i kapitel 10

[1] Adobe pausar utveckling för mobiltelefoner

Reuters är en stor engelsk nyhetsbyrå

[2] Steve Jobs tankar om Flash

Steve Jobs egna ord på Apples hemsida. Detta är en väldigt trovärdig referens.

[3] Kalla kriget mellan Adobe och Apple

CNN är en internationellt erkänd nyhetsbyrå som anses vara trovärdig inom nyhetsbranschen

[4] Adobe anklagar Apple

IDG är en svensk tidning som ger ut IT-relaterad information. I denna artikel som hänvisas till har en intervju med en av Adobes chefer gjorts.

[5] Stackoverflow om xfl koordinathantering

Detta forum är används mycket inom databranschen. All information hämtad härifrån har testats och fungerar.

[6], [7], [8] w3schools om HTML5, Javascript och XML

w3schools är en känd och inom databranschen vida använd webbsida. All information som hämtats har även testats och fungerar

[9] Författarens tråd om GradientEntry

Detta är min tråd jag skapade på ”stackoverflow ”där jag sökte hjälp om GradientEntry. Att enbart en person svarade visar hur lite kunskap det finns om xfl-formatet.

[10] Dokumentation om jQuery

Ett av de mest använda Javascriptbiblioteken. Alla funktioner som använts har testats och fungerar som utlovat.

[11] EaselJS dokumentation

Alla funktioner som använts har testats och fungerar som utlovat.

[12] CoffeScript

De exempel som använts har testats och fungerar som utlovat.

[13] Adobe om matrix till Actionscript

Adobes webbsida. De har utvecklat Flash produkter och bör veta hur de fungerar.

[14] Doug Crockfords föreläsningar om Javascript

En erkänd Javascriptexpert som föreläser om Javascript. All information som använts har testats och fungerar som utlovat.

[15] Windows7 Flash to HTML5 converter 6.4

Microsoft beskriver sin egen produkt.

[16] Flash CS6

Adobe beskriver sin produkt Flash CS6.

[17] Bezierkurva, Dr Thomas Sederberg

Dr Thomas Sederberg är en professor i Datavetenskap som förklarar bezierkurvor. Dr Sederberg har vunnit pris för sin expertis inom datorgrafik.

[18] Google Swiffy

Googles beskrivning av sin egna produkt.

[19] W3C

W3C:s hemsida anses vara trovärdig.

[20] Adobe Flash

Tillverkaren Adobes egen beskrivning av sin produkt Flash.

4.2 Analys av informationen som inhämtats i kapitel 3 (metod).

4.2.1 Javascript

XML Parser

XML Parser är en funktion som konverterar ett XML-dokument till ett XML-DOM objekt som kan användas med Javascript. Eftersom xfl-formatet lagras i en XML-fil behövs en funktion för analysering (parsing) av XML. Funktionen läser innehållet ur en XML-fil.

```
if (window.XMLHttpRequest)
{
        xhttp=new XMLHttpRequest();
}
else // for IE 5/6
{
        xhttp=new ActiveXObject("Microsoft.XMLHTTP");
}

xhttp.open("GET","DOMDocument.xml",false);
xhttp.send();
```

Lagra XML-filens innehåll med jQuery

All information ur XML-filen lagras som en sträng hos en variabel med hjälp av jQuery.

```
var xml = $(xhttp.responseXML);
```

Array

En array i Javascript fungerar ungefär som en vektor i Java. En skillnad är att det i förväg inte behöver bestämmas vad för sorts typ den array som skapas skall innehålla. Eftersom Javascript är ett svagt typat språk kan värden av olika typer lagras i en array, det går även att ändra typ av värde som lagras i array efter att den skapats. Det finns en mängd bra funktioner att använda sig av i array-klassen som `push()` och `shift()` för att nämna några av de mest flitigt använda i detta arbete. Några bra funktioner för hantering av array hämtas ur hjälpbiblioteket jQuerys klasser.

```
this.domShapeArray = new Array();
```

String

Klassen string används för att skapa en sträng och har många funktioner som kommer till användning. Funktionerna `substring()`, `charAt()` och `slice()` är några av de som används mest i arbetet.

```
currentAction = edgesCoordinatesString.charAt(0);
```

Globala anropet ”this.”

En global array, global sträng eller annat globalt objekt skapas genom att skriva ”this.” framför namnet på det objekt som skapats. Varje gång objektet refereras till skall ”this.” skrivas framför namnet på objektet. Det finns flera sätt att i Javascript deklarera globala variabler, i detta arbete har ”this.”-varianten använts.

```
this.strokeColor = strokeColor;
```

Skapa klasser

Det går i Javascript inte att skapa klasser på samma sätt som i Java. Javascript är inte uppbyggt på detta sätt. Istället skapas en funktion som döps till ett variabelnamn och i denna funktion skapas fler funktioner som används som klass-metoder används i Java. På så vis konstrueras stora funktioner med mindre funktioner inuti den stora funktionen och kan sedan användas som en klass. I rapporten kommer dessa klassliknande funktioner hädanefter att kallas för klasser fast de i själva verket inte är riktiga klasser.

Koden nedan motsvarar en klassdeklaration i Java

```
StrokeStyle = (function(){

     function StrokeStyle(){
        //tom konstruktor
     }
    StrokeStyle.prototype.setStrokeColor =
function(strokeColor){

           this.strokeColor = strokeColor;
    };
       return StrokeStyle;
}());
```

StrokeStyle är namnet på den klass som implementerats och den tomma funktionen StrokeStyle är konstruktorn som i fallet ovan är tom. När nya funktioner ska läggas till i klasserna måste namnet på klassen skrivas följt av prototype och sedan namnet på funktionen som i fallet ovan är setStrokeColor. I slutet av klassen skall allt som lagrats i klassen returneras för att kunna användas i andra klasser. Detta är StrokeStyle-klassen som implementerats och den har i detta fall kortats ner för att användas som exempel på hur klassliknande funktioner kan implementeras i Javascript.

Hjälpbiblioteket jQuery

De olika jQuery-funktionerna som används finns på jQuerys hemsida [10] där det även finns exempel på hur de skall användas samt dokumentation om dem.

Parsing av XML med jQuery var den mest använda funktionen i arbetet

```
$(xml).find("DOMShape").each(function(){

          domShape = NodeIterator.prototype.parseDomShape($
          (this), includedMatrix);
          tempList.push(domShape);
});
```

4.2.2 Xfl

Koordinathantering och operatorer

Om ett utropstecken ”!” (”!” = ”moveTo”) finns framför koordinaterna skall markören enbart flyttas till den punkt som koordinaterna beskriver.

!4520 2020

Om en pipe ”|” (”|” = ”lineTo”) finns framför koordinaterna skall markören ritas från föregående punkt till punkten koordinaterna beskriver.

|5899 2020

mellanslaget skiljer x- och y-koordinaterna. I fallet ovan är 5899 x-koordinat och 2020 y-koordinat. Detta gäller även vid utropstecken då markören flyttas utan att ritas.

Om en rak inledande parentes ”[” (”[” = ”beziercurveTo”) finns framför koordinaterna skall en bezierkurva ritas.

[5701 2368 5735 2404

Även här skiljer mellanslagen koordinaterna men här står första koordinaten 5701 för nuvarande x-koordinat, 2368 för nuvarande y-koordinat, 5735 för x-slutkoordinat och 2404 för y-slutkoordinat.

Svårtolkat sätt att rita linjer i xfl

Hur xfl ritar linjer eller kurvor av olika slag beskrivs av koordinaterna nedan.

!6959 3080|4420 3080!4420 3080|4420 1980!4420 1980

Det som händer är att först flyttas markören utan att ritas till punkt (x=6959, y=3080) sedan ritas en linje från den punkten till punkt (x=4420, y=3080) därefter flyttas markören till samma punkt (x=4420, y=3080) och så vidare.

DomShape

DomShape är namnet på den nod som lagrar majoriteten av informationen som behövs för konvertering av en bild. Det kan finnas fler noder med namnet DomShape och det betyder att en bild kan ha flera bildobjekt som är uppbyggda på olika sätt. Detta betyder då att alla de olika DomShapes-noderna måste lagras separat så att rätt färger, index och koordinater hör ihop med rätt DomShapes-nod. DomShape innehåller barnnoder som FillStyle, StrokeStyle, edges och Matrix.

FillStyle (nod under DomShapes träd)

FillStyle lagrar ett index och innehar en barnnod som lagrar en färg skriven hexadecimalt. Färgen som lagras är den färg ett bildobjekt skall vara ifyllt med. Finns det fler bildobjekt som befinner sig i samma DomShape-nod med olika ifyllda färger kan flera syskonnoder använda sig av namnet FillStyle och det är där indexet har sin verkan. Indexet kan ha värdet 1 och sedan adderas med 1 för varje syskon-nod som heter FillStyle inuti DomShape-nodens träd. Detta index kommer senare i noden edges att jämföras så att rätt bildobjekt får rätt färg. Om en av FillStyle-noderna inte har någon färg representerad utan bara ett index så används defaultfärgen svart.

Exempel FillStyle-noden i XML

```
<FillStyle index="1">
            <SolidColor color="#0080C0"/>
</FillStyle>
```

StrokeStyle (nod under DomShapes träd)

StrokeStyle fungerar på likadant sätt som FillStyle dock är färgen som lagras är inte en färg för fyllnad av bildobjekt utan den färg som omger bildobjektet. Bildobjektet kan ha en ram med olika tjocklekar och olika former på ramen (fyrkantig, rund, med flera). De olika formerna på ramen har flera

attribut och värden, av dessa attribut används `Weight` flitigast och den lagrar ett flyttal som anger hur tjock den omgivande ramen skall vara. Om en `StrokeStyle`-nod inte har någon färg representerad utan bara ett index så används defaultfärgen svart.

Exempel på `StrokeStyle`-noden i XML

```
StrokeStyle index="1">
       <SolidStroke scaleMode="normal"
                    weight="8.5">
              <fill>
                  <SolidColor color="#400040"/>
              </fill>
      </SolidStroke
</StrokeStyle
```

`edges` (`DomShapes` barnnod)

`Edge` är en barnnod till `edges` och lagrar en `fillStyle1`, en `strokeStyle` och en `edges` variabel.

Variabeln `fillStyle1` kan även vid vissa tillfällen inneha namnet `fillStyle` 0,1,2 eller 3. Variabeln innehåller en siffra som passar in på indexet lagrat i de olika `FillStyle`-noderna och skall därmed jämföras med dessa index för att ange den rätta färgen bildobjektet som beskrivs av koordinaterna i `edges` skall vara ifyllt med.

Liknande sker med variabeln `strokeStyle` som skall jämföras med noden `StrokeStyles` index för att rätt omgivande färg och tjocklek på ram skall användas.

Variabeln `edges` innehåller koordinater för hur de olika bildobjekten är ritade.

