



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DEPARTMENT OF INFORMATION SYSTEMS

**TRANSFORMACE WEBOVÝCH STRÁNEK DO VEKTO-
ROVÉ GRAFIKY**

WEB PAGE TRANSFORMATION TO VECTOR GRAPHICS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

HOANG DUONG NGUYEN

VEDOUcí PRÁCE

SUPERVISOR

Ing RADEK BURGET, Ph.D.

BRNO 2019

Zadání bakalářské práce



21464

Student: **Nguyen Hoang Duong**
Program: Informační technologie
Název: **Transformace webových stránek do vektorové grafiky**
Web Page Transformation to Vector Graphics
Kategorie: Web

Zadání:

1. Prostudujte grafické vektorové formáty SVG a PDF a knihovny pro jejich generování na platformě Java.
2. Seznamte se s existujícím projektem WebVector a souvisejícími projekty.
3. Navrhněte rozšíření projektu WebVector o podporu výstupu ve formátu PDF a o podporu pokročilých vlastností CSS, zejména geometrických transformací.
4. Implementujte navržená rozšíření.
5. Proveďte testování implementovaných rozšíření na reálných webových stránkách.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Projekt WebVector: <https://github.com/radkovo/WebVector>
- Šafář, M.. Transformace dokumentů HTML na vektorovou grafiku SVG. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- Červinka, Z.: Generování PDF dokumentů z webových stránek, bakalářská práce, Brno, FIT VUT v Brně, 2015
- Scalable Vector Graphics, The World Wide Web Consortium, <http://www.w3.org/Graphics/SVG/>

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz <http://www.fit.vutbr.cz/info/szz/>

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 16. října 2018

Abstrakt

Táto bakalárska práca sa venuje problematike vykreslenia webových stránok pomocou vektorovej grafiky. Cieľom tejto práce je navrhnuť a implementovať rozšírenie projektu WebVector tak, aby umožňoval vytvoriť výstup vo formáte PDF a vykresliť vybrané pokročilé vlastnosti CSS (CSS3). Vysvetlené sú pojmy súvisiace s vektorovou grafikou a jej formátmi. Práca následne popisuje štruktúru a funkcie knižnice CSSBox, s ktorou projekt WebVector pracuje, spolu s ďalšími súvisiacimi knižnicami. Ďalej je popísaných niekoľko CSS3 vlastností, ich návrh a implementácia na platforme Java.

Abstract

This bachelor thesis is devoted to the problem of rendering websites with vector graphics. The goal of this thesis is design and implement an extension of the WebVector project, which allows creating the output in PDF format and rendering some specific CSS3 properties. The terms related to vector graphics and its formats are explained. This work describes the structure and features of the CSSBox library, which project WebVector works with, and other related libraries. Then some of the CSS3 properties and also their design and implementation in platform Java are detailedly described.

Kľúčové slová

WebVector, CSSBox, JStyleParser, PDF, SVG, grafika, Apache PDFBOX, Java, CSS3

Keywords

WebVector, CSSBox, JStyleParser, PDF, SVG, graphics, Apache PDFBox, Java, CSS3

Citácia

NGUYEN, Hoang Duong. *Transformace webových stránek do vektorové grafiky*. Brno, 2019. Bakalárska práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing Radek Burget, Ph.D.

Transformace webových stránek do vektorové grafiky

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Radka Burgeta, Ph.D.. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Hoang Duong Nguyen
15. mája 2019

Podakovanie

Chcel by som poďakovať pánovi Ing. Radkovi Burgetovi PhD. za odbornú pomoc, ochotu a užitočné poznámky pri tvorení tohoto projektu.

Obsah

1	Úvod	3
2	Počítačová grafika	4
2.1	Pojmy vektorová grafika a rastrová grafika	4
2.2	Výhody a nevýhody vektorovej grafiky	5
2.3	Formát SVG	5
2.4	Formát PDF	6
3	CSSBox a WebVector	10
3.1	Funkcionalita nástroja CSSBox	10
3.2	WebVector	11
3.3	jStyleParser	12
3.4	CSSBoxSvg	12
3.5	CSSBoxPdf	13
4	Kaskádové štýly	14
4.1	Syntax CSS	14
4.2	Transform	15
4.3	Gradientsy	16
4.4	Border-radius	18
4.5	Filter	19
5	Java knižnice pre tvorbu PDF	20
5.1	Apache PDFBox	20
5.2	iText	20
5.3	PDF Clown	21
6	Návrh implementácie CSS3 vlastností pre výstup vo formáte PDF	22
6.1	Návrh renderovania 2D transform vlastností	22
6.2	Návrh renderovania CSS3 gradientov	24
6.3	Návrh renderovania border-radius	28
6.4	Návrh renderovania filter vlastností	29
7	Implementácia uvedených CSS3 vlastností	31
7.1	Štruktúra rozšírenia projektu CSSBoxPdf	31
7.2	Implementáci dvojdimenzionálnych transformačných vlastností	31
7.3	Implementácia lineárneho a radiálneho gradientu	32
7.4	Implementácia filtrových vlastností	34

7.5	Implementácia vlastnosti border-radius	35
7.6	Rozšírenie implemetácie	36
8	Testovanie	38
8.1	Testovanie na vytvorených HTML súboroch	38
8.2	Zhodnotenie testovania a možnosť rozšírenia práce	39
9	Záver	40
	Literatúra	41
A	Obsah CD	43

Kapitola 1

Úvod

Webové stránky reprezentujú zdroje informácií v elektronickej podobe a už od doby vzniku patria medzi najdôležitejšie prvky internetu. V súčasnej dobe neustále rastie počet webových stránok a spolu s tým vzniká aj potreba premeniť tieto webové stránky do takého formátu, ku ktorému sa dá jednoduchšie prístupíť (nie je potrebný prístup k internetu) a ktorý poskytuje možnosť vytlačiť a uložiť daný formát. Existuje mnoho formátov počítačovej grafiky, do ktorých sa dá premeniť webové stránky. Medzi najpopulárnejšie formáty reprezentujúce elektronické dokumenty patrí *Portable Document Format* (**PDF**), ktorý vďaka svojej grafickej integrite a zabezpečeniu sa v poslednej dobe stáva dominantným formátom pre uloženie, vytlačenie a zdieľanie elektronických dokumentov.

Hlavným cieľom nášho projektu je vykresliť webové stránky na PDF výstup spolu s niektorými pokročilými CSS3 vlastnosťami ako 2D transformácia, gradienty, filter a rámček s okrúhlymi rohmi.

Táto bakalárska práca je rozdelená do niekoľko kapitol a podkapitol. Na začiatku bude vysvetlených niekoľko pojmov týkajúcich sa problematiky našej práce. Budeme popisovať význam vektorovej grafiky a jej formátov, základné funkcionality nástroja CSSBox, jazyk CSS a jeho pokročilé vlastností a zároveň Java knižnice pre tvorbu PDF dokument. Ďalej sa budeme zaoberať návrhom realizácie vybraných CSS3 vlastností a ich implementáciami. Na konci sa budeme venovať testovaniu výsledného projektu a možnostiam rozšírenia projektu v budúcnosti.

Kapitola 2

Počítačová grafika

V súčasnej dobe je dvojdimenzionálna počítačová grafika obvykle rozdelená na rastrovú alebo vektorovú grafiku. Vďaka svojej flexibilita a všestranosti je vektorová grafika častejšie používaná oproti rastrovej. Táto kapitola si kladie za cieľ vysvetliť princíp reprezentácie obrázkov u týchto dvoch typov spolu s ich výhodami a nevýhodami. Ďalej sú tu popísané dva z najvýznamnejších formátov vektorovej grafiky.

2.1 Pojmy vektorová grafika a rastrová grafika

Vektorová grafika reprezentuje počítačové grafické obrázky vytvorené zo základných geometrických útvarov, napríklad priamky, úsečky, mnohoúhelníky (trojuholník, štvorec) atď. Tieto útvary sú definované matematickými funkciami a skladajú sa z jednej alebo viacerých ciest (angl. paths). Každá cesta obsahuje minimálne dva 2D body¹ - počiatočný a koncový bod, ktoré určujú jeho polohu na obrázku. Jednotlivá cesta okrem súradnice je definovaná aj farbou, priehľadnosťou, odtieňmi farby a ďalšími vlastnosťami[5]. Pomocou týchto ciest je možné vytvoriť akékoľvek tvary. Medzi najbežnejšie formáty vektorových obrázkov patria *Scalable Vector Graphics (SVG)* a *Portable Document Format (PDF)*. Detail o týchto formátoch si objasníme v podkapitole 2.3 a 2.4.

