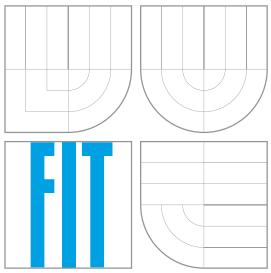
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TRANSFORMACE DOKUMENTŮ HTML NA VEKTOROVOU GRAFIKU SVG

HTML DOCUMENT TRANSFORMATION TO SCALABLE VECTOR GRAPHICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN ŠAFÁŘ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2016

Abstrakt

Tato diplomová práce řeší problematiku vykreslování HTML/CSS dokumentů pomocí vektorové grafiky ve formátu SVG. Cílem práce je návrh a implementace rozšíření knihovny CSSBox o možnost vektorového výstupu. Nejprve jsou popsány základy jazyka SVG a existující knihovny na platformě Java, které s SVG pracují. Dále je popsána knihovna CSSBox a postup vykreslování webových stránek v této knihovně. Poté následuje rozbor některých CSS3 vlastností. Hlavní částí této práce je návrh řešení renderování CSS3 vlastností jako jsou zaoblené rámečky, gradienty, nebo transformace. U jednotlivých CSS vlastností je popsána jejich specifikace podle standardu CSS3 a způsob řešení v jazyce SVG. Po návrhu následuje popis implementace výsledného rozšíření a nakonec je zhodnocena úspěšnost implementace při testování. V závěru práce jsou navržena možná rozšíření této práce.

Abstract

This diploma thesis deals with the topic of rendering HTML/CSS documents using the Scalable Vector Graphics (SVG) language. The goal of this thesis is to design and implement an extension for the CSSBox library, which will be able to generate a vector output. First, we provide a description of the SVG language and some Java libraries that can be used for creating SVG documents. After that, there is description of the CSSBox library. Then, we perform an analysys of selected CSS3 features. The main part of this thesis is the design of a solution for rendering various CSS3 attributes such as rounded corners, gradients or transformations using SVG. After the design, there is a description of the implementation and evaluation of the achieved results using various tests. The conclusion offers some possibilities of extending this thesis.

Klíčová slova

CSSBox, Vektorová grafika, SVG, Java, HTML, CSS3, Apache Batik, Swing

Keywords

CSSBox, Vector graphics, SVG, Java, HTML, CSS3, Apache Batik, Swing

Citace

ŠAFÁŘ, Martin. *Transformace dokumentů HTML na vektorovou grafiku SVG*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Transformace dokumentů HTML na vektorovou grafiku SVG

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Šafář
23. května 2016

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Radku Burgetovi Ph.D. za odbornou pomoc při tvoření této práce.

Obsah

Kapitola 1

Úvod

Nástroj CSSBox [8], implementovaný čistě v jazyce Java, slouží k analýze HTML/CSS dokumentů a k následnému exportu jejich obrazu do vektorové, nebo rastrové grafiky. Výstup nástroje lze uložit také do souboru. Ukládat lze ve formátech SVG [15] (Scalable Vector Graphics) pro vektorovou grafiku, nebo PNG (Portable Network Graphics) pro rastrovou grafiku. Rastrovou grafiku lze dále zobrazit v prohlížeči (BoxBrowser, případně SimpleBrowser), který je integrovaný v projektu.

Zobrazovat výstup HTML dokumentů je ovšem vhodnější pomocí vektorové grafiky. Oproti rastrové grafice lze s vektorovou grafikou mnohem jednodušeji pracovat a provádět v ní změny. Vektorová grafika zároveň poskytuje velkou výhodu v tom, že lze v dokumentu vyhledávat a indexovat text přímo ve zdrojovém souboru. V aktuální verzi nástroj CSSBox nepodporuje pro vektorový výstup CSS3 vlastnosti jako jsou například zaoblené rámečky, stíny, gradienty, nebo transformace. Pro zobrazování HTML dokumentu v integrovaném prohlížeči se používá rastrový výstup a ne vektorový. V současné verzi se v CSSBoxu generuje SVG kód přímo textově a přidává se do výsledného řetězce. Vhodnější bude upravit generování SVG výstupu do DOM objektu.

V kapitole 2 jsou nejprve popsány základy vektorové grafiky a její srovnání s rastrovou grafikou. Dále je popsán jeden ze způsobů pro popis vektorové grafiky - jazyk SVG (Scalable Vector Graphics). Na závěr je podrobněji vysvětlena funkčnost elementů a konstrukcí jazyka SVG, které budou potřeba pro implementaci renderování vektorového výstupu.

Kapitola 3 popisuje tři zkoumané knihovny na platformě Java, které umožňují práci s vektorovou grafikou v jazyce SVG. Konkrétně jde o Apache Batik, Salamander SVG a jFreeSVG. Zároveň je v této kapitole srovnání možnosti využití knihoven pro vytváření a následné renderování vektorového výstupu webových stránek v projektu CSSBox.

Kapitola 4 se zabývá projektem CSSBox, popisuje jeho účel a možnosti použití. Dále je popsán aktuální způsob renderování webových stránek pomocí rastrové grafiky za použití rozhraní Graphics2D v Jave. Nakonec je v této kapitole popsán aktuální způsob renderování výstupu ve vektorové grafice.

Kapitola 5 popisuje vybrané vlastnosti definované ve standartu CSS3. Jde především o vlastnosti týkající se grafického výstupu stránek. Například zaoblené rohy, gradienty, stínování, transformace nebo použití obrázku jako pozadí pro rámeček. Kromě syntaxe zápisu těchto CSS3 atributů je popsána i základní funkčnost, případně některé speciální případy, které mohou nastat.

Kapitola 6 obsahuje návrh rozšíření nástroje CSSBox o renderování vektorového výstupu i s CSS3 vlastnostmi. Pro každou z vybraných CSS3 vlastností je nejprve podrobně popsána funkčnost tak, jak je definována podle specifikací W3C (World Wide Web Consortium). Poté

jsou rozebrány možnosti, jakými by bylo možné danou vlastnost implementovat v jazyce SVG. Nakonec je popsán způsob, jak obecně převést zadané hodnoty z CSS do SVG.

V kapitole 7 jsou popsány detaily implementace renderování jednotlivých CSS3 vlastností.

Kapitola 8 obsahuje informace o testování. Testování bylo rozděleno na tři části: Testy jednotlivých vlastností na samostatných HTML elementech, testování na webových stránkách a testy rychlosti běhu aplikace.

Na závěr jsou zhodnoceny veškeré poznatky z této práce, popis dosažených výsledků a dále možnosti rozšíření této diplomové práce.

Kapitola 2

Vektorová grafika a jazyk SVG

V této kapitole je nejprve popsán vektorový způsob reprezentace obrázků a jeho porovnání s rastrovou reprezentací. Poté následuje popis značkovacího jazyka SVG, definice některých základních pojmů a rozbor obsahu dokumentu SVG. Nakonec jsou popsány některé základní grafické elementy, včetně ukázky jejich použití.

2.1 Popis rastrové a vektorové grafiky

Pro reprezentaci grafických dat v počítačích existují dva hlavní přístupy: Rastrová (bitmapová) grafika a vektorová grafika. V rastrové grafice je obrázek reprezentován pomocí tzv. bitmapy neboli dvourozměrného pole pixelů. Pro každý z těchto pixelů je pak různými způsoby uložena barva[1].

Vektorová grafika ukládá obrázky pomocí elementárních geometrických prvků (jako například bod, úsečka, křivka, tvar nebo mnohoúhelník). Všechny tyto geometrické prvky lze reprezentovat pomocí matematických výrazů. [1]

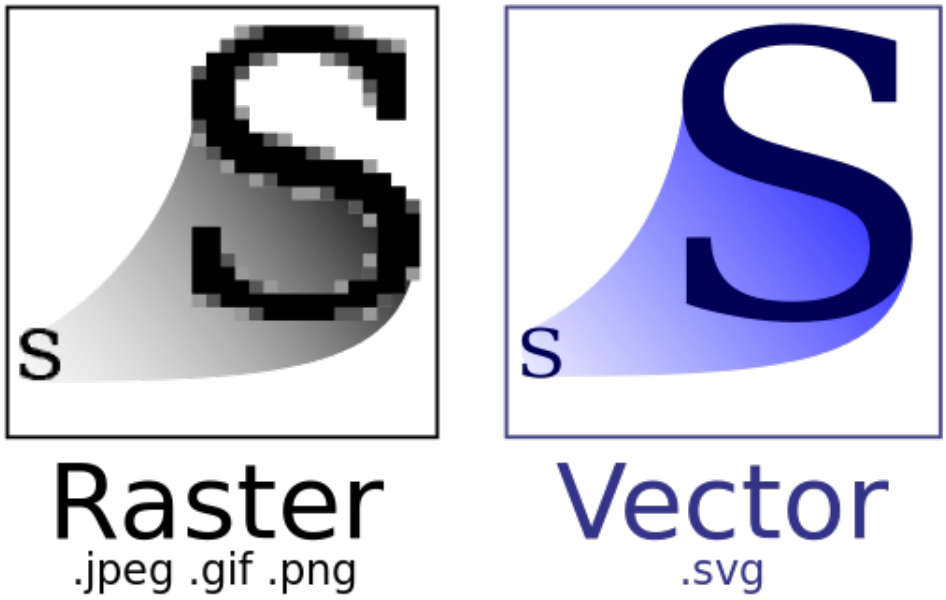
Na rozdíl od rastrových obrázků mají ty vektorové několik výhod. Vektorové obrázky se neskládají z jednotlivých bodů na plátně, ale z matematicky popsaných geometrických elementů. Díky tomu lze obrázky reprezentované pomocí vektorové grafiky libovolně zvětšovat i zmenšovat bez ztráty kvality (viz obrázek 2.1). Pokud bychom chtěli zvětšit rastrový obrázek, došlo by při tom nenávratně k rozmazání obrazu.

Velkou výhodou vektorové grafiky je určité porozumění obsahu obrázku. Právě tím, že je obsah definován pomocí základních geometrických útvarů, je možné jednodušeji analyzovat obsah. Stejně tak například identifikace a vyhledávání v textu je mnohem jednodušší, protože text je ve vektorové grafice popsán pomocí speciálního elementu, ve kterém je textový obsah přímo vložen.

Další výhodou vektorové grafiky je velmi nízká velikost souboru pro jednodušší obrázky. Velmi složité obrázky mohou vyžadovat tolik vektorových elementů, že bude velikost výsledného souboru mnohem větší než u rastrové reprezentace. [19]

Vektorová grafika má ale zároveň nevýhody. Jak už bylo zmíněno výše, pokud je obrázek příliš složitý, může být výsledný soubor velmi velký a renderování obrázku do 2D rastrové grafiky může být náročné na výpočetní zdroje počítače. Zároveň není tak jednoduché obrázky ve vektorové grafice vytvořit. Většinou jsou tyto obrázky vytvářeny pomocí grafických

[1] Barva může být uložena pomocí hodnoty RGB, případně odkazem na příslušnou barvu do speciální tabulky s barvami.



Obrázek 2.1: Rozdíl mezi vektorovou a rastrovou grafikou při zvětšování obrázků [13]

editorů, případně mohou být generovány v různých aplikacích. Pro získání rastrového obrázku stačí fotoaparát, případně skener.

Převádění z vektorové grafiky do rastrové je mnohem jednodušší než při opačném převodu, protože rastrová reprezentace je vlastně pouze vyrenderování vektorové grafiky na plochu (monitor, papír, atd.) pomocí tzv. rasterizace (převod do rastrové reprezentace). Převod z rastrové grafiky do vektorové, neboli vektorizace, je velmi složitý, protože musíme identifikovat jednotlivé objekty na obrázku a pak je matematicky popsat.

U webových stránek se rastrová grafika používá především v podobě digitálních fotografií (ať už jako pozadí, nebo jako obsah webu). Další využití má rastrová grafika na fotografie, tisk, zobrazování na monitoru, skenování, atd.

Vektorová grafika se u webových stránek používá především u elementu *<canvas>* [2] a dále pak na animace, loga, ikony, písma, nebo jednoduché grafické elementy. Mimo webové stránky se vektorová grafika používá hodně na technické výkresy pro projektanty (CAD - Computer Aided Design), případně pro vytváření zadání pro tiskárny s vysokým rozlišením.

2.2 Jazyk SVG

Scalable Vector Graphics (SVG) je jazyk používaný pro reprezentaci dvourozměrných vektorových obrázků a podporuje kromě základního kreslení i animace a dokonce i interakci (například lze používat hypertextové odkazy). SVG vychází z jazyka Extensible Modeling Language (XML), takže výsledný soubor má podobu textového souboru[1]. Jazyk SVG je open-source standardem a je vyvíjen společenstvím World Wide Web Consortium (W3C) již od roku 1999. Poprvé byl jazyk SVG oficiálně představen v roce 2001 a je inspirován svými předchůdci Vector Markup Language (VML) a Precision Graphics Markup Language (PGML). I když oba tyto standarty byly a stále jsou používány některými velkými firmami,

[2] Element canvas umožňuje během chodu aplikace v javascriptu vykreslovat různé grafické elementy. http://www.w3schools.com/html/html5_canvas.asp

W3C doporučuje pro vektorovou grafiku právě SVG.

Aktuální verze jazyka SVG je 1.1 a v současné době tuto verzi SVG podporují téměř všechny používané internetové prohlížeče. Pouze Internet Explorer verze 8 a starší SVG nepodporuje. Pro nepodporované prohlížeče je možné definovat alternativní obsah například textově. Díky tomu, že HTML5 také vychází z jazyka XML, je možné vložit SVG obrázek přímo do HTML dokumentu. Takový obrázek se vkládá do tagu *<svg>*.

SVG definuje několik základních prvků. Patří mezi ně grafické elementy (například křivky, tvary, úsečky, atd.), rastrové obrázky a text. Se všemi prvky je možno dále pracovat. Je možné například zvětšit nebo zmenšit elementy, u textu lze zvolit typ písma či barvu, u grafických elementů je možné nastavit tloušťku čáry či barvu čar a výplně. U všech elementů je možné aplikovat 2D transformace a animace[1].

Jazyk SVG dále poskytuje množství filtrů. Ty umožňují použít rozmazávání nebo stínování pomocí různých typů osvětlení či metod vykreslování.

2.3 Definice pojmů v jazyce SVG

Než budou popsány základní konstrukce jazyka SVG, je vhodné objasnit několik výrazů. Plátno (canvas) je plocha, na kterou se vykresluje SVG kód. Teoreticky je tato plocha neomezeně velká, ale prakticky je omezena výřezem (viewport), což je plocha, na které může uživatel vidět vykreslované SVG elementy (jednotlivé grafické prvky definované pomocí grafických elementů SVG, výčet těchto elementů se nachází dále). Rozměry a poloha výřezu jsou určeny samotným SVG dokumentem a současně jeho rodičovským elementem. Pokud jsou některé elementy vykreslovány mimo výřez nebo například přesahují ven, nejsou tyto části vidět.

Jednotky rozměrů v SVG mohou být definovány pomocí několika různých identifikátorů:

- *em* - určeno podle velikosti písma, 2em znamená dvakrát větší než velikost písma,
- *ex* - určeno podle výšky písma - 2ex znamená dvakrát větší než výška písma,
- *in* - 1 palec,
- *px* - pixel, 1px je roven 1/96 palce,
- *pt* - point, 1pt je roven 1/72 palce,
- *pc* - picas, 1pc = 12pt,
- *cm* - centimetr,
- *mm* - milimetr,
- *%* - procenta.

Jazyk CSS podporuje všechny výše zmíněné jednotky pro definování rozměrů. Navíc podporuje ještě několik dalších jednotek. Zajímavé jsou například jednotky *vw, vh, vmin* nebo *vmax*. Tyto jednotky se vypočítávají jako jedno procento buď z výšky, nebo ze šířky výřezu. Pro *vw* je to jedno procento šířky, pro *vh* je to jedno procento výšky, *vmin* jedno procento z nižší z těchto dvou hodnot a *vmax* naopak vyšší z těchto dvou hodnot. Dále jsou to ještě jednotky *rem* - totéž jako *em*, ale bere se velikost písma v kořeni dokumentu a *ch* - opět závisí na velikosti písma, ale vypočítává se podle šířky znaku 0 (nula). [4]

2.4 Základní informace o dokumentu SVG

Jak již bylo zmíněno, jazyk SVG je odvozený od jazyka XML, tudíž na začátku dokumentu můžeme specifikovat verzi XML a dále nastavit Document Type Definition (DTD) pro validátor. Jazyk SVG má velké množství společných vlastností s jazykem HTML a to hlavně díky tomu, že oba jazyky vyvíjí společenství W3C.