Vidare finns ytterligare en variabel i noden `Edge` vid namn `Cubics`, denna variabel innehåller koordinater men har ingen synlig funktion när de konverteras, inte heller har de en synlig funktion i Flash då de vid flera tillfällen raderades med oförändrat resultat i ursprungsbilden ritad i Flash.

Exempel på xfl formatets edgesnod i XML

```
<edges>
     <Edge fillStyle1="1" strokeStyle="1"
           edges="!4520 2020
                  |5899 2020!5899 2020|5899 2560
                  !5899 2560
                  |4520 2560!4520 2560|4520 2020"/>
</edges>
```

`Matrix` (DomShapes barnnod)

Noden `matrix` har en barnnod vid namn `Matrix` som är en matris beskrivande translation i x och/eller y led, skalning, rotation samt skjuvning av ett bildobjekt

Exempel på en `matrix`-nod i XML

```
<matrix>
                <Matrix a="0.565673828125"
                        b="0.565673828125"
                        c="-0.565673828125"
                        d="0.565673828125"
                        tx="287.6" ty="-111.05"/>
</matrix>
```

Matrisen har attributen a, b, c, d, tx och ty. Tx beskriver ett bildobjekts förflyttning i x-led, ty beskriver ett bildobjekts förflyttning i y-led.

$$\begin{bmatrix} a & c & t_x \\ b & d & t_y \\ u & v & w \end{bmatrix}$$

Variablerna a och d beskriver hur mycket bildobjektet skall skalas. Vid värden mindre än 1 skall bildobjektet förminskas och vid värden större än 1 skall bildobjektet förstoras förutsatt att variablerna b och c är lika med noll. Finns variablerna b och c ej representerade i matrixnoden är de i själva verket lika med noll och skrivs då ej ut av xfl. Variablerna b och c beskriver en skjuvning.

Finns variablerna a, b, c, och d representerade med värden kan flera olika kombinationer av rotation, skalning och skjuvning av bildobjektet ske.

Variablerna u, v och w används vid projicering men har i detta arbete ej påträffats.

Vid rotation innehar a, b, c, och d värden som på bilden [13].

$$\begin{bmatrix} \cos(q) & -\sin(q) & 0 \\ \sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

LinearGradient

Noden `LinearGradient` innehåller en `matrix`-nod samt `GradientEntry`-noder med färger och kvoters värden. En gradient behöver en start och en slutpunkt som matrixnoden skall tillgodose.

Exempel på en `LinearGradient`-nod i XML

```
<LinearGradient>
   <matrix>
      <Matrix a="0.0125732421875"
              b="0.013031005859375"
              c="0.013031005859375"
              d="-0.0125732421875"
              tx="609" ty="401.05"/>
   </matrix>
   <GradientEntry color="#E63426"
                  ratio="0.00392156862745098"/>
   <GradientEntry color="#CA271E"
                  ratio="0.36078431372549"/>
   <GradientEntry color="#B31D19"
                  ratio="0.749019607843137"/>
   <GradientEntry color="#AB1917" ratio="1"/>
</LinearGradient>
```

`LinearGradient` sätter startpunkten för gradienten till 0 och slutpunkten till 1. Ytan mellan 0 till och med punkten som representeras i första kvoten(ratio) beskriven i den första `GradientEntry`-noden färgas med färgen #E63426. Mellan kvoten i den förra `GradientEntry`-noden till och med nästa `GradientEntry`-nods kvot skall färgas med färgen #CA271E som är den nodens färgattribut(color) och så fortsätter det till sista `GradientEntry`-noden där kvotens värde alltid är 1 och färgen som beskrivs är den färg som avslutar gradienten.

DOMSymbolInstance

Denna nod används när det finns en eller flera inkluderade filer i huvudfilen. Här finns en bland annat en matrix-nod som beskriver den inkluderade filen så noggrannhet bör beaktas för att inte blanda ihop olika matrix-noder när matrix eftersöks i DomShape-noden. Den inkluderade filens DomShape-nod kommer inte att innehålla en matrixnod eftersom den befinner sig i huvudfilens nod DOMSymbolInstance.

DOMSymbolInstance innehåller även libraryItemName som är attributnamnet som skall användas för jämförelse med attributnamnet beläget i den inkluderade filen. Detta för att säkerställa att rätt bifogad fil läses och bearbetas

Beskrivning av hur xfl-inkludering av filer fungerar

I början av xfl filen finns sökvägar för de olika inkluderade filerna, om det finns några. Det kan finnas en eller flera inkluderade filer och varje fil måste hanteras för sig. För att försäkra sig om att rätt inkluderad fil bearbetas vid rätt tillfälle använder xfl-formatet sig av variabelnamnet ”libraryItemName”. Detta variabelnamns värde skall jämföras med motsvarande variabelnamns värde som finns beläget i den inkluderade filen. Om båda variabelnamnens värden är identiska kan vetskap bekräftas att rätt inkluderad fil bearbetas.

De inkluderade filerna innehåller till skillnad från huvudfilerna inte någon matrix-nod. Den matrix-nod som finns i noden DOMSymbolInstance hör till den bifogade filen som beskrivs i DOMSymbolInstance-noden.

4.2.3 HTML5 canvas

Att skapa en canvasyta

På canvasytan ritas bilderna ut i webbläsaren, den skall skapas i Javascriptprojektets HTML fil.

```
<canvas id="canvas" width="550" height="400"
style="background:#EEE">No Canvas</canvas>
```

Olika funktioner för att rita bildobjekt, bestämma färg på bildobjekt och en mängd andra funktioner finns tillgängliga att använda.

4.3 Bearbetning av information för val av programmeringsspråk

4.3.1 Valet mellan Java och Javascript

Eftersom Jadestone gav två alternativ att skriva koden i begränsades valmöjligheterna till Java eller Javascript.

Javascript exekveras direkt i webbläsaren och då kan xfl-formatets XML-fil läsas av, konverteras och sedan ritas ut i webbläsaren med Javascript.

Javakoden skall läsa in xfl-formatets XML-fil och sedan konvertera den. Nästa steg är att javakoden skall generera ett Javascript som krävs för att kunna använda HTML5-canvasfunktioner.

Vid kodning med Java, måste Java skrivas vid inmatning och Javascript vid utmatning av data. Det blir även mindre arbete vid Javascript då det mesta av koden skrivs i ett språk och körs direkt utan mellanvägar i webbläsaren.

Valet blev Javascript eftersom det kan minska risken för felskriven kod på grund av den ökade komplexiteten då två språk som skall användas. Arbetets-mängden minskar med Javascript och det var det huvudsakliga skälet till varför Javascript användes.

5 Genomförande

Genom att testa olika bildobjekt har hela tiden nya nod-namn, attribut och värden som behövs för konvertering tillkommit. Därmed har koden ofta ändrats och klasserna gjorts om genom att nya funktioner uppstått, ändrats eller tagits bort. Eftersom Javascript inte använder sig av klasser utan klassliknande funktioner som implementeras (kap 4.2.1) kommer jag hädanefter att kalla dessa klassliknande funktioner för klasser för att spara tid och utrymme i texten. Det är alltså inte i sig klasser utan klassliknande funktioner som kan användas på samma sätt som klasser används i andra programmeringsspråk.

5.1 Klasser och viktiga funktioner

Detta avsnitt beskriver de olika klasserna, några viktiga funktioner samt hur arbetet påbörjades. Samtliga klasser under arbetets gång implementerats.

5.1.1 Klassen `Rekursion`

Klassen `Rekursion` motsvaras av Javascript-funktionen `recursiveThroughTree` som hanterar de rekursiva anropen.

Funktionen `recursiveThroughTree`

Den rekursiva koden (nedan) är en urklippt del av den totala koden för att visa enbart den nödvändiga delen för rekursion. I funktionen finns sedan kod som hanterar de olika nodernas namn för att hämta och lagra data om färger och koordinater i olika arrayer som behövs vid konvertering.

```
myCode.prototype.recursiveThroughTree = function (action,
index){
        currentNode = action.localName;
        myFillStyles = [];
        myStrokeStyles = [];
        if(action.childNodes.length > index ){
               myCode.prototype.recursiveThroughTree
               (action.childNodes[index], 1);
         myCode.prototype.recursiveThroughTree(action,
index+2);
       }
};
```

5.1.2 Klassen `FillStyle`

`FillStyle` är en klass som lagrar färgen som ett bildobjekt skall vara ifyllt med och ett index (kap 4.2.2) som räknar hur många olika färger som finns tillgängliga i noden `DomShape` (kap 4.2.2). Den har två get- och två set-funktioner för färg respektive index.

5.1.3 Klassen `StrokeStyle`

`StrokeStyle` är en klass som lagrar färgen som omger ett bildobjekt. Den hanterar xfl-noden `strokeStyles` (kap 4.2.2) värden och attribut. Färgen `index` och `weight` är några av de viktigare attributen med värden som lagras och funktionerna som hanterar dessa är get- och set-funktioner för de olika attributen.

5.1.4 Klassen `Edge`

`Edge` lagrar attribut och värden som behövs för att rita ett bildobjekt. Här lagras objekt av klasserna `StrokeStyle` och `FillStyle` med alla deras innehållande värden och attribut samt de rätta färgernas kod skrivna i hexadecimala tal som behövs för att rita ett bildobjekt. Koordinaterna ur noden `edges` (kap 4.2.2) finns lagrade i denna klass och här finns även en array i vilken koordinaterna är sorterade efter huruvida markören enbart skall flyttas eller flyttas och ritas eller om bezierkurvor skall användas.

5.1.5 Klassen `Coordinate`

`Coordinate` hanterar koordinaterna som finns i den array i klassen `Edge` (kap 5.1.4) där de sorterade koordinaterna lagras. Denna klass används av klassen `DomShapeModifier` då den parsar alla koordinaterna för ett bildobjekt. `Coordinate` lagrar attributet ”act” som beskriver vilken operation som skall utföras (bezierkurva, flytta markör utan att rita eller flytta markör + rita). Även x- och y-koordinater lagras samt de fyra koordinaterna som beskriver en bezierkurva lagras i klassen.

5.1.6 Klassen `Matrix`(används ej)

Klassen `Matrix` används inte eftersom EaselJS innehar en klass vid namn `Matrix2D` som i stort sett gör samma sak som denna implementerade klass. `Matrix2D`-klassens konstruktor är uppbyggd på ett sådant sätt att det enkelt sätts in värden som hämtas ur xfl:s-`matrix` nod (kap 4.2.2) när ett `Matrix2D`-objekt skapas.

```
var matrix = new Matrix2D(a, b, c, d, tx, ty);
```

`Matrix2D` är med andra ord mer lättanvänt än den `Matrix`-klass som implementerats. `Matrix2D` innehar även en mycket viktig funktion vid namn `decompose()` som hanterar rotation, skalning, skjuvning och translation i x- och/eller y-led.

5.1.7 Klassen `DomShape`

`DomShape` implementeras för att i olika arrayer lagra nödvändiga värden som behövs för konvertering. Arrayer med noderna `fillStyle`-, `strokeStyle`-, `edges`- och `matrix`-värden lagras för att nämna de allra viktigaste.