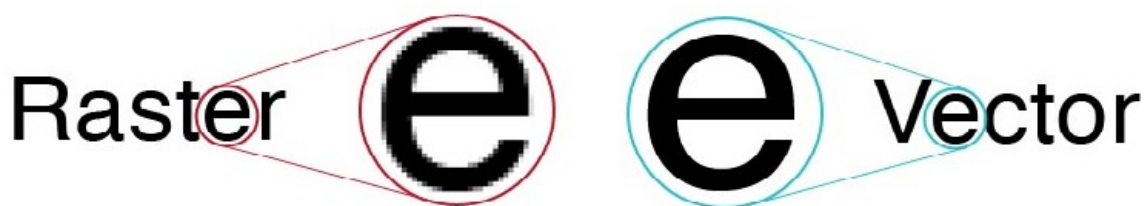
Rastrová grafika narozdiel od vektorovej vykresľuje obrázok ako súbor malých štvorčekov usporiadaných do tvaru mriežky. Takýto štvorček sa nazýva *pixel* a každý pixel je možno identifikovať podľa svojich súradníc, ktoré určujú, na ktorom riadku a stĺpci sa daný pixel nachádza. Každý štvorček taktiež obsahuje hodnoty farby a jej odtiene. Rozlíšenie rastrového obrázku je definovaný počtom pixelov nachádzajúcich sa v danom obrázku. To znamená, že čím je počet pixelov vyšší, tým je obrázok kvalitnejší. Z tejto vlastnosti vzniká hlavná nevýhoda rastrovej grafiky, a to je strata kvality pri zväčšovaní obrázku. Keď sa zväčšuje obrázok, zväčšuje sa aj rozmer jednotlivých pixelov, čo vedie k rozmazanosti obrázku. Aby nedošlo k tomuto problému, obrázok musí mať vyššie rozlíšenie (viac pixelov), čo vedie k zväčšeniu veľkosti obrázku. Takéto obrázky sú obvykle veľmi finančne nákladné.

V porovnaní s vektorovou grafikou je rastrová grafika z týchto dôvodov menej flexibilná a pomalšia pri zobrazení. V súčasnej dobe sa rastrová grafika používa najmä pri vytvorení fotografických obrázkov, skenovaní a vektorová sa používa pre tvorbu počítačových výkresov a animácií. Bežné rastrové formáty sú JPEG (JPG), PNG, GIF a pod.

¹dvojdimenzionálne body, ktoré sú definované x-ovou a y-ovou súradnicou

2.2 Výhody a nevýhody vektorovej grafiky

Na rozdiel od rastrovej grafiky je možné vektorový obrázok ľubovoľne zväčšovať alebo zmenšovať, pričom kvalita a veľkosť obrázku ostávajú nezmenené. Je to vďaka tomu, že polohy a rozmery jednotlivých útvarov vo vektorovom obrázku sú definované matematickými rovnicami. Napr. pri zmene rozmeru kruhu, ktorý je definovaný rovnicou $(x-h)^2 + (y-k)^2 = r^2$, kde h a k sú súradnice stredu kruhu a r je rádius, musí počítač iba zistiť novú hodnotu pre rádius a dosadiť to do vzorca[4]. Z toho dôvodu je vektorová grafika veľmi efektívna pre vytvorenie log a je menej náročná pre operačnú pamäť. Ďalšia výhoda je to, že vektorová grafika si nemusí zapamätáť informácie od všetkých pixelov vyskytujúcich sa na obrázku. Preto je veľkosť súborov tejto grafiky oveľa menšia ako rastrové súbory s rovnakou kvalitou.



Obr. 2.1: Kvalita rastrovej a vektorovej grafiky pri zväčšení obrázku [4].

Jednou z nevýhod vektorovej grafiky je nevhodnosť pre vytváranie fotorealistických obrázkov. Nedokáže dokonale vykresliť jemné odtiene farieb vo fotkách ako rastrová grafika. Z tohoto dôvodu väčšina vektorových obrázkov má kreslenú a nerealistickú podobu. Tvorba takýchto obrázkov je obvykle časovo náročná a žiada vysokovýkonný procesor pre načítavanie a zobrazenie na monitore. Počítač musí byť dostatočne silný, aby bolo možné vykonávať výpočty pre vykreslenie obrázku. Ďalšia nevýhoda je, že nie je možné premeniť súbory v iných formátoch do vektorových formátov bez použitia špecifických softvérov, aplikácií.


2.3 Formát SVG

Scalable Vector Graphics je vektorový formát, ktorý bol navrhovaný a vyvinutý od roku 1999 spoločnosťou *World Wide Web Consortium (W3C)*[13]. Reprezentuje dvojdimenzionálne vektorové grafické objekty. SVG súbory sú definované jazykom SVG, ktorý je založený na základe jazyka *eXtensible Markup Language (XML)*. Podobne ako XML je SVG značkovacím jazykom (angl. markup language²), tzn. že používa značky (tags) pre definovanie objektov v dokumente. Syntax tohto jazyka je veľmi jednoduchý, používa štandardné slová a preto je ľahko čitateľný ľudským okom. Výsledný SVG dokument je obyčajný textový súbor a jeho obsah sa dá upraviť aj pomocou bežných jednoduchých textových editorov napr. Notepad. Obsah SVG dokumentov sa dá zmeniť aj dynamicky(počas pustení) pomocou skriptovacieho jazyka *Javascript*³[13]. Príklad jazyka SVG môžeme vidieť na obrázku 2.2.

²https://techterms.com/definition/markup_language

³<https://www.javascript.com/>

SVG súradnicový systém je rovnakým systémom ako systém, ktorý používa výstupné zariadenia pre zobrazenie vektorovej grafiky. Tento systém sa nazýva *device space* (tento pojem si vysvetľujeme v odstavci [Dvojdimenziálny súradnicový systém](#) v podkapitole 2.4). Jednotky na súradnicových osiach sú merané v pixeloch[13]. Napr. ak sa objekt nachádza na pravej strane vo vzdialenosti 100 pixelov od počiatočného bodu (x=0, y=0) tak jeho súradnice budú mať tvar x=100, y=0.

SVG code	Illustration
<pre data-bbox="295 707 911 768"><rect x="0" y="0" width="200" height="150" fill="#FFFF00" stroke="blue" stroke-width="5" /></pre>	

Obr. 2.2: Príklad objektu popísaného jazykom SVG a jeho vykreslenia [13]

2.4 Formát PDF

Portable Document Format (PDF) je súborový formát založený spoločnosťou *Adobe*. Používa sa pre prezentovanie a výmenu dokumentov, pričom nezávisí na type operačnej pamäti, softvéru alebo hardvéru. To znamená, že tieto súbory nezáležia na programe a prostredí, kde boli vytvorené alebo zobrazené - vždy sa zobrazujú rovnako vo všetkých operačných pamätiach (MAC OS, Windows) a zariadeniach (mobil, počítač). PDF dokument môže obsahovať rôzne typy dát ako napr. text, obrázky, grafiky a elementy, ktoré sú plne funkčné iba v elektronickej reprezentácii, ako napr. odkazy, animácie, video atď. Obsahuje aj informácie o rozložení stránky, ktoré charakterizuje veľkosť a tvar jednotlivých stránok a lokalitu každého prvku, ktorý sa tam nachádza. Autor PDF dokumentov môže svoju prácu zašifrovať nastavením hesla, potom sa obsah dokumentu zobrazuje iba po zadaní tohoto hesla. Existujú dva typy hesla - *user password* a *owner password*. Použitím *owner password* pri otvorení súboru získava užívateľ plný prístup vrátane možnosti zmeniť heslo. Pri použití *user password* užívateľ môže vykonávať špecifické činnosti, ktoré sú obmedzené autorom práce a ktoré sú bez použitia hesla nedostupné[1].

PDF dokument má tvar textového súboru a skladá sa z viacerých operátorov a operandov jazyka PDF. Obsah dokumentu je zobrazený pomocou webových prehliadačov alebo pomocou aplikácie *Adobe Acrobat Reader*, ktorá podporuje aj upravovanie, komentovanie a podpísanie dokumentu. V súčasnej dobe je PDF dominantným formátom pre uloženie a zdieľanie elektronických dokumentov. Jazyk PDF je vybudovaný na základe jazyka *PostScript*⁴ a preto ich syntaxe sa veľmi podobajú. Narozdiel od *PostScript* je jazyk PDF neúplný. Nenachádzajú sa tam premenné, procedúry a riadiace štruktúry (if-else, for(), while()).

Štruktúra PDF súborov sa skladá zo štyroch prvkov:

⁴<https://www.adobe.com/products/postscript.html>

- **Hlavička** - je reprezentovaná prvým riadkom v PDF súbore. Hlavička identifikuje PDF verziu, ktorá sa používa pre tvorbu súboru. Príklad hlavičky s PDF verziou 1.7: `%PDF-1.7`
- **Telo** - skladá sa z objektov, ktoré spolu tvoria dokument ako napr. stránky, typ písma.
- **Tabuľka odkazov** - obsahuje odkazy na objekty v súbore a umožňuje rýchly prístup k určitým častiam dokumentu.
- **Pätička** - určuje lokalitu tabuľky odkazov a ďalšie špeciálne objekty [1].

Vzhľad každej PDF stránky je popísaný pomocou tzv. *content stream*, ktorý zahŕňa všetky grafické objekty, ktoré budú vykreslené na stránke. Tieto objekty sú definované operátormi a operandmi, ktoré sú tiež súčasťou content stream.

Grafika PDF

Z hľadiska počítačovej grafiky môže byť formát PDF vektorový alebo rastrový. Jeden PDF dokument môže obsahovať aj vektorovú grafiku aj rastrové obrázky. Grafika v PDF súboroch je tvorená so sekvenciou operátorov. Tieto operátory spolu vytvoria 6 hlavných skupín:

- **Graphics state operators** - operátory, ktoré manipulujú s dátovou štruktúrou tzv. *graphics state*. Táto štruktúra obsahuje parametre, ktoré reprezentujú aktuálny stav grafiky stránky v súbore, napr. aká je aktuálna farba krivky, pozadia, aktuálny stav textu (veľkosť písma, druh písma, aktuálna šírka riadku atď)[1].
- **Path constructions operators** - operátory, ktoré sa používa pre vytvorenie ciest (kriviek), pomocou ktorých sa vykresľuje rôzne útvary vo vektorovej grafike[1].
- **Path-painting operators** - vyfarbujú krivky alebo priestore okolo nich.
- **Other painting operators** - vyfarbujú grafické objekty ako obrázky, geometrické útvary.
- **Text operators** - tieto operátory slúžia pre vytvorenie textu, definujú veľkosť medzier medzi jednotlivými slovami a písmenami, veľkosť písma, typ písma a pod.
- **Marked-content operators** - neovplyvňujú pohľad stránky, spájajú špecifické informácie s objektami v content stream.

