Technologie SVG a HTML5 objekt Canvas jako perspektivní metody vkládání dynamické grafiky do www stránek

SVG technology and HTML5 Canvas object as perspective methods for inserting dynamic graphics into www pages

Bakalářská práce
Tomáš Trantýr
Vedoucí bakalářské práce: PaedDr. Petr Pexa
Jihočeská univerzita v Českých Budějovicích
Pedagogická fakulta
Katedra informatiky
Rok 2011/2012

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tomáš TRANTÝR**
Osobní číslo:	**P08280**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie ve vzdělávání**
Název tématu:	**Technologie SVG a HTML5 objekt Canvas jako perspektivní metody vkládání dynamické grafiky do www stránek**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

Cílem bakalářské práce bude porovnat techniku pro vkládání dynamické bitmapové grafiky na webové stránky pomocí nového HTML5 objektu Canvas s alternativní technologií SVG, která je primárně určena pro vkládání grafiky vektorové. Do nedávné doby byly možnosti vložení grafiky do webového prostředí značně omezené, weboví designéři měli pouze dvě varianty - vložit klasický statický obrázek, nebo si pomoci např. Adobe Flash pluginem. S existencí nových moderních verzí webových prohlížečů bude ale již možné častěji využívat vektorový formát SVG a bitmapový Canvas, který je součástí nového standardu HTML5. Výstupem práce bude sada originálních praktických příkladů, analýza výhod a nevýhod obou technologií, otestování podpory v aktuálních verzích webových prohlížečů a popsány i možnosti kombinace obou zmíněných variant.

Rozsah grafických prací:	**CD ROM**
Rozsah pracovní zprávy:	**60**
Forma zpracování bakalářské práce:	**tištěná**
Seznam odborné literatury:	**viz příloha**

Vedoucí bakalářské práce:	**PaedDr. Petr Pexa** Katedra informatiky
Datum zadání bakalářské práce:	**12. dubna 2011**
Termín odevzdání bakalářské práce:	**30. dubna 2012**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 8. dubna 2010

Příloha zadání bakalářské práce

Seznam odborné literatury:

1. **HTML5: Up and Running [online]. First Edition. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'REILLY — Google Press, 2010 [cit. 2011-03-31]. Dostupné z WWW: <http://oreilly.com/catalog/9780596806033>. ISBN 978-0-596-80602-6.**

2. **LUBBERS, Peter; ALBERS, Brian; SALIM, Frank. Pro HTML5 Programming : Powerful APIs for Richer Internet Application Development [online]. Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013 : Apress, 2010 [cit. 2011-03-31]. Dostupné z WWW: <http://www.apress.com/book/view/1430227907>. ISBN 978-1-4302-2791-5.**

3. **MALÝ, Martin. Zdroják [online]. 27. 9. 2010 [cit. 2011-03-31]. SVG, nebo Canvas? Vyberte si. Dostupné z WWW: <http://zdrojak.root.cz/clanky/svg-nebo-canvas-vyberte-si/>.**

4. **W3schools [online]. c2011 [cit. 2011-04-02]. Dostupné z WWW: <http://www.w3schools.com/svg/svg_inhtml.asp>.**

5. **W3C. W3C [online]. 2010, 30 March 2011 [cit. 2011-04-02]. Dostupné z WWW: <http://dev.w3.org/html5/spec/Overview.html>.**

6. **ROWELL, Eric. Html5CanvasTutorials [online]. c2011 [cit. 2011-04-02]. Dostupné z WWW: <http://www.html5canvastutorials.com/>.**

7. **JENKOV, Jakob. Jenkov [online]. 2009 [cit. 2011-04-02]. Dostupné z WWW: <http://tutorials.jenkov.com/svg/index.html>.**

8. **DAILEY, David. W3C [online]. c2010 [cit. 2011-04-02]. Dostupné z WWW: <http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html>.**

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské/diplomové práce, a to v nezkrácené podobě pedagogickou fakultou elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích, dne 20. dubna 2012

Tomáš Trantýr

Anotace

V mé bakalářské práci se zabývám porovnáním technologie SVG a HTML5 objektu Canvas pro vkládání dynamické grafiky na webové stránky. Díky neustálému vývoji prohlížečů a především díky užitným hodnotám dynamických prvků začínají být tyto techniky stále více podporovány a využívány. Má práce by měla objasnit principy tvorby dynamických grafických prvků pomocí obou zmiňovaných metod, přehledně analyzovat jejich výhody a nevýhody a také vhodnost použití v reálných situacích. Součástí je také sada originálních příkladů demonstrující použití SVG a Canvasu v praxi, což by mělo čtenáři ještě více přiblížit obě prostředí a jejich funkcionalitu. Rovněž byla otestována funkčnost obou technologií v aktuálních verzích nejrozšířenějších webových prohlížečů.

Klíčová slova: vektor, bitmapa, rastr, SVG, HTML5, Canvas, JavaScript

Abstract

In my thesis, I deal with comparison of SVG technology and HTML5 Canvas object for insertion of dynamic graphics on web pages. Thanks to continuous development of web browsers and especially thanks to the usability of dynamic elements, those techniques are being supported and used increasingly among the Internet. My work describes principles of creation of dynamic graphic elements with use of both methods mentioned earlier, clearly analyse their advantages and disadvantages as well as the suitability of utilization in real situations. Also a set of original examples demonstrating the use of SVG and Canvas in practice is a part of my work, which should bring the audience even closer to those environments and their functionality. The functionality of both technologies in current versions of wide spread web browsers was tested as well.

Keywords: vector, bitmap, raster, SVG, HTML5, Canvas, JavaScript

Poděkování

Poděkovat bych chtěl především panu PaedDr. Petru Pexovi za jeho vstřícný přístup, trpělivost, odborné rady a poskytnutou zpětnou vazbu.

Rovněž bych chtěl poděkovat všem, kteří mi umožnili studium na Jihočeské univerzitě.

Obsah

1 Úvod

S grafickými prvky se na webu setkáváme již velice dlouho. S neustálým zdokonalováním internetových prohlížečů vznikají nové technologie usnadňující práci s grafickými formáty a především nabízející nové možnosti jak využívat dynamických možností vložené grafiky.

HTML5 objekt Canvas a SVG jsou technologie dovolující vytvářet bohatou grafiku na webové stránce, ale jsou od sebe značně odlišné. SVG je vektorový grafický formát založený na XML, naproti tomu HTML5 Canvas je univerzální JavaScriptové aplikační prostředí dovolující kódovat programové kreslící operace.

1.1 Současná situace

Když se porozhlédneme na webu, zjistíme, že míra využití těchto moderních technologií není nijak vysoká. Bohužel se ještě stále často setkávám například se známými sloupcovými grafy z prostředí Microsoft Excel vloženými jako obyčejný statický obrázek. V takovém obrázku samozřejmě není možné měnit žádné z měřených hodnot, porovnávat či jinak vyhodnocovat zdrojová data, a pokud chceme nějaká porovnání demonstrovat, je nutné vložit těchto grafů několik. S tím samozřejmě klesá i přehlednost, protože orientace v několika třeba i desítkách téměř stejných grafů je obtížná. Naproti tomu dynamický graf s možností volby vstupních dat, který je vyhodnocuje prakticky okamžitě, mně osobně vyhovuje daleko více.

1.2 SVG

Scalable Vector Graphics byl představen již v roce 1999. Jedná se o vektorový grafický formát založený na technologii XML. Dnes nachází na webu využití především jako interaktivní prvek, například uživatelské rozhraní (GUI – Graphical User Interface) ve webových aplikacích. Webem však pole působnosti tohoto formátu nekončí. Dočkal se takového rozšíření, že se z něj stal jakýsi standard pro přenášení vektorové grafiky napříč různými platformami a aplikacemi. Velmi známý je například zdarma dostupný SVG editor InkScape.

1.3 Canvas

Párový HTML tag Canvas se objevil nově ve specifikaci formátu HTML5. Jedná se vlastně o prostředí, pomocí něhož a sady JavaScriptových funkcí můžeme vkládat dynamické grafické prvky do webových stránek.

Veškerá funkčnost Canvasu je tedy prováděna pomocí JavaScriptového API (Application Programming Interface), které implementuje mj. funkce pro vykreslování základních geometrických tvarů.

2 Cíle práce

Cílem mé práce je porovnání rozdílného přístupu k dynamické grafice na webu vkládané pomocí technologií SVG a HTML5 objektu Canvas. Díky tomu, že každá z obou zmiňovaných technologií disponuje jinými možnostmi použití je i zvolení té správné velice důležitým rozhodnutím zásadně ovlivňujícím očekávaný výsledek.

Výstupem práce bude sada originálních praktických příkladů, na kterých budu demonstrovat možnosti obou formátů. Vzhledem k rozdílnosti SVG a Canvas očekávám, že příklady budou jasně znázorňovat vhodnost použití té dané technologie pro konkrétní situace.

Dalším cílem je analýza výhod a nevýhod obou technologií, možnosti jejich kombinace a otestování podpory v aktuálních verzích webových prohlížečů, zejména se zaměřím na následující: Opera, Mozilla Firefox, Safari, Internet Explorer, Google Chrome.

2.1 Metodika vypracování

Vzhledem k povaze mé bakalářské práce byla velká většina mých znalostí čerpaná z Internetu. Podíval jsem se na již hotové praktické ukázky na webových stránkách. Ze svých zkušeností vím, že pokud návštěvníkovi na webu nabídnu něco takzvaně navíc oproti konkurenci, bude mít důvod na stránce zůstat déle a ideálně ji navštívit v budoucnosti znovu. Z tohoto hlediska se zdá dynamická grafika na webu ideálním lákadlem.