5.1.8 Klassen `XflIteration`

`XflIteration` anropas och används ifall det finns bifogade filer i den ursprungliga xfl-filen med innehållande noder och information som används när ett bildobjekt skall ritas ut. Ibland kan vissa bildobjekt ha flera mindre lager av bildobjekt ritade i sig, det är då xfl använder sig av bifogade filer.

Det som görs i denna klass är att en XML parser (kap 4.2.1) anropas och all

data ur filen lagras i en sträng som sedan parsar och lagrar nödvändig data på samma sätt som ur den ursprungliga xfl-filen. Förutom en sträng med all data ur den bifogade xfl-filen lagras även den bifogade filens attributnamn med vilken man i klassen `NodeIterator` kommer att jämföra med noden `DomSymbolInstances` attributnamn ”`libraryItemName`” som enbart är ett namn att jämföra med och inte någon sökväg för inläsning av filen (kap 4.2.2).

5.1.9 Klassen `NodeIterator`

`NodeIterator` fyller funktionen som den rekursiva koden fyllde innan den skrotades. Den söker genom trädet efter vissa nod-namn som behövs för konvertering av bild. Den använder sig av funktioner för att finna och sedan parsa noden `DomShapes` träd. Sedan eftersöks nödvändiga noder innehållande värden som behövs för att fullfölja konverteringen samt att lagra dessa värden i de olika klasserna som nämndes tidigare i detta kapitel. Varje barnnod till `DomShape` som hittas vid parsingen anropar i sin tur en metod som parsar den noden tills värdena som behövs för konvertering av bildobjekt blir åtkomliga och kan lagras.

`NodeIterator` kontrollerar även om det finns någon referens till en bifogad fil som hör ihop med den aktuella filen som hanteras. Ett objekt av typen `XflIteration` (kap 5.1.8) används i en av funktionerna i `NodeIterator` som hanterar bifogade filer. Kapitel 11.1.1 visar en bild som beskriver de olika objekten som lagras i `NodeIterator` samt hur de ser ut i Javascripts logg.

Inläsning av data ur `DomShape`

En av klassens funktioner kombinerar två jQuery funktioner(`find()` och `each()`) för att finna alla noder vid namn `DomShape` (kap 4.2.2).

```
$(xml).find("DOMShape").each(function(){

   domShape =
   NodeIterator.prototype.parseDomShape(
   $(this),includedMatrix);
   tempList.push(domShape);

});
```

För var och en av de funna noderna anropas en funktionen `parseDomShape()` som skall parsa noder belägna i `DomShapenoden`.

Parsing av `DomShape`

Funktionen för att parsa hela `DomShapes` (kap 4.2.2) kombinerar två jQuery funktioner(`find()` och `each()`) som letar upp det aktuella nod-namnet som eftersöks och hanterar en nod i taget ifall flera noder med samma namn hittas. De olika eftersökta noderna är `FillStyle`, `StrokeStyle`, `Edge` och `matrix`, även dessa noder parsas i olika funktioner för att i slutändan leverera olika värden för lagring i respektive klasser beskrivna tidigare i kapitlet.

Hantering av inkluderade filer

I denna klass finns även ett antal funktioner för hantering av inkluderade filer.

En funktion som kontrollerar om någon bifogad fil existerar i huvudfilen och en funktion som lagrar ett objekt av typen `XflIteration` (kap 5.1.8) i en array. Det antal platser som är upptagna i denna array är även det antal bifogade filer som hänvisas till i huvudfilen. Vidare finns funktioner som parsar noden `DOMSymbolInstance` (kap 4.2.2) samt jämför attributnamnet ur noden med attributnamnet ur den bifogade filen. Ytterligare några funktioner är värda att nämna, de lokaliserar och parsar den `matrix`-nod som är barnnod till `DOMSymbolInstance` och lagrar de värden som finns samt anropar klassen `Draws` ritfunktion.

Försiktighet bör iakttagas så att `matrix`-noderna från `DomShape` och `DOMSymbolInstance` inte blandas ihop.

5.1.10 Klassen `DomShapeModifier`

Denna klass är som namnet avslöjar en klass som modifierar noden `DomShapes` värden. Dess olika funktioner anropas i `NodeIterator` och fyller klassen `Edge` (kap 5.1.4) med dess attribut och värden. Dess funktioner lägger till `fillColor`- och `strokeColor`-färger till `Edge`, parsar koordinaterna till den array som beskrivs i `Edge` kapitlet. Denna array lagrar koordinaterna i `Coordinate`-objekt (kap 5.1.5). Klassen ordnar även fel som kan uppstå när bilder ritade i xfl-format skall ritas ut i HTML5.

Funktioner för parsing av koordinater

I klassen finns det funktioner som var för sig parsar x-koordinater, y-koordinater, slutkoordinat för x och slutkoordinat för y ifall bezierkurvor används. Dessa tar hänsyn till ifall hexadecimala tal använts i koordinaterna och i det fallet anropas en funktion som konverterar hexadecimala tal till vanliga decimaltal.

Även de fallen då S2, S4 eller S6 står bland koordinaterna hanteras av de olika parsingfunktionerna.

De oförklarliga fallen med S2, S4 eller S6

Ibland händer det att någon av koordinaterna består av S2, S4 eller S6 vilket leder till att funktionen som användes för att rita linjer i HTML5-canvas inte ritar bildobjektet på ett korrekt sätt. Detta medför att någon eller några (beroende på hur många platser som ”infekterats” av S följt av någon siffra) linjer inte konverteras rätt och inte överensstämmer med den ursprungliga bilden. Detta problem löstes genom att helt enkelt radera tecknet S och allt som står efter tecknet fram tills x- eller y-koordinatet är slut som i exemplen nedan, mellanslag mellan koordinatvärden skiljer ju som bekant x- och y-koordinaten.

Före	Efter
!2539 1100S2	!2539 1100
!2234S4 1134	!2234 1134

Koordinater skrivna med hexadecimala siffror

Vid vissa tillfällen väljer xfl att skriva enstaka koordinater med hexadecimala siffror vilket gör att konverteringen inte fungerar eftersom funktionerna som används för att rita i HTML5-canvas använder sig av decimaltal. Detta har vid de tillfällen då de upptäckts inträffat när bezierkurvor ritas. Koordinaterna som beskrivs i hexadecimala tal kan vara enbart en x-koordinat medan tillhörande y-koordinaten skrivs med vanliga decimaltal. Som koden nedan visar är hälften av koordinaterna skrivna i hexadecimala tal och hälften i decimaltal.

[#1F29.2C #11CC.D9 7976 4556.5

En funktion i klassen `DomShapeModifier` implementerades därför för att konvertera hexadecimala tal till vanliga decimaltal.

Problemet när en ny rad bryter en koordinat i xfl

En del koordinat-rader som beskriver ett bildobjekt i noden `edges` (kap 4.2.2) kan vara väldigt långa, då kan xfl göra en radbrytning. När en radbrytning uppstår tolkas det som dubbla mellanslag vilket leder till att den konverterade bilden inte ritas korrekt. Detta går att lösa med Javascripts metod `replace`().

5.1.11 Klassen `Draw`

Denna klass ritar bildobjekten i HTML5. Den har en `NodeIterator` som inparameter innehållande data som behövs för att kunna rita en korrekt konverterad bild och använder sig av EaselJS funktioner för att rita bildobjekten. `Draw` håller reda på om ett bildobjekt skall ha en strokeColor och/eller en fillColor för att kunna påbörja en strokeStyle eller en fillStyle som krävs av EaselJS funktion för att ett bildobjekt skall ha en viss omgivande färg eller vara ifylld med en viss färg.

Med dessa anrop påbörjas färger för strokeStyle och fillStyle.

```
shape.graphics.beginFill(fillColor);
shape.graphics.setStrokeStyle().beginStroke(strokeColor);
```

Anropen kräver sedan avslutsanrop som vi ser i koden nedan.

```
shape.graphics.endFill();
shape.graphics.endStroke();
```

EaselJS klassen `shape` används för att rita bildobjekt och via `shape` kan man komma åt funktioner ur klassen `graphics` utan att skapa ett `graphics`-objekt. Funktionerna i koden ovan är `graphics`-funktioner.

EaselJS `shape` och `stage`

Från början användes klassen `graphics` enbart för att rita objekt men efter att rotation och andra matrix-funktioner behövde konverteras skapades objekt av klassen `Shape` och objekt av klassen `Stage`. `Stage`-objektet behövde en inparameter som beskrev vilken canvasyta som skulle användas och sedan behövde `stage` en `addChild()` funktion med `shape`-objektet som inparameter. Vad detta betyder är att en scen skapats och att nuvarande bildobjekt hanteras på denna scen. Detta behövs när `Matrix2D` funktionen `decompose()` skall användas, utan denna funktion måste komplicerade uträkningar göras för att förstå matrisens värden för rotation och andra matrixfunktioner. Funktionen `decompose()` sköter rotation, skalning, skjuvning och translation, funktionen använder `shape`-objekt som inparameter. Det som exempelvis roteras är hela scenen man skapat som består av en `shape` och det är bildobjektet klassen `Draw` för tillfället hanterar.

```
this.canvas = document.getElementById("canvas");
stage = new Stage(this.canvas);

var shape = new Shape();
stage.addChild(shape);

currentMatrix.decompose(shape);
```

Detta var även skälet till varför EaselJS klass `Matrix2D` användes istället för den klass med samma namn som från början implementerades (kap 5.1.6).

En viktig detalj som bör nämnas är att ingenting kommer att ritas om inte `stage` uppdateras innan nästa bildobjekt skall ritas. Uppdateringen sker med följande anrop.

```
stage.update();
```

Funktionen som ritar bildobjekt

När bildobjektet skall ritas är det viktig att tänka på att dividera xfl-koordinater med kvoten ”tjugo” för att rita med HTML5. Koordinaterna i xfl har tjugo gånger högre värde än vad koordinaterna i HTML5-canvas använder sig av.

Xfl koordinaterna

|2539 1100

tolkas av HTML5 som.

```
shape.graphics.lineTo(126.95, 55);
```

För att detta skall fungera i alla lägen måste xfl koordinaterna genast divideras med kvoten.

Javascriptet som det ser ut i den implementerade koden.

```
shape.graphics.lineTo(xCoor, yCoor);
```

En annan viktig sak är hur xfl ritar och flyttar markören (kap 4.2.2). Detta fungerar inte i HTML5 eftersom funktionerna som används i HTML5 kräver en startfunktion för `fillColor` och `strokeColor` samt även en slutfunktion för att färgerna skall användas i hela bildobjektet. När markören flyttas utan att ritas efter varje gång funktionen har ritat en linje kommer inte färgerna att fylla eller omge ett bildobjekt på ett korrekt sätt.

Koden implementerades så att flytt av markören inträffar förrän hela bildobjektet ritats färdigt.

5.2 Ett principiellt vägval

En tredjedel av tiden gick åt att försöka implementera med hjälp av rekursion. Här beskrivs den rekursiva implementationen, den mer objektorienterade implementationen samt skälen till varför rekursionen slutade användas.