PDF grafické objekty sú definované týmito operátormi a delia sa na šesť typov:

- **Path object** - sú to rôzne útvary tvorené z úsečiek, geometrických útvarov a *Bézier* kriviek⁵. Každý *path object* je definovaný jedným alebo viacerými operátormi, ktoré špecifikujú, s akou farbou a hrúbkou budú okraje objektu vykreslené (stroke) alebo s akou farbou bude pozadie objektu naplnené (fill).
- **Text object** - skladajú sa z jedného alebo viacerých znakových reťazcov, ktoré identifikujú sekvenciu glyfov⁶.

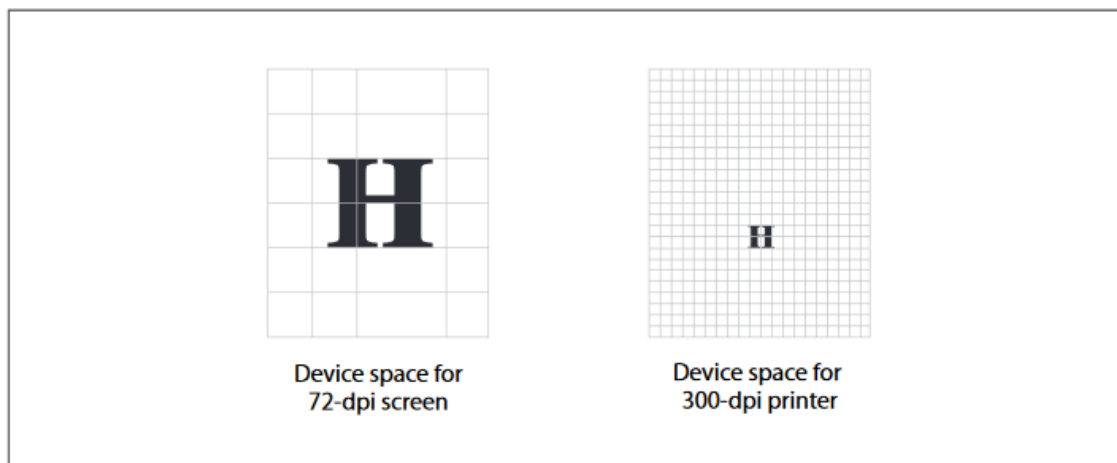
⁵tento pojem si podrobne vysvetľujeme pri návrhu implementácie border-radius

⁶glyf je grafická realizácia podoby grafému (písmena, číslice, znaku, ...)[1].

- **External object** (XObject) - sú to objekty, ktoré sú definované mimo content stream a každý z týchto objektov má definovaný vlastný názov s odkazom naňho. Existujú rôzne typy XObject ako *image XObject* (reprezentuje rastrové obrázky), *form XObject* (reprezentuje celý content stream, ktorý je spracovaný ako jeden grafický objekt[1]), *reference XObject*, *group XObject*, *PostScript XObject* atď.
- **Inline image object** - definujú obrázky, ktoré sa nachádzajú v rámci PDF dokumentu.
- **Shading object** - reprezentujú geometrické útvary, ktorých farba je ľubovoľná funkcia určujúca pozície v rámci útvaru[1]. *Shading* môže byť definovaný ako farba pre vyplňovanie grafických objektov. V našom projekte je Shading používaný pre tvorbu lineárneho a radiálneho gradientu.

Dvojdimenziálny súradnicový systém

Súradnicový systém v PDF určuje pozíciu, smer a veľkosť všetkých objektov, ktoré sa nachádzajú na stránkach dokumentu (text, obrázky atď.). Typ súradnicového systému, ktorý používa výstupné zariadenia ako tlačiareň a monitor pre zobrazenie PDF stránok, sa nazýva *device space*. Tento systém je závislý na vlastnostiach zariadenia, v dôsledku čoho pri zobrazení na výstupe sa objekty môžu nachádzať v inej polohe ako je v zdrojovom dokumente. Táto vlastnosť je načrtnutá na obrázku 2.3. Device space sa odlišuje od typu používaného na PDF stránkach tým, že jeho počiatočný bod súradnice($x=0$, $y=0$) je umiestnený v ľavom hornom rohu stránky. X-ová súradnica sa pohybuje smerom doľava a y-ová smerom dole pri zvýšení ich hodnoty.



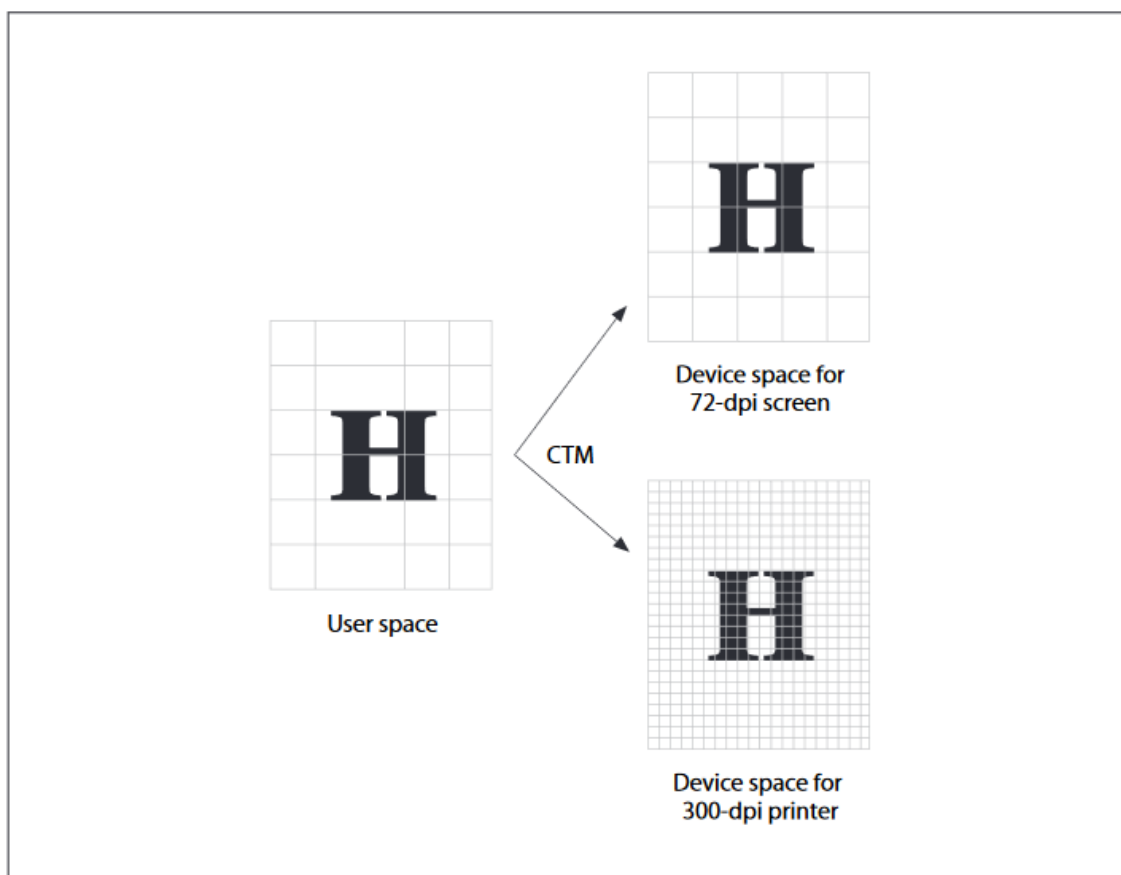
Obr. 2.3: Zobrazenie objektov pri rôznych zariadeniach pomocou device space [1].

Aby nedošlo k nezhodnému zobrazeniu PDF dokumentov na rôznych výstupných zariadeniach, PDF používa súradnicový systém tzv. *user space*, ktorý je nezávislý na type zariadenia, tzn. že objekty dokumentu sa vždy rovnako zobrazujú vo všetkých výstupných zariadeniach (veľkosti a pozície objektov sa nemenia). Počiatočný súradnicový bod v user space sa nachádza v ľavom dolnom rohu stránky. Hranicu PDF stránok určuje obdĺžnikový útvar s názvom *MediaBox*. Pri zväčšení hodnoty x-ovej súradnice sa objekt posunie smerom doprava a pri zväčšení y-ovej sa objekt posunie smerom hore. User space implicitná jednotka

reprezentuje veľkosť $1/72$ inch (pridaním 72 do súradníc znamená posunutie objektu o 1 inch (2.54cm) v skutočnosti). Užívateľ si môže hodnotu tejto jednotky nastaviť ľubovoľne podľa potreby v položke *UserUnit*.