Soubor s grafikou v jazyce SVG lze stylovat buď pomocí kaskádových stylů CSS (nicméně pouze do verze CSS2), nebo pomocí XSL transformací. Vzhledem k tomu, že se celý projekt týká webových dokumentů, bude samozřejmě logické používat stylování pomocí CSS. Kromě stylů společných s CSS navíc SVG definuje několik dalších stylovacích parametrů, které lze použít přímo jako atributy elementu. Mezi takové parametry patří například:

- *stroke* a *stroke-width* - pro určení barvy a šířky čar a rámečků,
- *fill* a *fill-opacity* - pro určení barvy a průhlednosti výplně elementu,
- nebo *text-anchor* pro zarovnávání textu.

Mezi další parametry, které je možné vkládat k SVG elementům, patří parametry pro optimalizaci rychlosti a kvality renderování obrázků, tvarů a textů. Konkrétně se jedná o atributy *image-rendering*, *shape-rendering* a *font-rendering*.

SVG podporuje různé animace a vychází z animací definovaných ve specifikaci Synchronized Multimedia Integration Language (SMIL).

2.5 Základní konstrukce v jazyce SVG

Základní prvky SVG dokumentu jsou grafické elementy. Ty se dále člení na tvary (tagy *<circle>*, *<ellipse>*, *<line>*, *<polyline>*, *<polygon>*, *<rect>* a *<path>*), obrázky (tag *<image>*) a text (tag *<text>*).

Příkaz *<svg>* obaluje obsah SVG dokumentu, může ale sloužit i pro rozčlenění obsahu a díky tomu i k lepší znovupoužitelnosti částí SVG grafiky. Elementy *<svg>* lze libovolně zanořovat a mohou být i prázdné. Zároveň mohou tyto elementy sloužit ke specifikaci výřezu v dokumentu. Příkaz *<g>* umožňuje seskupovat jednotlivé grafické elementy do skupin. To napomáhá další strukturalizaci dokumentu. Jako motivaci k tomu, aby byly obrázky v SVG správně strukturalizovány, uvádí W3C ve specifikaci SVG možnost připojit k jednotlivým skupinám i popisky a titulky, díky nimž se vytváří další možnost interpretace dokumentu (například řeč - audio výstup).

Dále lze používat odkazy pomocí tagu *<a/>*, *<pattern/>* pro definici použití obrázku nebo jiného grafického elementu jako pozadí k ostatním prvkům. Zvláštním grafickým elementem je příkaz *<use />*. Ten umožňuje pomocí odkazu na jiný symbol v dokumentu (například přes ID prvku) vkládat základní grafické prvky, případně i celé skupiny objektů. Díky tomu lze ušetřit čas nejen při vytváření elementů, ale i při jeho renderování.

Odkazování se přes ID prvku je dále využíváno u výplně (atribut *fill*) nebo při využití ořezového elementu *<clipPath/>* a stejnojmenného atributu (použití ořezávání pomocí elementu *<clipPath/>* bude v následující podkapitole).

Pro odkazování v dokumentu se používá skupina atributů *xlink*. Odkaz se vyplňuje v atributu *xlink:href*. Atribut *xlink:title* zase umožňuje vyplnit popis odkazu, případně objektu, na který odkaz směřuje. Některé další XLINK atributy slouží pouze k zajištění zpětné kompatibility se staršími verzemi SVG. [14]

2.6 Důležité grafické SVG elementy

V této podkapitole jsou popsány některé základní grafické elementy v jazyce SVG, které budou klíčové pro následné vykreslování CSS3 vlastností.

2.6.1 Obdélníky

Vzhledem k tomu, že CSS Box model je založen na obdélnících, jeden z důležitých SVG elementů bude element reprezentující obdélník:

```
<rect x="10" y="10" width="100" height="100" style="fill:red;"/>
```

Atributy *width* a *height* definují rozměry obdélníku a atributy x a y definují počáteční bod vykreslování. Do atributu *style* lze vložit kromě barvy výplně (atribut *fill*) také například šířku a barvu okraje (*stroke*) nebo průhlednost elementu (*opacity*). Výše definovaný SVG element tedy bude čtverec s červenou výplní o délce jedné strany 100 pixelů. Počátek (levý horní roh) tohoto čtverce bude umístěn na souřadnicích *(100,100)*.

2.6.2 Cesty - path

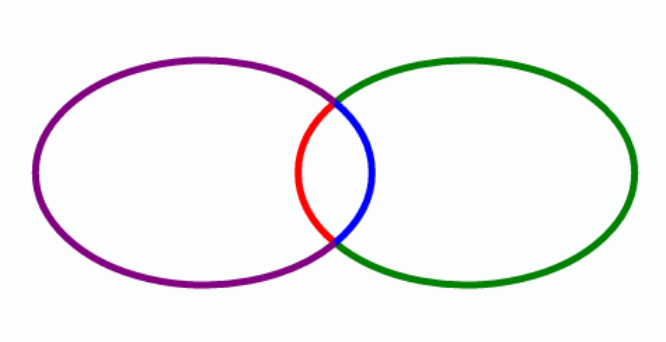
Dalším důležitým elementem je cesta - *<path/>*, pomocí něhož lze definovat prakticky jakýkoliv jiný tvar. Hlavním atributem tohoto elementu je atribut *d* (path data). V tomto atributu se nachází libovolná sekvence příkazů definující tvar výsledného grafického elementu. Mezi příkazy, které lze použít, patří:

- příkaz *M x y* (moveto) - tento příkaz umožňuje nastavit souřadnice aktuálního bodu vykreslování, tedy bod, kde "položíme štětec"na plátno.
- Příkaz *Z* (closePath) - použitím tohoto příkazu se vytvoří úsečka z aktuálního bodu do počátečního bodu aktuálně vykreslované části (poslední bod definovaný pomocí příkazu moveto).
- Příkaz *L x y* (lineto) vykreslí úsečku z aktuálního bodu do bodu definovaného souřadnicemi v parametru příkazu. Tento příkaz má ještě dvě varianty - *V* a *H* pro vertikální, respektive horizontální úsečky. Tyto příkazy mají jen jeden parametr (y, respektive x).
- Příkaz *A rx ry x-rotation large-arc-flag sweep-flag x y* (eliptical arc) slouží k vykreslení eliptického oblouku. Parametry *rx* a *ry* určují hlavní a vedlejší poloosu elipsy, která definuje tvar křivky a parametry x a y definují cílový bod křivky. *X-rotation* definuje, jak je elipsa otočená oproti ose x. Poslední dva zbývající parametry *large-arc-flag* a *sweep-flag* jsou pouze přepínače (mohou mít hodnotu 0 nebo 1) a určují, jaká část elipsy se bude vykreslovat. Fungování těchto přepínačů lze vidět na obrázku 2.2, kde každá barva reprezentuje jednu kombinaci hodnot parametrů.
- Dále lze použít příkazy pro kubické a kvadratické beziérovy křivky (příkazy *C,S,Q a T*). Pomocí těchto křivek lze vytvořit všechny možné tvary, ale pro použití vykreslovaní HTML dokumentů pravděpodobně nebudou potřeba, je zbytečné se jimi tedy zabývat.

Každý ze zmíněných parametrů má i verzi (záměnou velkého písmenka za malé), která používá relativní souřadnice místo absolutních. Ty ale kvůli přehlednosti výstupu SVG kódu nebudou použity.

Pro ukázku, čtverec popsaný v podkapitole 2.6.1 pomocí cesty by tedy v SVG vypadal následovně:

```
<path d="M 10 10 L 110 10 L 110 110 L 10 110 Z"
                                        style="fill:red;" />
```



Obrázek 2.2: Ilustrace fungování přepínačů u příkazu pro eliptický oblouk elementu *<path>* převzato z [16]

2.6.3 Gradienty

Gradienty slouží k vytvoření plynulého přechodu barev. Stejně jako v CSS3, v SVG jsou jak lineární, tak i radiální gradienty. Společným charakterem obou gradientů je, že se nejedná o elementy, které by byly přímo vykresleny na plátně. Gradienty musí být specifikovány uvnitř speciálního SVG elementu *<defs/>* a jsou používány výhradně přes odkazy.

Hlavní SVG elementy pro gradienty se používají takto:

```
<linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%" />
<radialGradient id="grad2" cx="50%" cy="50%"
                    r="50%" fx="50%" fy="50%" />
```

a takto by vypadalo použití například lineárního gradientu jako pozadí čtverce:

```
<rect width="100" height="100" style="fill:url(#grad1);" />
```

Parametry lineárního gradientu jsou dva body, definované pomocí souřadnic v procentech (souřadnice se dopočítávají podle rozměrů elementu, na kterém se vykreslují) a určují směr gradientu. V uvedeném příkladu je například směr z levé strany na pravou.

U radiálního gradientu určují souřadnice *cx* a *cy* střed vnější kružnice. Parametr *r* určuje poloměr kružnice a parametry *fx* a *fy* určují střed vnitřní kružnice (bod, odkud gradient začíná). Tvar gradientu (buď elipsa, nebo kružnice) určuje tvar objektu, na který je gradient aplikován.

Procenta se u všech parametrů používají z toho důvodu, že při vytváření gradientu uvnitř definičního elementu nelze vědět, kde všude a na jaké elementy bude gradient použit.

Uvnitř elementu gradientu se poté pomocí SVG elementu *<stop/>* definují jednotlivé hraniční barvy a jejich umístění rovněž v procentech.

Kapitola 3

SVG knihovny na platformě Java

Pro práci s SVG existuje v jazyce Java hned několik knihoven. V projektu bude potřeba jednak generovat textový výstup ve formátu SVG a případně tento výstup zobrazovat v grafickém rozhraní nástroje. Nástroj CSSBox je implementován pomocí knihovny Swing. Bude proto vhodné najít knihovnu, která podporuje toto grafické uživatelské rozhraní (GUI).

V této kapitole budou postupně rozebrány tři knihovny na platformě Java, které jsou určeny pro práci s grafikou ve formátu SVG. Konkrétně jde o Apache Batik, jFreeSvg a Salamander SVG. Knihovna Apache Batik je ze tří zmíněných vyvíjena nejdéle a má v porovnání s ostatními mnohem více funkčnosti. To lze vidět i na prezentačních webech obou dalších projektů, kde autoři sami srovnávají svůj nástroj s Apache Batik. Knihovna jFreeSVG je naopak vyvíjena nejkratší dobu (poprvé veřejně vydaná v roce 2013).

Na konci této kapitoly jsou jednotlivé knihovny porovnány a je vybrána nejvhodnější knihovna pro implementaci.

3.1 Apache Batik

Apache Batik (aktuálně ve verzi 1.8) je knihovna na platformě Java, která umožňuje pracovat s obrázky ve formátu SVG. Celá knihovna Batik je rozdělena na několik modulů, se kterými lze pracovat samostatně. Základní funkce, které tato knihovna nabízí, jsou generování, úprava a renderování obrázků v SVG. [3] Kromě základních funkcí ale Batik nabízí i kompletní aplikace, jako je například Squiggle (samostatný prohlížeč SVG). Dalším nástrojem je SVG Rasterizer, který umožňuje převádět obrázky z formátu SVG do rastrových formátů. Výhoda těchto nástrojů je především v tom, že je lze jednoduše rozšířit a tím pádem ušetřit mnoho času při implementaci vlastních aplikací. Batik také poskytuje syntax parser pro SVG soubory. Tato funkce by se mohla hodit při analýze HTML dokumentů pro nalezení a zobrazení chyby v SVG obrázku.

3.2 jFreeSVG

Knihovna jFreeSVG poskytuje podobnou základní funkčnost jako Apache Batik, ovšem dále už neposkytuje tolik rozšiřujících nástrojů. To se také projevuje na výsledné velikosti celé knihovny. Knihovna jFreeSVG má celkově cca 44KB[10], přičemž knihovna Apache Batik má téměř čtyřikrát více pouze se základní funkčností. I na stránkách projektu jFreeSvg je tento fakt zmíněn. Vývojáři zároveň prezentují tuto knihovnu jako "lehkou"a rychlou. Toto

tvrzení zároveň podporují testem, ve kterém srovnávají rychlost renderování v Apache Batik a v jFreeSVG.[11]

3.3 Salamander SVG

Projekt SVG Salamander nabízí podobnou funkčnost jako dva předchozí. Navíc ale nabízí možnost použít převod SVG obrázků do rastrových v ant skriptu. Dále podporuje používání některých prvků ve formulářích v GUI knihovně Swing. Oproti ostatním dvěma knihovnám je SVG Salamander cílený především pro použití při implementaci her na platformě java. [17]

3.4 Srovnání knihoven

Všechny zmíněné knihovny poskytují základní funkčnost, která by vyhovovala pro generování, renderování a analýzu SVG. Pro použití v tomto projektu bude ale zřejmě nejlepší použít knihovnu Apache Batik. Především díky možnostem rozšiřitelnosti kódu. Později, pokud by se ukázalo, že je třeba použít knihovnu, která bude s SVG pracovat rychleji, bylo by možné upravit implementaci tak, aby CSSBox mohl používat jak jFreeSVG, tak i Apache Batik.

Kapitola 4

CSSBox

V této kapitole je popsána základní struktura projektu CSSBox a jsou zmíněny některé důležité nástroje patřící do tohoto projektu. Dále je rozebrán základní postup zpracovávání HTML dokumentů a generování grafického výstupu. Nakonec je popsán způsob, jakým je aktuálně generován výstup ve formátu SVG.

4.1 Základní informace o projektu CSSBox

CSSBox je nástroj pro zobrazování dokumentů v jazyce HTML (případně i XML) společně s kaskádovými styly (CSS). Jeho hlavní funkčností je zpracování HTML dokumentů i se styly a poskytnutí kompletních informací, se kterými je možné dál pracovat. Tento nástroj obsahuje několik ukázkových demo souborů (respektive integrovaných aplikací), které lze používat a případně rozšiřovat. CSSBox umí vytvořit výstup jak v rastrové, tak i ve vektorové grafice. Zobrazení dokumentu je ale aktuálně možné pouze v rastrové grafice. Vektorový výstup může uživatel uložit do souboru. Díky použití formátu SVG je možné dále vektorový výstup upravovat ve vektorových editorech.

Pro lepší využití při analýze grafického rozhraní by bylo výhodné renderovat dokument pomocí vektorové grafiky. Klíčovou výhodou při zobrazování webových stránek ve vektorové grafice je možnost vyhledávání v textu. Text je ve vektorové grafice zpravidla reprezentován pomocí samotného řetězce, typu a velikosti písma, případně pak souřadnic umístění a barvy. Další výhodou je fakt, že vektorové obrázky uvnitř HTML dokumentu není třeba rasterizovat zvlášť, ale lze je přímo začlenit do dokumentu.

4.2 Podprojekty CSSBox

Celý projekt CSSBox obsahuje několik podprojektů, které spolu přímo či nepřímo souvisí. Například projekt jStyleParser je využíván pro zpracování kaskádových stylů. Projekt SwingBox slouží k zobrazování webových stránek v prostředí knihovny pro grafické rozhraní Swing. Dalším projektem je WebVector, což je aplikace, která převádí HTML dokumenty do rastrových a vektorových obrázků (umožňuje výstup v PNG nebo v SVG).

4.3 Přehled funkcí nástroje CSSBox

Kromě nástrojů zmíněných v kapitole 4.2 nabízí CSSBox několik dalších funkcí implementovaných v demo souborech přímo v projektu CSSBox.

Demo soubory StyleImport.java a ComputeStyles.java pracují s CSS. První zmíněný demo soubor nahradí všechny soubory CSS, které jsou do dokumentu vloženy přes HTML tag *<link/>* jejich obsahem do tagu *<style>*. Demo soubor ComputeStyles.java zase vloží vypočtené kaskádové styly ke každému elementu do atributu *style*.

SimpleBrowser.java slouží jako prohlížeč webových dokumentů. Zpracovávanou stránku renderuje na plátno v grafickém rozhraní Swing. Podobnou funkčnost nabízí BoxBrowser.java, který ale navíc obsahuje i adresní řádek a DOM Tree HTML dokumentu (v levém panelu aplikace), který je možné procházet. Grafický výstup je rastrový, ale umožňuje interakci, protože informace o poloze a rozměrech všech elementů jsou drženy po celou dobu zobrazení dokumentu. Demo aplikace BoxBrowser umožňuje vybírat elementy klikáním v dokumentu. O vybraných elementech zobrazuje v pravém panelu základní informace.