5.2.1 Den rekursiva eran

Traversering genom nod-träd sker med rekursion, detta på grund av att rekursion är ett bra sätt att använda sig av när man skall traversera sig genom ett nod-träd. Andra alternativ finns att tillgå men eftersom man med lite kod kan traversera genom hela trädets noder känns rekursion som effektivast att börja med.

Var det gick bra

Rekursivitet fungerade väldigt bra i början då det var få attribut som behövdes och bilderna som ritades var väldigt enkla. Det gick att med lite kod besöka varje nod i trädet samt hämta attributnamn och värden ur noderna. Koden blev dock längre och längre i takt med att mer avancerade bilder testritades. Det behövdes fler noder att söka efter och fler värden att lagra i de olika arrayer som används. I det stora hela så fungerade det ändå väldigt bra så länge endast en `DomShape`-nod fanns i XML dokumentet.

Var det gick dåligt

I takt med att mer avancerade bilder började ritas tillkommer svårigheter. Koden är inte längre kort och fler if-satser används för att söka efter nytillkomna noders namn som behövdes vid konvertering. De riktigt stora problemen tillkommer när bilder som innehåller flera bildobjekt används. Varje `Domshape`-nod beskriver ett bildobjekt och ju fler bildobjekt desto fler `DomShape`-noder finns representerade (kap 4.2.2). Eftersom olika array lagrar `fillStyle`- och `strokeStyle`-färger och deras index färgas alla nya bildobjekt med samma färger som det första bildobjektets `fillStyle`- och `strokeStyle`-färger.

Detta på grund av att vid sökning av array för att hitta index- värdet ”1” påträffades alltid den första platsen i array med värdet ”1”. Finns det då flera

`Domshape`-noder kan flera färger inneha index-värdet ”1” och lagras i array. Det medför att vid konvertering kommer alla bildobjekt fyllas med samma färger som det första bildobjektet som beskrivs i XML-filen.

En alternativ lösning av detta problem är att tömma array efter att bildobjekten som beskrivs i den första `DomShape`-noden konverteras och ritats klart. Detta fungerar inte och efter att array tömts på data förblir de tomma. Flera olika försök att lösa dessa problem misslyckas då nya fel uppkommer efter varje försök. Vid det här laget har den rekursiva funktionen fyllts med en mängd if-satser och koden är inte längre liten och nätt. Den har blivit komplicerad och rörig även efter försök till refaktorisering.

Den slutgiltiga skrotningen av rekursionen

Problemen som beskrivits var bara toppen av isberget. När en bild med fler `DomShape`-noder ritades tog det väldigt lång tid att exekvera koden. Det tog från början sjutton sekunder innan bilder helt och hållet ritades ut i canvasen. Tiden kortades ner till åtta sekunder efter ändring i koden och sedan till tre sekunder efter ytterligare ändring vilket är för lång tid för en konvertering. Skälet till den långa exekveringstiden är att vid rekursion kan en nod besökas väldigt många gånger även om data redan hämtats ur den så kommer den att besökas och analyseras efter olika värden eller jämföras med andra värden som lagras i olika arrayer. Detta medför att tidskomplexiteten blir väldigt dålig (snabbt växande) ju mer avancerad bild som testas för konvertering. Vid användning av rekursion är det även komplicerat att identifiera var problemen finns och hur man på bästa sätt efter identifiering av problemen löser problemen utan att nya uppstår. När rekursion används måste utvecklaren tänka ”baklänges” och det kan bli lite rörigt och svårt om det är mycket som händer i koden. Beslutet att skrota rekursionen togs därmed för att testa ett nytt tillvägagångssätt.

5.2.2 Den nya objektorienterade eran

Det finns funktioner i jQuery som kan användas för att jämföra en valfri sträng med en nods namn. Kombinerar man dessa sökfunktioner med funktionen `each()` kan man upprepa arbetet för varje sökt nods namn.

Uppbyggnad

När objektorientering i stället används så skapas flera klasser för att lagra de olika värdena som behövdes för att konvertera en bild. NodeIterator var en sådan som fyllde samma funktion som den rekursiva koden fyllde. Den letade reda på var i XML-dokumentet DomShape-noderna fanns och parsade genom den nodens hela familj i funktionen nodeTreeStoring.

En skillnad i hur data lagras jämfört med rekursion är att olika färgers värden med tillhörande index lagrades i olika array. När objektorientering används så lagras objekt i array dessa arrayer lagras i sin tur i andra objekt som skapats.

5.3 Arbetsbeskrivning

1. Skapa en canvasyta

En canvasyta skapas. I webbsidans canvasyta publiceras de konverterade bilderna.

```
<canvas id="canvas" width="550" height="400"
style="background:#EEE">No Canvas</canvas>
```

2. Läsa xfl formatet i XML kod

Med hjälp av funktionen XML parser och jQuery funktionen som lagrar XML-filens innehåll i en sträng (kap 4.2.1) läses hela xfl-filen och sparas i variabeln ”xml”.

3. Olika bildobjekt testas

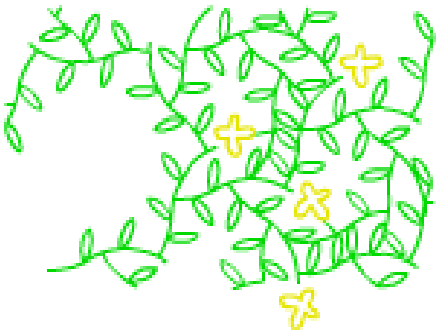
Olika bildobjekt ritades i Flash med anledning av att se hur koden ser ut i xfl för att sedan bedöma vilka attribut, noders namn och värden som behövs i konverteringen.

När olika rektanglar som den i figur 5.1 ritades erhölls en inblick i hur xfl-koden kunde ändras och se ut, samt även flera viktiga noders namn.

Figur 5.1
Bilden i figur 5.1 är ritad i Flash och har en svart ram med blå fyllnadsfärg.

4. Mer avancerad bild

En bild med många blommor som den i figur 5.2 var den första bilden där programmets prestanda verkligen testades. Ur denna bild upptäcktes många fel i den rekursiva implementationen. Det var även denna bild som blev början på slutet för det rekursiva tillvägagångssättet.



Figur 5.2
Figur 5.2 är ritad i Flash med bezierkurvor och föreställer en buske med blommor och blad.

5. Iterativ testning

Vidare fortsattes testning av olika bilder, studerande av xfl-formatets kod i XML-filen och implementering av kod för konvertering. Detta steg fortsattes till slutet av arbetet.

6. Rita objekt med kod

Bilder ritades även genom att i XML-filen manuellt skriva koordinater och andra funktionsanrop som xfl använder sig av. Detta medför att en större förståelse erhölls för hur xfl-formatet använder olika noders attribut och värden när olika bildobjekt ritas.

6 Resultat

Resultaten från arbetet är att olika bilder som ritas i Flash konverterats och ritats ut i en webbläsare med HTML5-canvasobjekt.

6.1 Bildexempel

6.1.1 Bildresultat när tecknet ”S” dök upp i koordinater

Vid vissa tillfällen kunde bokstaven ”S” följt av siffrorna 2, 4 eller 6 blandas inuti koordinaterna som beskrivs i kapitel 5 (kap 5.1.10). Koden kunde då se ut på följande sätt.

!2539 1100S2

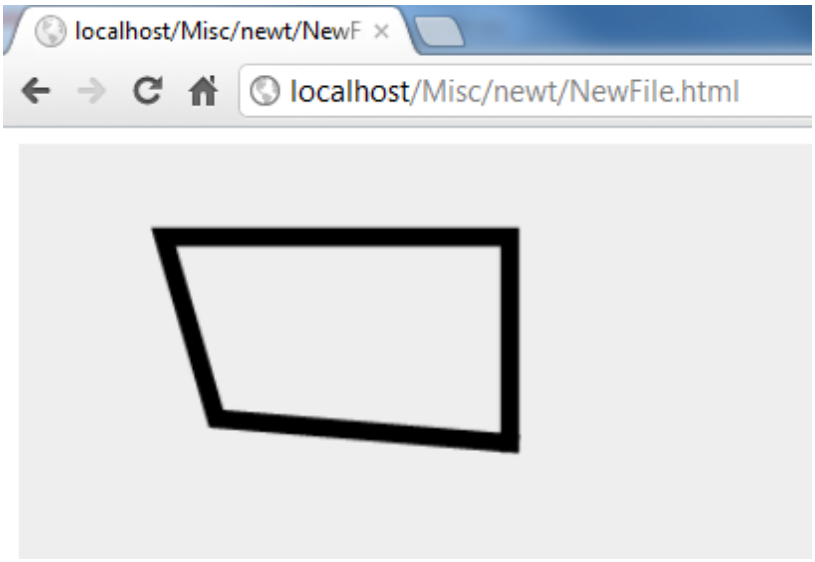
Figur 6.1

Bilden i figur 6.1 ritad i Flash (xfl)

Icke-korrekt konverterad bild ritad i webbläsaren innan lösning på fallet ”S”.

Figur 6.2
Bilden i figur 6.2 ritad i webbläsaren med HTML5-canvasobjekt innan lösningen på problemet med ”S” beskriven i kapitel 5 (kap 5.1.10).

Korrekt konverterad bild ritad i webbläsaren med lösning på fallet ”S”.



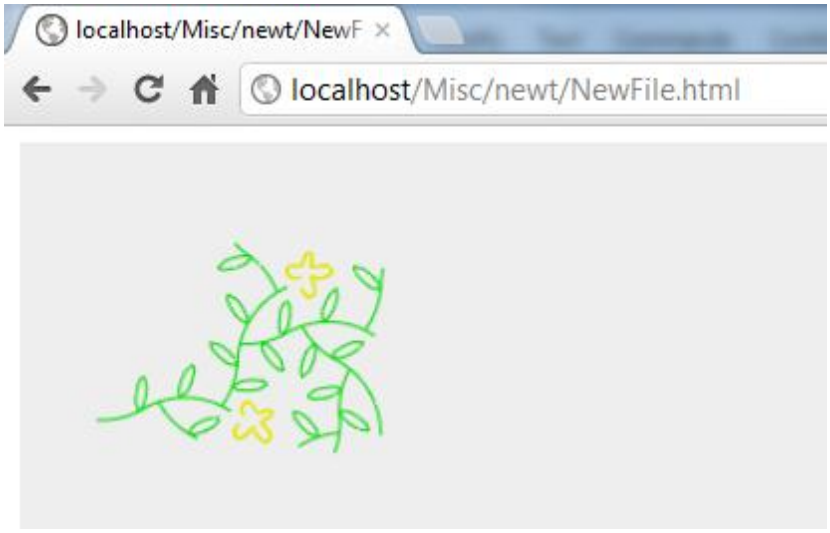
Figur 6.3
Bild i figur 6.3 konverterad med funktionell lösning på fallet ”S”.