Z hľadiska matematiky môžeme definovať user space ako nekonečnú rovinou, z ktorej iba malá časť je zobrazená na výstupnom zariadení[1]. Táto časť sa nazýva *CropBox* a je definovaná obdĺžnikovým útvarom. Všetky grafické objekty, ktoré sa nachádzajú v tejto oblasti, budú vykreslené na stránke PDF.

Pred zobrazením stránok na výstupnom zariadení dochádza k transformácii user space na device space pomocou matice *current transformation matrix* (CTM)⁷. PDF aplikácia na základe vlastností výstupných zariadení (rozlíšenie, typ súradnicového systému) upravuje túto maticu tak, aby sa zachová PDF vlastnosť *nezávislosť* na zariadení. Upravením hodnoty CTM môžu byť objekty posunuté (translation), otočené (rotate), zmenené mierkou (scale). Tieto transformácie sa podobajú transformačným vlastnostiam CSS a budú rozobrané v kapitole 4.



Obr. 2.4: PDF user space a transformácia do device space [1]

⁷https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf

Kapitola 3

CSSBox a WebVector

CSSBox je (X)HTML/CSS renderovací nástroj napísaný čisto v programovacom jazyku Java[7]. Renderovanie (anglicky rendering) v tomto projekte znamená vytvorenie vizualizácie HTML/CSS alebo XML dokumentov. Táto kapitola sa zameriava na objasnenie funkcionality nástroja CSSBox, jeho štruktúry a postupu renderovania dokumentu. Ďalej je popísaná aplikácia *WebVector*, ktorá je užívateľským rozhraním nástroja CSSBox. Na konci sa venujeme významu súvisiacich projektov.

3.1 Funkcionalita nástroja CSSBox

Hlavným účelom nástroja CSSBox je poskytnúť všetky informácie o renderovaných stránkach vhodných pre ďalšie spracovanie[7]. Vstupom tohoto nástroja je DOM dokument, ktorý je charakterizovaný stromovou štruktúrou a hlavným koreňovým uzlom tohoto stromu je uzol s názvom *Document*. CSSBox vytvorí tento DOM dokument¹ rozoberaním (angl. parse) vstupného XML alebo HTML súboru pomocou triedy *DOMSource*, ktorá je implementovaná na základe nástrojov *NekoHTML* a *Xerces 2* a je súčasťou CSSBox. V priebehu analýzy budú zakomponované do dokumentu aj CSS vlastnosti, ktoré sú definované v CSS súboroch, ktoré sú uvedené v HTML súbore.

Vstupné HTML a XML súbory sú zadané v tvare URL reťazca. Výstupom je objektovo orientovaný model (model založený na objektoch) stránkového usporiadania (page layout), ktorý môže byť v podobe rastrovej (PNG formát) alebo vektorovej grafiky (SVG alebo PDF formát). Výstup vo vektorovej grafike je možné ďalej upraviť a spracovať v špecifických editoroch. Použitím podprojektu *SwingBox* sa môže CSSBox chovať ako interaktívny webový komponent prehliadača v niektorých z Java Swing aplikácií[7]. *SwingBox* reprezentuje komponent, ktorý je navrhovaný ako *JEditorPane*² s úmyslom efektívnejšieho renderovania. Služí pre zobrazenie (X)HTML súborov spolu so CSS vlastnosťami na obrazovke.

Postup spracovania vstupného dokumentu

CSSBox očakáva vstup v tvare DOM dokumentu, ktorý bol popísaný v predchádzajúcom odseku. Vkladanie kaskádových štýlov (CSS) do DOM dokumentu zaisťuje trieda *DOMAnalyzer*. Pomocou tejto triedy sú uložené do dokumentu všetky štýly, ktoré sú buď definované priamo v HTML súbore, alebo sú definované v externom CSS súbore, na ktorý odkazuje

¹<https://www.w3.org/TR/WD-DOM/introduction.html>

²<https://docs.oracle.com/javase/7/docs/api/javafx/swing/JEditorPane.html>

HTML súbor využitím elementu `<link>` nachádzajúceho sa v HTML hlavičke. DOMAnalyzer počas spracovania kaskádových štýlov používa typ *screen* ako implicitný typ media a používa implicitné parametre obrazovky (šírka, výška, rozlíšenie, a pod.) pre nastavenie tzv. *media queries*³. Podľa media queries sa vykresľujú hranice výstupu a jeho elementov spolu s ich CSS vlastnosťami tak, aby nedochádzalo k chaotickému usporiadaniu elementov na *viewport*⁴ pri zobrazení webových stránok v rôznych typoch zariadení (mobil, počítač). Existujú štyri hodnoty pre typ media: *screen*, *print*, *speech*, *all*, pričom *screen* reprezentuje počítačové, mobilné, tabletové obrazovky. Parametre obrazovky a typ media, ktoré používa DOMAnalyzer, sa dajú nastaviť pomocou triedy *MediaSpec*, ktorá je súčasťou podprojektu *jStyleParser* (podkapitola 3.3). Po inicializácii triedy DOMAnalyzer a vložení štýlov vzniká výsledný DOM dokument, ktorý obsahuje HTML elementy a ich príslušné CSS vlastnosti spolu s ich media hodnotami.

Celý proces vykreslenia grafického rozvrhnutia (angl. layout) výstupu zaisťuje trieda *BrowserCanvas*, ktorá môže byť používaná pre zobrazenia výstupného dokumentu (je odvodená z triedy *javax.swing.JPanel*) alebo pre získaní vytvoreného layout modelu[7]. *BrowserCanvas* pomocou svojich funkcií postupne vytvorí layout dokumentu. Výsledkom tohoto procesu je stromová štruktúra objektov, ktorá reprezentuje usporiadanie HTML elementov v dokumente. Tieto objekty sú inštanciami mnohých tried rozšírených z abstraktnej triedy *Box*. Každý objekt reprezentuje jeden obdĺžnikový priestor na výslednej stránke, ktorý odpovedá príslušne renderovaný HTML element. Hierarchia typov týchto objektov je dostupná na stránke <http://cssbox.sourceforge.net/manual/>. Koreň stromu objektov reprezentuje objekt *Viewport*, ktorý definuje viewport dokumentu a má vždy jedného potomka s názvom *rootbox*, ktorý reprezentuje koreňový element vstupného HTML dokumentu, ktorý je obvykle `<body>` element[7]. Tento strom potom využívajú tieto tri nasledujúce triedy pre vykreslenie výstupného dokumentu spolu s jeho grafikou do triedy *BrowserCanvas* v troch príslušných formátoch: trieda *GraphicRenderer* pre vykreslenie rastrového formátu PNG, trieda *SVGDOMRenderer* pre SVG vektorový formát a trieda *PDFRenderer* pre formát PDF.

Detaily o funkciách pre konfiguráciu triedy *BrowserCanvas* a pre získanie hodnoty z jednotlivých *Box* objektov sú popísané v manuáli *CSSBox*[7].

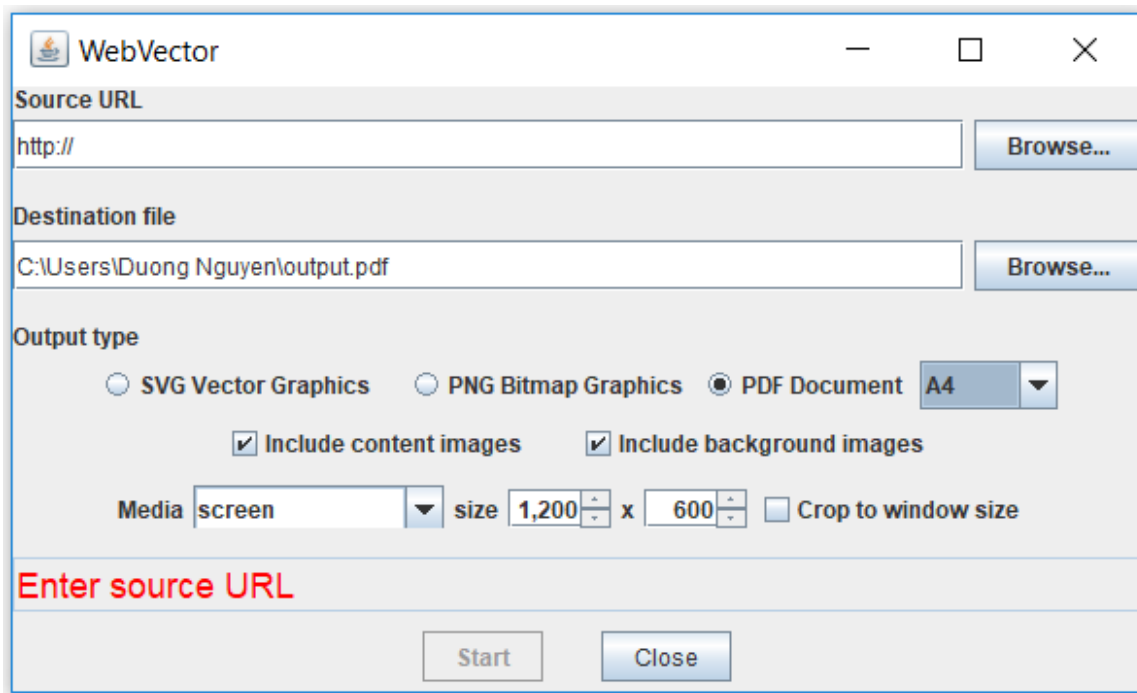
3.2 WebVector

WebVector je Java Swing aplikácia, ktorá pôvodne slúžila pre premenu HTML dokumentu do rastrového obrázku vo formáte PNG alebo do vektorového vo formáte SVG. V najnovšej verzii *WebVector* 3.4 pribúdala možnosť vytvoriť vektorový výstup aj vo formáte PDF. Rozhranie tejto aplikácie môžeme vidieť na obrázku 3.1.