4.4 Průběh zpracování dokumentu

Při zpracovávání HTML dokumentu je nejprve načten dokument podle zadaného identifikátoru URI. Tento dokument je následně zpracován a uložen do třídy reprezentující Document Object Model (DOM). Poté je dokument zpracován třídou *DOMAnalyzer*. Tato třída zpracovává styly, které jsou buď přímo vloženy v dokumentu, nebo které jsou vloženy do dokumentu pomocí tagu*<link/>*. Zároveň je u této třídy možno nastavit základní styly, kódování, nebo atribut *media* pro CSS (defaultní je screen). Ve třídě *DOMAnalyzer* jsou poté kromě elementů v dokumentu uloženy i jejich CSS styly. V tuto chvíli máme tedy kompletní reprezentaci dokumentu a následuje buď její vykreslení do Swing komponenty, nebo vygenerování souboru s výstupem.

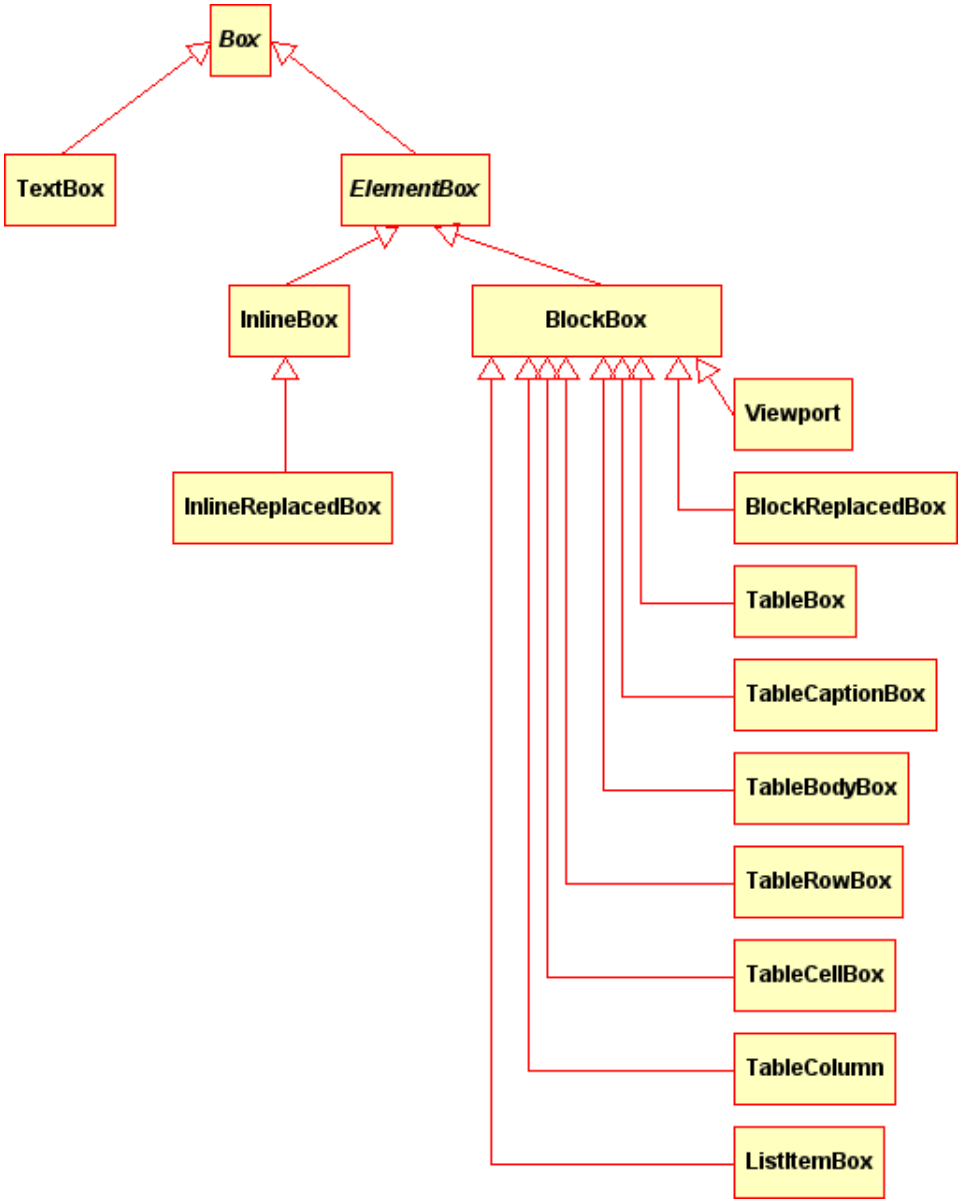
K rendrování grafického výstupu slouží třída *BrowserCanvas*. V této třídě je postupně dokument převeden na strom objektů, které jsou všechny instancemi tříd oděděných od třídy *Box*. Struktura těchto tříd je na obrázku 4.1. Jak je vidět na obrázku, základní třídy jsou *ElementBox* (reprezentující blokové a řádkové elementy) a *TextBox* pro textové elementy. Každý tento element je zabírá ve výsledném dokumentu obdélník podle svého umístění a rozměrů. Jednotlivé elementy se mohou překrývat. Po převedení dokumentu na strom "Box"objektů následuje vykreslení celého dokumentu pomocí třídy *Graphics2D*, což je třída do které jsou ukládány grafické elementy pomocí základních příkazů *drawRect*, *drawText*, atd. Třída *GraphicsRenderer* nakonec vykresluje grafiku z *Graphics2D* přímo do komponenty *BrowserCanvas* v grafickém rozhraní. [8]

Tento postup se provádí při generování rastrového výstupu. Generování výstupu ve formátu SVG bude popsáno v následující kapitole.

4.5 Renderování výstupu ve vektorové grafice

Renderování výsledného dokumentu do vektorové grafiky je aktuálně implementováno tak, že se výsledné elementy generují přímo jako textové příkazy jazyka SVG. Jazyk SVG vychází z jazyka XML a bylo by tedy možné výsledný soubor generovat a následně ukládat v rámci aplikace pomocí DOM. Tento přístup umožňuje dále manipulovat s vygenerovaným SVG obrázkem. Na rozdíl od toho při aktuálním způsobu již následná manipulace s dokumentem není možná, protože není vytvořena žádná struktura.

Aby byla při generování elementů zachována viditelnost a CSS vlastnost overflow je před každým elementem vygenerován SVG tag *<clipPath/>* a do něj je vložen ořezový obdélník podle rozměrů boxu. Obsah elementu je poté obalen tagem *<g/>*, který tvoří skupinu SVG



Obrázek 4.1: Struktura tříd dědících od třídy *Box*. (převzato z [12]

elementů. Pozadí elementů je vytvořeno pomocí dalšího obdélníku, kterému je nastavena výplň. V případě, že na pozadí elementu je obrázek, je vložen pomocí SVG tagu *<image/>*. Aby bylo možné zobrazovat okraje elementů samostatně a ne všechny najednou, jsou renderovány pomocí čar (SVG tag *<path>*). Šířka a barva okrajů jsou nastaveny pomocí atributů *stroke*. Ze stylů okrajů jsou podporovány následující: solid, dotted dashed a none. Ostatní styly okrajů zatím nejsou implementovány. Styly dotted a dashed jsou implementovány pomocí atributu *stroke-dasharray*.

4.6 Implementace nového rendereru pro vektorový výstup

V aktuální verzi nástroje CSSBox generují výstup dvě třídy: *GraphicsRenderer* pro rastrový výstup a *SVGRenderer* pro vektorový výstup. Obě tyto třídy implementují rozhraní *BoxRenderer*. Toto rozhraní deklaruje šest metod. Tyto metody jsou postupně volány při generování výstupu. Generování elementu začíná voláním metody *startElementContents()*. Před touto metodou není vygenerován žádný obsah vykreslovaného elementu, ale může již být vykreslené pozadí pomocí metody *renderElementBackground()*. Ta generuje pozadí a rámeček elementu. Metody *renderTextContent()* a *renderReplacedContent()* slouží ke generování obsahu elementu. Metoda *finishElementContents()* je volána po vykreslení elementu a všech jeho potomků. Nakonec po vykreslení všech elementů je volána metoda *close()*, která ukončí výstup.

V rámci této práce bude třeba vytvořit novou třídu pro renderování vektorového výstupu. Jak je popsáno v kapitole 4.5, třída *SVGRenderer* aktuálně generuje ve výše zmíněných metodách SVG kód přímo jako řetězec. Nová třída pro generování vektorového výstupu bude vycházet z třídy *SVGRenderer*, nicméně výstup bude ukládán do stromu DOM objektů a až nakonec bude převeden na řetězec.

Kapitola 5

CSS a standart CSS3

Webové stránky a obecně všechny HTML dokumenty jsou dnes stylovány především pomocí kaskádových stylů CSS, které jsou založeny na tzv. CSS Box modelu. V tomto modelu je každý element v HTML dokumentu definován jako obdélníkový box. Každý takový box má definované rozměry pomocí šířky a výšky, dále pak šířku rámečku, případně velikost vnitřního a vnějšího okraje (padding a margin). Dále je pak ke každému elementu možné definovat různé vlastnosti týkající se písma, rámečků, pozadí, atd. Vzhledem k tomu, že se způsoby vykreslování základních CSS vlastností tato práce nezabývá, nebudou zde popsány. Detaily týkající se vykreslování lze najít na stránkách W3C[1]

Standart CSS3 obsahuje velké množství nových prostředků ke stylování stránek, které zatím v nástroji CSSBox pro vektorový výstup nejsou podporovány. Například zaoblené rohy u rámečků, gradienty, dále 2D a 3D transformace, nebo stínování (box-shadow).[2]

V této kapitole jsou postupně popsány některé CSS3 vlastnosti, včetně ukázek a popisu některých speciálních vlastností.

5.1 Rámečky a zaoblené rohy

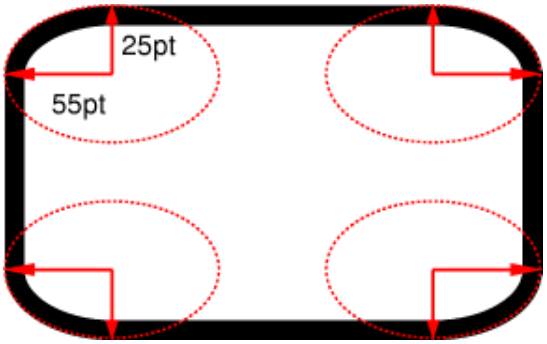
V CSS lze definovat barvu, styl a šířku rámečku pro každou stranu zvlášť. Navíc ve standartu CSS3 lze definovat zaoblení rohů rámečku. Zaoblení se lze nastavit pro každý roh zvlášť a to jak ve směru osy x, tak i ve směru osy y. Vnější okraje rámečku jsou pak definovány elipsou, jejích poloosy jsou definovány právě velikostí zaoblení. Vnitřní okraj je rovněž definovaný elipsou, ale její poloosy jsou kratší o šířku rámečku v příslušném směru. Na obrázku 5.1 je zaoblení rámečku, definované následujícím CSS kódem:

```
border: 8pt solid black;
border-radius: 55pt / 25pt;
```

To znamená, že okraj bude široký 8pt, nepřerušovaný a černý. Rozměry elipsy určující tvar okraje budou následující: délka poloosy elipsy ve směru osy x bude $55pt$ a délka poloosy ve směru osy y bude $25pt$. Vnitřní okraj bude definován elipsou, která bude mít hlavní poloosu o délce $47pt$ a vedlejší poloosu o délce $17pt$.

Speciálním případem je vykreslení rámečku v případě, že šířka sousedního rámečku je $0px$. V tom případě se rámeček postupně ztenčuje až do bodu, kde končí zaoblení rámečku (viz obrázek 5.2).

[1] `https://www.w3.org/Style/CSS/specs.en.html`



Obrázek 5.1: Eliptický tvar vykreslení okraje zaobleného rámečku podle zadaných hodnot. Převzato z [5]

Obrázek 5.2: Vykreslení obdélníku s levým rámečkem a zaoblenými rohy v aktuálních verzích používaných prohlížečů (příklad je z prohlížeče Chrome)

Dalším speciálním případem je, když mají sousední rámečky stejnou šířku, ale rozdílnou barvu. V tom případě je přechod mezi barvami realizován v místě, kde diagonála vykreslovaného obdélníku protíná spoj obou rámečků (viz obrázek 5.3).

Obrázek 5.3: Vykreslení obdélníku s levým a spodním rámečkem (oba rámečky mají stejnou šířku) a zaoblenými rohy v aktuálních verzích používaných prohlížečů (příklad je z prohlížeče Chrome)

Pokud sousední okraje nemají stejnou šířku jako vykreslovaný okraj, je třeba podle definice vykreslovat v rozích okraje tak, že se postupně ztenčují (respektive rozšiřují), aby se uprostřed setkaly se stejnou šířkou. Pro příklad: v případě, že sousední okraje mají šířky $20px$ a $40px$, tak se v rohu se budou setkávat oba s šířkou $30px$ (viz obrázek 5.4). V takovém

případě není přechod barev na diagonále obdélníku, ale na přímce, jejíž směrnice je určena podle poměru šířek jednotlivých stran [5]. Pro výše zmíněný příklad bude tedy přechod cca ve dvou třetinách, blíže k tenčímu rámečku.

Obrázek 5.4: Vykreslení obdélníku s levým a spodním rámečkem (rámečky mají různou šířku) a zaoblenými rohy v aktuálních verzích používaných prohlížečů (příklad je z prohlížeče Chrome)

V případě, že není nastaveno zaoblení rámečku, jsou rohy vykresleny pomocí trojúhelníků (viz obrázek 5.5).

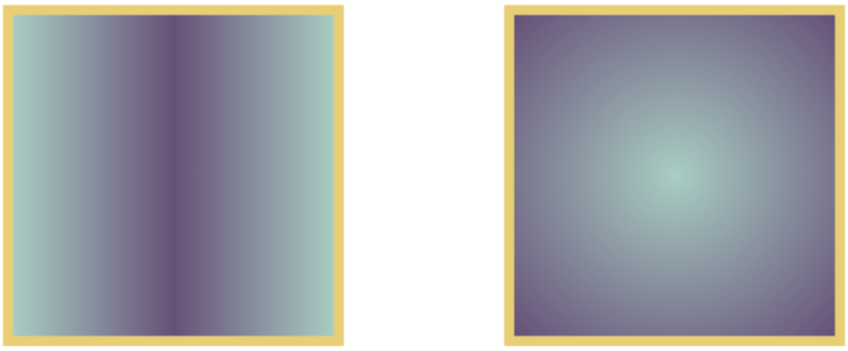
Obrázek 5.5: Vykreslení obdélníku s levým a spodním rámečkem (rámečky mají různou šířku) bez zaoblených rohů v aktuálních verzích používaných prohlížečů (příklad je z prohlížeče Chrome)

5.2 Gradienty

V CSS3 lze používat dva typy gradientů: lineární a radiální. Radiální gradient přechází od středu do krajů a může mít buď kružnicový, nebo eliptický tvar. U lineárního gradientu je možné nastavit směr: Nahoru, dolů, doleva, doprava, diagonálně nebo případně pomocí úhlu, který definuje směr gradientu.

U radiálního gradientu lze nastavit tvar gradientu (kružnice, nebo elipsa), jeho střed a rozměry. Střed gradientu může být definován buď v procentech nebo konkrétní vzdáleností a počítá se od levého horního rohu. Navíc je možné umístění středu definovat pomocí klíčových slov (například *top right* pro pravý horní roh elementu). Rozměry elementu mohou být nastaveny třemi způsoby. Kromě konkrétní vzdálenosti a procentuální hodnoty lze ještě rozměry definovat pomocí klíčových slov *closest-corner, closest-side, farthest-corner, farthest-side.*

Příklady lineárního a radiálního gradientu jsou na obrázku 5.6.



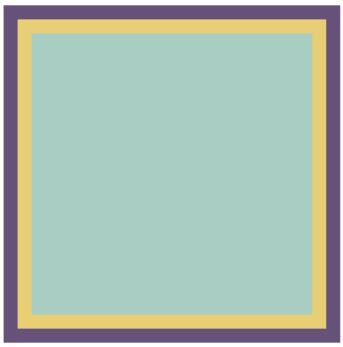
Obrázek 5.6: Příklad lineárního a radiálního gradientu CSS3 (příklad je z prohlížeče Chrome).