6.1.2 Bildresultat med hexadecimala

När bezierkurvor ritas kan det ibland hända att xfl formatet använder sig av hexadecimala siffror för att beskriva vissa av koordinaterna (kap 5.1.10). Det var även denna typ av bild (figur 6.4) där det upptäcktes att den rekursiva koden inte var en bra lösning

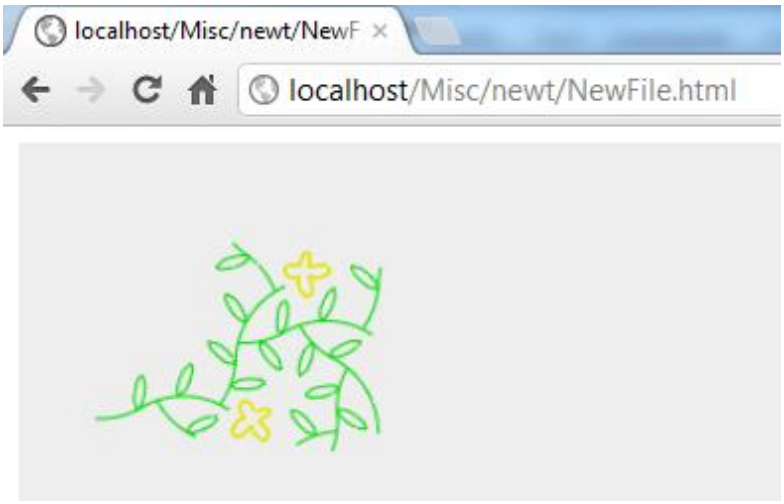
Figur 6.4
Bild i figur 6.4 med bezierkurvor ritad i Flash (xfl)

Bilden i figur 6.5 ritades i webbläsaren med HTML5-canvasobjekt innan funktionen som konverterar hexadecimala tal var implementerad. Detta medför därmed att bilden inte blir korrekt konverterad från xfl-formatet. Det är ganska svårt att se men de gula blommorna är inte hela och är därmed inte på ett korrekt sätt ritade enligt den ursprungliga xfl-filen.



Figur 6.5
Figur 6.5 beskriver en bild innan funktionen som konverterar hexadecimala siffror till decimaltal implementerats. De gula blommorna är inte helt ritade.

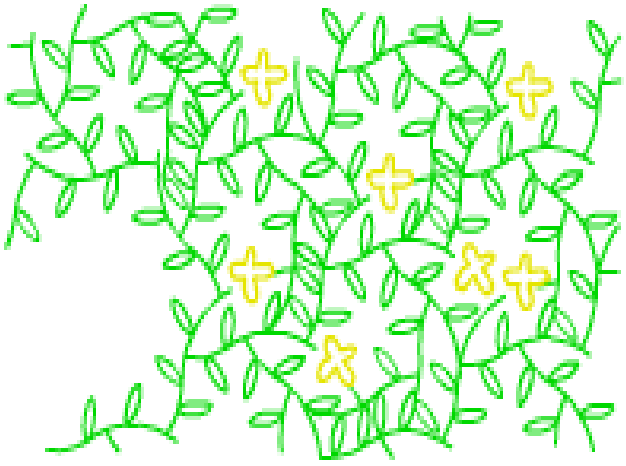
Bilden i figur 6.6 ritades i webbläsaren med HTML5-canvasobjekt när funktionen som konverterar hexadecimala tal till decimaltal var färdigimplementerad. Om man noggrant jämför med bilden i figur 6.5 och bilden i figur 6.6 kan små skillnader upptäckas på de gula blommorna, blommorna är inte helt utritade i figur 6.5.



Figur 6.6
Figur 6.6 beskriver den korrekt konverterade bilden ur figur 6.4.

6.1.3 Ny rad

Långa koordinatrader kan behövas i xfl-formatet om en bild består av många detaljer. Vid sådana tillfällen kan xfl-formatet skapa en ny rad som ibland delar en koordinathandling som exempelvis en bezierkurva. Som beskrivet i kapitel 5.1.10 uppstår då ett problem som gör att bilden inte konverteras på ett korrekt sätt i HTML5. Ny rad tolkas som dubbla mellanslag och medför därmed att konverteringen blir inkorrekt.



Figur 6.7
Bilden i figur 6.7 är ritad i Flash (xfl)

Bilden i figur 6.8 är en konverterad bild av flera bezierkurvor som inte är korrekt konverterade på grund av problemet med att ny rad tolkas som dubbla mellanslag. Om man tittar syns det att bladen inte är hela och inte ser exakt likadana ut som i bilden ur figur 6.7.

Figur 6.8
Figur 6.8 beskriver en icke-korrekt konverterad bild ur figur 6.7.
Vissa av de gröna löven är inte hela som i figur 6.7.

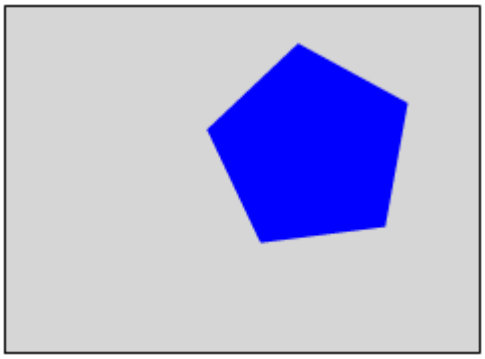
Bilden i figur 6.9 är en korrekt konverterad bild av den som visas i figur 6.8. En funktion för att lösa problemet med ny rad används. Vid noggrann granskning kan man upptäcka att felen som uppstod i figur 6.8 inte har upprepats och alla gröna löv är hela som den ursprungliga bilden ur figur 6.7.

Figur 6.9
Figur 6.9 beskriver en bild som är en korrekt konverterad bild av bilden i figur 6.7.

6.2 Bildförändringar med matrix

6.2.1 Bilden ritad i Flash

Bilden i figur 6.10 ritad i Flash (xfl) och hänvisas till som originalbild i kapitel 6.2. Den gråa bakgrunden har lagts till för att bakgrunden skall synas bättre.



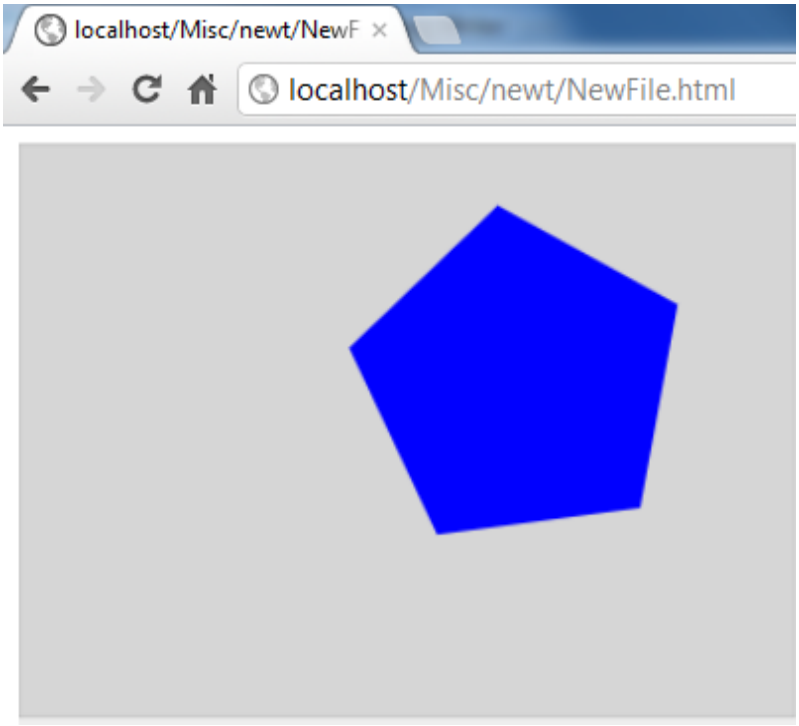
Figur 6.10
Figur 6.10 ritad i Flash (xfl). Kommer att kallas för originalbilden hädanefter.

6.2.2 Konverterad originalbild

XML-koden beskriver xfl-formatets kod

```
<DOMShape>
      <fills>
         <FillStyle index="1">
           <SolidColor color="#0000FF"/>
          </FillStyle>
      </fills>
      <strokes>
          <StrokeStyle index="1">
               <SolidStroke scaleMode="normal"
                  weight="0.1">
                  <fill>
                    <SolidColor/>
                  </fill>
              </SolidStroke>
          </StrokeStyle>
      </strokes>
      <edges>
        <Edge fillStyle1="1" strokeStyle="1"
           edges="!4179 3716|3010
          2372!3010 2372|3927 845!3927 845|5663 1245!
          5663 1245|5819 3020!5819 3020|4179 3716"/>
      </edges>
</DOMShape>
```

Bilden i figur 6.11 är den konverterade originalbilden ritad i webbläsaren med HTML5.



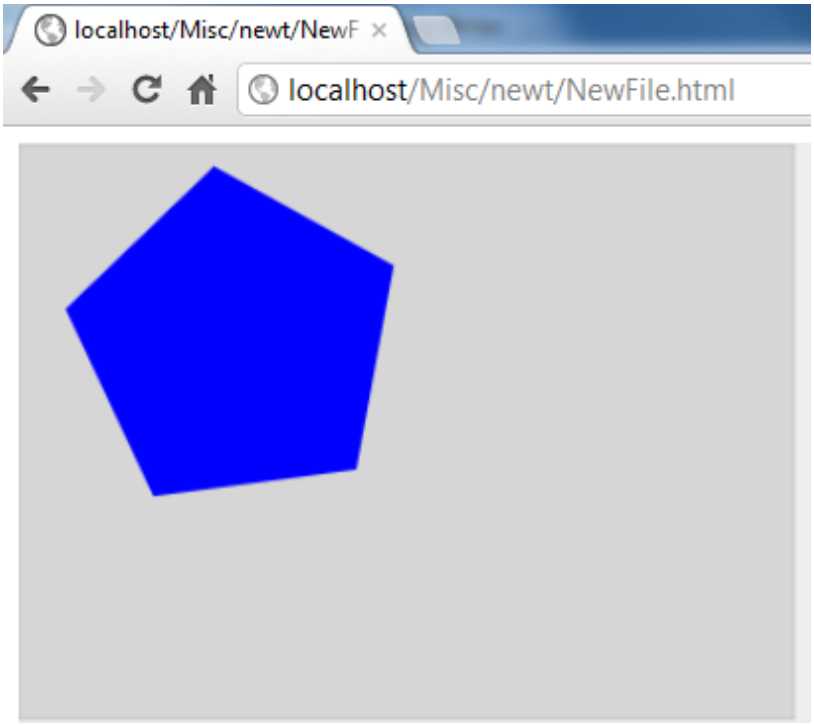
Figur 6.11
Figur 6.11 beskriver den konverterade bilden ur figur 6.10 som även kallas för ”originalbilden”.