Jelikož zdrojů v češtině se vyskytuje poměrně malé množství, čerpal jsem především ze zahraničních materiálů psaných v angličtině. Zpočátku jsem si obě metody tvorby grafiky vyzkoušel, vytvořil několik praktických ukázek a otestoval jejich funkčnost. Jakmile jsem považoval své nabyté zkušenosti za dostatečné, začal jsem popisovat rozdíly v přístupu obou variant k dynamické grafice.

V druhé části své práce prezentuji okomentované praktické příklady a důvody volby konkrétní technologie pro danou situaci.

Třetí část práce je analýza výhod a nevýhod obou technologií, ze které vyplynula vhodnost použití pro konkrétní reálné situace.

3 Vektorová a bitmapová grafika

Abychom dobře pochopili důvod používání vektorové grafiky na webu, je nutné vysvětlit si rozdíl mezi vektorovým a bitmapovým datovým typem obrázku. Je vhodné zmínit si i výhody a nevýhody obou typů, jejich typické použití a nejrozšířenější datové formáty používané pro ukládání takových souborů.

3.1 Obecné aspekty

Vektorová a bitmapová grafika je v zásadě odlišná popisem obrazových informací.

Bitmapový (někdy také označovaný jako rastrový) obrázek je obyčejně generován fotoaparátem, skenerem či jiným digitalizačním zařízením. Takové zařízení se prakticky vždy snaží zachytit co nejvěrohodnější realitu. Naproti tomu vektorové obrázky vznikají prakticky vždy pomocí nějakého grafického programu, typickými zástupci mohou být Adobe Illustrator či Corel Draw. Vektorová grafika je používána především pro vytváření abstraktních obrázků, příkladem může být logo, billboard, leták či jiné reklamní poutače.

Technicky jsou oba formáty zcela odlišné. Výsledný obraz může ale vypadat velice podobně a často se setkáme s výtvory, kdy je velice těžké na první pohled určit, zda se jedná o rastrový či vektorový obraz.

Obrázek 1 - fotorealistický vektorový obrázek[1]

[1] Autor: Yukio Miyamoto, zdroj www.prepressure.com

3.2 Převod bitmapy na vektor

Bitmapový obrázek je možné převést na vektorový pomocí techniky zvané trasování, anglicky tracing.

Obrázek 2 - vlevo vektor, vpravo originální rastr

Obrázek 2 ukazuje výsledek vektorizace bitmapového originálu pomocí programu Adobe Illustrator. Výsledek je výborný, nebýt stínu na bílém plastu kapotáže motocyklu, těžko bychom v tomto rozlišení hledali výraznější rozdíl. To nás ale dovádí k faktu, že proces vektorizace je vždy doprovázen ztrátou informace a výsledný vzhled obrázku tak přímo závisí na kvalitě použitého nástroje.

Vektorizace je hojně používána především v oboru geologie, kdy jsou územní mapy převáděny do vektorové podoby pro následný import do grafických programů (typicky AutoCAD).

Můžeme se setkat se třemi typy vektorizace:

1) Automatická, která je prováděna bez zásahu operátora a je velmi závislá na kvalitě předlohy.

2) Poloautomatická vektorizace je náročnější na obsluhu, jsou nutné ruční zásahy do procesu. Používá se například při převodu vrstevnicových plánů do vektorové podoby.

3) Ruční vektorizace je zcela manuální, jedná se o obtažení originální předlohy. Časově je samozřejmě nejnáročnější.

Vektorizace je hojně využívaná v odvětví digitalizace dokumentů, kdy jsou jednotlivé naskenované předlohy dále zpracovávány technikou OCR[2] a uloženy v textovém formátu. Tím je umožněna další práce s dokumentem, jeho úpravy či vyhledávání zadaných řetězců znaků.

3.3 Převod vektoru na bitmapu

Stejně jako lze konvertovat bitmapový obrázek na vektorový, je možné provést i obrácený proces zvaný rasterizace.

Rasterizace se používá například při potřebě převodu trojrozměrného modelu do standardní 2D podoby a jeho následující zobrazení na monitoru či odeslání na tiskové zařízení.

3.4 Bitmapová (či pixelová) grafika

Bitmapový obrázek je definován jednotlivými pixely, které jsou jednoznačně určeny jejich polohou, barvou (RGB), případně barvou a průhledností (RGBa), přičemž malé písmenko „a“ označuje „alpha channel“, jinými slovy barevný kanál popisující průhlednost daného pixelu. Typickým zástupcem bitmapové grafiky, se kterým se setkáváme například na poli digitální fotografie, je formát JPEG. Důvodem, proč se pro digitální fotografie nevolí vektorový formát, je prostý fakt, že na fotografiích nenalezneme pravidelně se opakující tvary, které by bylo možné definovat geometrickými obrazci. Proto je nutné (a v tomto případě i výhodné) každý jednotlivý pixel obrázku popsat zvlášť.

[2] Optical Character Recognition – optické rozpoznání znaků

Obrázek 3 - Bitmapový obrázek extrémně zvětšený

Obrázek 3 ukazuje skutečnou podobu rastrových souborů. Každý čtvereček, tedy pixel, má přesně danou svou polohu a barvu. Zde se v praxi uplatňuje nedokonalost lidského oka, a pokud má obrázek dostatečně vysoké rozlišení, jednotlivé pixely nevnímáme.

3.4.1 Formát JPEG

Takové obrázky jsou obyčejně do webových stránek vkládány pomocí tagu <img> a nejčastěji se jedná o formáty JPEG, GIF či PNG. Formát JPEG se, jak jsem již psal, nejčastěji používá pro vkládání fotografií a je vždy kompresní a zároveň ztrátový. I při nastavení nejvyšší možné kvality je tedy určité množství informace ztraceno, typicky při převodu z jiného, bezztrátového, ale podstatně objemnějšího formátu (např. TIFF). Za tuto daň nás ale formát JPEG odmění výborným poměrem datového objemu a kvality obrázku. Modernějším formátem, který ještě bohužel ale nemá tak dobrou podporu ze strany softwarových vývojářů a je tedy problematické použít ho jako univerzální grafický formát, je JPEG2000. Proti staršímu JPEG formátu dosahuje ale ještě lepšího poměru objemu dat a kvality, a navíc umí být i bezztrátový. Bohužel enkodéry pro tento typ datového souboru jsou pomalejší a více energeticky náročné, což je zároveň důvod, proč JPEG2000 není k vidění v digitálních fotoaparátech i přes jeho nesporné výhody.

3.4.2 Formáty PNG a GIF

Grafický formát PNG je dokonalejším nástupcem GIFu. GIF používá podobně jako TIFF kompresní algoritmus LZW (Lempel–Ziv–Welch). PNG využívá k bezztrátové kompresi metody DEFLATE, podporuje alpha channel (kanál průhlednosti pixelu) a téměř ve všech případech dosahuje nižšího datového objemu než stejný obrázek ve formátu GIF. To jsou i důvody, proč se s tímto formátem napříč internetovými stránkami setkáváme častěji a proč je webová grafika vytvářena převážně v PNG. Problém je jen u velice starých prohlížečů (typicky Internet Explorer 6), které si s průhledností PNG neporadí a mohou tak nejednomu nedostatečně informovanému programátorovi učinit horké chvíle.

PNG přišel s drastickým zlepšením v možnosti počtu barevných odstínů, které dokáže pojmout. Starší GIF umí 8-bit přesnost, tedy 256 barev, PNG dokáže až 16.7 miliónů odstínů, jinak také označovaných jako 24-bitová barevná hloubka[3].

Formáty GIF a PNG jsou používány výhradně pro abstraktní grafiku, jejich použití pro fotografické potřeby není vhodné.

3.5 Vektorová grafika

Naproti tomu vektorový obrázek se neskládá z pixelů, ale ze základních geometrických obrazců jako jsou čáry, kružnice a křivky. Tyto obrazce jsou určeny vždy minimálně počátečním bodem, dále směrem (čáry), poloměrem (kružnice), případně několika body (křivky). Díky tomu, že jsou jednotlivé tvary popsány obecně, nikoliv na úrovni jednotlivých pixelů, je možné je libovolně zvětšovat, aniž by se tato transformace podepsala negativně na jejich kvalitě.

Tedy vektorová grafika jsou obecně obrázky vytvořené na základě matematického popisu, který určuje pozici, délku a směr, kterým jsou jednotlivé čáry vykresleny.

[3] 24-bitová barevná hloubka je někdy označována také jako truecolor

Obrázek 4 - původní obrázek

Obrázek 5 - původní obrázek zvětšený, vlevo bitmapa, vpravo vektor

Jak je vidět z ilustračních obrázků, zvětšení bitmapového obrázku způsobí velmi znatelnou degradaci původní kvality, zatímco vektorový obrázek si svou kvalitu plně zachoval i po více než sedminásobném zvětšení.

4 Výhody použití vektorové grafiky

4.1 Možnost zvětšení

Vektorové obrázky, případně jejich část, mohou být bez obtíží zvětšeny se zachováním stoprocentní kvality výstupu. Zvětšování bitmapového obrázku oproti tomu vede k rozmazaným okrajům a ke vzniku tzv. obrazových artefaktů.

4.2 Menší velikost souboru

Ve vektorové grafice jsou ukládány pouze matematické koordináty vlastních vektorů, ne však jednotlivé pixely. Díky tomu mohou být takové obrázky menší a méně náročné na médium zajišťující přenos dat.

4.3 Nezávislost na výstupním zařízení

Vektorové obrázky jsou na výstupu vždy ostré a je možné je vytisknout v jakémkoli rozlišení je potřeba, ať už je to 300 či 1200 dpi.