Užívateľ tejto aplikácie ako vstup môže zadať URL webovej stránky alebo lokalitu HTML súboru, ktorý sa nachádza na jeho počítači. Koncovka výstupného súboru sa automaticky generuje podľa formátu grafiky, ktorý užívateľ vybral (.png, .svg, .pdf). Po zvolení PDF formátu je možné vybrať aj formát stránky dokumentu (od A0 do A9), pričom každý má definované svoje rozmery a po jeho zvolení budú tieto rozmery automaticky nastavené v dvoch *JSpinner* reprezentujúcich dĺžku a šírku stránky výsledného dokumentu. Tieto rozmery sa dajú nastaviť aj ručne. Užívateľ sa môže rozhodnúť, či má záujem o vykreslenie obrázkov na pozadí alebo všetkých obrázkov, ktoré sa vyskytujú v HTML súbore. Aplikácia

³https://www.w3schools.com/css/css_rwd_mediaqueries.asp

⁴definuje výrez webovej stránky, ktorý je zobrazený prehliadačom a je viditeľný užívateľom.



Obr. 3.1: WebVector verzia 3.4

podporuje dva typy medií z možných štyroch typov, a to sú *screen* a *print*. Pre spracovanie PNG výstupu WebVector používa demo aplikáciu *ImageRenderer* dostupnú v projekte CSSBox, pre SVG výstup je použitý projekt *CSSBoxSvg* (podkapitola 3.4) a pre PDF je použité *CSSBoxPdf* (podkapitola 3.5).

3.3 jStyleParser

Projekt jStyleParser slúži ako Java knižnica pre analýzu kaskádových štýlov (CSS alebo časť CSS3) [8] a priradenie týchto vlastností do HTML alebo XML elementov, do ktorých patria. Existujú tri možné spôsoby ako získať CSS súborov: z vzdialeného súboru (cez URL webovej stránky), z lokálneho súboru alebo z reťazcov typu *String* - sú to štýly, ktoré sú definované medzi prvkami `<style>`. V triede *CSSFactory* sú definované tri statické funkcie, pričom každá z týchto funkcií reprezentuje jeden z uvedených spôsobov. Výsledkom parsovania je objekt *StyleSheet*, ktorý obsahuje súbor CSS pravidiel a je vstupom pre proces DOM analýzy. Cieľom tohoto procesu je mapovanie do jednotlivých DOM elementov reprezentujúcich HTML/XML elementy ich príslušné CSS vlastnosti. Tieto vlastnosti sa ukladajú do štruktúry tzv. *NodeData*. *NodeData* poskytuje dve metódy *getProperty()* a *getValue()*, pomocou ktorých je možné zistiť hodnoty jednotlivých CSS vlastností a spolu s týmito hodnotami sa vykresľuje výsledný dokument.

3.4 CSSBoxSvg

CSSBoxSvg je súčasťou nástroja CSSBox, ktorý slúži pre generovanie výstupu v SVG formáte. Tento projekt je výsledkom diplomovej práce, ktorej autorom je pán Ing. Martin Šafár [6]. Jadro tohoto projektu je trieda *SVGDOMRenderer*, ktorá je vylepšená verzia

triedy SVGRenderer v CSSBox. Obidve triedy majú rovnaký účel - vytvoriť výstupný SVG súbor zložený z elementov SVG jazyka a obidve implementujú rozhranie BoxRenderer. SVGDOMRenderer postupne prevedie SVG elementy, ktoré sú vygenerované z elementov DOM dokumentu počas jeho analýzy, do tvaru objektov typu DOM Element⁵. Následne tieto objekty uloží do stromovej štruktúry a nakoniec prebieha ich prevod do tvaru reťazca reprezentujúceho SVG príkazov. Tento spôsob je oproti SVGRenderder elegantnejší, ktorý priamo generuje príkazy SVG v textovej podobe z objektov DOM dokumentu. Navyše SVGDOMRenderer dokáže renderovať niekoľko CSS3 vlastností - gradienty, transformácie, border-radius, tieňovanie, priehľadnosť.

3.5 CSSBoxPdf

Podobne ako CSSBoxSVG je CSSBoxPdf podprojektom nástroja CSSBox, ktorého hlavnou úlohou je generovať vektorový výstup vo formáte PDF. Proces renderovania výstupu zabezpečuje trieda PDFRenderer. Táto trieda vytvorí pre každý objekt vyskytujúci sa v DOM dokumente, ktorý bol generovaný nástrojom CSSBox z vstupného HTML/XML dokumentu, príslušnú inštanciu triedy *Node*, ktorá obsahuje typ tohoto objektu (element, text, obrázok) a navyše aj jeho rodič a jeho ekvivalentný uzol v štruktúre TREE. Ďalej sa tieto inštancie uložia do dvoch štruktúr - TREE a LIST, pričom štruktúra TREE reprezentuje stromové usporiadanie (rodič - potomkovia) týchto inštancií a štruktúra LIST reprezentuje ich poradie. PDFRenderer ďalej vytvorí tabuľky *breakTable* a *avoidTable*. Pomocou dát v týchto tabuľkách a pomocou konca stránok určeného rozmerom stránky dochádza k zalomeniu stránok. Dáta o zalomení stránok sa uložia do štruktúry TREE[9]. Nakoniec sa postupne vykresľujú všetky uzly uložené v štruktúre LIST do podoby PDF a realizuje sa stránkovanie pomocou štruktúry TREE. Pred vykreslením uzlov do PDF dokumentu je potrebné premeniť ich súradnice. To je z dôvodu, že CSSBox používa súradnicový systém device space a PDF používa user space (tieto systémy boli popísané v podkapitole 2.4). V rámci našej bakalárskej práce je potrebné doplniť do triedy PDFRenderer schopnosť vykresliť niektoré CSS3 vlastností. pričom vybrané tieto vlastnosti: transformácia, gradienty (lineárne, radiálne), niektoré filter funkcie (brightness, opacity, invert) a border-radius. Projekt CSSBoxPdf je prácou pána Ing. Zbynka Červinky, ktorú vytvoril v rámci svojej bakalárskej práce[9].

⁵https://www.w3schools.com/jsref/dom_obj_all.asp

Kapitola 4

Kaskádové štýly

Cascading Style Sheets CSS bol navrhnutý a publikovaný v roku 1996 spoločnosťou *World Wide Web Consortium (W3C)*. CSS je jednoduchý jazyk, ktorý popisuje vzhľad webových stránok definovaných značkovacím jazykom ako HTML, XHTML, XML atď. Určuje rôzne vlastnosti týkajúce sa elementov webovej stránky (písmo, pozadie) ako napr. typ a veľkosť písma, farba pozadia, umiestnenie a pod. V tejto práci sa budeme konkrétne zaoberať CSS pre jazyk HTML. CSS vlastnosti môžu byť definované priamo v HTML dokumente ako atribút elementu (`inline-style`), medzi značkami `<style>` `</style>` nachádzajúcimi sa v hlavičke dokumentu, alebo v externom súbore s koncovkou `.css`, na ktorý sa stránka odkazuje pomocou značky `<link>`. Jeden CSS súbor sa dá používať vo viacerých webových stránkach.

Existujú tri verzie CSS, pričom CSS3 je momentálne najnovšia verzia a pravdepodobne nevznikne verzia CSS4, keďže CSS3 rozčleňuje všetky štýly do jednotlivých modulov, ktoré sa dajú samostatne rozšíriť. Preto v budúcnosti pravdepodobne vychádzajú iba jeho aktualizované verzie. CSS3 poskytuje veľké množstvo nových vlastností, medzi ktoré patria transformačné vlastnosti, gradienty pre pozadie, okrúhle rohy u rámečkov, flex pozícia, animácie a mnoho ďalšie. V tejto kapitole si najprv vysvetlíme syntax jazyka CSS. Ďalej sú popísané niektoré CSS3 vlastnosti, ktoré sú implementované do projektu CSSBoxPdf tak, aby bolo možné tieto vlastnosti renderovať do formátu PDF, čo je aj účelom tejto práce.

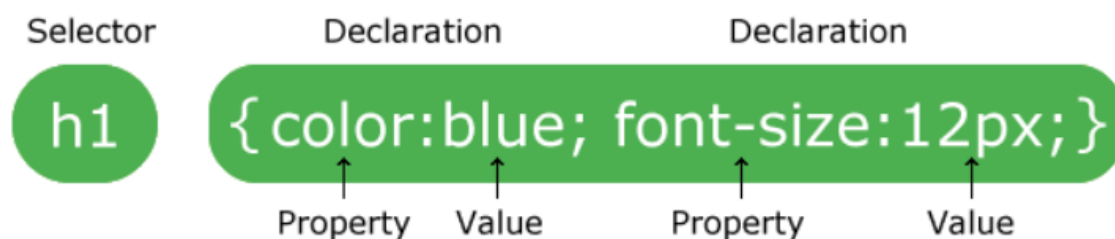
4.1 Syntax CSS

Kaskádové štýly sú definované pomocou pravidiel (rule), ktoré sa skladajú zo selektorov a skupiny (angl. block) deklarácií [12]. Príklad pravidla CSS a jeho komponenty môžeme vidieť na obrázku 4.1. Každá skupina deklarácie začína a končí so zloženými zátvorkami `{}` a obsahuje jednu alebo viac deklarácií, pričom každá deklarácia je zložená z jednej CSS vlastnosti spolu s jej hodnotou a končí bodkočiarkou. Jednotlivá deklarácia v rámci jednej skupiny musí byť oddelené bodkočiarkou, bez toho by nedošlo k jej prekladu a zobrazeniu. Selektor reprezentuje ukazovateľ na konkrétne elementy HTML dokumentu. Pomocou selektorov sa dá zistiť, ku ktorým HTML elementom patria dané CSS štýly. Existujú rôzne typy selektorov, pričom medzi základné typy patria tieto:

- **ID selector** - ukazuje na element pomocou hodnoty jeho atribútu *id*. Hodnota tohoto atribútu by mala byť jedinečná v rámci celého HTML súboru. ID selektor začína znakom hash (`#`) a za ktorým nasleduje hodnota atribútu *id* elementu, na ktorý chceme ukázať. Príkladom je `#headline1`.