5.3 Stínování

Zvláštním případem gradientu je CSS příkaz *box-shadow*, neboli stínování. Tento CSS příkaz má několik parametrů:

```
box-shadow: h-shadow v-shadow blur spread color [inset] ;
```

Parametry *h-shadow* a *v-shadow* umožňují posouvat stín horizontálně, respektive vertikálně. *Blur* přidává rozmazání a zároveň roztahuje stín směrem od hrany. Parametr *color* určuje barvu stínu. Hodnota nastavená u parametru *spread* určuje šířku stínu před tím, než se začne rozmazávat. V případě, že necháme hodnoty *h-shadow*, *v-shadow* a *blur* na 0px, určuje nám prakticky hodnota parametru *spread* tloušťku rámečku, který lze použít jako alternativu k rámečku pomocí CSS příkazu *border*. Na rozdíl od klasického rámečku, *box-shadow* nezvětšuje velikost elementu. Lze použít oba dva příkazy a mít tak dva rámečky (viz obrázek 5.7).



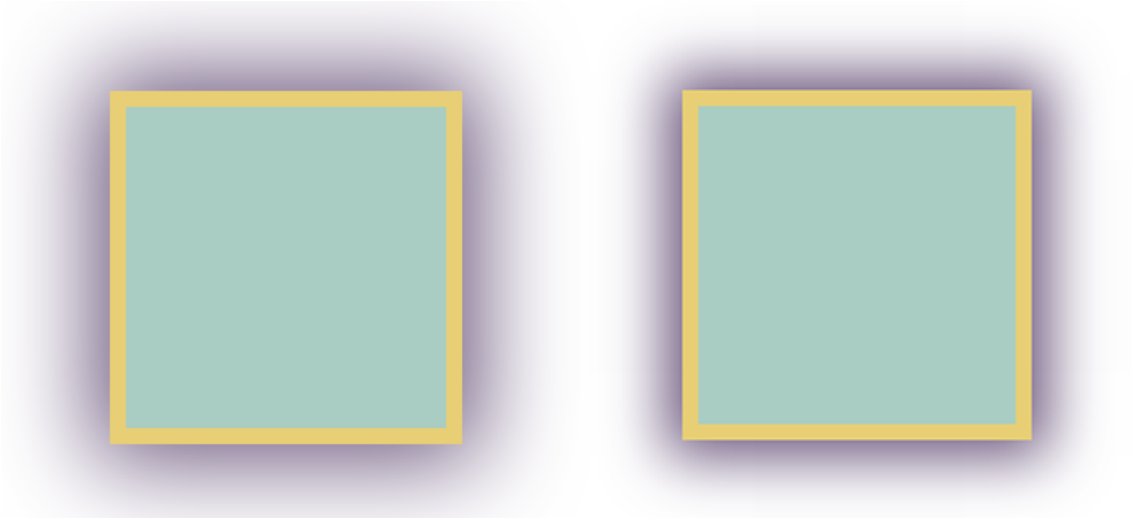
Obrázek 5.7: Vykreslení čtverce s rámečkem a stínováním. Žlutý rámeček je vytvořen pomocí atributu *border*, fialový rámeček je vytvořen pomocí atributu *box-shadow* (příklad je z prohlížeče Chrome).

Volitelný parametr *inset* určuje, zda bude stín směřovat ven od okrajů elementu, nebo naopak dovnitř.

U této CSS vlastnosti se na rozdíl od předchozích moderní prohlížeče velmi liší při vykreslování stínu. Jak je vidět na obrázku 5.8, kde je porovnáno renderování stínu v prohlížečích Chrome a Mozzila Firefox, jsou stíny vykresleny se značnými rozdíly. Především už od pohledu je stín v prohlížeči Chrome výrazně tmavší. V prohlížeči Firefox je stín světlejší, ale zároveň je přechod stínu pomalejší a dosahuje dále než v prohlížeči Chrome. Rozdíl vykreslení je dán tím, že ve specifikacích CSS3 pro implementaci stínování není přesně určen algoritmus, který se má na rozmazání použít. Doporučený algoritmus je Gaussovo rozmazání a je pouze definováno, že barva jednotlivých bodů výsledného stínu se může lišit maximálně o 5% od Gaussova rozmazání.

Čtverce v obou obrázcích mají šířku a výšku 200px a mají nastavený CSS atribut *box-shadow* následovně:

$$box - shadow: \ 0px\ 0px\ 60px\ 10px\ \#665178$$



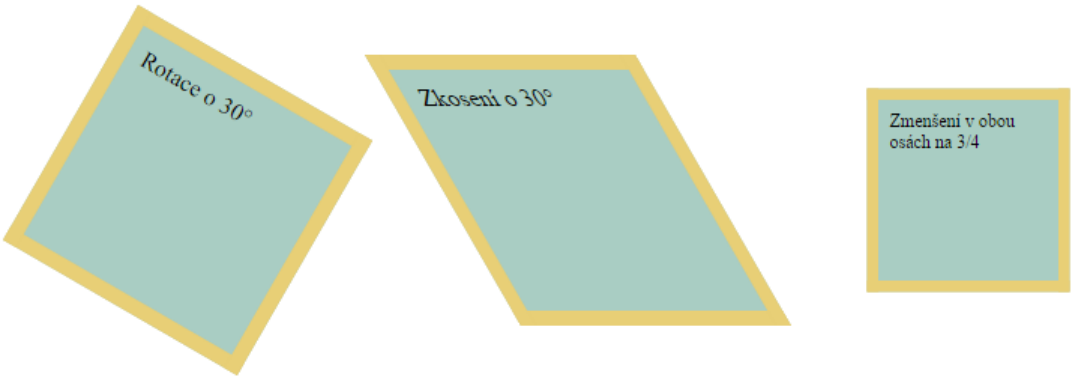
Obrázek 5.8: Porovnání vykreslení stínování mezi prohlížeči Mozilla Firefox (levý obrázek) a Chrome (pravý obrázek)

5.4 2D transformace

2D transformace ve standartu CSS3 umožňují aplikovat na elementy HTML dokumentu následující efekty:

- Změnu velikosti ve směru os x a y (funkce *scale()*, respektive *scaleX()* a *scaleY()*),
- rotaci o určený počet stupňů okolo středu elementu (funkce *rotate()*),
- posun o určený počet pixelů ve směru os x a y (funkce *translate()* respektive *translateX()* a *translateY()*),
- zkosení o určitý počet stupňů, opět lze aplikovat ve směrech obou os (funkce *skew()*, respektive *skewX()* a *skewY()*).

Kromě zmíněných funkcí lze aplikovat také 2D transformační matici pomocí funkce matrix(), pomocí níž lze realizovat jak jednotlivé výše zmíněné funkce, tak i jejich kombinace.



Obrázek 5.9: Příklady některých 2D transformací v CSS (rotace, zkosení a zmenšení).

5.5 3D transformace

3D transformace nabízejí podobné funkce jako 2D transformace, ale navíc přidávají třetí osu Z. To znamená, že přibývají funkce *scaleZ()*, *translateZ()*, atd. Bohužel 3D transformace nejsou v jazyce SVG podporovány. 3D transformace by bylo možné využít z CSS3, nicméně aktuální standart SVG (verze 1.1) zatím nepodporuje CSS3 vlastnosti. To znamená, že 3D transformace by byly použitelné pouze při zobrazování SVG v moderních webových prohlížečích, které podporují CSS3 a lze transformovat pouze celé elementy SVG.

5.6 Průhlednost

Standart CSS3 umožňuje nastavit průhlednost elementů pomocí atributu *opacity*. Hodnota tohoto atributu může být od 0 do 1, přičemž číslo vyjadřuje úroveň průhlednosti. Pokud je hodnota rovna 0, element je zcela průhledný. Pokud je hodnota rovna 1, element je naopak zcela neprůhledný.

5.7 Další CSS3 vlastnosti

Mezi další vlastnosti definované ve standartu CSS3 patří například animace, stínování písma, nebo filtry umožňující různé operace s elementy (rozmazání, převod do odstínů šedi, saturaci, nebo například sépiové tónování). Do CSS3 dále také patří vlastnosti, které upravují "tok dokumentu"- tedy například flexbox, nebo vícesloupcový layout pomocí příkazů *column-**.

Kapitola 6

Návrh implementace pokročilého SVG výstupu pro vybrané CSS3 vlastnosti

V této kapitole je navržen způsob pro implementaci generování SVG výstupu pomocí SVG DOM a jeho napojení na aktuální verzi nástroje CSSBox. Zároveň jsou zde popsány návrhy na implementaci jednotlivých CSS3 vlastností pomocí SVG.

6.1 Napojení SVG rendereru na nástroj CSSBox

Celý projekt bude mít vlastní repozitář git[1] na serveru BitBucket [2] a bude dostupný na této webové adrese [3].

Projekt bude fungovat jako rozšíření pro CSSBox. Bude obsahovat dvě demo aplikace, které jsou i v projektu CSSBox. Konkrétně jde o BoxBrowser a ImageRenderer. Jejich funkčnost je shodná s demo aplikacemi popsanými v kapitole 4, nicméně renderují výstup ve vektorové grafice s podporou vykreslování vybraných CSS3 vlastností. Hlavním rozdílem oproti těmto aplikacím je použití třídy *SVGDOMRenderer*, která implementuje rozhraní *BoxRenderer* popsané v kapitole 4.5.

Implementace generování SVG výstupu v aktuální verzi CSSBoxu vytváří SVG kód přímo v textové podobě do třídy *PrintWriter*, jejíž obsah je nakonec pouze vložen do výsledného souboru. Proto je nutné generovat jak počáteční tagy, tak i ukončující. Tento způsob by obzvlášť u složitějších souborů a s přibývajícími funkcemi mohl dělat problémy. U tohoto stylu zápisu je také nutnost zvlášť ošetřovat hodnoty vkládané do atributů SVG tagů, aby nedošlo k chybám v SVG kódu.

Z výše zmíněných důvodů je třeba implementovat generování SVG pomocí DOM objektu. K tomu bude vhodné využít třídu *SVGDOMImplementation* z java knihovny Apache Batik. Tato třída je oddděděná od třídy *DOMImplementation* z balíčku org.w3c.dom a přidává navíc podporu SVG objektů.

Generování výsledného SVG DOM objektu ve třídě *SVGDOMRenderer* bude tedy probíhat následovně: V konstruktoru bude vytvořen hlavní kořenový uzel DOM struktury (tedy "dokument"). V konstruktoru navíc proběhne inicializace zásobníku, který bude sloužit pro

[1] open source verzovací systém (`https://git-scm.com/`)
[2] Webová služba poskytující řešení pro verzovací systém git (`https://bitbucket.org/`)
[3] `https://tomasam@bitbucket.org/tomasam/cssbox-vectorrenderer.git`

ukládání aktuálního SVG elementu. Díky tomu bude výsledný dokument lépe strukturovaný [4]. Na vrchol zásobníku bude vložen kořenový element a bude vygenerována hlavička SVG souboru. Poté budou volány metody rozhraní *BoxRenderer* podle vykreslované webové stránky (tato logika se nachází v nástroji CSSBox). Po dokončení všech elementů bude nakonec volána metoda *writeFooter* (rovněž z rozhraní *BoxRenderer*), která ovšem bude prázdná, protože není třeba ukončovat žádné elementy.

6.2 Návrh tříd pro renderování SVG výstupu

Jak bylo popsáno v předchozí kapitole, realizaci celého renderování bude mít na starosti třída *SVGDOMRenderer*. Pro implementaci generování jednotlivých CSS3 vlastností bude třeba vytvořit třídy pro jejich reprezentaci. Tyto třídy budou současně sloužit pro zapouzdření náročnějších operací a výpočtů.

Konkrétně bude třeba vytvořit třídy pro reprezentaci následujících vlastností:

- třídy pro rámečky a pro zaoblené rámečky, které budou obsahovat jednotlivé body pro vykreslení zaoblených rámečků (viz dále),
- třídu pro transformace,
- a třídu pro radiální a lineární gradienty a pro jednotlivé hodnoty gradientu (gradient stop).

Vzhledem k tomu, že k implementaci některých CSS vlastností bude třeba vytvářet zanořené SVG elementy, bude v třídě *SVGObjRenderer* zásobník, který bude na svém vrcholu udržovat vždy aktuální SVG element.

6.3 Návrh implementace renderování rámečků a zaoblených rohů

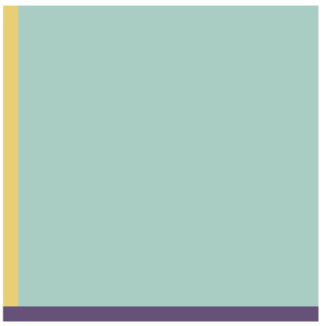
Zaoblené rohy lze v SVG realizovat pomocí atributů *rx* a *ry* u elementu *<rect»*. Ty jsou ale pro realizaci zaoblených rohů, tak jak je definuje CSS3, nevhodné. Určují zaoblení všech okrajů najednou, kdežto v CSS3 je možné definovat zaoblení rohů pro každý okraj zvlášť. Navíc obdélník nelze použít pro renderování rámečků proto, že by pak nebylo možné definovat různě široké rámečky pro každou stranu zvlášť.

Řešení tohoto problému, které se nabízí jako první, je místo úseček pro zaoblené rohy použít křivku. Toto řešení by fungovalo pro okraje o šířce $1px$. V případě, že by byl okraj širší a byl by nastaven pouze pro jednu stranu (např. CSS atribut *border-left: 5px solid red;*), nastal by problém v rozích, kde je nutné okraj v zaoblení postupně ztenčovat do špičky (takto je nadefinováno vykreslování CSS3 rámečků).

V původní verzi SVG rendereru jsou rámečky implementovány pomocí elementu *<path>*, pro každou stranu rámečku zvlášť. Úsečka definovaná elementem *<path>* je vedená středem rámečku a šířka rámečku je nastavována pomocí atributu *stroke-width*, který definuje šířku vedené křivky. Barva je definována pomocí atributu *stroke*. V případě, že je nastavena i css vlastnost border-style na dotted, nebo dashed (tečkovaný, případně čárkovaný okraj),

[4] Teoreticky by bylo i možné generovat většinu elementů bez strukturalizace "vedle sebe". Výhodou by mohla být jednodušší struktura dokumentu, nicméně při generování by bylo nutné například zvlášť řešit ořezávání elementů v případě, že například nějaký rodičovský element v HTML dokumentu má nastavenou vlastnost *overflow:hidden* (tato vlastnost skrývá veškerý obsah, který přesahuje rozměry elementu).

použije se SVG atribut *stroke-dasharray* [5]. Tento způsob není vhodný pro renderování rámečků, protože v rozích nevytváří plynulý přechod, ale vždy poslední renderovaná strana rámečku viz obrázek 6.1.



Obrázek 6.1: Ukázka překrytí rohu rámečku při použití elementu *<path>* s nastavenou šířkou linek.

Pravděpodobně nejlepším řešením bude renderovat rámečky každý zvlášť pomocí tagu *<path>* a dopočítat jednotlivé krajní body. Rámeček tedy nebude vykreslován pomocí křivek a jejich šířky, ale pomocí polygonů, případně dalších objektů a pro vykreslení rámečku s příslušnou barvou bude sloužit atribut *fill* (barva pozadí objektu). Výpočet jednotlivých bodů, potřebných pro vykreslení rámečku, bude rozebrán v následujících podkapitolách.

6.3.1 Třída pro reprezentaci rámečků

Celý rámeček bude reprezentovat třída *Border*, která bude obsahovat původní reprezentaci, tedy pomocí objektu třídy *Rectangle*. Dále bude obsahovat barvy pro každou stranu rámečku a nakonec i pro každý roh jednu třídu *CornerRadius*, obsahující vypočtené body potřebné k vykreslení rohu. V této třídě navíc bude veškerá logika potřebná k výpočtu všech bodů.