5 SVG

5.1 SVG jako skriptovaná grafika

Jelikož SVG je založeno na jazyce XML, může tak být grafika tohoto typu vytvářena „on the fly“ či „real-time“, tedy volně přeloženo „za běhu“ nebo „v reálném čase“ pomocí jakéhokoliv skriptovacího jazyka (např. Javascript či Java samotná), s prakticky libovolným zdrojem dat, textovým souborem počínaje a SQL databází konče. Díky tomu mohou být jednotlivé obrázky dynamicky upravované aktuálně dostupnými daty. Příkladem může být interaktivní mapa ilustrující aktuální dopravní situaci na frekventovaných silnicích či dálnicích, meteorologický snímek demonstrující oblačnost a mnoho dalšího.

5.2 SVG jako interaktivní grafika

SVG může být jednoduše animováno, interaktivní a stylováno pomocí tzv. stylesheetů, tedy textových souborů obsahujících informace popisující vzhled dokumentu. SVG je rovněž kompatibilní s řadou dalších standardů založených na jazyku XML, jako jsou DOM (Document Object Model) dobře známý z Javascriptu, či SMIL (Synchronised Multimedia Integration Language) používaný pro multimediální prezentace, popisující časování, rozvržení, animace a jiné.

5.3 Vkládání SVG do webových stránek

SVG grafika může být do webových stránek vkládána několika metodami.

5.3.1 Vložení pomocí <embed> tagu

```
<embed src="obrazek.svg" type="image/svg+xml" />
```

Příklad 1 – vložení pomocí <embed> tagu

Výhoda způsobu využití embed tagu je v tom, že je podporován napříč nejrozšířenějšími prohlížeči a dovoluje skriptování.

Nevýhoda je porušená validita stránky v případě použití v HTML4 a XHTML. Je ale součástí specifikace HTML5.

5.3.2 Vložení pomocí <object> tagu

```
<object data="obrazek.svg" type="image/svg+xml"></object>
```

Příklad 2 - vložení pomocí <object> tagu

Tag <*object*> je součástí specifikace HTML4, XHTML i HTML5 a je podporován všemi nejrozšířenějšími webovými prohlížeči.

Nevýhodou tohoto použití je nemožnost skriptování.

5.3.3 Vložení pomocí <iframe> tagu

```
<iframe src="obrazek.svg"></iframe>
```

Příklad 3 - vložení pomocí <iframe> tagu

Tag <*iframe*> je podporován všemi nejrozšířenějšími prohlížeči.

Jeho použití je ale problematické, protože vytváří orámovaný objekt uvnitř stránky bez možnosti stylování, záleží tedy vždy na implementaci v konkrétním prohlížeči. Rovněž generuje validační chyby v případě použití uvnitř webové stránky napsané v HTML4 Strict a XHTML Strict.

5.3.4 Vložení SVG kódu přímo do webové stránky

```
<html>
 <body>
 <svg xmlns="http://www.w3.org/2000/svg" version="1.1">
   <rect x="10" y="10" width="20" height="50" />
 </svg>
 </body>
 </html>
```

Příklad 4 - vložení SVG kódu přímo do webové stránky

Velká pětka prohlížečů[4] v aktuálních verzích dokáže SVG kód správně zobrazit i v případě, že je vložen přímo do webové stránky.

5.3.5 Odkaz na externí SVG soubor

```
<a href="obrazek.svg">SVG obrázek<a>
```

Příklad 5 - vložení SVG pomocí odkazu na externí soubor

Posledním způsobem jak vložit SVG soubor na webovou stránku je pomocí odkazu na externí soubor.

5.4 Přehled aktuálních SVG editorů

Dnes se vyskytuje na trhu velké množství editorů podporujících SVG, proto vybrat několik zástupců je poměrně problematické. Seznam jsem rozdělil na volně dostupné a komerční produkty.

5.4.1 Volně dostupné SVG editory ve formě desktopové aplikace

1) Inkscape – asi nejznámější SVG editor poskytovaný zcela zdarma. Díky nativní podpoře SVG poskytuje celou řadu pokročilých funkcí. Autoři prohlašují, že jejich aplikace má podobné schopnosti jako komerční Adobe Illustrator či CorelDraw.[5]

2) Sketsa SVG Editor[6] – Další volně dostupná aplikace s nativní podporou SVG. Umožňuje export či rasterizaci do formátů JPEG, PNG a TIFF. Sketsu je dále možné vylepšovat různými pluginy.

3) Amaya[7] – komplexní (nejen) SVG editor od tvůrců z řad konsorcia W3C. Původně začal jako HTML+CSS editor, v průběhu času byl ale rozšířen o podporu různých XML jazyků jako např. XHTML, MathML a SVG.

[4] Internet Explorer, Google Chrome, Opera, Firefox, Safari

[5] Zdroj - http://inkscape.org

[6] Dostupný z http://www.kiyut.com/products/sketsa/download.php

5.4.2 Volně dostupné online SVG editory

1) ScriptDraw[8] – online SVG editor vyvinutý pomocí moderní technologie Adobe Flex.

2) svg-edit – Dle slov autorů se jedná o rychlý webový SVG editor založený na javascriptové technologii, který funguje v jakémkoliv moderním prohlížeči[9].

5.4.3 Komerční SVG editory

1) Oxygen xml editor – Asi nejznámější profesionální aplikace pro úpravu (nejen) XML formátů je Oxygen xml editor. Mimo celé řady dostupných nástrojů obsahuje i modul pro editaci a zobrazení SVG souborů.

5.5 Základní kostra SVG dokumentu

V úvodu jsme si již říkali, že SVG je formát založený na XML technologii (Extensible Markup Language). To znamená, že veškeré grafické objekty jsou popsány textem.

Takto vypadá hlavička SVG dokumentu:

```
<?xml version="1.0" standalone="no"?>
<svg width="300" height="200"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
     <!-- vlastní kód -->
</svg>
```

Příklad 6 - hlavička SVG dokumentu

Jelikož SVG je aplikace XML formátu, měla by hlavička dokumentu vždy obsahovat úvodní deklaraci XML jako zde na prvním řádku.

---

[7] Dostupný z http://www.w3.org/Amaya/User/BinDist.html

[8] Dostupný z http://scriptdraw.com/

[9] Zdroj - http://code.google.com/p/svg-edit/

<svg> tag na druhém řádku říká prohlížeči, že se jedná o SVG dokument. Viditelná část SVG dokumentu je definována parametry *width* a *height*. Pokud bychom tyto dva parametry nedefinovali, SVG obrázek by zabral celé okno prohlížeče.

Třetí řádek definuje XML namespace použitý pro tento dokument.

Následuje veškerý SVG obsah, který musí být uvnitř tagů *<svg> </svg>*.

5.6 První příklad

Jelikož už víme, jak vypadá hlavička SVG dokumentu a co znamenají jednotlivé části, můžeme si ukázat jednoduchý obrazec vytvořený krátkým kódem.

```
<?xml version="1.0" standalone="no"?>

<svg width="300" height="200"

     xmlns="http://www.w3.org/2000/svg" version="1.1">

     <circle cx="50" cy="50" r="30" fill="#ff0000"  stroke="#000000"
     stroke-width="3">

     <title>Prvni obrazek</title>

     <desc>Kruh</desc>

     </circle>

</svg>
```

Příklad 7 - náš první SVG obrázek

Tento jednoduchý kód nám nakreslí přesně takový kruh, jaký je obsažen v ilustraci Obrázek 4 - původní obrázek. Rozeberme si tedy nejdříve nové části kódu, které nám přibyly.

5.6.1 Elementy <title> a <desc>

SVG dokument umožňuje jakémukoliv elementu přidělit element *<title>* a *<desc>*. Tyto popisné informace nejsou nutně viditelné – záleží na implementaci SVG procesoru, který se stará o zobrazování SVG dokumentů.

5.6.2 Element <circle>

Element <*circle*> je náš první grafický element, který se stará o zobrazení kruhového obrazce. Podíváme se na jednotlivé atributy popisující náš kruh.

Povinné atributy:

- *cx* – vzdálenost středu kruhu od levého okraje dokumentu
- *cy* – vzdálenost středu kruhu od horního okraje dokumentu
- *r* – poloměr kruhu

Poznámka: Veškeré rozměry jsou definovány v pixelech

Nepovinné atributy:

- *fill* – barva výplně
- *stroke* – barva okraje kruhu
- *stroke-width* – šířka okraje kruhu

Poznámka: Barvy v příkladu jsou definovány v šestnáctkové soustavě, je však možné využít i základních odstínů jako grey, blue, red atp.