- **Class selector** - Podobne ako ID selector, tento typ selektoru ukazuje na všetky elementy s definovanou hodnotou atribútu *class*. Vzhľadom na to, že hodnota atribútu *class* nemusí byť unikátna, tak ju môže mať viacero elementov. Tento typ selektoru sa skladá z bodky a hodnoty atribútu *class* elementu a môže vypadať nasledovne: *.block1*
- **Element selector** - použitím názvu typu elementov ukazuje na všetky elementy s rovnakým typom napr *h1*, *p*.

Tento syntax neplatí pre kaskádové štýly definované ako *inline-style*, ktoré sú priamo vložené do určitého HTML elementu ako hodnota jeho atribútu *style*.



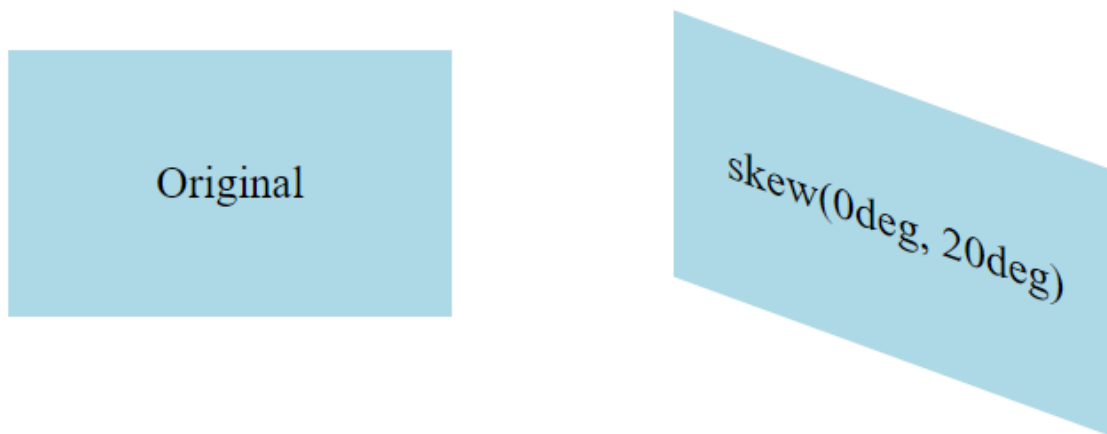
Obr. 4.1: Pravidlo CSS [12]

4.2 Transform

Pri použití transform vlastností dochádza k modifikovaniu súradníc objektov dokumentu, na ktorý sú tieto vlastnosti aplikované. Pomocou CSS3 transform je možné objekty otáčať, posúvať, zmeniť rozmery a pod. Existujú dvojdimenzionálne (2D) a trojdimenzionálne (3D) transform vlastnosti, pričom kvôli náročnosti pri realizovaní sa v tejto práci nebudeme venovať 3D vlastnostiam. 2D transform vlastnosti pracujú s horizontálnymi (x) a vertikálnymi (y) súradnicovými osami. Transform vlastnosti sa aplikujú okolo bodu, ktorý sa implicitne nachádza v centre daného elementu. Tento bod sa nazýva *transformation origin* a jeho pozíciu sa dá zmeniť pomocou CSS3 vlastnosti *transform-origin*. Medzi 2D transform hodnoty patria nasledujúce funkcie:

- **rotate(angle)** - otočí daný element podľa hodnoty parametru *angle*. Ak je táto hodnota kladná, element sa otočí v smere hodinových ručičiek. Ak je záporná, tak sa otočí v opačnom smere. Stred otáčania sa nachádza implicitne v centre elementu. Hodnotu *angle* je možné zapísať v tvare celého a desatinného čísla a jej jednotka môže byť jednou z týchto štyroch jednotiek: *deg* (stupne), *turn* (otočka), *rad* (radián) a *grad* (gradient). Jedna plná rotácia sa rovná 360 deg, 1 turn, 400 grad a približne 6.2832 rad.
- **scale(x, y)** - pomocou tejto funkcie je možné zväčšiť alebo zmenšiť rozmery elementu. Parameter *x* reprezentuje násobok šírky elementu a parameter *y* násobok dĺžky. Implicitná hodnota parametru *x* a *y* je 1, čo znamená, že rozmer sa nezmení. Pri zápornej alebo nulovej hodnote sa nevykresluje daný element, keďže jeho rozmery sa tým pádom rovnajú nule alebo sú menšie ako nula. CSS3 poskytuje navyše funkcie *scaleX()* a *scaleY()*, pomocou ktorých sa dá zmeniť buď iba šírka alebo iba výška elementu.

- **skew(*angle*x, *angle*y)** - táto funkcia sa používa pre zošikmenie elementu na oboch súradnicových osiach stránky x a y. Implicitné hodnoty parametrov sú 0. Vlastnosti týchto hodnôt sa podobajú vlastnostiam parametru u funkcii rotate(). Podobne ako u scale() sa tu dá samostatne zošikmiť na x-ovej osi pomocou funkcie skewX() a na y-ovej osi pomocou skewY(). Príklad tejto funkcie je uvedený na obrázku 4.2.

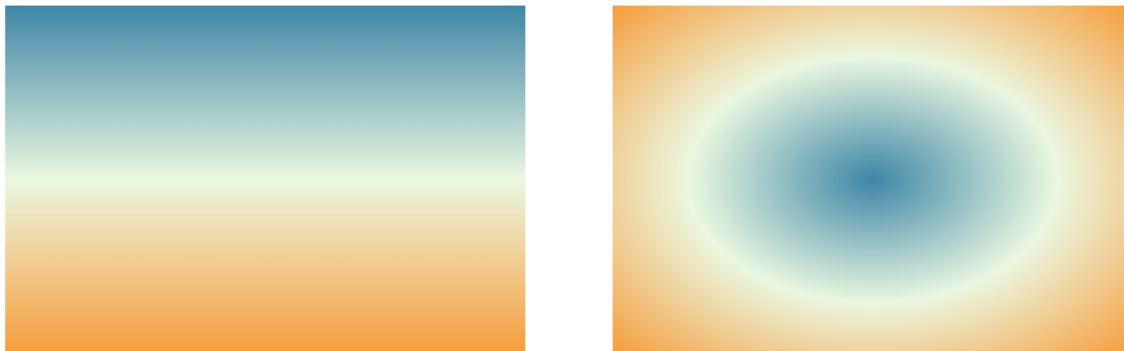


Obr. 4.2: Príklad CSS3 vlastnosti transform: skew(0deg, 20deg).

- **translate(x, y)** - je to funkcia, ktorá slúži pre posunutie elementov z ich pôvodnej pozície. Parameter x reprezentuje hodnotu, o ktorú sa element posunie v smere x-ovej osi dokumentu a y definuje hodnotu, o ktorú sa posunie element v smere y-ovej osi. Kladné hodnoty x,y posunú element doprava a hore a pri ich záporných hodnotách sa element posunie doľava a dole. Parametre x a y môžu byť zadané vo všetkých CSS metrických jednotkách napr. px, em, pt atď., z týchto jednotiek je px najpoužívanejšie. Tieto parametre je možné zadať ako percentá, pričom 100% u x sa rovná šírke elementu a 100% u y sa rovná jeho dĺžke.
- **matrix()** - všetky CSS3 2D transform vlastnosti sú definované maticou, pomocou ktorej sa určuje, ktoré vlastnosti sa majú aplikovať na daný element. Pomocou funkcie matrix() sa priamo nastavujú hodnoty tejto matice a je možné všetky 2D transform vlastnosti realizovať naraz. Táto funkcia má šesť parametrov, ktoré reprezentujú vyššie uvedené funkcie - matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY()) [12].