6.3.2 Výpočet krajních bodů pro jednotlivé strany rámečku

Jak bylo popsáno v kapitole 5.1, ve specifikacích pro CSS3 rámečky a pozadí [5] je implementace zaoblených rohu popsána takto: Pro každý z rohů rámečku lze definovat dvě hodnoty následovně:

$$\text{border-top-right-radius} : r\ [px|\%]\ /\ s\ [px|\%];$$

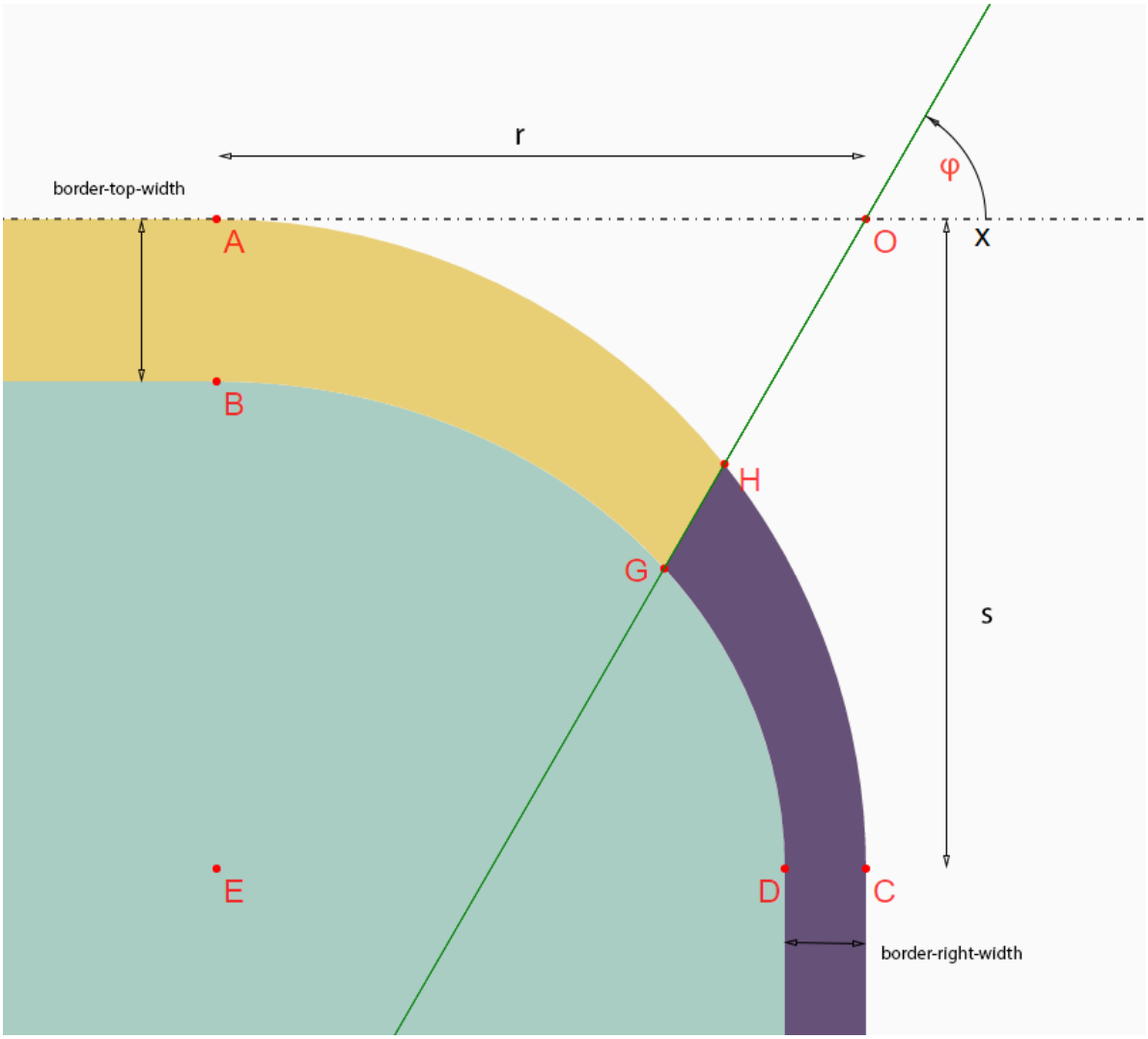
Hodnoty r a s definují délku hlavní a vedlejší poloosy elipsy, jejíž výřez určuje vnější okraj rámečku. První hodnota určuje délku poloosy v horizontálním směru, druhá hodnota určuje délku poloosy ve vertikálním směru. Pokud je jedna z hodnot nulová, vykresluje se rámeček bez zaoblení. Pokud druhá hodnota není nastavena, použije se stejná jako u první.

Vnitřní okraj rámečku je také definován pomocí elipsy, ale délky jejích poloos se získají odečtením šířky rámečku v horizontálním směru pro hodnotu r, respektive odečtením

[5] Atribut *stroke-dasharray* umožňuje vytvořit na lince čárkování. Má dva parametry, přičemž první určuje délku čar a druhý určuje délku mezer. V případě, že chceme například tečkovaný okraj, nastaví se oba parametry na stejnou hodnotu a to konkrétně na tloušťku rámečku.

šířky rámečku ve vertikálním směru pro hodnotu s. V případě, že rozdíl hodnot je alespoň v jednom ze směrů menší nebo roven 0, je roh vykreslen v pravém úhlu.

Body, které budou potřeba pro vykreslení rámečku jsou na obrázku 6.2. Bod O je v rohu elementu, pro který je rámeček vykreslován. Body A, B, C, D a E lze dopočítat přičtením, nebo odečtením (to záleží na tom, pro který z rohů je rámeček vykreslován) příslušných šířek rámečků, nebo hodnot r a s od souřadnic bodu O. Způsob získání bodů G a H bude popsán v následujících podkapitolách.



Obrázek 6.2: Detailní zobrazení zaobleného rohu rámečku, včetně všech bodů potřebných k jeho vykreslení.

6.3.3 Přechod mezi rámečky s různou barvou

Body A, B, C, D a E stačí pro vykreslení rámečku v případě, že je celý rámeček jednobarevný. Pokud ale má každá strana rámečku jinou barvu, je třeba v určitém místě vytvořit přechod těchto barev. Ve specifikaci pro CSS rámečky [5] je řečeno následující: Přechod mezi styly jednotlivých stran se musí nacházet v oblasti vymezené zaoblením a zároveň

tento přechod musí být na monotónní spojité funkci, která odpovídá poměru šířek sousedních stran rámečku. Zjednodušeně se přechod musí nacházet v oblasti vymezené body $AOCE$ na obrázku 6.2 a střed přechodu mezi barvami nachází na přímce, jejíž směrnice je určena poměrem šířek rámečků. Přesná poloha této přímky je již ponechána na interpretu.

Pro vykreslení přechodu bude tedy potřeba získat krajní body úsečky, která bude tvořit hranici mezi stranami rámečku (na obrázku 6.2 body G a H). Tyto body lze získat výpočtem průsečíku přímky s vnější elipsou (bod G) a průsečíku přímky s vnitřní elipsou (bod H). Obecně jsou průsečíky přímky s elipsou samozřejmě dva (pokud přímka není tečnou elipsy, v tom případě je jen jeden průsečík). Potřebné body se nachází ve čtverci $AOCE$.

Zbývá tedy získat rovnici přímky, na které budou ležet zmíněné hraniční body. Tato přímka má podle specifikace definovanou směrnici, ale nemá definovaný výsek na ose y (koeficient q ve směrnicovém tvaru přímky). Přímka může procházet buď středem elipsy (bod E) která definuje okraj, nebo krajním rohem obdélníku (bod O). Směrnici přímky získáme jako tangens úhlu mezi přímkou a osou y (úhel φ v obrázku 6.2). Velikost úhlu v radiánech [6] a následně směrnice k budou vypočteny pomocí následujících vzorců:

$$\varphi = \frac{\text{border-top-width}}{\text{border-top-width} + \text{border-right-width}} \frac{\pi}{2}$$
$$k = tan(\varphi)$$

Směrnicový tvar přímky:

$$y = kx + q$$

získáme dopočítáním koeficientu q. Ten získáme dosazením směrnice a souřadnic bodu O do rovnice:

$$q = x_O - ky_O$$

Je třeba vzít v potaz, že funkce tangens není definovaná pro $\frac{\pi}{2}$. Tento případ ale bude ošetřen speciálně, protože úhel mezi osou x a hraniční přímkou může být roven $\frac{\pi}{2}$ radiánů jen v případě, že jedna ze stran rámečku má šířku rovnu *0*.

6.3.4 Průsečík přímky a elipsy

Pro nalezení obou průsečíků přímky a elipsy je třeba vyřešit soustavu dvou rovnic o dvou neznámých. První rovnice je směrnicový tvar přímky a druhá rovnice je rovnice elipsy v kanonickém tvaru (viz vzorec 2). Soustava rovnic by v případě, že ani jedna z poloos elipsy není nulová, měla by mít vždy dvě řešení (viz předchozí podkapitola).

$$y = kx + q$$
$$\frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2} = 1$$

Soustavu je nutné vyřešit obecně, abychom získali vzorce pro dopočítání souřadnic průsečíků. Nejprve je potřeba oddělit neznámé v rovnici elipsy:

$$b_2x^2 - 2b_2x_0x + a^2y^2 - 2a^2y_0y = a^2b^2 - b^2x_0^2 - a^2y_0^2$$

[6] Základní matematické funkce v balíčku *java.lang.Math* pracují s úhly pouze ve formě radiánů.

Do upravené rovnice elipsy se nahradí y za pravou stranu rovnice přímky. Výsledná rovnice se postupně upraví až do tvaru $ax^2+bx+c=0$, tedy kvadratická rovnice v základním tvaru, ze které pak pomocí determinantu lze dopočítat kořeny rovnice:

$$\frac{(b^2+a^2k^2)}{2}x^2+(a^2kq-b^2x_0-a^2ky_0)x-\frac{a^2b^2-b^2x_0^2-a^2y_0^2+2a^2qy_0-a^2q^2}{2}=0$$

Kořeny kvadratické rovnice se vypočítají následovně:

$$x_{1,2}=\frac{-b\pm\sqrt{D}}{2a},$$

kde D je determinant a získáme ho podle vzorce:

$$D=b^2-4ac$$

Po dosazení:

$$D=(-b^2x_0+kqa^2-a^2kq_0)^2-4\frac{(b^2+a^2k^2)}{2}\frac{a^2b^2-b^2x_0^2-a^2y_0^2+2a^2qy_0-a^2q^2}{2}$$

$$D=a^2b^4-a^2b^2y_0^2-2a^2b^2qx_0+2a^2b^2x_0y_0+2a^2b^2qy_0-a^2b^2q^2+a^4b^2k^2-a^2b^2k^2x_0^2$$

Nyní je ještě vhodné vytknout a^2b^2 a zavést substituci $f=q+kx_0$:

$$D=a^2b^2(a^2k^2+b^2-y_0^2-f^2-2fy_0)$$

Nyní tedy máme x-ové souřadnice průsečíků přímky s elipsou (hodnoty x_1 a x_2). Stejným způsobem lze získat i souřadnice průsečíků na ose y (opět jde o řešení kvadratické rovnice a řešením jsou hodnoty y_1 a y_2).

Řešením soustavy rovnic jsou tedy dva body $M(x_M,y_M)$ a $N(x_N,y_N)$. Nyní už stačí jen vybrat ten bod, který se nachází v obdélníku vymezeném body $AOCE$.

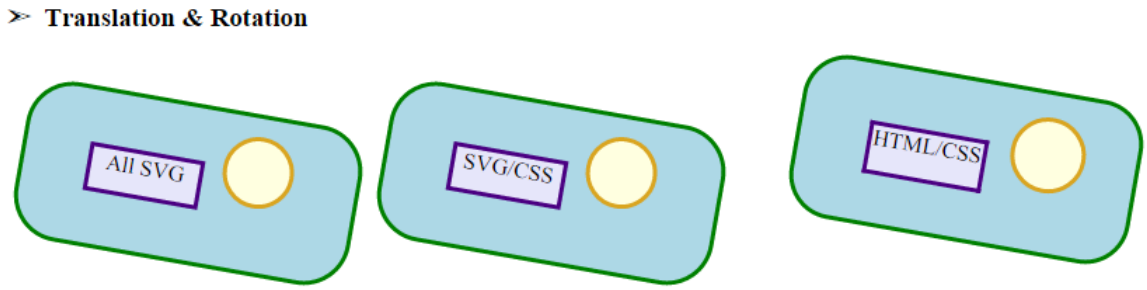
6.3.5 Konečné vykreslení rámečku

Se všemi potřebnými body lze nyní vykreslit obě dvě části zaobleného rámečku. Pro vykreslení rámečku bude použit SVG element *<path>* a bude se skládat ze dvou rovných čar a ze dvou eliptických křivek (příkaz *A*, popsaný v 2.6.2). Jedna část bude mezi body A,B,G a H a druhá část mezi body C,D,G a H. Zbytek rámečků (mezi rohy) lze vykreslit buď pomocí elementu *<path>* o dané šířce rámečku, případně vytvořit obdélník pomocí krajních bodů.

Rámečkům v CSS lze nastavit různé styly (atribut *border-style*). Nejpoužívanější je styl solid (plná čára), dále ale může být například dotted (tečkovaný), dashed (čárkovaný), double (dvojitý rámeček), inset (obsah elementu se opticky zdá být propadlý do stránky) a další. U rovných čar by pro tečkovaný a čárkovaný styl bylo možné využít SVG atribut *stroke-dasharray*, ale ostatní styly by bylo velmi náročné implementovat pomocí SVG, obzvlášť pro zaoblené rohy. Proto nakonec bylo rozhodnuto, že styly rámečků nebudou implementovány.

6.4 Návrh implementace renderování 2D transformací

Jazyk SVG podporuje 2D transformace podobně jako CSS3, takže by neměl být problém převést 2D transformace na vektorový výstup. Na obrázku 5.9 jsou příklady některých transformací. Jediný problém by mohl být v posunu při transformaci. Počáteční bod pro transformace (atribut *transform-origin*) je totiž možné u transformací v CSS libovolně nastavit. Oproti tomu SVG má vždy počáteční bod transformace v levém horním rohu dokumentu [18] a kromě rotace elementu (transformační funkce *rotate*) není možné počáteční bod měnit. Na obrázku 6.3 lze vidět porovnání transformací při použití stejných SVG transformací a CSS transformací bez jakékoliv úpravy. Vlevo je transformace čistě v jazyce SVG, tedy pomocí SVG transformací. Uprostřed je SVG objekt, na který jsou aplikovány CSS transformace. Poslední obrázek je transformace HTML elementu pomocí CSS. Pro implementaci transformací tedy bude potřeba použité transformační funkce upravovat.



Obrázek 6.3: Porovnání transformace (rotace) při použití SVG transformace v SVG, CSS transformace v SVG a CSS transformace v HTML. (Převzato z [18])

Co se týče samotných funkcí pro 2D transformace, tak jsou téměř shodné s těmi v CSS3. Jediná funkce, která je navíc, je zkosení (*skew*) podle obou os najednou. To je pouze v CSS3, nicméně díky funkci *matrix* lze jednoduše nahradit.

Jak bylo zmíněno výše, problémem při renderování 2D transformací do SVG je fakt, že CSS3 transformace využívají jiný počáteční bod transformace, respektive umožňují si tuto hodnotu nastavit pomocí zmíněného CSS3 atributu. Rozdíl lze vyřešit tím, že nejprve transformovaný element přeneseme pomocí funkce *translate* tak, aby počáteční bod SVG transformace odpovídal počátečnímu bodu v CSS3 a po provedení příslušných transformací ho opět pomocí stejné transformační funkce vrátíme na původní místo.

6.5 Návrh implementace renderování průhlednosti

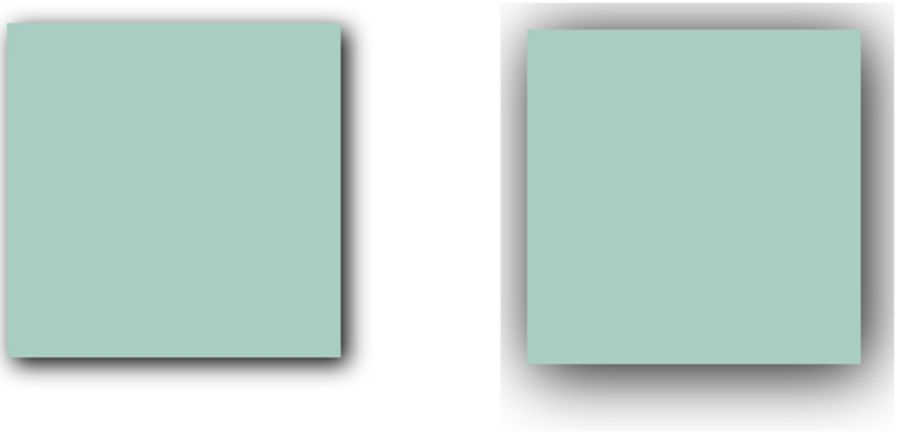
Průhlednost lze v SVG nastavit pomocí atributu *opacity* a funguje stejně jako v CSS3, tedy hodnota musí být desetinné číslo mezi 0 a 1, kde 0 znamená, že element bude zcela průhledný (nebude vidět vůbec), a 1 znamená, že element nebude průhledný. Stačí tedy informaci z CSS3 stylů přenést do SVG atributů.

Je třeba dát pozor na to, že průhlednost se týká i rámečku a ten se v nástroji CSSBox renderuje zvlášť od obsahu v metodě *renderElementBackground*. Vzhledem k tomu, že tato metoda může a nemusí být volána před metodou *startElementContents* bude nejvhodnější si ukládat informaci o tom, jestli již byl vytvořen obalovací element s nastavenou průhledností.