5.7 Stylování SVG dokumentů

V předchozím případě jsme použili dostupné XML atributy pro vybarvení obrázku a jeho okraje. Jak jsem se zmínil v bodě 5.1.1, SVG může být stylováno pomocí stylovacího jazyka, typicky CSS (Cascading Style Sheets). Pro tento způsob stylování máme k dispozici tři různé metody:

5.7.1 Deklarace CSS vlastnosti tzv. inline

Stejně jako v HTML či XHTML specifikaci je i zde možné vlastnosti CSS deklarovat uvnitř SVG elementu pomocí atributu *style.*

```
<circle cx="50" cy="50" r="30" style="fill: #ff0000; stroke:
#000000; stroke-width :3">
```

Příklad 8 - inline deklarace CSS vlastnosti

5.7.2 Reference na interní CSS definice

Sadu interních definicí přímo na stránce, kde se vyskytuje SVG kód, je nutné specifikovat uvnitř elementu <*defs*>.

```
<defs>
      <style type="text/css">
      circle {
            fill: #ff0000;
            stroke: #000000;
            stroke-width :3
      }
      </style>
</defs>

<circle cx="50" cy="50" r="30" />
```

Příklad 9 - reference na interní CSS definice

5.7.3 Reference na externí CSS soubor

Máme vytvořený soubor style.css s následujícím obsahem:

```
circle {
            fill: #ff0000;
            stroke: #000000;
            stroke-width :3
}
```

Příklad 10 - soubor style.css

Reference na externí CSS soubor v SVG dokumentu je umístěna pod úvodní xml tag a vypadá následovně:

```
<?xml-stylesheet href="style.css" type="text/css"?>
```

Příklad 11 - reference na externí CSS soubor umístěný mimo stránku s SVG kódem

Celý SVG dokument tedy vypadá takto:

```
<svg width="300" height="200"
xmlns="http://www.w3.org/2000/svg" version="1.1">
<?xml-stylesheet href="style.css" type="text/css"?>
<circle cx="50" cy="50" r="30" />
</svg>
```

Příklad 12 - příklad SVG dokumentu s referencí na externí CSS soubor

5.8 Seskupování objektů

SVG poskytuje párový element <*g*> , který slouží k vytváření skupin grafických objektů. Skupina pak dědí všechny atributy nebo vlastnosti definované v elementu <*g*>.

```
<g style="fill:#000000;stroke:#00ff00">
    <circle cx="50" cy="100" r="20" />
    <rect x="150" y="200" width="70" height="60" />
</g>
```

Příklad 13 - seskupování objektů pomocí elementu <g>

V tomto případě tedy aplikujeme černou výplň a zelený okraj objektům ve skupině, konkrétně kruhu a obdélníku.

5.9 Referencování objektů v SVG

SVG implementuje dva typy referencí na objekty – lokální (relativní) a nelokální (absolutní) reference. Rozdíl mezi těmito dvěma metodami je zřejmý, a sice v případě lokální reference je

odkazovaný objekt umístěn ve stejném dokumentu, kdežto v případě absolutní reference je daný objekt v jiném dokumentu.

5.9.1 Element <defs>

Element <*defs*> jsme již použili v bodě 5.4.2 pro ilustraci interního CSS stylování. Element <*defs*> je využíván k definování objektů, které budou referencovány dále v dokumentu. Objekty definovány uvnitř tohoto elementu samozřejmě nejsou ihned vykreslovány, ale je na ně později odkazováno.

```
<defs>
   <linearGradient id="lGradient">...</linearGradient>
</defs>

<circle cx="50" cy="50" r="30" style="fill:url(#lGradient)" />
```

Příklad 14 - element <defs>

Element <*defs*> je v dokumentu vždy použit pouze jednou.

5.9.2 Element <use>

Element <*use*> slouží pro „znovupoužití“ již definovaných objektů. Referencování je možné interní i externí, jediná podmínka je, že referencovaný element musí být potomkem elementu <*defs*>.

```
<defs>
   <circle id="Kruh" cx="50" cy="50" r="30"/>
</defs>
<use x="2cm" y="3cm" xlink:href="#Kruh" />
```

Příklad 15 - element <use>

5.9.3 Element <image>

Pro vkládání obrázků do SVG dokumentů se používá element <*image*>. Ten definuje obdélníkovou oblast pro vložení bitmapového obrázku (např. JPEG, PNG, GIF) nebo SVG souboru.

```
<image x="100" y="100" width="50px" height="50px"
xlink:href="vkladany_obrazek.jpeg" />
```

Příklad 16 - element <image>

5.10 Základní tvary

Jazyk SVG definuje několik základních tvarů pro vykreslování grafických objektů.

5.10.1 Obdélník

```
<rect x="x" y="y" width="width" height="height" />
```

Příklad 17 - obdélník

Obdélník je definován odsazením od levého horního rohu (parametry *x* a *y*) a jeho šířkou a výškou.

5.10.2 Kruh

```
<circle cx="cx" cy="cy" r="polomer" />
```

Příklad 18 - kruh

Příkaz *circle* vykreslí kruh definovaný velikostí poloměru a vystředěný na souřadnice cx a cy.

5.10.3 Elipsa

```
<ellipse cx="cx" cy="cy" rx="x-polomer" ry="y-polomer" />
```

Příklad 19 - elipsa

Vykreslí elipsu s danými hodnotami poloměrů opět vystředěná na souřadnice *cx* a *cy*.

5.10.4 Čára

```
<line x1="pocatecni-x" y1=" pocatecni -y" x2="konecne-x"
y2="konecne-y" />
```

Příklad 20 - čára

Čára je vykreslena pomocí souřadnic počátku (*x1, y1*) a konce (*x2, y2*).

5.10.5 Křivka

```
<polyline points="x1,y1 x2,y2 x3,y3" />
```

Příklad 21 - křivka

Křivka je vytvořena pomocí série čar určených souřadnicemi *x* a *y*.

5.11 Element <path> pro vytváření vlastních grafických objektů

Path, tedy česky cesta, je grafický objekt definovaný sekvencí příkazů. Tato cesta reprezentuje obrys objektu, který může být dále např. vyplněn barvou. Vykreslený objekt může být neuzavřený (čára), nebo uzavřený (polygon) a vytvořen z jednotlivých čar, křivek a segmentů.

Každý segment se skládá z řady příkazů, kde první určuje počáteční bod, následuje typ čáry/křivky a její konečný bod. Příklad jednoduché čáry může vypadat takto:

```
<path d="M 0 0 L 50 50" />
```

Příklad 22 - element <path>

Atribut *d* definuje cestu. Příkaz *M 0 0* určuje začátek cesty, tedy v tomto případě je začátek v bodě 0,0. Příkaz *L 50 50* pak dále vykreslí čáru (Line) do bodu 50,50.

5.11.1 Čáry

Příkazy L a M již známe, k dispozici jsou ale i další. Příkaz H vytváří horizontální čáru a přijímá jediný parametr – x-ovou souřadnici konečného bodu. Obdobně funguje příkaz V vykreslující vertikální čáru a akceptující jediný parametr ypsilonové souřadnice konečného

bodu. Posledním příkazem je Z, který jen uzavře vykreslovaný tvar rovnou čárou do počátečního bodu.

5.11.2 Křivky

Podobným způsobem jako jednoduché čáry jsou vytvářeny i různé křivky, samozřejmě s rozdílným složením vstupních parametrů.

Beziérova křivka je generována pomocí příkazu C a tří párů souřadnicových parametrů. Obecná syntaxe tedy vypadá následovně:

```
<path d="M mx,my C x1,y1 x2,y2 x,y">
```

Příklad 23 - Beziérova křivka

Parametry *mx* a *my* určují počáteční bod, *x1* a *y1* jsou kontrolním bodem na počátku křivky, *x2* a *y2* jsou kontrolní bod na konci křivky a *x, y* jsou koncové body.

Kvadratická křivka je speciální případ Beziérovy křivky, která má jeden kontrolní bod a je vytvářena příkazem Q. Syntaxe příkazu vypadá takto:

```
<path d="M mx, my Q x1,y1 x,y">
```

Příklad 24 - kvadratická křivka

5.12 Vybarvování a jiné efekty

SVG poskytuje možnost vykresleným objektům přidat barevnou či vzorkovou výplň. Toto vybarvování (anglicky „painting“) můžeme v zásadě aplikovat na vnitřek objektu (fill) a jeho vnější okraj (stroke).

K dispozici jsou opět různé způsoby výplní.

5.12.1 Výplň jedinou barvou

```
<rect width="10" height="15" style="fill:black" />
```

Příklad 25 - výplň jednou barvou

Kromě slovního označení barvy je možné využít hexadecimálního zápisu barev.

```
<rect width="10" height="15" style="fill: #00ff00" />
```

Příklad 26 - výplň jednou barvou s využitím hexadecimálního zápisu barvy

Výsledná barva je určena kombinací červených, modrých a zelených odstínů – RGB. Každý odstín je určen dvojicí znaků nebo číslic. V našem případě je výplň sytě zelená.

Posledním způsobem jak určit barvu výplně je zapsat ji pomocí hodnot světlosti jednotlivých RGB složek.

```
<rect width="10" height="15" style="fill: rgb(255, 0, 0)" />
```

Příklad 27 - výplň jednou barvou popsána světlostí jednotlivých RGB odstínů

V tomto případě jsou barvy popsány světlostí jednotlivých odstínů RGB, které mohou nabývat hodnot od 0 do 255. Výsledná barva je v našem příkladu sytě červená.

5.12.2 Tzv. gradient, česky barevný přechod (lineární a radiální)

Barevný přechod může být lineární stupňující se horizontálně nebo vertikálně, a radiální, vytvářející kruhový barevný přechod. Gradient je vždy definován uvnitř elementu <*defs*>.

```
<linearGradient id="horizontalni_linearni" x1="0%" y1="0%" x2="100%"
y2="0%" >

 <stop offset="0%" style="stop-color:red" />

 <stop offset="100%" style="stop-color:black" />

 </linearGradient>
```

Příklad 28 - lineární horizontální gradient červeno-černý

```
<linearGradient id="vertikalni_linearni" x1="0%" y1="0%" x2="100%"
y2="0%" >

 <stop offset="0%" style="stop-color:yellow" />

 <stop offset="100%" style="stop-color:blue" />

 </linearGradient>
```

Příklad 29 - lineární vertikální gradient žluto-modrý

```
<radialGradient id="radialni " cx="50%" cy="50%" r="50%" >
 <stop offset="5%" style="stop-color: green" />
 <stop offset="100%" style="stop-color: pink" />
</radialGradient>
```

Příklad 30 - radiální gradient zeleno-růžový

Atributy *x1, y1, x2, y2* u lineárních gradientů určují počáteční a koncové souřadnice přechodové výplně. Pokud se souřadnice *y1* a *y2* shodují a *x1* a *x2* odlišují, bude vytvořen horizontální gradient. Vertikální gradient je vytvořen, pokud se x-ové souřadnice shodují a y-ové liší.