4.3 Gradienty

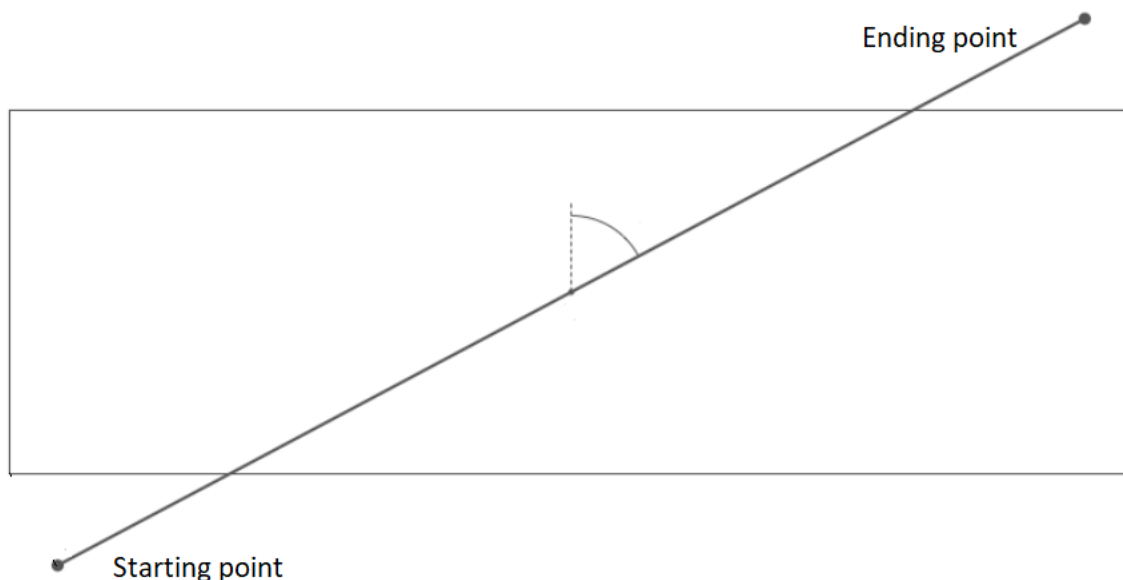
CSS gradienty sú funkcie, ktoré sa používajú ako deklarácie pre CSS vlastnosti *background-image* a *background*. Gradient je zobrazený ako prechod medzi dvoma alebo viacerými farbami (jeho vzhľad je zobrazený na obrázku 4.3). Existujú tri typy gradienty: *linear gradient*, *radial gradient* a *conic gradient*, pričom conic gradient momentálne nie je oficiálne publikovaný. Tento gradient zatiaľ podporujú iba dva významné prehliadače Chrome a Safari a preto sa budeme zaoberať iba o dvoch zvyšných typov gradientu.



Obr. 4.3: Príklad lineárneho (vľavo) a radiálneho gradientu (vpravo).

Linear gradient

Linear gradient je vykreslený ako prechod medzi farbami v smere rovnej priamky. Táto priamka sa nazýva *gradient line* a spolu s vertikálnou priamkou, ktorá začína od stredného bodu elementu smerom hore, zvierajú uhol gradientu (obrázok 4.4). Tento uhol je definovaný voliteľným prvým parametrom vo funkcií `linear-gradient()`, ktorý môže byť zadaný buď v tvare hodnoty spolu s uhlovou jednotkou (napr. `160deg`) alebo s jedným zo špecifických kľúčových slov napr. `to left`, `to right`, `to top right` a pod. Ak nie je tento parameter zadaný, uhol má implicitnú hodnotu `to bottom`, ktorá je ekvivalentná hodnote `180deg`. Na gradient line sa nachádzajú dva body, ktoré sa nazývajú *starting point* a *ending point*. Starting point určuje miesto, kde začína prvá farba a ending point určuje miesto, kde končí posledná farba. Syntax funkcie `linear-gradient()` a spôsob, ako získať súradnice týchto bodov spolu so spôsobom, ako vypočítať dĺžku gradient line a ako sa umiestňujú farby, budú detailne popísané v podkapitole 6.2.



Obr. 4.4: Náčrt komponentov lineárneho gradientu. ¹

Radial gradient

Radial gradient reprezentuje prechod medzi dvoma alebo viacerými farbami, ktorý začína od určitého bodu v elemente tzv. center point a rozširuje sa okolo tohoto bodu smerom von. Tvar radiálneho gradientu je reprezentovaný tzv. *ending shape*, ktoré môže mať kruhový alebo elipsový tvar. Ak funkcia *radial-gradient()* nemá nastavený parameter určujúci tvaru, tak pri elemente majúcom štvorcový útvar bude mať prechod kruhový tvar a pri elemente s obdĺžnikovým obvodom bude prechod vykreslený v tvare elipsy. Detail o syntaxe tejto funkcie bude vysvetlený v podkapitole 6.2.

4.4 Border-radius

Border-radius je CSS3 vlastnosť, pomocou ktorej sa vykresľujú okrúhle rohy rámčeka elementu. Tieto rohy na základe zadaného počtu rádiusu pre každý roh môžu byť kruhové (zadaný 1 rádius) alebo elipsovité (2 rádiusy). Vlastnosť border-radius reprezentuje tieto CSS3 vlastnosti: *border-top-left-radius*, *border-top-right-radius*, *border-bottom-left-radius* a *border-bottom-right-radius*. To znamená, že nastavenie border-radius spôsobí rovnaký efekt ako nastavenie všetkých vyššie uvedených vlastností spolu.

Syntax border-radius má nasledujúci tvar [12]:

border - radius : < lengthorpercentage > 1 - 4 [/ < lengthorpercentage > 1 - 4]

Ak vlastnosť border-radius neobsahuje znak /, tak rohy rámčeka budú mať kruhový tvar a ich rádiusy sú definované zadanými parametrami (1 až 4). Inak majú rohy tvar elipsy a prvé 1 až 4 hodnoty pred znakom / určujú horizontálne rádiusy (x-rádiusy) a 1 až 4 hodnoty na konci určujú vertikálne rádiusy (y-rádiusy) pre jednotlivé rohy. Parametre sú zadané buď v metrických hodnotách alebo v percentách, pričom x-rádius s hodnotou 100% sa rovná polovici šírky elementu a y-rádius s hodnotou 100% sa tiež rovná polovici dĺžky elementu.



Obr. 4.5: Príklad elementu s rámčekom obsahujúcim okrúhle rohy. ²

¹obrázok je dostupný na <https://medium.com/@patrickbrosset/do-you-really-understand-css-linear-gradients-631d9a895caf>

²obrázok je dostupný na https://www.w3schools.com/cssref/css3_pr_border-radius.asp

4.5 Filter

CSS3 filter vlastnosť umožňuje aplikovať grafické efekty ako napr. rozmazanosť, zmena jas, prehľadnosti atď. na daný element stránky. Filter vlastnosť sa väčšinou používa pre `` elementy, avšak môže byť aplikovaná aj pre pozadie a rámčeky. Hodnoty tejto vlastnosti sú definované filter funkciami. Každá funkcia definuje jeden špecifický grafický efekt. Do nášho projektu sú implementované tieto filter funkcie:

- **brightness(number or percentage)** - táto funkcia zmení jas obrazu. Parameter tejto funkcie môže byť zadaný v tvare čísla alebo v percentách. Hodnota parametru je priamoúmerná hodnote jas obrazu. Po zadaní hodnoty 0% (0) bude výsledný obrázok kompletne čierny a pri hodnote 100% (1) bude rovnaký ako pôvodný obrázok. Parameter s hodnotou nachádzajúcou sa v rozsahu 0% do 100% (0 do 1) spôsobuje zníženie jas obrázku a pri hodnotách vyšších ako 100% dochádza k zvýšeniu jas. Negatívne hodnoty nie sú podporované.
- **opacity(number or percentage)** - je to funkcia, ktorá ovplyvňuje priehľadnosť obrazu. Táto funkcia ako svoj parameter akceptuje kladné číslo alebo percentá podobne ako u `brightness()`. Parameter s hodnotou 100% zanechá obraz v pôvodnom stave a parameter 0% spôsobí totálnu priehľadnosť obrazu, čo znamená, že výsledný obraz sa síce vykresluje, ale nebude viditeľný. Pri hodnotách vyšších ako 100% bude výsledný efekt rovnaký ako pri 100%.
- **invert(number or percentage)** - pomocou tejto funkcie sa vytvorí výsledný obrázok s obrátenou farbou oproti pôvodnej farbe (napr. čierna na bielu). Podobne ako u `opacity()` a `brightness()` môže byť parameter zadaný v čísle alebo percentách. Pri hodnote 0% bude pôvodný obrázok nezmenený, pri 100% bude kompletne obrátený a pri 50% bude celý pôvodný obrázok nahradený sivou farbou. Ak je parameter definovaný hodnotou väčšou ako 100%, tak výsledok bude rovnaký ako pri 100%. Hodnoty v rozsahu 0% do 100% sú priamoúmerné efektu (spôsob, ako vypočítať takéto obrátené farby bude uvedený neskôr v podkapitole 6.4).



Obr. 4.6: Prehľad zmien obrázkov po použití niektorých filter vlastností.³

³ Pre vytvorenie tohoto obrázku je použitý obrázok `pineapple.jpg`, ktorý je dostupný na stránke <https://www.w3schools.com/>

Kapitola 5

Java knižnice pre tvorbu PDF

V súčasnej dobe existuje niekoľko knižníc, ktoré sa používajú pre tvorbu a spracovanie PDF dokumentu na platforme Java. Medzi najvýznamnejšie knižnice patria PDFBox, iText a PDF Clown. Všetky tieto knižnice sú open-source¹ a každá z nich má svoje špeciality a silné stránky. Keďže cieľom tejto práce je doplniť rozšírenie do existujúceho projektu CSSBoxPdf, ktorý umožňuje renderovať CSS3 vlastnosti, tak je vhodné použiť knižnicu, ktorú tento projekt používa, a to je knižnica PDFBox publikovaná pod Apache licenciou. V tejto kapitole sa budeme zaoberať funkčnosťou knižnice PDFBox a jej významných súčastí. Ďalej sú stručne popísané dve jej alternatívy.