6.6 Návrh implementace renderování stínování

Stínováni by teoreticky bylo možné v SVG implementovat pomocí kombinace několika filtrů. Fungování SVG filtrů není v této práci popsáno, ale jejich funkčnost je definována zde [9].

Stínování by bylo možné realizovat pomocí filtru *feFlood*, který vytvoří výplň pozadí s požadovanou barvou (i s případnými převisy, pokud je nastaven parametr *spread*), dále pak filtr *feOffset*, který by realizoval posun stínu v horizontálním a vertikálním směru a nakonec filtr *feGaussianBlur*, který by vytvářel rozmazání stínu. Na obrázku 6.4 vlevo lze vidět stínování čtverce vytvořené pomocí SVG.



Obrázek 6.4: Stínování vytvořené pomocí filtrů v SVG (vlevo). Omezení plochy filtru v SVG (vpravo).

Na obrázku 6.4 vpravo lze vidět omezení plochy filtru, které je dáno implicitním rozměrem filtru, který je nastaven na 10% šířky, respektive výšky ve všech směrech. Pro správné zobrazování filtru je třeba tyto hodnoty přenastavit tak, aby nebylo omezeno vykreslení stínu.

6.7 Návrh implementace renderování gradientů

Na začátek je třeba zmínit, že ani lineární, ani radiální gradienty nejsou implementovány v nástroji CSSBox. Již v nástroji jStyleParer, který zpracovává CSS nejsou informace o gradientech ukládány. Při implementaci a testování gradientů tedy bude potřeba simulovat styly gradientů. Zároveň bude třeba vytvořit i třídy, do kterých se budou gradienty ukládat. Při případném doimplementování gradientů do CSSBoxu bude potřeba tyto třídy napojit na ty z nástroje jStyleParser.

Lineární i radiální gradienty jsou definovány i v SVG. Lineární gradient se v SVG definuje pomocí elementu *<linearGradient/>* a radiální elementem *<radialGradient/>*. Definice všech gradientů musí být v elementu *<defs>* [7]. Ani lineární ani radiální gradient nemohou být nikdy vykresleny přímo ve stránce a lze je použít jen pomocí parametrů *fill* nebo *stroke*. U lineárního gradientu se nastavují čtyři parametry v procentech: x_1, y_1, x_2

[7] Element *<defs>* je zkratkou pro "definitions"a slouží pro definici různých elementů, například tvarů, gradientů, filtrů, masek, animací nebo i celých podčástí svg dokumentu. Tyto elementy pak mohou být používány a odkazovány v rámci dokumentu.

a y_2, které definují souřadnice dvou bodů (začátek a konec gradientu). U radiálního gradientu se nastavuje pět parametrů: c_x, c_y, které určují střed vnitřního kruhu, f_x, f_y, které určují střed vnějšího kruhu. Poslední parametr r určuje poloměr vnějšího kruhu a také značí hranici gradientu.

Oba typy gradientů pak mají v obsahu definovaný seznam barev a jejich vzdálenost pomocí tagů *<stop>*

```
<stop offset="20%" style="stop-color: red;">
```

Atribut *offset* určuje umístění barvy a může být definován v procentech, nebo jako desetinné číslo mezi 0 a 1. Oba dva způsoby určují umístění zastávky gradientu v poměru k celkové velikosti gradientu. Zastávky gradientu mohou být v rozmezí 0 až 100%

6.7.1 Třídy pro ukládání gradientu

Pro lineární i radiální gradient bude třeba vytvořit zvlášť třídy, které budou ukládat informace o gradientu. Tedy pro lineární gradient úhel, souřadnice počátečního a konečného bodu a velikost elementu, na který se gradient aplikuje. Dále bude obsahovat seznam jednotlivých gradientových zastávek, které budou reprezentovány vlastní třídou. U radiálního gradientu je třeba ukládat větší množství informací, protože radiální gradient v CSS má mnoho různých možností zápisu.

Vzhledem k tomu, že je třeba generovat pro různé velikosti elementů zvlášť gradienty, bylo by vhodné se pokusit gradienty použít vícekrát, pokud to bude možné. Hodilo by se tedy ukládat identifikátory vygenerovaných gradientů společně s informacemi o velikosti elementu, směru gradientu a jednotlivých zastávek a v případě, že se gradient opakuje, negenerovat nový, ale použít ten uložený.

6.7.2 Návrh implementace lineárního gradientu

Lineární gradient je v CSS3 definován úhlem určujícím směr gradientu a dále pak jednotlivými barvami a jejich hranicemi. Pro převedení lineárního gradientu z CSS3 do SVG bude tedy potřeba dopočítat souřadnice počátečního a konečného bodu vyjádřených v procentech vzhledem k velikosti objektu.

Instrukce k implementaci lineárního gradientu ve specifikacích [6] určují vykreslení gradientu následovně: Střed gradientu se nachází uprostřed boxu, ve kterém je gradient vykreslován. Na přímce (osa gradientu), která prochází středem a svírá s osou y určený úhel gradientu, se nachází počáteční a konečný bod gradientu. Podle definovaného úhlu jsou následně určeny rohy (pro 45° jsou to levý spodní a pravý horní roh), kterými jsou vedeny kolmice na osu gradientu. Průsečíky kolmic s osou gradientu jsou počáteční a konečný bod gradientu.

Vzhledem k tomu, že výsledek musí být v procentech, bude třeba po provedení výpočtu převést výsledek na procenta.

6.7.3 Výpočet počátečního a konečného bodu lineárního gradientu

Pro výpočet souřadnic obou bodů bude nejprve třeba získat směrnicový tvar osy gradientu. Jak bylo popsáno výše, osa gradientu prochází středem obdélníku, na kterém se gradient vykresluje a svírá s osou x určený úhel. Tento úhel bude třeba posunout o 90° oproti úhlu definovanému v CSS3. Je to proto, že úhel lineárního gradientu se v CSS3 počítá od osy y

po směru hodinových ručiček a úhel pro výpočet směrnice se počítá od osy x proti směru hodinových ručiček.

Po přepočtení úhlu získáme pomocí funkce tangens směrnici osy gradientu a dosazením souřadnic a směrnice do směrnicového tvaru přímky získáme tzv. výsek na ose y (hodnotu q).

$$y = kx + q$$

Po získání směrnicového tvaru osy gradientu je třeba vypočítat ještě směrnicové tvary pro kolmice procházející vybranými rohy. Směrnici kolmice na přímku k^n (tzv. normály) lze vypočítat podle vzorce:

$$k^n = -\frac{1}{k}$$

kde k je směrnice původní přímky a k^n je směrnice normály. Nakonec už stačí jen dosadit příslušné body do rovnice k získání hodnot q obou kolmic.

Výsledný počáteční a koncový bod gradientu vypočítáme jako výsledek řešení soustavy dvou rovnic o dvou neznámých:

$$k_1 x - y + q_1 = 0$$
$$k_2 x - y + q_2 = 0$$

Nejprve odečteme druhou rovnici od první a postupně získáme x. Pak už jen x dosadíme do jedné z rovnic a tím získáme y

$$(k_1 - k_2)x + q_1 - q_2 = 0$$
$$x = \frac{q_2 - q_1}{k_1 - k_2}$$
$$y = k_1 * (\frac{q_2 - q_1}{k_1 - k_2}) + q_1$$

Po výpočtu průsečíků už stačí jen přepočítat souřadnice na procenta.

6.7.4 Úprava vypočítaných souřadnic podle rozměrů elementu

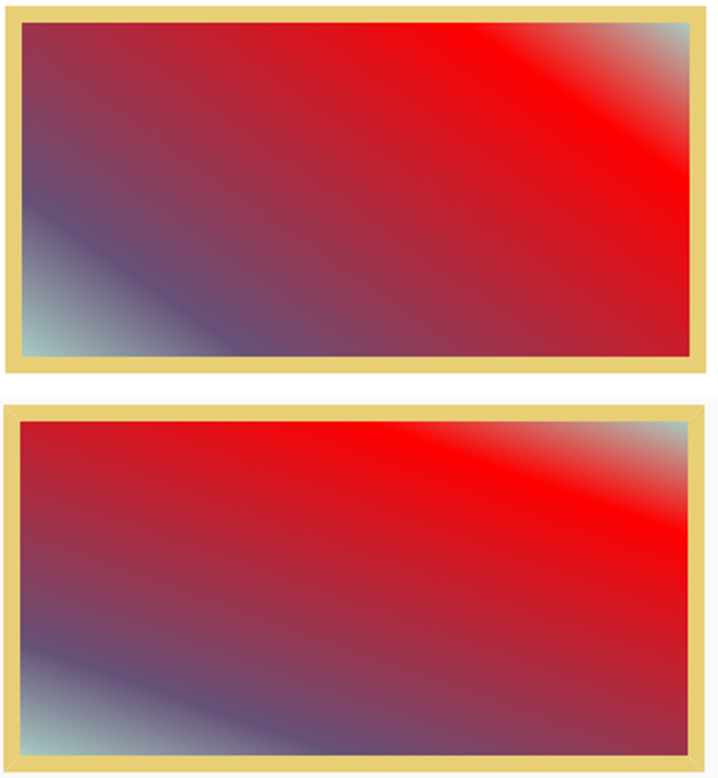
Výpočet popsaný v předchozí podkapitole funguje správně pouze v případě, že obdélník vymezující element, na který se gradient vykresluje, je zároveň čtverec. Při použití takto vypočtených hodnot se směr gradientu změní, pokud změníme poměr šířky a výšky elementu, na který se gradient aplikuje (viz obrázek 6.5).

Pro vyřešení zmíněného problému je třeba upravit směrnici osy gradientu. Nová směrnice se vypočte podle poměru šířky (w) a výšky (h) elementu následovně:

$$k' = \frac{k}{\frac{w}{h}},$$

kde k' je nová směrnice a k je původní směrnice.

Kvůli tomu, že je potřeba přepočítávat souřadnice pro většinu elementů zvlášť, bude možnost znovupoužití již nadefinovaných gradientů výrazně omezená. I tak by teoreticky bylo možné si ukládat již vygenerované gradienty, včetně poměru stran elementu. Poté při renderování dalšího gradientu by se vždy porovnávalo s předchozími gradienty v SVG a v případě, že by se shodoval s některým již vyrenderovaným, pouze by se na něj odkázalo pomocí jeho identifikátoru.



Obrázek 6.5: Porovnání vykreslení gradientu na obdélníkovém elementu. Nahoře je verze HTML vygenerovaná prohlížečem Chrome a na spodním obrázku je gradient vygenerovaný v SVG pro čtvercový element aplikovaný na obdélníkový element.

6.7.5 Návrh implementace radiálního gradientu

CSS umožňuje definovat dva tvary radiálního gradientu: Kružnici a elipsu. Oproti tomu SVG nabízí různé možnosti manipulace s tvarem výsledného gradientu díky tomu, že umožňuje, kromě středu gradientu, definovat i střed vnějšího tvaru (ať už je to kružnice nebo elipsa). Lze pak dosáhnout různých zajímavých tvarů. Pro renderování CSS3 radiálních gradientů pomocí SVG ale budou vždy oba středy na stejných souřadnicích.

SVG bohužel neumožňuje rozlišovat mezi kružnicovým a eliptickým gradientem. Tvar výsledného gradientu záleží na tvaru elementu, pro který je gradient použit (pro čtvercový element má gradient tvar kružnice a pro obdélníkový má tvar elipsy).

V případě, že je gradient definovaný jako kružnice, stačí pouze dopočítat jeho umístění a poloměr kružnice. Vzhledem k tomu, že hodnoty mohou být zadány jak v procentech, tak i konkrétní hodnotou, je nutné znát rozměry elementu, pro který je gradient generován. Umístění středu gradientu je buď definováno konkrétní vzdáleností od levého horního rohu, nebo jedním z klíčových slov (*top*, *left*,...). Tyto hodnoty je třeba přepočítat na absolutní vzdálenosti, respektive na souřadný systém, kde levý horní roh je počátek se souřadnicemi $(0, 0)$. Po získání souřadnic středu $S(S_x, S_y)$ gradientu již lze dopočítat výsledný poloměr. Jak bylo zmíněno v kapitole 5.2, poloměr kružnice může být definován jedním z klíčových slov (*closest-corner, closest-side*,...). V tom případě je potřeba spočítat vzdálenost od středu k rohům, respektive ke stranám. Souřadnice rohů lze získat ze šířky a výšky elementu, pro který je gradient generován. Všechny čtyři vzdálenosti se pak dopočítají pomocí Pythago-

rovy věty. Vzdálenost středu gradientu od stran lze spočítat jako absolutní hodnotu rozdílu příslušných souřadnic od šířky, případně výšky elementu.

U eliptického gradientu bude výpočet velmi podobný jako u kružnicového. Hlavním rozdílem je, že se místo poloměru dopočítají délky hlavní a vedlejší poloosy elipsy.

Po získání všech potřebných hodnot je třeba souřadnice a rozměry přepočítat na procenta, vzhledem k původnímu elementu. Pro aplikaci radiálních gradientů bude vždy použit SVG element *<rect/>*. Tvar tohoto elementu pro kružnicové gradienty bude čtvercový a pro eliptické gradienty bude obdélníkový.

Kapitola 7

Implementace renderování

V této kapitole jsou popsány konkrétní detaily implementace. Nejprve jsou popsány použité technologie a struktura aplikace. Poté následuje popis implementace generování SVG kódu pro jednotlivé CSS3 vlastnosti.

7.1 Struktura aplikace

Aplikace je rozdělena do jednotlivých balíčků po vzoru balíčkové struktury nástroje CSSBox. Konkrétně aplikace obsahuje balíčky *render*, *demo* a *layout*. V prvním zmíněném balíčku se nachází třída *SVGDOMRenderer*, která zpracovává po jednotlivých elementech vstupní HTML dokument a z těchto elementů pak postupně vytváří výsledný SVG dokument, který ukládá do třídy *java.io.PrintWriter*.

V demo balíčku se nacházejí demo aplikace ImageRenderer a BoxBrowser. Obě tyto třídy jsou téměř shodné se stejnojmennými třídami v balíčku *cssbox.demo*, ale pro renderování výsledného obrázku používají na rozdíl od CSSBoxu nově vytvořenou třídu *SVGDOMRenderer*. Demo aplikace BoxBrowser navíc místo původního plátna *BrowserCanvas* [1], které je použito v původní implementaci, využívá JSVGCanvas z knihovny Apache Batik, který umožňuje vykreslit SVG soubor do Swing komponenty. Díky tomu je možné experimentálně testovat a případně i používat vektorový výstup přímo v běžící aplikaci.

Posledním zmíněným balíčkem je balíček *layout* obsahující třídy, které slouží k reprezentaci dat potřebných ke správnému vykreslování SVG elementů. Tyto třídy budou postupně popsány v následujících podkapitolách.

7.2 Použité technologie

Z použitých technologií je třeba zmínit především knihovnu Apache Batik, která poskytuje velké množství nástrojů, modulů, nebo i standalone aplikací pro práci s SVG. Z této knihovny je používána především třída *SVGDOMImplementation*, která vychází z rozhraní *DOMImplementation* z balíčku *org.w3c.dom* (základní rozhraní pro DOM objekty v jazyce Java). Další modul, který byl při implementaci použit je již zmíněná Swing komponenta JSVGCanvas z balíčku *batik.swing* [2].

[1] *BrowserCanvas* je Java Swing komponenta definovaná v nástroji CSSBox. Tato třída je odděděná od třídy *JPanel*.

[2] Balíček *batik.swing* není součástí základní knihovny Apache Batik (batik-svg-dom), ale nachází se v balíčku batik-swing.

Další použité technologie jsou úzce svázány s tím, že implementovaný renderer pro svg výstup vychází z nástroje CSSBox. Mezi použitými knihovnami a nástroji jsou GUI knihovna Swing, nebo knihovna pro parsování a zpracovávání kaskádových stylů CSS jStyleParser.

7.3 Implementace renderování rámečků

Implementace renderování rámečků byla svým způsobem nejnáročnější z celé implementace. Aby bylo možné výsledný SVG kód generovat společně pro všechny rohy, je třeba jednotlivé body potřebné k vykreslení rámečku dopočítávat pro každý roh zvlášť. Díky tomu jsou některé části kódu velmi rozsáhlé.