Radiální barevný přechod je řízen středovými souřadnicemi *cx* a *cy* a poloměrem *r*, určujícím konečnou velikost přechodu.

Element <*stop*> slouží pro nastavení barev, které budou na počátku a konci přechodu. Každý <*stop*> element má dva klíčové atributy – *offset* a *stop-color*. První určuje bod na přechodu, kde je zobrazena definovaná barva. Druhý atribut určuje počáteční, respektive konečnou barvu přechodu.

5.12.3 Výplň vzorem

Pro vyplnění objektu (nebo dokonce jeho hrany) je možné použít i grafický vzor. Ten může být dále opakován pro vyplnění celého objektu.

Použití výplně pomocí definovaného vzoru je svázáno s několika pravidly. Výplň je nutné definovat pomocí elementu <*pattern*>. Jelikož se jedná o definici, není takový objekt rovnou vykreslován, ale je na něj dále odkazováno při skutečném použití vzoru.

```
<pattern id="vzor_obdelnik" x="0" y="0" width="20" height="25"
 patternUnits="userSpaceOnUse">
 <rect x="10" y="10" width="20" height="25"
style="fill:red;stroke:blue;"/>
 </pattern>
```

Příklad 31 - definice výplně vzorem

Atributy *x* a *y* určují souřadnice počátku výplně ve vztahu k vyplňovanému objektu. Tedy 0,0 je levý horní roh objektu, který má být vzorem vyplněn. Atributy *width* a *height* pak dále určují rozměr oblasti, kam bude každá část vzoru vložena.

Atribut *patternUnits* určuje typ souřadnicového systému pro atributy *x, y, width* a *height.*

Následně je možné pro daný objekt použít referenci na výplň pomocí atributu *style*.

```
<rect x="10" y="10" width="120" height="60"
style="fill:url(#vzor_obdelnik)" />
```

Příklad 32 - aplikace výplně pomocí vzoru

Z příkladu je tedy vidět, že mezi klíčové atributy výplně vzorem patří i její identifikátor, jelikož je na něj dále odkazováno. Zároveň z toho vyplývá, že se nemusíme omezovat jedním typem výplně, ale můžeme si jich nadefinovat celou řadu.

5.13 Filtry

Součástí specifikace SVG jsou filtry, tedy speciální efekty, které je možné aplikovat na tvary a text. K dispozici je sada primitivních filtrů, které mohou být dále kombinovány pro vytvoření komplexního efektu.

Efekt	Filtry
Smíšení, spojení	<feBlend>, <feComposite>, <feMerge>
Barevný	<feColorMatrix>, <feComponentTransfer>, <feFlood>
Speciální efekty	<feConvolveMatrix>, <feDisplacementMap>, <feTurbulence>, <feGaussianBlur>,

	<feOffset>, <feImage>, <feMorphology>, <feTile>
Nasvětlovací efekty	<feDiffuseLighting>, <feSpectularLighting>
Efekty využívající zdroje světla	<fePointLight>, <feDistantLight>, <feSpotLight>

5.13.1 Společné atributy pro většinu základních filtrů

- *x,y* – určují počáteční bod oblasti, na kterou bude filtr aplikován
- *width, height* – určují šířku a výšku oblasti, na kterou bude filtr aplikován
- *result* – přiřadí jméno filtru
- *in* – určuje vstup pro daný filtr, který může být jedním ze šesti předdefinovaných hodnot nebo může být také dříve použité jméno filtru - atribut *result*

  Předdefinované hodnoty pro vstup filtru:

  - SourceGraphic
  - SourceAlpha
  - BackgroundImage
  - BackgroundAlpha
  - FillPaint
  - StrokePaint

5.13.2 Definice a použití filtrů

Filtry jsou definovány uvnitř elementu <*defs*> a každý má své unikátní ID.

```
<defs>
 <filter id="filtr">
...
 </filter>
</defs>
 <text style="filter:url(#filtr)">Text s aplikovaným filtrem</text>
```

Příklad 33 - definice a ukázka použití filtru

Filtry samozřejmě zachovávají perfektní škálovatelnost SVG, tedy i objekt s aplikovaným filtrem je možné bez ztráty kvality zvětšovat či zmenšovat.

5.14 Možnosti transformace

Umístění objektů na stránce je v SVG definováno pomocí souřadnicového systému. Ten může být stejně jako objekty v něm určitým způsobem transformován.

Příkaz	Efekt
Translate	Posune systém souřadnic
Scale	Mění velikost objektu
Rotate	Rotuje objekt
Skew	„zkreslí“ objekt

5.15 Dynamické SVG

SVG je silný formát pro vytváření nejrůznějších animací a interaktivních aplikací. Animace je možné vytvářet prostřednictvím využití animačních elementů nebo pomocí DOM (Document Object Model).

5.15.1 Animace

Atributy dostupné přímo v SVG pro vytváření animačních efektů jsou vypsány v následující tabulce.

Atribut	Funkce
<animate>	Modifikuje určitý atribut elementu v čase.
<set>	Zkratka pro atribut <*animate*>, animuje nenumerické hodnoty.
<animateMotion>	Animace pohybem – rozpohybuje určitý element podél předdefinované cesty po určitou dobu.
<animateColor>	Barevná transformace elementu.
<animateTransform>	Aplikuje více transformací na element.

5.15.2 Interaktivita

Interaktivita SVG dokumentů může být zajištěna třemi způsoby – odkazování, události a skripty.

Odkazování je prováděno pomocí XLink a XPointer, které dovolují vytvářet hyperlinky odkazující na existující dokumenty (XLink), respektive specifické části dokumentu (XPointer).

Události jsou standardě rozděleny podle vstupního zařízení (myš, klávesnice) či změny stavu (např. stav načtení SVG souboru).

Posledním způsobem jak zajistit interaktivitu SVG dokumentu je skriptování pomocí skriptovacího jazyka (např. Javascript či Java samotná).

6 Canvas

Canvas je nový párový tag, který se objevil spolu se specifikací HTML5. Díky této značce je možné uvnitř prostoru definovaného šířkou a výškou plátna provádět programové kreslící operace pomocí skriptovacího jazyka, jmenovitě JavaScriptu. Canvas implementuje API se sadou užitečných funkcí, se kterými lze jednoduše vytvářet základní geometrické tvary. Zjevný velký rozdíl proti SVG je přístup k tvarům pouze prostřednictvím javascriptového API. Syntaxe vykreslování objektů je tak zcela odlišná, protože nevyužívá párových či nepárových XML elementů jako je tomu v případě SVG, ale primitivní javascriptové funkce.

Část W3C specifikace mluví o Canvasu takto (volně přeloženo) :

„Element poskytující skripty s bitmapovým plátnem závislým na rozlišení, který může být použit pro vykreslení grafů, herní grafiky, nebo dalších vizuálních obrazů v reálném čase.“[10]

Další pěkná specifikace je k nalezení na stránkách Wikipedia (volně přeloženo):

„Canvas se skládá z kreslící oblasti definované v HTML kódu pomocí atributů určujících šířku a výšku takové oblasti. JavaScriptový kód přistupuje k této oblasti skrze sadu kreslících funkcí podobných funkcím používaných v jiných 2D API, čímž dovoluje vytvářet dynamicky generovanou grafiku. Některé předpokládané využití Canvasu zahrnuje vytváření grafů, animací, her a obrázkových kompozicí.“[11]

6.1 Princip funkce

Abychom mohli začít vytvářet grafiku, musíme mít na stránce minimálně jeden párový element <*canvas*>. V každém tomto elementu je možné využít tzv. kontext, ve kterém lze vykonávat jednotlivé javascriptové příkazy pro vykreslení grafických objektů. Dnešní moderní prohlížeče dokáží zobrazit klasický dvourozměrný obsah elementu <*canvas*>,

[10] Zdroj - http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html

[11] Zdroj - http://en.wikipedia.org/wiki/Canvas_element#Usage

existují ale i experimentální verze prohlížečů (Opera, Firefox – podpora pomocí pluginu), které dovolují zobrazit třídimenzionální kontext canvasu.

6.2 Základní kostra HTML dokumentu s elementem Canvas

Jelikož Canvas je součástí DOM dokumentu, je potřeba nejdříve vytvořit element Canvas.

```
<canvas id="platno" width="300" height="300">
      ...
</canvas>
```

Příklad 34 - element Canvas

Je důležité definovat ID elementu, protože se na něj budeme dále v kódu javascriptu odkazovat, abychom mohli získat kontext Canvasu, díky kterému je možné provádět kreslící operace.

```
var canvas = document.getElementById('platno');
var context = canvas.getContext('2d');
context.fillRect(0, 0, 50, 50);
```

Příklad 35 - První příklad Canvas

Na prvním řádku kódu definujeme proměnnou, do které přiřadíme vytvořený Canvas element. Druhým řádkem inicializujeme kontext tohoto elementu, a na třetím řádku už provádíme první kreslící operaci.

Je důležité vědět, že pro jeden element Canvas můžeme inicializovat pouze jeden kontext, na jedné stránce však může být více elementů Canvas.

Aby nám tento příklad na stránce fungoval, je potřeba nějakým způsobem zajistit inicializaci Canvasu a jeho kontextu ihned po načtení stránky. Existuje jako ve většině případů více způsobů, zde se inicializace provádí pomocí javascriptu.