6.2.3 Konverterad originalbild som förflyttats med matris

Koden beskriver ett blått ifylld fem-kantigt objekt som är förflyttat i x- och y-led med matrisvärden. Koden ser likadan ut som i kap 6.2.2 med undantaget av värden ur matrisnoden.

```
<matrix>
        <Matrix tx="-136" ty="-29"/>
</matrix>
```

Bilden i figur 6.12 är den konverterade originalbilden. Bildobjektet är förflyttat i x- och y-led



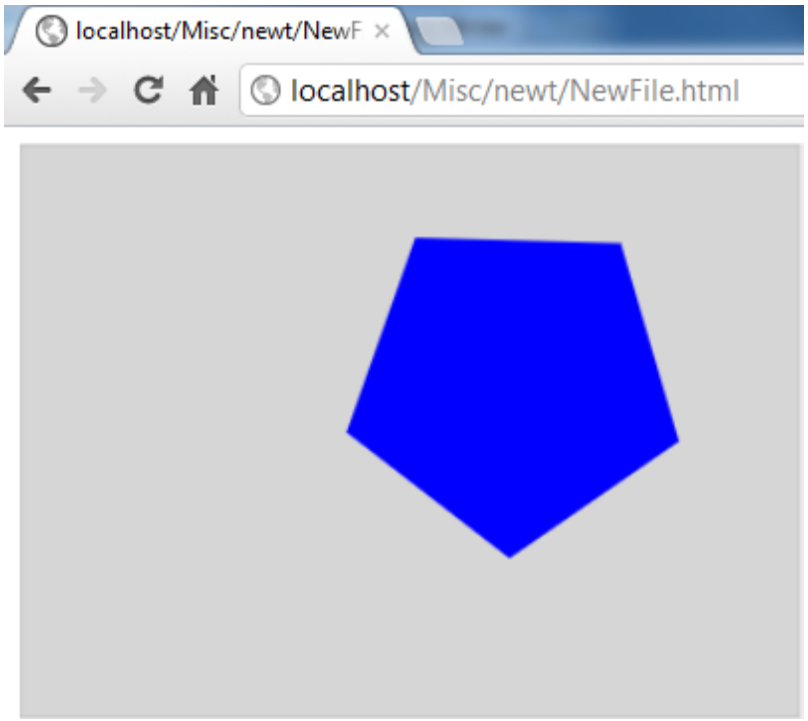
Figur 6.12
Bilden i figur 6.12 beskriver originalbilden som konverterats och förflyttats i både x- och y-led.

6.2.4 Konverterad originalbild som roterats med matrix

Koden beskriver matrixvärden som roterat bildobjektet 45 grader.

```
<matrix>
     <Matrix a="0.707107543945313"
             b="0.707107543945313"
             c="-0.707107543945313"
             d="0.707107543945313"
             tx="139.55" ty="-130.05"/>
 </matrix>
```

Bilden i figur 6.13 beskriver originalbilden efter rotation 45 grader. Bilden är konverterad och ritad i webbläsaren.



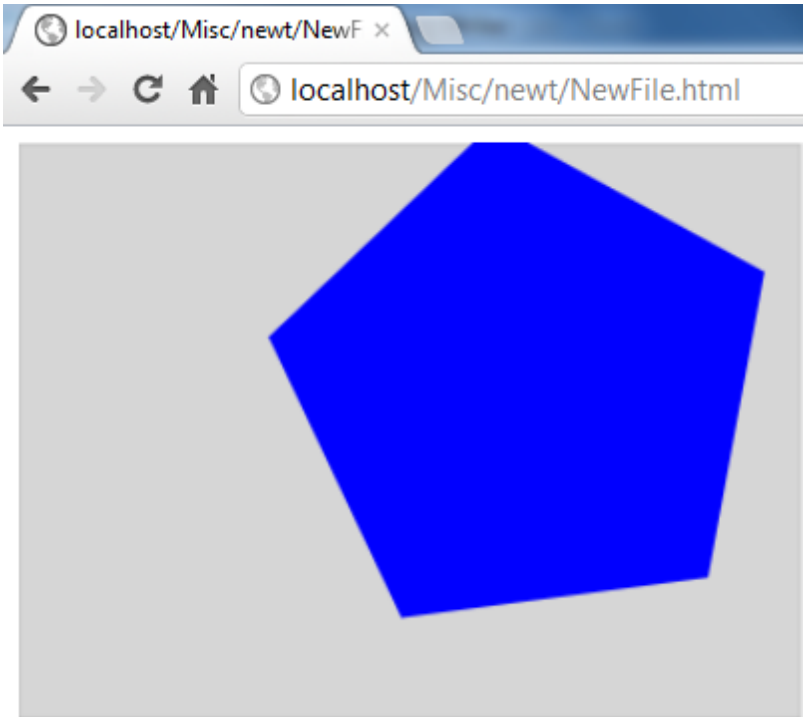
Figur 6.13
Bilden i figur 6.13 beskriver originalbilden som konverterats och roterats 45 grader.

6.2.5 Konverterad originalbild som förstorats med matrix

Koden beskriver matrixvärden som förstorat originalbilden samt förflyttat den i x- och y-led.

```
<matrix>
          <Matrix a="1.5" d="1.5" tx="-113.4"
                  ty="-51.75"/>
</matrix>
```

Bilden i figur 6.14 har förstorats och förflyttats. Bilden har konverterats och ritats i webbläsaren.



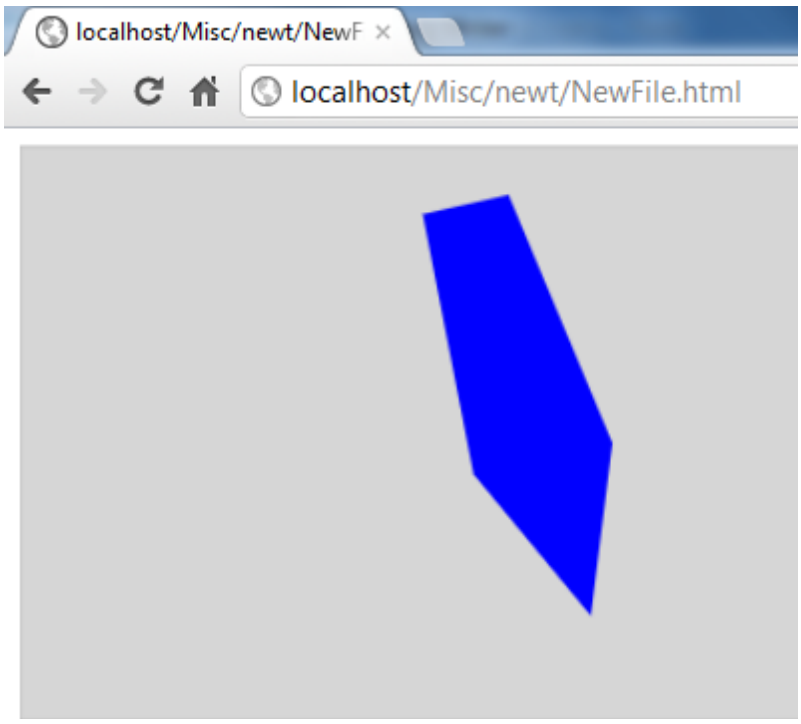
Figur 6.14
Bilden i figur 6.14 beskriver hur originalbilden har konverterats och förstorats med 150 procent. Bilden har även förflyttats i både x- och y-led.

6.2.6 Konverterad originalbild efter skjuvning

Koden beskriver originalbilden efter skjuvning och förflyttning.

```
<matrix>
          <Matrix a="0.573577880859375"
                  b="0.81915283203125"
                  tx="96.7" ty="-185.75"/>
</matrix>
```

Bilden i figur 6.15 har skjuvats och förflyttats. Bilden har även konverterats och ritats i webbläsaren.



Figur 6.15
Bilden i figur 6.15 beskriver hur originalbilden skjuvats och förflyttats i både x- och y-led. Bilden har konverterats till HTML5.

6.3 Misslyckade konverteringar

Dessa typer av bilder misslyckades vid försök till konvertering.

6.3.1 Bild med gradienter

Bilder med olika gradienter misslyckades vid konvertering. Ett bildexempel på när det misslyckades syns i figur 6.16

Figur 6.16
Bilden i figur 6.16 beskriver en bild med gradienter ritad i Flash (xfl).

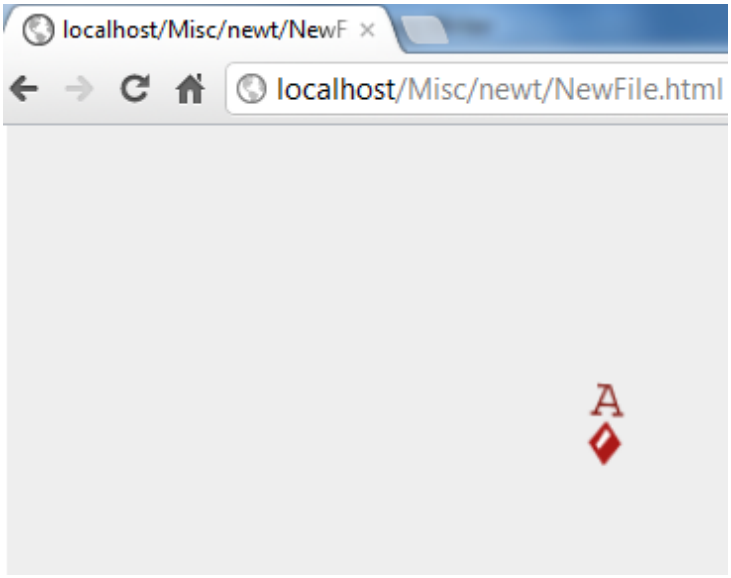
Xfl-koden för denna lilla bild (figur 6.16) var stor och innehöll 150 rader kod vilket var mer än koden som beskriver bilderna i kapitel 6.1 och 6.2. Koden

som presenteras nedan är en av många noder som används för att rita bilden i figur 6.16.

```
LinearGradient>
    <matrix>
        <Matrix a="0.0125732421875"
                b="0.013031005859375"
                  c="0.013031005859375"
                  d="-0.0125732421875" tx="609"
                  ty="401.05"/>
    </matrix>

<GradientEntry
            color="#E63426"
            ratio="0.0039215686274509 8"/>
    <GradientEntry
                color="#CA271E"
                ratio="0.36078431372549"/>
    <GradientEntry
                color="#B31D19"
                ratio="0.749019607843137"/>
    <GradientEntry color="#AB1917" ratio="1"/>
</LinearGradient>
```

Bilden i figur 6.17 är en konverterad bild med gradienter som dessvärre inte blir korrekt och innehar vissa skillnader mot bilden i figur 6.16.



Figur 6.17
Bilden i figur 6.17 beskriver bilden ur 6.16 efter konvertering.
Konverteringen misslyckades.

6.3.2 Bilder icke korrekt ifyllda

Vissa av de mer avancerade bildobjekten blev inte på ett korrekt sätt ifyllda som den i figur 6.18. Bilden är ritad i Flash (xfl).