7.3.1 Popis pomocných tříd

Pro dopočítávání a vykreslování rámečků slouží dvě třídy. Třída *Border* obsahuje atribut typu *LengthSet*, ve kterém jsou uloženy šířky jednotlivých stran rámečků. Pro každou ze stran rámečku je zároveň uložena její barva v atributech typu *TermColor* z balíčku jStyleParser. Atribut *borderBounds* typu *Rectangle* určuje počáteční bod a rozměry obdélníku, který tvoří ohraničení vykreslovaného elementu (po přičtení šířek rámečků). Dále jsou ve třídě *Border* souřadnice krajních bodů ve kterých přechází rovná část rámečku na zaoblenou. Tyto body jsou využity pro vykreslení rovných částí rámečku. Poslední čtyři atributy ve třídě *Border* jsou typu *CornerRadius*, jeden pro každý z rohů.

Třída *CornerRadius* obsahuje údaj o zaoblení rámečku v horizontálním a ve vertikálním směru. Dále jsou v této třídě uloženy souřadnice všech bodů popsaných v kapitole 6.3.3. Body A, B, C, D, E a O jsou celočíselné a body G a H jsou v číslech s plovoucí čárkou. Třída dále obsahuje atribut typu *Rectangle*, který vymezuje prostor, ve kterém je zaoblení vykresleno (čtverec vymezený body *AOCE*, ve kterém budou průsečíky obou elips s přímkou). Poslední dva atributy určují hodnoty k a q ze směrnicového tvaru přímky.

7.3.2 Příprava dat k vykreslování rámečku

Rámečky vykreslovaných elementů se renderují společně s pozadím v metodě *renderElementBackground* třídy *SVGDOMRenderer*. Vnější okraj rámečku definuje okraje pozadí elementu. Je tedy potřeba mít připravené okrajové body, respektive ořezový element *<clipPath/>*, jehož tvar bude definovaný pomocí grafického elementu *<path>* .

Výpočet souřadnic bodů potřebných k renderování byly v první verzi implementovány do jedné metody (*WriteBorderCorners*) a pro zjednodušení bylo renderování implementováno pouze pro jeden roh. Byl vybrán pravý horní roh ze dvou důvodů: Zaprvé, protože na tomto rohu byl proveden návrh. Druhým důvodem byl jednodušší výpočet směrnice hraniční přímky mezi dvěma stranami rámečku a tím pádem i průsečíků přímky s elipsami. Tato volba sice nejprve přinesla relativně rychlou cestu k funkčnímu prototypu, ale díky jednoduššímu výpočtu směrnice bylo pak nutné dodělávat generování směrnice přímky i pro ostatní rohy. Při dopočítávání hodnot pro ostatní rohy postupně narůstala velikost zmíněné metody a při dokončení dosáhla cca 350 řádků, což už bylo pro jakoukoliv další úpravu či další použití neúnosné. Z toho důvodu bylo veškeré dopočítávání potřebných dat přesunuto zvlášť do třídy *Border*.

Nakonec tedy výpočet veškerých potřebných dat probíhá v konstruktoru třídy *Border*. Konstruktor má celkem tři parametry. Prvním parametrem jsou šířky jednotlivých stran

rámečku, druhým je obdélník vymezující vnější okraje rámečku. Poslední parametr je typu *ElementBox*, což je abstraktní třída definovaná v nástroji CSSBox a reprezentuje DOM elementy ze zpracovávaného HTML dokumentu. Parametrem tedy může být jedna ze dvou tříd odděděných od třídy *ElementBox*, tedy *InlineBox*, nebo *BlockBox* viz 4.1. Třetí parametr slouží především k získání konkrétních hodnot CSS atributu *border-radius* (respektive jeho částí - *border-top-left-radius*, atd) a stejně tak k získání barev jednotlivých stran rámečku.

V konstruktoru tedy dojde nejprve k získání potřebných informací. Poté následuje dopočítání souřadnic všech bodů, které budou sloužit k vygenerování výsledného SVG elementu. Nejprve se dopočítají krajní body rovných částí rámečku v metodě *calculateBorderPoints*. Jak již bylo zmíněno výše, pro každý z rohů se liší dopočítávání krajních bodů a směrnic hraničních přímek. Výsledná metoda *calculateRadiusPoints* má tedy nakonec stejně něco málo přes sto řádků, ale i přesto je relativně čitelná (výpočet je oddělen pro každý z rohů zvlášť) a dala by se tedy případně alespoň rozčlenit do čtyř metod (pro každý roh zvlášť). Alternativou k tomuto řešení by bylo ukládat šířky rámečků a velikost zaoblení v záporných hodnotách tam, kde se hodnoty mají odečítat. Pak by bylo možné mít společný výpočet pro všechny rohy.

7.3.3 Výpočet hraničních bodů v zaobleném rámečku

Pro jeden samostatný roh probíhá dopočítávání bodů následovně: Body A, B, C, D a E jsou získány pouze ze základních údajů (souřadnice bodu O, šířka rámečku, případně hodnota atributu *border-radius*). Směrnice přímky a výsek na ose y (hodnoty k a q ze směrnicového tvaru přímky) jsou dopočítány podle vzorců uvedených v kapitole 6.3.3. Výpočet se pouze liší pro pravý dolní a levý horní roh, kde je třeba vynásobit hodnotu směrnice k číslem -1. To je způsobeno tím, že úhel, který se vypočítává poměrem šířek sousedních rámečků, je v rozmezí $0° - 90°$, ale úhly, které svírají hraniční přímky těchto rohů s osou x, jsou v rozmezí $90° - 180°$ a výsledná směrnice je tedy kolmicí na vypočítanou směrnici.

Po získání přímky je třeba získat rovnice obou elips v kanonickém tvaru (respektive hodnoty z nich) a dopočítat jejich průsečíky s přímkou. Protože průsečíky s elipsami budou vždy dva, je nutno vybrat ten, který se nachází v obdélníku $AOCE$. Výsledné průsečíky jsou tedy body G a H.

Existuje několik speciálních případů, kdy se body G a H nevypočítávají způsobem popsaným výše. První případ nastává, pokud je zaoblení rámečku nastaveno na implicitní hodnotu (0px). V takovém případě jsou souřadnice bodu H shodné s bodem O a souřadnice bodu G jsou ve vzdálenosti šířky rámečků (na ose x šířka levé nebo pravé strany a na ose y šířka horní nebo spodní strany) směrem dovnitř elementu. Přechod mezi stranami rámečku je tedy vedený na přímce přímo protínající vnitřní a vnější roh rámečku (viz obrázek 5.5).

7.3.4 Vykreslení rámečku pomocí SVG

Výsledné vykreslení jedné poloviny zaobleného rohu je realizováno následující sekvencí příkazů v SVG elementu *<path/>*.

```
M Bx By
A r' s' 0 0 1 Gx Gy
L Hx Hy
A r s 0 0 0 Ax Ay
L Bx By
```

Hodnoty r a s jsou délky poloos vnější elipsy. r' a s' jsou délky poloos vnitřní elipsy. Konkrétní význam jednotlivých příkazů a jejich parametrů je popsán v kapitole 2.6.2, ale zjednodušeně jde o umístění počátečního bodu na souřadnice bodu B, následované eliptickým obloukem na vnitřní straně rámečku. Po prvním oblouku následuje úsečka do bodu H a nakonec vnější eliptický oblouk vedený do bodu A. Poslední příkaz (úsečku z bodu A do bodu B) lze vynechat, protože v případě, že koncový bod elementu *<path>* v SVG není shodný s počátečním bodem, je použita úsečka z koncového bodu do počátečního.

Pro určení barvy je elementu *<path>* kromě atributu *d* nastaven i atribut *fill* s příslušnou barvou. Atribut *stroke-width* je nastaven na šířku 0 a *stroke-width* (okraj tvaru) má nastavenou barvu na transparentní.

V případě, že šířka jedné ze stran rámečku je větší než zaoblení v příslušném rohu, je rámeček vykreslen následovně:

```
m Bx By
l Gx Gy
l Hx Hy
a r s 0 0 0 Ax Ay;
l Bx Oy
```

Rozdíl je především v tom, že zaoblení se vykresluje pouze na vnějším okraji a na vnitřním je ostrý roh.

Element *<path/>* (i s příslušnými atributy) je nakonec přidán do dokumentu. Poslední, co zbývá vykreslit jsou rovné části, na které je opět použit element *<path/>*.

7.4 Implementace 2D transformací

Transformace jsou celé implementované ve třídě *Transform* v balíčku *ms.layout*. V případě, že jsou v CSS pro některý z vykreslovaných elementů definované CSS transformace, je vytvořen objekt třídy *Transform* a následně je volána metoda *createTransform*. V této metodě je nejprve podle rozměrů a umístění elementu vygenerována transformace *translate*, která posune element do bodu počátku transformace (*transform-origin*). Následně jsou v cyklu projity všechny transformace, tak jak byly vloženy v CSS atributu a jsou pro ně vygenerovány příslušné příkazy do SVG transformací. Před vygenerováním každé transformace je ještě zkontrolován počet a typ vstupních parametrů. Kvůli přehlednosti kódu je generování transformačních funkcí zapouzdřeno zvlášť ve vlastních metodách. Jak již bylo popsáno v kapitole 6.4, 2D transformační funkce v CSS a v SVG jsou téměř shodné. CSS obsahuje navíc oproti SVG funkci *skew*, tedy zkosení ve směru obou os najednou podle zadaných úhlů. Tuto funkci nelze implementovat pomocí kombinace *skewX* a *skewY*, protože výsledná transformace se liší. Nicméně všechny transformace vychází z transformační matice, takže zmíněnou funkci lze nahradit následující transformační maticí [7]:

$$\begin{pmatrix} 1 & tan(\alpha) & 0 & 0 \\ tan(\beta) & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

kde úhel α je ve směru osy x a úhel β je ve směru osy y. Zmíněnou transformaci pak stačí vložit pomocí příkazu *matrix* do řetězce s transformačními funkcemi.

Po aplikaci všech transformací je ještě přidána transformační funkce *translate*, která element posune zpět na jeho původní umístění.

7.5 Implementace průhlednosti

Implementace renderování průhlednosti byla až na jednu výjimku bez problémů. Jediný problém byl v tom, že metoda *renderElementBackground* (metoda vykreslující pozadí aktuálního HTML elementu) může být volána ještě před začátkem vykreslování elementu. Aby nemusel být atribut *opacity* přidělen více elementům, je celý element i se svým pozadím (vygenerovaným v metodě *renderElementBackground*) obalen elementem *<g/>*. Tomuto elementu je zároveň kromě průhlednosti nastaven i atribut s transformacemi.

7.6 Implementace gradientů

Parser kaskádových stylů v nástroji CSSBox bohužel v aktuální verzi gradienty nepodporuje. Bylo by zbytečné implementovat parser pro gradienty zvlášť, protože jStyleParser zpracovává CSS styly pomocí frameworku ANTLR[3], což hodně ulehčuje zpracování vstupních stylů. Při implementaci a testování tedy byla využita simulace. Gradient, který byl nadefinovaný v CSS, bylo potřeba se stejnými hodnotami vytvořit i do tříd pro reprezentaci gradientů. Zobrazení testovaného gradientu je pak realizováno u příslušného elementu na základě nastavené CSS třídy (případně identifikátoru) u elementu nebo například podle nastavené konkrétní hodnoty barvy. Na reálných webových stránkách bohužel gradienty testovat nelze.

Gradienty jsou generovány v rámci metody *renderElementBackground*. Oba typy gradientů jsou obaleny v SVG tagu *<defs/>* a je u nich generován identifikátor, na který se později odkazuje. Aby byly identifikátory unikátní, je pro generování použit inkrementální čítač.

Nakonec nebyl implementovaný způsob ukládání a následného znovupoužití opakujících se gradientů za účelem optimalizace SVG kódu (tak jak byl popsaný v kapitole 6.7.4).

7.6.1 Popis pomocných tříd pro implementaci gradientů

Pro implementaci gradientů byly vytvořeny celkem tři třídy. Třída *GradientStop* reprezentuje hraniční zastávky s barvou, podle toho jak jsou definovány v CSS. Tato třída je společná pro oba typy gradientů a obsahuje atribut typu *Color*, který určuje barvu zastávky a dále celočíselný parametr *i*, který určuje v procentech umístění zastávky.

Třída *LinearGradient* slouží k reprezentaci a k výpočtům pro lineární gradienty. Mezi atributy je úhel, který určuje směr gradientu. Původně v této třídě byla i implementace převodu ostatních definic směru gradientu na úhel (například *left to right* na 90∘). Tato logika ale pravděpodobně bude později implementována již v rámci nástroje jStyleParser, takže byla nakonec odstraněna. Úhel se definuje pouze pomocí celočíselného parametru.

Dále jsou ve třídě čtyři číselné parametry (typu *double*), které určují souřadnice začátku a konce gradientu. Tyto souřadnice jsou dopočítávány podle úhlu gradientu a zároveň podle velikosti elementu. Poslední atribut má název *data* a je typu *ArrayList<GradientStop>*, tedy pole objektů typu *GradientStop*. Jednotlivé zastávky jsou v poli uložené v pořadí, ve kterém byly vyplněny v CSS.

Poslední třídou, která je využívána pro implementaci gradientů je třída *RadialGradient*. Ta slouží k renderování radiálních gradientů. Stejně jako třída pro lineární gradienty, má i třída pro radiální gradienty pole s jednotlivými zastávkami gradientu. Třída *RadialGradient*

[3]Framework ANTLR slouží ke generování lexikálního a syntaktického analyzátoru, které jsou následně využity k implementaci překladačů a interpretů.

dále obsahuje hodnoty gradientu, tak jak jsou reprezentované v CSS (délky poloos elipsy a souřadnice umístění) a SVG atributy *cx*, *cy*, *fx*, *fy*, *r* (čísla typu *double*, jejich význam je popsán v kapitole 6.7).

7.6.2 Generování lineárního gradientu

Jak bylo popsáno výše, generování lineárního gradientu je implementováno v metodě *simulateLinearGradient*. Jediným parametrem je objekt typu *ElementBox*, který obsahuje informace o rozměrech a umístění gradientu. Tato metoda, kromě elementu s gradientem vygeneruje i grafický element, který tvoří pozadí elementu. V případě, že by byly gradienty implementovány i v jStyleParseru a byly by napojeny přímo na CSSBox, bylo by potřeba toto pozadí případně zkombinovat s pozadím, které se vykresluje aktuálně. Mohlo by totiž dojít k překrytí gradientu pozadím (například obrázkem), které mělo být pod gradientem.

Při vytváření gradientu je nejprve vytvořen objekt příslušné třídy a do něj se vloží příslušné gradientové zastávky reprezentované objekty *GradientStop*. Následně se volá metoda *setAngleDeg*. Tato metoda má tři parametry. První je úhel, který určuje směr gradientu a další dva jsou šířka a výška elementu, pro který se gradient generuje.

V metodě *setAngleDeg* je nejprve přepočítán zadaný úhel gradientu na úhel, podle kterého bude vypočítána směrnice osy gradientu (viz 6.7). Z šířky a výšky se získá poměr stran *wRatio* a oba rozměry se nastaví na hodnotu 100, aby bylo možné počítat v procentech. Poté následuje ošetření dvou hodnot přepočítaného úhlu - 90◦ a 90◦, protože funkce tangens není pro tyto úhly definovaná. Zároveň jsou ošetřeny hodnoty 0◦ a 180◦, protože pro ty je zbytečné souřadnice počítat (nezáleží na poměru stran, gradient je vždy ve vodorovném směru). V původní implementaci, která nebrala v úvahu poměr šířky a výšky, byly ošetřeny i ostatní násobky úhlu 45◦. Nakonec ale zůstaly mimo výpočet pouze čtyři zmíněné hodnoty.

Pokud je přepočítaný úhel mimo násobky 90◦, vypočítá se tangens úhlu a z něj se pomocí poměru *wRatio* se získají směrnice osy gradientu a obou kolmic, které značí začátek a konec gradientu. Dosud vypočtené hodnoty jsou společné pro všechny směry gradientu. Dále je ale výpočet hodnot do rovnic směrnicových tvarů kolmic na osu gradientu rozdělen podle směru. V každém ze směrů totiž rovnice kolmic procházejí jinými body. Například pro úhel 45◦ jsou to body D a B.

Po získání směrnicových tvarů přímek už jsou pouze dopočítány průsečíky osy s oběma kolmicemi. Tyto průsečíky jsou výsledné souřadnice počátku a konce gradientu v procentuálním vyjádření.

Po získání souřadnic je v metodě *simulateLineatGradient* vytvořen element *<defs/>*, do něj je vložen příslušný SVG element pro gradient. Obsahem elementu *<linearGradient/>*, jsou ještě elementy *<gradientStop/>*, které jsou vkládány v cyklu, který prochází všechny zastávky definované v objektu třídy *LinearGradient*. Celý obalovací element je vložen do dokumentu a ještě je vygenerován obdélníkový element, kterému je přes identifikátor gradientu vloženo pozadí (atribut *fill*). Tato část je realizována stejně i u radiálního gradientu.