1) Před uzavírací tag *</script>* vložit následující řádek

```
window.addEventListener("load", vykreslovaciFunkce, true);
```

2) Použití funkce *window.onload*

```
window.onload = function()
```

Celý příklad tedy může vypadat takto (podle první možnosti):

```
<script type="text/javascript">
      function ctverec() {
            var canvas = document.getElementById('platno');
            var context = canvas.getContext('2d');
            context.fillRect(0, 0, 50, 50);
      }
window.addEventListener("load", ctverec, true);
</script>
```

Příklad 36 – Inicializace Canvasu pomocí EventListeneru

Inicializace Canvasu podle druhé možnosti vypadá následovně:

```
<script type="text/javascript">
      window.onload = function() {
            var canvas = document.getElementById('platno');
            var context = canvas.getContext('2d');
            context.fillRect(0, 0, 50, 50);
      }
</script>
```

Příklad 37 - inicializace Canvasu pomocí funkce window.onload

6.3 Primitivní tvary

Canvas, obdobně jako SVG, poskytuje sadu příkazů pro vykreslování primitivních tvarů.

6.3.1 Šablona zdrojového kódu

Abych nemusel stále opakovat celý kód, budu používat tuto šablonu jako statickou a měnit pouze část, kde je pomocí komentáře označeno, kam jednotlivé kreslící operace patří.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript">
window.onload = function() {
        var platno = document.getElementById("platno");
        var context = platno.getContext("2d");
        //zde budeme doplňovat jednotlivé kreslící operace
      };
</script>
</head>
<body>
<canvas id="platno" width="500" height="400"></canvas>
</body>
</html>
```

Příklad 38 - Šablona zdrojového kódu pro Canvas

6.3.2 Obdélník

```
context.rect(x, y, width, height);
```

Příklad 39 - Obdélník

Obdélník je jednoznačně určen počátečním bodem (souřadnice *x* a *y*) a jeho šířkou a výškou.

6.3.3 Čára

```
context.moveTo(x, y);

context.lineTo(x, y);

context.stroke();
```

Příklad 40 - Čára

Čára je vytvářena kombinací příkazů *moveTo*, *lineTo* a *stroke*. První příkaz posune pomyslný kurzor na počáteční souřadnici určenou hodnotami x a y a druhý příkaz z tohoto bodu vykreslí čáru do souřadnic definovaných v něm. Poslední příkaz pak udělá čáru viditelnou tím, že jí přiřadí barvu (výchozí barva *stroke* je černá).

6.3.4 Oblouk

```
context.arc(centerX, centerY, polomer, pocatecniUhel, konecnyUhel,
protiSmeruHodRucicek);
```

Příklad 41 - Oblouk

Pro vytvoření oblouku použijeme metodu *arc*. Oblouk je jednoznačně určen středovým bodem (souřadnice *centerX* a *centerY*), poloměrem (*polomer*), počátečním a konečným úhlem (*pocatecniUhel*, *konecnyUhel*) a boolovskou proměnnou určující, zda má být vykreslen po nebo proti směru hodinových ručiček (*protiSmeruHodRucicek*). Hodnota false tedy znamená, že oblouk bude vykreslen po směru hodinových ručiček.

6.3.5 Kruh

```
context.arc(centerX, centerY, polomer, 0, 2*Math.PI, false);
```

Příklad 42 - Kruh

Poměrně zajímavé je, že javascriptové API nemá vlastní metodu pro vykreslení kruhu. Ten je potřeba vytvořit pomocí metody *arc*, která vytváří oblouk pouhou změnou parametrů. Pro vytvoření oblouku je potřeba zadat počáteční úhel 0 a koncový úhel 2∏.

6.3.6 Elipsa

```
context.save();

context.scale(nasobekSirky, nasobekVysky);

context.beginPath();

context.arc(centerX, centerY,polomer,0,2*Math.PI,false);

context.stroke();

context.closePath();

context.restore();
```

Příklad 43 - Vykreslení elipsy pomocí několika příkazů

Elipsa nemá přímo definovanou primitivní funkci ve specifikaci Canvas. Řešení je však jednoduché – nakreslíme kruh pomocí již známého *context.arc(),* a následně aplikováním *context.scale()* změníme jeho poměr stran. Jediné, co musíme mít na paměti, je fakt, že tato změna bude aplikována na všechny další objekty vykreslené pomocí daného kontextu. Proto si příkazem *context.save()* na počátku uložíme aktuální stav, vykreslíme elipsu, a nakonec příkazem *context.restore()* vrátíme původní nastavení kontextu.

6.3.7 Kvadratická křivka

```
context.moveTo(x, y);

context.quadraticCurveTo(kontrolniBodX, kontrolniBodY, konecnyBodX,
konecnyBodY);
```

Příklad 44 - Kvadratická křivka

Kvadratická křivka je vytvářena pomocí dvou příkazů, kdy *moveTo* nám posune kurzor na požadovanou souřadnici, která se zároveň stane začátkem křivky a *quadraticCurveTo* nám pak vykreslí křivku podle souřadnic kontrolního bodu a konečného bodu.

6.3.8 Beziérova křivka

```
context.bezierCurveTo(kontrolniBodX1, kontrolniBodY1,
kontrolniBodX2, kontrolniBodX2, konecnyBodX, konecnyBodY);
```

Příklad 45 - Beziérova křivka

Beziérova křivka se vytváří podobně jako kvadratická, s tím rozdílem, že pro její vytvoření je potřeba určit dva kontrolní body.

6.4 Vybarvování a jiné efekty

Stejně jako v případě SVG máme pro výplň vykreslených obrazců k dispozici sadu standardních metod.

6.4.1 Výplň jednou barvou

```
context.fillStyle=[hodnota];

context.fill();
```

Příklad 46 - Výplň jednou barvou

6.4.2 Přechodová výplň (tzv. gradient, k dispozici je lineární a radiální)

```
var gradient=context.createLinearGradient(pocatecniBodX,
pocatecniBodY, konecnyBodX, konecnyBodY);
```

Příklad 47 - Přechodová výplň

```
gradient.addColorStop(offset, barva);
```

6.4.3 Výplň vzorem

```
var vzor = context.createPattern(obrazek, repeatOption);

context.fillStyle = vzor;

context.fill();
```

Příklad 48 - Výplň vzorem

6.5 Vložení obrázku do elementu Canvas

```
context.drawImage(obrazek,cilovyBodX,cilovyBodY, sirka, vyska);
```

Příklad 49 - Vložení obrázku do Canvasu

Pro „vykreslení“ obrázku do elementu Canvas potřebujeme existující obrázek a souřadnice cílového bodu, případně můžeme zadat i šířku a výšku obrázku, pokud je chceme změnit oproti originálu.

Celý příklad může vypadat takto:

```
window.onload = function() {
      var platno = document.getElementById("platno");
      var context = platno.getContext("2d");
            var cilovyBodX = 20;
            var cilovyBodY = 20;
            var sirkaObrazku = 145;
            var vyskaObrazku = 117;
      var obrazek = new Image();
      obrazek.src = "html5_logo.gif";
      obrazek.onload = function(){
            context.drawImage(obrazek, cilovyBodX, cilovyBodY,
            sirkaObrazku, vyskaObrazku);
            }
       };
```

6.6 Text v Canvasu

Stejně jako v SVG můžeme pomocí Canvasu nechat vykreslit text. Obrovským rozdílem, který vyplývá z principu technologie, je ale fakt, že takový text je stále jen pouhým obrázkem a nelze tedy označit kurzorem a kopírovat, a ani vyhledávače jej nenajdou.

6.6.1 Vykreslení textového řetězce

```
context.fillText("Žluťoučký kůň", pocatecniBodX, pocatecniBodY);
```

Příklad 50 - Vykreslení textu v Canvasu

Příkaz *context.fillText* přijímá tři atributy, a sice samotný textový řetězec pro vypsání a souřadnicové body *x* a *y* sloužící pro určení počátečního bodu.

6.6.2 Font a velikost textu

```
context.font = "50px Tahoma";
```

Příklad 51 - Font a velikost textu

Pro úpravu velikosti textu a použitého fontu slouží příkaz *context.font*. Dále následuje v uvozovkách uzavřený pár hodnot, kdy první hodnota udává velikost písma a druhá použitý font. Pokud se tento příkaz explicitně nepoužije, ve výchozím nastavení bude použit font typu sans-serif, tedy bezpatkový, o velikosti 10 pixelů.

6.6.3 Řez písma

```
context.font = "bold 50px Tahoma";

context.font = "italic 50px Tahoma";
```

Příklad 52 - Řez písma

Obdobně jako v CSS je možné určit i v Canvasu řez písma. Příklad ukazuje v prvním případě tučné písmo a v druhém případě kurzívu.

6.6.4 Barva písma

```
context.fillStyle = "#ff0000";
```

Příklad 53 - Barva písma

Pro ovlivnění barvy písma vykreslovaného textu použijeme příkaz *context.fillStyle*. Výchozí hodnotou je barva černá.

6.6.5 Ohraničení textu

```
context.lineWidth = 2;

context.strokeStyle = "#0000ff";

context.strokeText("Žluťoučký kůň", pocatecniBodX, pocatecniBodY);
```

Příklad 54 - Ohraničení textu

Pro ohraničení textu je nutné definovat barvu ohraničení a dále místo dosavadně užívaného příkazu pro vykreslení textu *context.fillText* použít *context.strokeText*. Příkaz *context.lineWidth* není povinný, určuje šířku orámování textu. Výchozí nastavení je šířka 1 pixel.

6.6.6 Rozměr vykresleného textu

Zajímavým příkazem je *context.measureText(text)*, který vrátí objekt obsahující rozměry textu.