Figur 6.18
Bilden i figur 6.18 beskriver en bild ritad i Flash (xfl).

En mycket liten del av den totala koden som används av xfl för att rita bilden ur figur 6.18 för att visa omfattningen av kod för en så tillsynes enkel bild.

```
<DOMShape selected="true" isDrawingObject="true">
    <matrix>
        <Matrix tx="-185.95" ty="-59.8"/>
    </matrix>
    <fills>
      <FillStyle index="1">
         <SolidColor color="#020203"/>
      </FillStyle>
   </fills>
   <edges>
     <Edge fillStyle1="1" edges="!8856 5213S2
        |9225 5212!9225 5212
        [9331 5212 9379 5231!9379 5231
        [9427 5249 9454 5292!9454 5292
        [9480 5334 9480 5383
        !9480 5383[9480 5430 9456 5476!9456 5476
        [9432 5521 9384 5552!9384 5552
        [9336 5582 9261 5599
        !9261 5599[9282 5629 9326 5685!9326
        5685|9465 5868!9465 5868[9518 5938 9535 5954
        !9535 5954|9545 5960!9545 5960
        [9552 5962 9581 5962
        !9581 5962|9581 5998!9581 5998
        [9540 5994 9508 5994
        !9508 5994[9475 5994 9429 5998!9429 5998
        [9396 5961 9308 5836!9308 5836
        [9268 5779 9225 5725
        !9225 5725[9170 5655 9114 5590
        !9114  5590|9119 5574
        !9119 5574|9163 5576!9163 5576
        [9263 5576 9315 5528
        !9315 5528[9368 5481 9368 5403!9368 5403
        [9368 5335 9323 5294!9323 5294
```

```
            [9277 5254 9187 5254
            !9187 5254[9129 5254 9076 5272
            !9076 5272|9068
            5733!9068 5733[9068 5800 9071 5879!9071 5879
            [9073 5932 9079 5942!9079 5942
            [9086 5952 9099 5957
            !9099 5957[9114 5962 9179 5962
            !9179 5962|9179 5998
            !9179 5998[9100 5994 9034 5994!9034 5994
            [8981 5994 8856 5998!8856 5998
            |8856 5962!8856 5962
            [8921 5962 8935 5957!8935 5957
            [8949 5952 8955 5943
            !8955 5943[8961 5933 8963 5890
            !8963 5890|8966 5478
            !8966 5478[8966 5403 8964 5330!8964 5330
            [8962 5277 8956 5269!8956 5269
            [8950 5259 8935 5254
            !8935 5254[8922 5250 8856 5248
            !8856 5248|8856
              5213"/>
    </edges>
</DOMShape>
```

Bilden som visas i figur 6.19 konverterades men blev inte som originalbilden i figur 6.18

Figur 6.19
Bilden i figur 6.19 beskriver bilden ur figur 6.18 som konverterats till HTML5. Konverteringen blev inkorrekt.

6.3.3 Cirklar

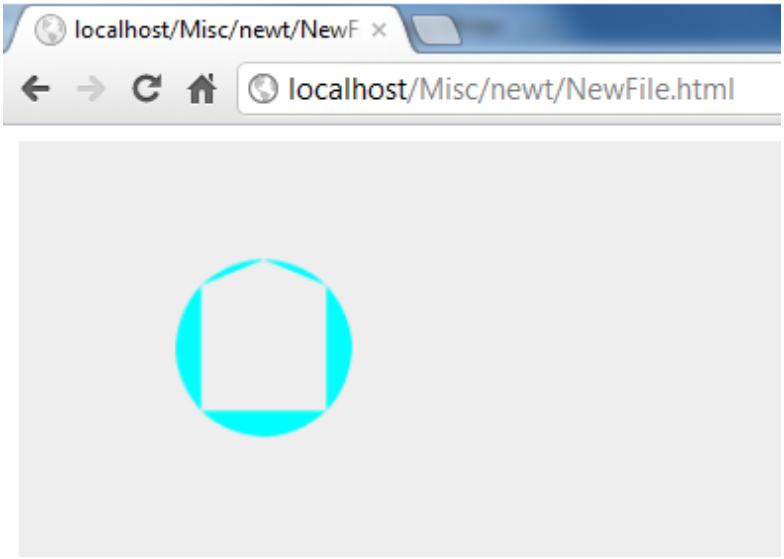
Bilden i figur 6.20 på en cirkel med ifylld färg ritad i Flash

Figur 6.20
Bilden i figur 6.20 beskriver en cirkel ritad i Flash (xfl).

Xfl koden beskriver cirkeln i figur 6.20.

```
<DOMShape>
  <fills>
     <FillStyle index="1">
        <SolidColor color="#00FFFF"/>
     </FillStyle>
   </fills>
   <edges>
       <Edge fillStyle1="1" edges="!2250 1080
             [2586 1080 2823 1317!2823 1317
             [3060 1554 3060 1890
             !3060 1890[3060 2226 2823 2463
             !2823 2463[2586 2700 2250 2700
             !2250 2700[1914 2700 1677 2463
             !1677 2463[1440 2226 1440 1890
             !1440 1890[1440 1554 1677 1317
             !1677 1317[1914 1080 2250 1080"/>
   </edges>
</DOMShape>
```

Konverterad bild av bilden i figur 6.20 visas i figur 6.21.



Figur 6.21
Bilden i figur 6.21 beskriver bilden ur figur 6.20 som konverterats och ritats ut i webläsaren med HTML5. Bilden blev inte korrekt konverterad.

7 Slutsats

7.1 Resultat

Huvudproblemet från kapitel 1.3 löstes delvis då de flesta bilder ritade i Flash konverterades och ritades ut i webbläsaren med HTML5-canvasobjekt. Långt ifrån alla möjliga bilder testades på grund av arbetets omfattning. De typer av bilder som inte konverterats på ett korrekt sätt nämns i kapitel 8 Framtida Utvecklingsmöjligheter.

7.1.1 Val av programmeringsspråk

De två språkalternativen Jadestone gav studerades och Javascript var det bättre alternativet. Detta val baserades på det faktum att om Java hade valts som språk att arbeta med hade programmet behövt generera ett Javascript för att använda HTML5-canvas. Då var det ett enkelt val att skriva koden direkt i Javascript. Kapitel 4.2 tar upp frågan och jämför de olika språken.

7.1.2 Dokumentation av xfl

Det finns ingen dokumentation av xfl på internet och väldigt lite information om xfl överhuvudtaget. Sökningen efter information om xfl gjordes på google och på ett antal olika forum och det lilla som hittades var användbart men informationen täckte endast en bråkdel av det som behövs för att kunna arbeta effektivt.

På forumet stackoverflow finns det tråd som behandlar ämnet xfl-dokumentation och endast ett svar på tråden finns publicerat. Detta svar bidrar med en länk till en artikel där en påstådd intervju ägt rum med en person vid namn Richard Galvan som enligt artikeln är en chef på Adobe. Intervjun är från Mars 2008 där Galvan berättar att Adobe har för avsikt att släppa en dokumentation för xfl men det sägs dock ingenting om när detta skall ske.

7.1.3 Sammanfattning av resultaten

Bildobjekten som konverterades hade ofta olika mindre defekter som skulle åtgärdas för att de skulle bli konverterade på ett korrekt sätt. Dessa typer av problem var utmärkande för hur arbetet fortgick. Ofta hände det att xfl-formatet överraskade med olika ändringar i XML-koden (kap 5.1.10) utan att någon synlig ändring gjorts i ursprungsbilden.

7.1.4 Korrekt ritade bilder

Byte från rekursion

Bytet från rekursion till mer objektorienterad programmering var ett väldigt lyckat byte som gjorde det enklare att både skriva kod och upptäcka och korrigera fel i samband med konverteringen.

Jadestone anställdas tips och råd

Arbetets handledare på Jadestone samt även andra utvecklare på Jadestone med erfarenhet av Flash gav väldigt bra tips på var några av felen som uppstod kunde befinna sig. De kände inte till hur xfl fungerade men hade väldigt goda kunskaper om hur Flash gamla format fungerade.

Iterativ testning

Testning av olika bilder ritade i Flash och sedan en analys av xfl-formatets kod ger en ökad förståelse för vad som behöver implementeras för en lyckad konvertering. De olika bilderna som testas i Flash konverteras och ritas i webbläsaren med HTML5-canvasobjekt. Därefter jämförs ursprungs-bilderna och de konverterade bilderna för att se om något inte konverterats på ett korrekt sätt. Vid icke-korrekt konvertering analyseras xfl-koden för att ge en inblick i vad som behöver ändras eller läggas till i implementationen.

7.1.5 Bilder som inte konverterades korrekt

Arbetets storlek

Arbetets omfattning gjorde att det inte fanns tid att testa alla möjliga kombinationer av bilder för att få ett bättre resultat. Det finns så många noder och attribut som behövs för en konvertering och vid varje ny typ av bild uppstod det nya noder och attribut som skall lagras och bearbetas innan konvertering. Med stor sannolikhet kan det antas att en stor mängd noder och attribut ännu inte upptäckts eftersom långt ifrån alla kombinationer av bilder och bildobjekt testats.

Tidsbrist

Med tanke på storleken på arbetet att producera en konverterare behövs det fler utvecklare och mycket mer tid för att kunna presentera en fungerande

slutprodukt.

Cirklarna (kap 6.3.3) som inte fylldes helt med vald färg beror på dels tidsbrist när olika bildobjekt testades i Flash och dels på tidsbrist när implementationen av ifylld färg för cirkelformade objekt skulle implementeras.

Brist på dokumentation

Arbetet har försvårats avsevärt på grund av brist på dokumentation av hur xfl och dess funktioner fungerar. Bilder med gradienter misslyckades på grund av att det inte gick att förutse hur xfl tilldelar start- och slutpunkt på gradienten, en dokumentation som förklarar detta hade kunnat förenkla denna problematik.

Adobe gör xfl komplicerat att konvertera

Adobe kan ha gjort xfl väldigt komplicerat för att det skall vara svårt att skapa en konverterare till HTML5. Detta grundas på att xfl-formatet innehåller en mängd oförklarliga överraskningar när konvertering skall göras. Markörflytt efter varje ritad linje är onödigt och används inte i det äldre formatet. Markörflytten gör att implementationen som hanterar bildobjekt som skall vara fyllda med en färg försvåras. Koden som hanterar detta problem ändrades ett flertal gånger. Mer ingående förklaring om detta problem finns i kapitel 7.3.4 och kapitel 8.1.2.

Kapitel 5.1.10 tar upp fler svårigheter som dyker upp vid försök till konvertering. Dessa svårigheter finns enligt Jadestones Flash-programmerare inte i de äldre Flash-formaten.

7.1.6 Skillnad i hur xfl och HTML5 ritar bildobjekt

Det är svårt att utan dokumentation tolka xfl koordinathantering och handlingar men lyckligtvis finns det bra förslag man kan utgå från ur en tråd på ett internetforum [5]. Det går även om man testar sig fram själv genom att rita olika bilder och studera hur och när koden ändras men det är väldigt tidskrävande.