7.6.3 Generování radiálního gradientu

U radiálních gradientů je velké množství vstupních parametrů, které se mohou různě kombinovat. Navíc je možné parametry zadávat v procentech, v absolutních délkách nebo pomocí klíčových slov. V aktuální implementaci je renderování radiálních gradientů realizováno pomocí několika různých metod. Metody jsou rozděleny do dvou skupin podle toho, jestli generují kružnicový gradient nebo eliptický gradient.

Pro kružnicový gradient je jedna základní metoda *setCircleData*, která podle rozměrů elementu, definované velikosti gradientu a jeho umístění (všechny hodnoty v pixelech) dopočítá hodnotu průměru gradientu (atribut r). Ostatní metody slouží pro případy, kdy nejsou všechny hodnoty zadány v pixelech. V případě, že je některý ze vstupních parametrů zadán jinak (v procentech, nebo pomocí klíčového slova), jsou nejprve hodnoty přepočítány na vzdálenost v pixelech a až poté je použita metoda *setCircleData*. V případě, že je poloměr gradientu zadán pomocí klíčových slov (například *closest-corner*), jsou zvlášť dopočítány vzdálenosti do všech čtyř rohů a z nich je poté vybrána ta nejkratší (podobně pro *closest-side*). Výpočet parametrů eliptického gradientu probíhá podobně jako kružnicový a navíc se dopočítává i poměr velikostí obou poloos elipsy.

Po získání parametrů pro SVG gradient je nutné vygenerovat příslušně velký obdélník, který bude tvořit pozadí elementu. U kružnicového gradientu se generuje čtverec, jehož délka strany odpovídá většímu z rozměrů původního elementu. Výsledný čtverec je oříznut pomocí elementu *<clipPath/>* na příslušné rozměry. Totéž je prováděno u eliptického gradientu a rozměry obdélníku jsou určeny poměrem délek poloos elipsy. Vygenerování elementu s gradientem probíhá stejně jako u lineárních gradientů.

Kapitola 8

Testování

V této kapitole je popsáno testování celé aplikace, které je rozděleno na tři části. V první části je popsáno testování aplikace na jednoduchých grafických elementech. Tyto elementy byly většinou obyčejné obdélníky s různě definovanými vlastnostmi a vždy přesně cílené na otestování jedné funkce. Poté následuje popis testování renderování na reálných webových stánkách. Následně je zhodnocena časová náročnost běhu aplikace na reálných webových stránkách a je porovnána s vykreslováním rastrové verze v nástroji CSSBox. Na konci této kapitoly jsou zhodnoceny výsledky testování naimplementované funkčnosti.

8.1 Testy na jednotlivých grafických elementech

Testování pomocí základních HTML elementů s jednoduchými styly CSS bylo důležité již při vývoji. Díky tomu, že dnešní internetové prohlížeče mají zabudovanou podporu jazyka SVG, bylo možné výstup přímo testovat oproti zadání, tedy testovacímu HTML dokumentu. Pro testování při vývoji byl vždy používán prohlížeč Chrome.

Po dokončení určité části funkčnosti byla vždy tato část testována a porovnána se zobrazením HTML verze i v prohlížeči Mozilla Firefox. SVG výstup byl pak kromě prohlížeče Chrome testován i v open source vektorovém editoru Inkscape [1].

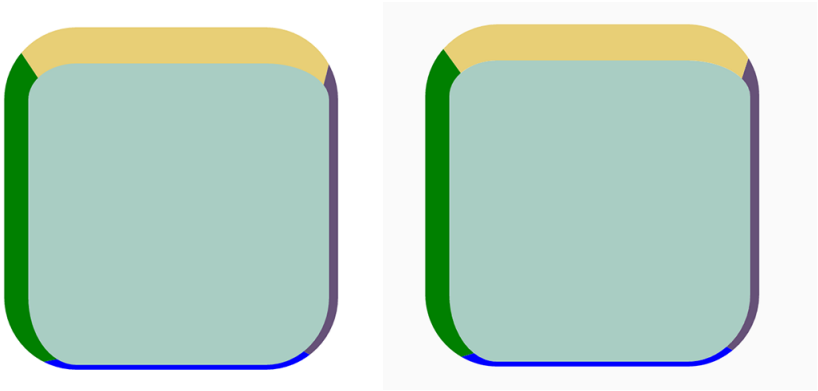
SVG výstup naimplementované funkčnosti gradientů, průhlednosti a transformací se v testovaných případech při vykreslení shoduje s vykreslením HTML verze v obou prohlížečích. SVG výstup zaoblených rámečků se trochu liší od zobrazení HTML dokumentu v prohlížeči.

Zobrazení zaoblených rámečků v SVG se od zobrazení HTML v prohlížeči Chrome liší umístěním hraniční přímky přechodu mezi barvami. Rozdíl lze vidět na obrázku 8.1.

Při implementaci zaoblených rámečků byla snaha se co nejvíce přiblížit grafickému výstupu prohlížeče Chrome. Bohužel se nakonec nepodařilo úplně napodobit renderování ve zmiňovaném prohlížeči, takže nakonec bylo implementováno řešení popsané v kapitole 6.3.3. Řešení popsané a implementované v rámci této práce samozřejmě odpovídá specifikacím pro CSS a od prohlížeče Chrome se pouze liší volitelné umístění přímky, která dělí roh rámečku.

Při testování navíc vznikl u transformací rozdíl mezi zobrazením transformace HTML dokumentu a vygenerovaného SVG dokumentu. Jednalo se konkrétně o posun textu ve vertikálním směru uvnitř transformovaného elementu. Toto ovšem bylo způsobeno tím, že

[1] Inkscape je multiplatformní, open sourcový vektorový editor, který nabízí velké množství funkcí. V rámci tohoto projektu byl využit pouze pro zobrazování SVG. Editor je volně dostupný na `https://inkscape.org/`



Obrázek 8.1: Rozdíl mezi zaoblenými rámečky v HTML dokumentu s CSS styly (vlevo) a SVG verzí vygenerovanou pomocí tohoto nástroje (vpravo)

prohlížeče (a stejně tak i CSSBox) mají různě nastavenou implicitní hodnotu CSS atributu *line-height*, tedy výšku řádku. Při specifikování této hodnoty v CSS byl problém odstraněn.

8.2 Testy na reálných webových stránkách

Testování nové implementovaného generování vektorového výstupu bylo prováděno i na reálných webových stránkách. Mezi tyto stránky patří například `www.vutbr.cz`. `www.seznam.cz`, `www.idnes.cz`, a další. Pro ověření fungování transformací, které na webových stránkách nebývají tak často používány, byl nástroj otestován na tutoriálových stránkách W3C, které ukazují použití transformací. Testování bylo prováděno experimentálně a byl pouze vizuálně porovnáván grafický výstup mezi zobrazením vstupního HTML dokumentu a vyrenderovaným SVG obrázkem.

Při testování bylo odhaleno několik dalších chyb a rozdílů vyrenderovaného SVG výstupu oproti HTML/CSS verzi. Například špatné vkládání pozadí pomocí obrázku, nebo příliš tenké vykreslení zaobleného rámečku, při nastavené šířce rámečku na $1px$ (kvůli tomu občas rámeček nebyl téměř vidět). Tyto chyby byly opraveny. Další část chyb a rozdílů vznikala z toho důvodu, že CSSBox nepodporuje některé CSS vlastnosti (například textové ikonky font-awesome [2]). Další rozdíly byly způsobeny technologickými překážkami, například flashové animace a reklamy.

8.3 Testy rychlosti běhu aplikace

Při testování rychlosti běhu nově implementovaného nástroje pro generování SVG výstupu nebyly zjištěny žádné výrazné odchylky od generování rastrového výstupu v demo aplikaci nástroje CSSBox. Měření bylo prováděno pomocí metody *System.nanoTime()* v jazyce Java a byl měřen celý běh zaváděcí statické metody v demo aplikaci *ImageRenderer*.

Pro menší testovací HTML dokumenty (v tomto případě 6 čtverců s rámečky a aplikovanými transfromacemi) trvalo zpracování kolem jedné sekundy. Průměrná doba z deseti měření u generování rastrového výstupu v CSSBoxu byla 1 045ms. Při zpracování stejného dokumentu pomocí nově implementovaného generování SVG výstupu byla průměrná hod-

[2] http://fontawesome.io/

nota deseti měření 1 009ms. Rozptyl od průměru u všech naměřených hodnot nebyl větší než 50ms. To bylo i díky tomu, že tyto dokumenty byly načítány přímo z disku počítače.

Při zpracovávání rozsáhlejších reálných webových stránek trval běh aplikace 3 a více sekund v závislosti na rozsahu vykreslované stránky. Konkrétní měření bylo prováděno na webové stránce `www.seznam.cz`. Opět bylo provedeno deset měření v obou testovaných aplikacích. V generátoru rastrového výstupu byl průměr naměřených hodnot 10 834ms. Při testování v generátoru SVG výstupu byla průměrná doba 11 517ms. Z výsledných hodnot lze vyčíst, že u větších webových stránek se doba zpracování liší přibližně o 680ms, což je nárůst o cca 6,3% při generování vektorového výstupu. Aby bylo jisté, že nedošlo k chybě při měření, bylo provedeno totéž měření ještě jednou, s velmi podobnými výsledky (oba průměrné časy byly o cca 150ms kratší a zpracování vektorového výstupu bylo o cca 6,1% pomalejší).

Další měření bylo provedeno na hodně jednoduché webové stránce: `www.google.cz`. Zde vyšly průměrné hodnoty měření téměř shodně: 3 198ms pro vektorový výstup a 3 077ms pro rastrový výstup. Zde je generování SVG výstupu pomalejší o cca 4%.

8.4 Zhodnocení testování

Konečné testování na základních elementech neodhalilo téměř žádné chyby. To je především proto, že toto testování bylo využíváno neustále během implementace a většina chyb byla odhalena již při implementaci.

Při testování na reálných webových stránkách bylo odhaleno několik chyb. Některé z nich se vyskytovaly v části renderování vektorového výstupu, ale některé byly i v jádře nástroje CSSBox. Většina nalezených chyb byla opravena.

Z popsaných testů rychlosti zpracování vyplývá, že nově implementované řešení renderování vektorového výstupu je řádově o jednotky procent pomalejší než rastrové. Naimplementované renderování SVG výstupu má ale velké možnosti, co se týče optimalizace generování SVG kódu. Vygenerovaný SVG kód není v rámci této práce prakticky téměř vůbec optimalizován.

Kapitola 9

Závěr

V rámci této práce bylo navrženo a implementováno rozšíření pro knihovnu CSSBox, které generuje vektorový výstup zpracovávaných HTML dokumentů ve formátu SVG. SVG dokumenty je možné stylovat pomocí kaskádových stylů ve verzi CSS2. Pro HTML dokumenty je aktuálně podporován jazyk CSS ve verzi CSS3. Hlavním cílem této práce bylo navrhnout řešení implementace generování SVG kódu pro vybrané CSS3 vlastnosti a tento návrh implementovat. Po dohodě s vedoucím práce bylo rozhodnuto, že budou implementovány následující CSS3 vlastnosti: Zaoblené rámečky, transformace, průhlednost a případně i gradienty, které ovšem zatím nejsou zpracovávány parserem CSS vlastností v knihovně CSSBox a jejich použití bude třeba simulovat během generování SVG kódu.

Před začátkem implementace bylo nutné si zvolit knihovnu v jazyce Java, která bude podporovat práci s SVG. V rámci této práce byly prozkoumány celkem tři knihovny. Ze zkoumaných knihoven byla nakonec vybrána knihovna Apache Batik, protože je nejrobustnější a zároveň jednoduše rozšiřitelná. V případě, že by později bylo potřeba renderování co nejvíce zrychlit, bylo by možné přepsat renderování pomocí některé z "lehčích"knihoven, například jFreeSVG.

Až na výjimky byly všechny vybrané vlastnosti implementovány podle specifikací standartu CSS3. Průhlednost a gradienty jsou implementovány kompletně. 2D transformace jsou také plně funkční. 3D transformace jazyk SVG bohužel nepodporuje, takže nebylo možné jejich generování implementovat. Vzhledem k tomu, že většina dnešních internetových prohlížečů podporuje zobrazování SVG souborů a podporuje v nich i CSS3 vlastnosti, bylo by možné tohoto faktu využít a 3D transformace pouze přenést do SVG. Toto řešení by samozřejmě nefungovalo pro zobrazování SVG dokumentu v grafických editorech (Adobe Photoshop, Inkscape, apod.) a proto nebylo nakonec implementováno. Zaoblené rámečky fungují také správně podle specifikací, ale nepodařilo se naimplementovat styly rámečku (atribut *border-style*). Rámeček tedy může být jedině plný (*solid*).

Naimplementovaná funkčnost byla nakonec otestována na jednoduchých elementech (pro otestování konkrétních CSS vlastností) a poté i na reálných webových stránkách. Při závěrečném testování bylo odhaleno několik chyb, které byly následně opraveny.

Pokračováním této práce by mohla být například optimalizace vygenerovaného SVG kódu. Díky tomu by bylo možné výrazně snížit velikost výsledného SVG dokumentu. V případě, že bude do knihovny CSSBox doimplementováno zpracování gradientů, bude potřeba zde implementované řešení napojit. Další možností rozšíření by mohla být implementace generování CSS3 animací v SVG.

Literatura

[1] J. David Eisenberg, Amelia Bellamy-Royds: *SVG Essentials, 2nd edition.* O'Reilly Media, 2014, iSBN 978-1-4493-7435-8.

[2] McFarland David Sawyer: *CSS3: the missing manual, 3rd ed.* Sebastopol, CA: O'Reilly, 2013, iSBN 978-1-4493-7435-8.

[3] WWW stránky: Apache(tm) Batik SVG Toolkit. [Online]; 2015 [cit. 2016-01-10] `https://xmlgraphics.apache.org/batik/`.

[4] WWW stránky: Cascading Style Sheets. [Online]; 2016 [cit. 2016-01-10] `http://www.w3.org/Style/CSS/`.

[5] WWW stránky: CSS Background and Borders. [Online]; 2016 [cit. 2016-04-05] `https://www.w3.org/TR/css3-background/`.

[6] WWW stránky: CSS Image Values and Replaced Content Module Level 3. [Online]; 2016 [cit. 2016-04-05] `https://www.w3.org/TR/css3-images/`.

[7] WWW stránky: CSS Transforms Module Level 1. [Online]; 2016 [cit. 2016-05-10] `https://www.w3.org/TR/css-transforms-1/`.

[8] WWW stránky: CssBox - Java HTML rendering engine. [Online]; 2015 [cit. 2016-01-10] `http://cssbox.sourceforge.net/`.

[9] WWW stránky: Filter Effects. [Online]; 2016 [cit. 2016-05-10] `https://www.w3.org/TR/SVG/filters.html`.

[10] WWW stránky: jFreeSVG. [Online]; 2015 [cit. 2016-01-10] `http://www.jfree.org/jfreesvg/`.

[11] WWW stránky: jFreeSVG vs Apache Batik. [Online]; 2014 [cit. 2016-01-10] `http://www.object-refinery.com/blog/blog-20140423.html`.

[12] WWW stránky: Manual - CSSBox - Java HTML rendering engine. [Online]; 2016 [cit. 2016-03-08] `http://cssbox.sourceforge.net/manual/`.

[13] WWW stránky: Scalable Vector Graphics Wikipedia the free wiki. [Online]; 2015 [cit. 2016-01-10] `https://en.wikipedia.org/wiki/Scalable_Vector_Graphics`.

[14] WWW stránky: Scalable Vector Graphics (SVG) 1.1 (Second Edition). [Online]; 2011 [cit. 2016-01-10] `http://www.w3.org/TR/SVG/`.

[15] WWW stránky: Scalable Vector Graphics, The Wold Wide Web Consortium. [Online]; 2010 [cit. 2016-01-10] `http://www.w3.org/Graphics/SVG/`.

[16] WWW stránky: SVG Basics - Creating Paths With Curve Commands. [Online]; 2016 [cit. 2016-05-02]
http://vanseodesign.com/web-design/svg-paths-curve-commands/.

[17] WWW stránky: SVG salamander - SVG parser and player. [Online]; 2015 [cit. 2016-01-10] https://svgsalamander.java.net/.

[18] WWW stránky: SVG transform vs CSS transform. [Online]; 2015 [cit. 2016-01-10] http://codepen.io/AmeliaBR/pen/aDhrs.

[19] WWW stránky: Vector and Raster graphics in printing. [Online]; 2014 [cit. 2016-01-10] http://olypress.com/vector-vs-raster-graphics-in-printing/.