```
var rozmery = context.measureText(text);

var sirka = rozmery.width;

var vyska = rozmery.height;
```

Příklad 55 - získání rozměru textu

Z příkladu je vidět, že pro získání rozměru textu je potřeba textový řetězec uložit do proměnné, abychom na něj mohli odkazovat pomocí příkazu *context.measureText.*

Tento rozměr pak dále uložíme do dalších definovaných proměnných pro šířku a výšku, abychom s ním mohli nakládat podle potřeby.

Celý příklad pak může vypadat následovně:

```
var textovy_retezec = " Žluťoučký kůň.";

var pocatecniBodX = 10;

var pocatecniBodY = 10;

context.font = " 50px Tahoma ";
```

```
context.fillText(textovy_retezec, pocatecniBodX, pocatecniBodY);


var rozmery = context.measureText(textovy_retezec);

var sirka = rozmery.width;

var vyska = rozmery.height;

//nyní už máme k dispozici rozměry textu uložené v proměnných sirka
a vyska
```

Příklad 56 - Příklad získání rozměrů textu

7 Porovnání obou technologií

Jak vyplývá z předešlého popisu obou technologií, SVG je objektově založené (SVG elementy jsou podobné HTML elementům, dá se tedy říct, že XML je jejich společný „otec“), Canvas je orientovaný na jednotlivé pixely (v zásadě je to obrázkový element s kreslícím aplikačním programovým prostředím).

SVG, obdobně jako HTML, vytváří objektový model elementů, atributů a stylů (DOM – Document Object Model). Element <*svg*> je tedy v HTML stránce součástí objektového modelu. Tvary vykreslené pomocí SVG jsou vytvářeny elementy a upravovány pomocí stylovacího jazyka či programově pomocí skriptu. Interakce s SVG je implementována objektově na úrovni primitivních grafických elementů (např. čáry, kruhy, cesty atp.). SVG objekty přímo podporují přístupnost.

Canvas je bitmapa, která má tu vlastnost, že po zobrazení výsledné grafiky neudržuje žádné informace o vykreslených objektech, veškeré vykreslené obrazce jsou uloženy jako matice jednotlivých pixelů. Výsledný obraz se tedy chová obdobně jako tag <*img*> dobře známý z jazyka HTML. Grafika je vytvářena i upravována výhradně pomocí skriptovacího jazyka, protože Canvas, respektive jeho HTML5 specifikace, neobsahuje žádné elementy pro vytváření primitivních tvarů. Interakce s objekty vykreslenými pomocí Canvasu musí být programovány ručně pomocí souřadnic myši. Canvas nepodporuje přístupnost.

7.1 Vhodnost použití SVG a Canvas

Jelikož se tyto dvě technologie principiálně velice odlišují (vektor / bitmapa), jsou samozřejmě i velice odlišné scénáře jejich použití v reálných aplikacích.

7.1.1 SVG

Již z povahy vektorové grafiky je předem poměrně jasné použití SVG pro vykreslování grafických objektů, které je potřeba bez ztráty kvality zvětšovat, škálovat a posílat na vstup různých výstupních zařízení s různým rozlišením. Proto je tedy SVG velice vhodným formátem pro přípravu grafiky prezentací či pro tisk nejrůznějších dokumentů.

V dnešní době, kdy se rozlišení moderních počítačových monitorů nejčastěji pohybuje kolem tzv. FULL HD, tedy 1920 x 1080 pixelů, stoupá potřeba škálovatelnosti grafických objektů na webové stránce. Nezřídka se stane i mně, že si chci zvětšit písmo na webu pro lepší čitelnost (na dvaadvacetipalcovém monitoru je poměrně často používané písmo o velikosti 11px opravdu malé a velmi obtížně čitelné), ale po zvětšení se samozřejmě veškerá grafika webu rozostří, protože je bitmapová. SVG jako vektorový formát by dokázal řešit i tento neduh.

V neposlední řadě je také výhoda SVG ve „znovupoužitelnosti“ již vykreslených objektů. Pokud budu potřebovat stejný obrázek použít jinde v jiné velikosti, jednoduše ho bez ztráty kvality zvětším, což samozřejmě u bitmapového formátu nelze.

7.1.2 Canvas

Naproti tomu, Canvas je velice silný v nejrůznější manipulaci s vykreslenými pixely. To znamená, že pokud budeme chtít aplikovat nejrůznější efektové filtry na obrázky, Canvas je vhodným kandidátem.

Další výhoda Canvasu je v tom, že s přibývajícími objekty nenarůstá náročnost na výpočetní výkon tak strmě, jako je tomu v případě SVG, které všechny objekty vkládá do DOM. Typickým příkladem může být mapka aktuálního počasí, kde se nachází velké množství nejrůznějších tvarů, které jsou ještě modifikovány v tzv. reálném čase. Realizace pomocí SVG by byla z výše uvedeného důvodu extrémně náročná a neefektivní.

Dalším pěkným příkladem pro použití pixelově orientovaného Canvasu je tzv. „greenscreen replacement“. Greenscreen, tedy zelená plocha, je trik používaný ve filmech, kdy se za herce umístí zelené plátno a to je potom nahrazeno např. scenérií s vodopádem.

7.1.3 Máme tedy vítěze?

Nemáme. Obě technologie mají co nabídnout, záleží především na situaci, ve které se chystáme SVG nebo Canvas nasadit. Proto se nedá jednoznačně říci, která technologie je lepší.

8 Možnosti kombinování SVG a Canvas

Již tedy víme, že Canvas a SVG využívá velice rozdílného přístupu k vykreslované grafice. Esenciální principy obou technologií tedy nutně evokují otázku, zda by se nedalo využít výhod obou formátů pro dosažení co nejlepšího výsledku v požadované realizaci.

8.1 Použití Canvasu v SVG

Pro možnost použití Canvasu v SVG dokumentu je potřeba učinit jisté kroky k zajištění dobré funkčnosti.

V současné době je jediný možný způsob jak dostat Canvas do SVG dokumentu pomocí elementu <*foreignObject*>.

Kroky nutné pro vložení Canvasu do SVG:

1) Přidat XHTML namespace
2) Přidat element <*foreignObject*>
3) Přidat element <*xhtml:canvas*> jako potomek elementu <*foreignObject*>
4) Použít *getElementsByTagNameNS* pro získání reference na Canvas

Příklad celého kódu:

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xhtml="http://www.w3.org/1999/xhtml">
<title>Canvas v SVG</title>
<foreignObject width="300" height="150">
  <xhtml:canvas width="300" height="150" >
    Alternativni obsah pro prohlizece nepodporujici Canvas
  </xhtml:canvas>
</foreignObject>

<script type="text/javascript"><![CDATA[
  window.onload = function () {
    var xhtmlNS = "http://www.w3.org/1999/xhtml";
    var context = document.getElementsByTagNameNS(
                  xhtmlNS,'canvas')[0].getContext('2d');
    context.fillStyle = 'rgba(0,200,0,0.7)';
    context.fillRect(0,0,100,75);
  };
]]></script>

</svg>
```

9 Podpora v prohlížečích

Moderní prohlížeče již dnes nemívají problémy s novými standardy tak, jako tomu bylo v dřívějších dobách. Na vině byl především Microsoft se svým browserem Internet Explorer, který z části uznával pouze své interní standardy, a bylo tak buď potřeba použít jinou syntaxi, nebo hledat způsoby, jak donutit prohlížeč obsah zobrazit podle představ programátora. Asi nejproblémovější verzí byla šestá, která byla hodně rozšířená především díky integraci do instalace operačního systému Microsoft Windows XP. Ten se stal suverénně nejpoužívanějším operačním systémem na PC od jeho uvedení až prakticky po uvedení Windows verze 7. Z toho důvodu bylo velké množství webových stránek zobrazováno právě v tomto nedokonalém prohlížeči, i když už existovaly značně schopnější konkurenční produkty poskytované zdarma a bez reklam – na mysli mám Mozillu a její Firefox, případně Operu či Safari od Applu.

Od těchto dob už ale uplynulo hodně času, a i Microsoft se umoudřil a na své nejnovější deváté verzi Internet Exploreru zapracoval tak, aby již nebyl zdrojem problémů webových designérů. Předpokladem pro setřesení špatné pověsti je samozřejmě kvalitní podpora webových standardů a formátů, ať již starších, jako například SVG, tak těch novějších, jmenovitě specifikace HTML5 a její součást, element Canvas.

Pro vyzkoušení kompatibility jsem zvolil nejpoužívanější prohlížeče současné doby v aktuálních verzích, konkrétně se jedná o následující:

- Opera ve verzi 11.62
- Mozilla Firefox 11.0
- Google Chrome 18.0.1025.162 m
- Internet Explorer 9
- Safari 5.1.5

9.1 Podpora elementu Canvas

Pro vyzkoušení elementu Canvas jsem vytvořil jednoduchou XHTML stránku, která obsahuje formulář s tlačítkem. Tlačítko spustí javascriptovou funkci, která se pokusí vytvořit element Canvas a následně získat jeho kontext. Pokud se toto prohlížeči podaří, máme vyhráno a je jasné, že HTML5 specifikace elementu Canvas je prohlížečem podporována. Následně se už jen vykoná příkaz na oznámení výsledku testu formou výstražného okna, neboli javascriptové funkce *alert*.

9.1.1 Výsledky testu podpory Canvas

Prohlížeč	IE	Firefox	Opera	Chrome	Safari
Podporuje	ANO	ANO	ANO	ANO	ANO

Jak je z tabulky vidět, současné prohlížeče nemají s podporou HTML5 elementu Canvas žádný problém. Není tedy potřeba instalovat doplňky, pluginy či jiné zásuvné moduly pro správné zobrazení grafického obsahu kontextu Canvas.

9.2 Podpora SVG

Standard SVG byl ve vývoji již od roku 1999, první publikovaná verze se objevila o dva roky později. Vývojáři webových prohlížečů tedy měli dostatek času na to, aby se jim podařilo implementovat podporu pro tuto technologii do svých produktů.

V průběhu testování jsem nezaznamenal žádný problém s pěticí zkoumaných prohlížečů. Microsoft dodal podporu pro SVG až do poslední, deváté verze jeho browseru. Jak se říká, sice pozdě, ale přece.