HTML5:s canvas-klass och de hjälpbibliotek som används för att rita i en canvasyta använder sig av pixlar för hantering av koordinater medan xfl:s koordinater multiplicerar en pixel med siffran 20. Vidare ritas linjer och bildobjekt i xfl på ett sätt som ger svårigheter vid konvertering. Koden nedan beskriver xfl sätt att rita en rektangel. Det som händer är att först flyttas markören till en startpunkt, sedan ritas en linje från startpunkten till en

slutpunkt som anges, för att sedan flytta markören till slutpunkten. Man flyttar alltså en markör till den platsen där markören befann sig då en linje ritades (kap 4.1.2).

!1800 2480|1320 840!1320 840|4480 840!4480 840|4480 2700

Denna koordinathantering gör det svårare att rita rektangeln med HTML5-canvas speciellt om rektangeln skall vara ifylld med en färg. HTML5 kommer att tolka varje flytt av markören som ett nytt bildobjekt och därmed kommer inte någon färg att fylla rektangeln eftersom HTML5 inte uppfattat att en rektangel ritats, istället tolkas det som att enskilda linjer ritats. HTML5-canvas- och EaselJS funktioner som finns tillgängliga kräver att en färg för fyllnad av ett bildobjekt väljs innan själva bildobjektet börjar ritas. Detta görs med funktionerna nedan.

```
shape.graphics.beginFill(fillColor);
shape.graphics.endFill();
```

Detta var i synnerhet besvärligt när cirklar skulle ritas. De blev inte korrekt ifyllda (kap 6.3.3).

7.1.7 Användning av rekursion

Implementering med rekursion av denna konverterare var med facit i hand inte någon bra ide. Det gick till en början väldigt bra då bildobjekten var enkla, men ju fler detaljer bilderna använde sig av desto svårare blev det att hitta och korrigera fel som uppstod när bilder inte blev korrekt ritade i canvasytan. Mycket tid lades ner på att ändra och lägga till kod som sökte efter olika noders namn och dess attribut.

I den rekursiva implementeringen lagrades alla värden och attribut i olika arrayer som fick följa med som inparametrar till rit-klassen `Draw` och dess funktioner. Det kunde bli svårt att hålla reda på alla dessa arrayer efter ett tag.

Övergången till objektorienterad implementering var inte helt smärtfri men genast märktes hur pass mycket enklare det var att ändra i koden eller lägga till attribut och värden som skulle eftersökas. Flera klasser implementerades och klassernas objekt kunde innehålla andra klassers objekt. Vid utskrift (kap 11.1.1) är det enkelt att få en överblick över vilka värden de olika objekten lagrat.

Rekursion passar bättre till ett mindre sök-träd än till ett så stort och dynamiskt som används i detta arbete.

7.1.8 Reflektioner

När man läser om Flash och HTML5 kan man ana sig till en förbittring från Adobes sida eftersom HTML5 kan vara på god väg att konkurrera ut Flash helt och hållet. Den numera avlidne Applegrundaren Steve Jobs gav i april 2010 ut ett brev [2] med sina tankar om varför Flash inte används på flera av Apples största produkter. I brevet framhävs flera nackdelar med Flash, Flash var skapat för PC där en mus används för navigering och inte skapat för pekskärmar där fingrar används för navigering till att Flash-applikationer är den främsta anledningen till att Apples datorer kraschar. Flash behövs enligt Steve Jobs helt enkelt inte längre och är förlegat.

Företrädare för Adobe har vid flera intervjuer kritiserat Apple och har antytt att det är Apples fel att Flash för mobiltelefoner misslyckats [4]. Det har gått så långt att vissa medier kallar konflikten för ”det kalla kriget mellan Adobe och Apple” [3]. Ett resultat av det ”kalla kriget mellan Adobe och Apple” var en av anledningarna till att Adobe beslutade att lägga ner utvecklingen för Flash till mobiltelefoner [1].

Detta ”kalla krig” kan enligt min personliga åsikt vara ett av skälen till varför det inte finns någon dokumentation över xfl.

Eftersom Adobe nyligen släppte sin nya version CS6 med möjlighet till konvertering till HTML5 är min uppfattning att Adobe inte vill släppa någon dokumentation av xfl för att hindra konkurrenter att färdigställa en liknande konverterare. Det är även min uppfattning att Adobe gjort xfl så komplicerat som möjligt för att ytterligare försvåra för konkurrenter att förstå sig på xfl och därmed skapa en konverterare till HTML5.

8 Framtida utvecklingsmöjligheter

Skäl till varför vissa bildobjekt misslyckats vid konvertering samt tips på hur de kan lösas av utvecklare som ska fortsätta med implementering av en konverterare finns i detta kapitel.

8.1 Framtida arbete

8.1.1 Gradienter

Hantering av gradienter sker i noden `LinearGradient` (kap 4.1.2). Detta är något som absolut behöver implementeras för att en konverterare skall kunna bli färdigställd. Problemet är att det inte ur xfl- formatets kod kan fastställas en start och en slutpunkt för gradienten. Utan dessa punkter kan man inte slutföra implementering av gradienter på ett fungerande sätt.

Försök att ur matrixnoden som finns i `LinearGradientnoden` tolka vad som kunde vara start och slutpunkt gjordes utan framgång. I nuläget finns inte någon lösning till hur gradienterna skall implementeras. Jag har även skapat en tråd i forumet stackoverflow dock utan att erhålla användbara tips [9].

8.1.2 Cirklar och problem med `fillColor`

En typ av bildobjekt som misslyckades var cirklar (kap 6.3.3). Det var tidsbrist och arbetets stora omfattning som var huvudskälen till att detta inte implementerades korrekt.

Tips på lösning av cirklar med ifylld färg

När cirklar ska ritas med en ifylld färg får enbart markörflytt inträffa när bildobjektet har ritats klart. Det tolkas av HTML5 som att ett bildobjekt tagit slut varje gång markören enbart flyttas vilket innebär att bildobjektet inte kommer att fyllas med en färg på ett korrekt sätt.

!5533 2311

Enbart när ett bildobjekt ritats färdigt får markören flyttas utan att ritas för att kunna rita ett nytt bildobjekt. Kapitel 4.1.2 tar upp ämnet markörflytt, kapitel 5.1.11 och kapitel 7.3.4 tar upp ämnet om hur HTML-canvas fyller ett bildobjekt.

En implementation bör hantera att om markören mitt under ett bildobjekts gång flyttar utan att rita så ska denna handling förbises.

8.2 Adobe släpper Flash CS6

8.2.1 Flash CS6 export till HTML5

Just när detta arbete lider mot sitt slut och denna rapport skrivs släpper Adobe sin senaste version av Flash. Flash CS6 innehåller möjligheten att ladda ner ett kostnadsfritt programtillägg som verkar kunna konvertera något eller några av Flash egna filformat till att kunna användas med HTML5-canvasobjekt. Om det är xfl:s filformat som konverteras är ännu oklart men om det skulle vara fallet kan det vara en av de olika förklaringarna till att någon dokumentation av xfl inte har släppts av Adobe. Den 23 april 2012 släpptes Flash CS6 [16].

Med tanke på att CS6 kan konvertera Flash till HTML5 så är det mer lönsamt för ett företag som skulle behöva en konverterare att betala en uppgraderingskostnad för CS6 än att lägga ner tid och pengar på att skapa en egen konverterare.

Det krävs stort arbete och mycket tid för att utan dokumentation av xfl kunna slutföra en konverterare.

9 Terminologi

9.1 Olika termer

9.1.1 Bezierkurvor

En typ av kurva på parameterform som används inom datorgrafiken för att konstruera jämna kurvformer.

9.1.2 Fill color

fillColor hänvisar till färgen som ska fylla ett bildobjekt.

9.1.3 Stroke color

strokeColor hänvisar till färgen som ska omge ett bildobjekt. Exempelvis färger på en ritad rektangels linjer.

9.1.4 Hypertext

En text där läsaren kan navigera med hjälp av länkar för att läsa om det som intresserar denne. Behöver därmed inte läsa hela texten för att komma till det stycke som denne finner av intresse.

9.1.5 Klasser i rapporten

Eftersom klasser inte används i Javascript och istället klassliknande funktioner implementeras har det i denna rapport valts att kalla dessa klassliknande funktioner för klasser.

10 Referenser

[1] Adobe pausar utveckling för mobiler 2012-04-20

http://www.reuters.com/article/2011/11/10/us-adobe-apple-idUSTRE7A968A20111110

[2] Steve Jobs tankar om flash 2012-04-20

http://www.apple.com/hotnews/thoughts-on-flash/

[3] Kalla kriget mellan Adobe och Apple 2012-04-20

http://tech.fortune.cnn.com/2010/01/29/behind-the-adobe-apple-cold-war/

[4]Adobe anklagar Apple 2012-04-20

http://www.idg.se/2.1085/1.415870/apples-fel-att-flash-misslyckades

[5] Stackoverflow xfl koordinathantering 2012-02-05

http://stackoverflow.com/questions/4077200/whats-the-meaning-of-the-non-numerical-values-in-the-xfls-edge-definition

[6] w3schools HTML5 2012-02-02

http://www.w3schools.com/html5/default.asp

[7] w3schools Javascript 2012-02-02

http://www.w3schools.com/js/default.asp

[8] w3schools XML 2012-02-03

http://www.w3schools.com/xml/default.asp

[9] Författarens tråd om GradientEntry 2012-04-23

http://stackoverflow.com/questions/10280530/how-to-use-gradiententry-in-xfl

[10] Dokumentation jQuery 2012-02-06

http://docs.jquery.com/Main_Page

[11] EaselJS dokumentation 2012-02-08

http://www.createjs.com/Docs/EaselJS/

[12] CoffeeScript 2012-02-27

http://coffeescript.org/

[13] Adobe om matrix för Actionscript 2012-03-30

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/geom/Matrix.html

[14] Doug Crockfords föreläsningar om Javascript 2012-02-15

http://www.youtube.com/watch?v=JxAXlJEmNMg

http://www.youtube.com/watch?v=RO1Wnu-xKoY&feature=relmfu

http://www.youtube.com/watch?v=ya4UHuXNygM&feature=relmfu

[15] Windows7 Flash to HTML5 converter 6.4 2012-05-03

http://www.windows7download.com/win7-flash-to-html5-converter/seogqprq.html

[16] Abode Flash CS6 2012-05-03

http://www.adobe.com/se/products/flash/flash-to-html5.html

[17] Bezier kurvor 2012-03-07

http://www.tsplines.com/resources/class_notes/Bezier_curves.pdf

[18] Google Swiffy 2012-05-03

http://www.google.com/doubleclick/studio/swiffy/faq.html

[19] W3C

http://www.w3.org/

[20] Adobe Flash

http://www.adobe.com/products/flashplayer.html?promoid=ISMSA

11 Bilagor(Appendix)

11.1 Objekts bild

11.1.1 Bild av innehållet i en `NodeIterator`.