9.2.1 Výsledky testu podpory SVG

Prohlížeč	IE	Firefox	Opera	Chrome	Safari
Podporuje	ANO	ANO	ANO	ANO	ANO

9.3 Podpora obou technologií na mobilních zařízeních

Mobilní zařízení prodělaly v posledních letech velkou evoluci především díky zásluze amerického Applu, který trhu v roce 2007 představil revoluční mobilní telefon iPhone. Jeho dotykové ovládání a operační systém iOS spolu nastolily nový trend mobilních zařízení.

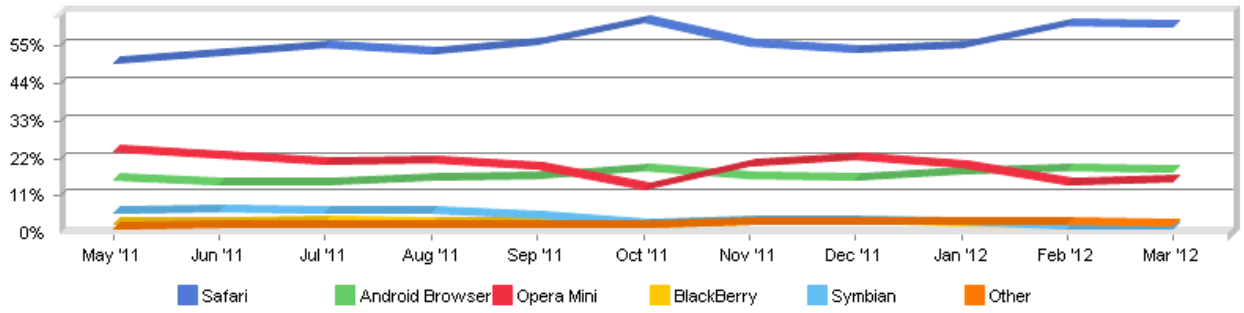
Standardní výbavou tohoto telefonu byla totiž bezdrátová síťová karta dovolující přístup k internetu a především opravdu dobře vymyšlené dotykové ovládání, díky kterému již nebyl problém webový obsah prohlížet pohodlně. Ostatní výrobci samozřejmě nemohli Applu tento velkolepý úspěch nezávidět, a tak se pustili do vývoje vlastních telefonů, které jsou a už asi vždy budou označované jako „iPhone like“, tedy volně přeloženo „podobný iPhonu“.

Tato revoluce tak způsobila markantní zvýšení přístupů na webové stránky z mobilních zařízení a jejich internetových prohlížečů.

9.3.1 Nejpoužívanější mobilní prohlížeče

Podle stránek www.netmarketshare.com [12] patří mezi tři nejpoužívanější prohlížeče na mobilních zařízeních Safari, Android browser a Opera Mini.

[12] Dostupné z http://www.netmarketshare.com/browser-market-share.aspx?qprid=1&qpcustomb=1



Obrázek 6 - graf používání mobilních prohlížečů

Jak je vidět z grafu, drtivě vede Safari, které je součástí operačního systému na všech zařízeních Applu. Opera Mini a výchozí prohlížeč na přístrojích s operačním systémem Android jsou poměrně vyrovnané, s tím, že aktuálně mírně vede Android.

Pro vyzkoušení podpory technologie Canvas a SVG jsem tedy zvolil tři nejpoužívanější prohlížeče podle stránek www.netmarketshare.com.

K testu jsem použil zařízení iPhone 3G (Safari iOS 4.3 a Opera Mini ve verzi 6.5). Bohužel jsem neměl k dispozici žádný přístroj s operačním systémem Android, proto budu vycházet z dat uvedených na stránce http://caniuse.com.

9.3.2 Výsledky testu

Mobilní zařízení v tomto případě nezklamaly. Technologie SVG ani Canvas jejich prohlížečům nejsou cizí a v tomto testu opravdu tak kapesní přístroje zabodovaly. Optimalizovat proto webové stránky pro použití mobilních prohlížečů dnes opravdu má smysl, protože kvalita softwarové výbavy dnešních telefonů se už začíná podobat té počítačové.

10 Praktické příklady

V rámci praktické části jsem vypracoval jak ukázkové příklady obou grafických formátů, tak webovou aplikaci, která jejich prohlížení činí jednodušším. Stránka obsahuje všechny ukázky v přehledné podobě spolu s jejich obecným popisem, syntaxí a ukázkou použitého zdrojového kódu.

11 Závěr

Pro vkládání dynamických prvků na webu máme v SVG a Canvasu velice silné nástroje. SVG je standardizován již řadu let, naproti tomu Canvas je součástí až nové specifikace jazyka HTML5.

Často bývají tyto dvě technologie předmětem otázek, kterou z nich by měl programátor pro svůj web či webovou aplikaci použít. Ve skutečnosti je ale pro informované rozhodnutí potřeba znát dobře požadavky takové práce, protože obě technologie se v přístupu ke zpracování grafických prvků znatelně liší.

Jelikož je jak SVG, tak Canvas součástí standardů, není s jejich podporou a zpracováním v moderních prohlížečích problém. Dokonce, jak jsme se přesvědčili v kapitole o mobilních prohlížečích, je podpora výborná i na přenosných zařízeních jako jsou chytré telefony (tzv. smartphony) či tablety.

V průběhu své práce jsem představil oba formáty, demonstroval jejich schopnosti na praktických příkladech a porovnal jejich výhody a nevýhody.

Pro přehlednější prohlédnutí praktických příkladů jsem vytvořil webovou stránku dostupnou na adrese www.fix-it.cz, kde je možné zobrazit si jednotlivé ukázky spolu s jejich základním popisem, obecnou syntaxí a zdrojovým kódem použitým v konkrétní ukázce.

Stanovené cíle bakalářské práce byly splněny.

Literatura

1. Rasterization. [online]. [cit. 2012-04-22]. Dostupné z: http://en.wikipedia.org/wiki/Rasterisation

2. Bitmap versus vector graphics and images. [online]. [cit. 2012-04-22]. Dostupné z: http://www.prepressure.com/library/file-formats/bitmap-versus-vector

3. SVG in HTML. [online]. [cit. 2012-04-22]. Dostupné z: http://www.w3schools.com/svg/svg_inhtml.asp

4. SVG tutorial. [online]. [cit. 2012-04-22]. Dostupné z: http://www.itk.ilstu.edu/faculty/javila/SVG/

5. Inkscape. Draw freely. [online]. [cit. 2012-04-22]. Dostupné z: http://inkscape.org

6. Svg-edit - A complete vector graphics editor in the browser. [online]. [cit. 2012-04-22]. Dostupné z: http://code.google.com/p/svg-edit/

7. ScriptDraw (TM) The free online SVG editor. [online]. [cit. 2012-04-22]. Dostupné z: http://scriptdraw.com/

8. Amaya. [online]. [cit. 2012-04-22]. Dostupné z: http://www.w3.org/Amaya/

9. KIYUT - Sketsa SVG Editor. [online]. [cit. 2012-04-22]. Dostupné z: http://www.kiyut.com/products/sketsa

10. SVG Editor. [online]. [cit. 2012-04-22]. Dostupné z: http://www.oxygenxml.com/svg_editor.html

11. HTML5: Up and Running [online]. First Edition. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'REILLY | Google Press, 2010 [cit. 2011-03-31]. Dostupné z WWW: <http://oreilly.com/catalog/9780596806033>. ISBN 978-0-596-80602-6

12. LUBBERS, Peter; ALBERS, Brian; SALIM, Frank. Pro HTML5 Programming : Powerful APIs for Richer Internet Application Development [online]. Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013 : Apress, 2010 [cit. 2011-03-31]. Dostupné z WWW: <http://www.apress.com/book/view/1430227907>. ISBN 978-1-4302-2791-5.

13. MALÝ, Martin. Zdroják [online]. 27. 9. 2010 [cit. 2011-03-31]. SVG, nebo Canvas? Vyberte si. Dostupné z WWW: <http://zdrojak.root.cz/clanky/svg-nebo-canvas-vyberte-si/>.

14. W3schools [online]. c2011 [cit. 2011-04-02]. Dostupné z WWW:

15. <http: //www. w3schools.com / svg/svg _inhtml.asp>.

16. W3C. W3C [online]. 2010, 30 March 2011 [cit. 2011-04-02]. Dostupné z WWW: <http: //dev.w3.org/html5/spec/Overview.html>.

17. ROWELL, Eric. Html5CanvasTutorials [online]. c2011 [cit. 2011-04-02]. Dostupné z WWW: <http://www.html5canvastutorials.com/>.

18. JENKOV, Jakob. Jenkov [online]. 2009 [cit. 2011-04-02]. Dostupné z WWW: <http://tutorials.jenkov.com/svg/index.html>.

19. DAILEY, David. W3C [online]. c2010 [cit. 2011-04-02]. Dostupné z WWW: <http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html>.

20. Seznámení s HTML5. [online]. [cit. 2012-04-22]. Dostupné z: http://interval.cz/clanky/seznameni-s-html-5/

21. Mobile SVG profiles: SVG Tiny and SVG Basic. [online]. [cit. 2012-04-22]. Dostupné z: http://www.w3.org/TR/SVGMobile/

22. Browsers market share. [online]. [cit. 2012-04-22]. Dostupné z: http://www.netmarketshare.com/browser-market-share.aspx?qprid=1&qpcustomb=1

Seznam obrázků

Seznam příkladů

Přílohy

1) CD – na přiloženém CD se nachází tato bakalářská práce s názvem bp_trantyr.pdf. Složka praktické příklady obsahuje vypracované ukázky grafických prvků zpracovaných pomocí SVG a Canvasu. Složka www obsahuje veškeré zdrojové kódy a grafiku webové stránky pro demonstraci příkladů.