5.1 Apache PDFBox

Knižnica Apache PDFBox je open-source Java nástroj pre prácu s PDF dokumentom. Táto knižnica poskytuje mnoho tried, ktoré sa používajú na vytvorenie dokumentu, manipulovanie² s dátami už existujúceho dokumentu a hlavne čerpanie (angl. extract) dát z dokumentu. PDFBox taktiež obsahuje aj nástroje pre prácu s príkazovým riadkom [2]. Medzi najdôležitejšie triedy, ktoré poskytujú funkcie pre vytvorenie PDF dokumentu patria: PDDocument - je to in-memory reprezentácia PDF dokumentu (priamo v operačnej pamäti), PDPage - reprezentuje stránku dokumentu, PDPageContentStream - kľúčová trieda, pomocou ktorej sa vykreslia všetky objekty, ktoré sa nachádzajú na stránkach dokumentu, triedy z balíka org.apache.pdfbox.pdmodel.graphics - tieto triedy obsahujú funkcie pre prácu s grafikou objektu, výstupy týchto funkcií sú potom predávané triede PDPageContentStream pre vykreslenie do dokumentu. Všetky tieto uvedené triedy sú súčasťou balíka org.apache.pdfbox.pdmodel, ktorý je kostrou knižnice PDFBox.

5.2 iText

iText je knižnica, ktorá pracuje na platforme Java a .NET a podobne ako PDFBox umožňuje vytvorenie a manipuláciu s PDF dokumentom [10]. V porovnaní s PDFBox je iText pokročilejší a populárnejší. Poskytuje funkcie, ktoré uľahčujú proces vykreslenia PDF dokumentu. Napr. iText poskytuje triedu s funkciami pre vykreslenie tabuľky, pričom u PDFBox musíme tabuľku vykresliť manuálne po jednotlivých čiarach. Avšak iText je bezplatný iba

¹bezplatne dostupné všetkým užívateľom.

²rozdeliť dokument na menšie dokumenty alebo zlúčiť viaceré dokumenty do jedného.

pod licenciou Affero General Public License (AGPL)³. To znamená, že ak nechceme platiť za iText, musíme zverejniť a zdieľať svoj zdrojový kód s ostatnými a nie je možné to využiť pre komerčné účely. Navyše, na rozdiel od PDFBox, užívateľ iText pod touto licenciou nemá právo túto knižnicu modifikovať podľa svojej potreby. Pre vytvorenie nášho projektu je knižnica PDFBox vhodnejšia a dostatočná.

5.3 PDF Clown

PDF Clown je bezplatná Java a .Net knižnica, ktorá rovnako ako PDFBox a iText dokáže vytvoriť a spracovať PDF dokument. Upravuje dokument prostredníctvom viacerých abstraktných vrstiev a je založená na základe špecifikácie PDF verzie 1.7 [11]. Na rozdiel od PDFBox a iText je PDF Clown neaktualizovaná. Najnovšia verzia 0.2.0 bola publikovaná už v roku 2015 ale doteraz ešte neprebíhalo žiadne vylepšenie, pričom už vychádzala PDF verzia 2.0 (PDF 1.7 už je zastaralé).

³<https://itextpdf.com/en>

Kapitola 6

Návrh implementácie CSS3 vlastností pre výstup vo formáte PDF

Ako bolo zmienené v zadaní, táto bakalárska práca je zameraná na rozšírenie projektu WebVector o podporu výstupu vo formáte PDF spolu s renderovaním pokročilých CSS3 vlastností do tohoto formátu. Vzhľadom na to, že v poslednej aktualizácii 3.4 bola pridaná do aplikácie WebVector možnosť vybrať PDF výstup a zároveň bolo nastavené napojenie projektu CSSBoxPdf (podkapitola 3.5) na túto aplikáciu, už je možné generovať PDF výstup pomocou WebVector. Z toho vyplýva, že našou úlohou ostáva doplniť schopnosť renderovať na PDF výstupe CSS3 vlastností, ktoré boli rozobrané v kapitole 4. V tejto kapitole budú postupne popísané návrhy pre realizovanie týchto vlastností vo formáte PDF. Implementácia týchto vlastností je vložená do existujúcej triedy PDFRenderer, ktorá je súčasťou projektu CSSBoxPdf.

6.1 Návrh renderovania 2D transform vlastností

Realizáciu dvojdimenzionálnych (2D) transform vlastností v CSS zaisťuje 2D matica (3x3), ktorá sa používa pre transformáciu súradníc elementov v HTML dokumente. Táto transformačná matica má nasledujúci tvar:

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Hodnoty a, b, c, d, e, f sú nastavené pomocou zadaných CSS3 transformačných funkcií rotate(), scale(), translate(), skew(), matrix(). Každá z týchto funkcií nastavuje určité hodnoty tejto matice. Hodnoty 0, 0, 1 na poslednom riadku sú pevne dané a nebudú nikdy zmenené. Proces transformácie elementu prebieha tak, že sa postupne vynásobujú súradnice x, y všetkých rohov daného elementu s maticou vytvorenou zo zadanej transformačnej funkcie (v prípade zadania viac funkcií naraz sa používa matica, ktorá vzniká zlúčením (angl. concatenate) všetkých matíc reprezentujúcich týchto funkcií). Pomocou novo vzniknutých rohov sa následne vykresľuje rámček a obsah výsledného elementu. Nové súradnice sa vypočítajú nasledujúcim spôsobom:

$$x' = a \times x + c \times y + e$$

$$y' = b \times x + d \times y + f$$

x a y sú pôvodné súradnice bodu, x' a y' sú jeho súradnice po transformácii a hodnoty a , b , c , d , e , f sú hodnoty matice, ktoré sú uvedené vyššie.

Ako bolo zmienené v podkapitole 4.2, CSS3 *transform* vlastnosť sa aplikuje okolo počiatočného bodu, ktorý sa nachádza v strede elementu a je možné tento bod nastaviť na inú pozíciu pomocou vlastnosti *transform-origin*. Avšak u PDF sa tento bod nachádza v ľavom dolnom rohu elementu. Z toho dôvodu je potrebné pred transformáciou posunúť element so zadanou transform vlastnosťou v PDF tak, aby jeho ľavý dolný roh odpovedal jeho počiatočnému bodu v CSS3. Po vykonaní transformácie je tento element posunutý späť o rovnaké hodnoty.

Pre realizáciu transformácie v PDF formáte na platforme Java musíme najprv získať spomenutú CSS3 transformačnú maticu. Pre vytvorenie tejto matice je vhodné používať triedu *AffineTransform*, ktorá je dostupná v balíku `java.awt.geom`. Táto trieda reprezentuje matematickú maticu s rozmerom 3x3 a obsahuje špecifické metódy, ktoré na základe zadaných transformačných funkcií a ich parametrov nastavuje príslušné hodnoty matice (obrázok 6.1). Napr. pri zadaní CSS3 vlastnosť *transform: scale(1.2, 2)*; používame metódu *scale()* v *AffineTransform*, pomocou ktorej sa dosadí do hodnoty a v *AffineTransform* hodnotu 1.2 a do hodnoty d ¹ hodnotu 2.

Po tvorení triedy *AffineTransform* je potrebné zlúčiť (angl. concatenate) túto triedu s *current transformation matrix* (CTM)², ktorá je jedným z viacerých parametrov globálnej dátovej štruktúry *graphics state* (odsek Grafika PDF v podkapitole 2.4). Zlúčenie *AffineTransform* s CTM je možné realizovať pomocou metódy *transform(Matrix)*, ktorá je súčasťou triedy *PDPageContentStream*³ z knižnice *PDFBox*. Parameter *Matrix* je trieda z *PDFBox*, ktorá podobne ako *AffineTransform* reprezentuje matematickú maticu a je možné ju vybudovať priamo z *AffineTransform*. Po zlúčení dochádza k vykresleniu elementov na výstup vo formáte PDF využitím *PDPageContentStream*.

scale(x,y)	scaleX(x)	scaleY(y)	translateX(x)	translateY(y)	translate(x,y)
$\begin{pmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} x & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}$
skewX(x)	skewY(y)	skew(x,y)	rotate(θ)		
$\begin{pmatrix} 1 & \tan x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ \tan y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & \tan x & 0 \\ \tan y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$		

Obr. 6.1: CSS3 transformačné funkcie a ich príslušné matice [15].

¹ a a d sú hodnoty transformačnej matice, jej tvar je definovaný na začiatku podkapitoly 4.2

² používa sa pre mapovanie objektov dokumentu v user space na device space v PDF a na základe hodnôt tejto matice sa zmenia súradnice objektov dokumentu. Pre viac informácií [1] kapitola 4.

³ *PDPageContentStream* obsahuje metódy, ktoré nastavujú parametre v štruktúre *graphics state* a obsahuje aj metódy, ktoré na základe dát v *graphics state* vykresľujú elementy na PDF formát.

6.2 Návrh renderovania CSS3 gradientov

V rámci našej práce bude potrebné implementovať možnosť renderovať lineárne a radiálne funkcie, ktoré sú deklaráciou CSS vlastnosti *background-image* alebo *background*. Pre lepšie pochopenie ich návrhu si najprv rozoberieme syntax jednotlivých funkcií.

Syntax radiálneho gradientu

radial - gradient([< shape > || < size >] [at < position >]?, | at < position > ,] ? < color - stop > [, < color - stop >]+) [14]

<shape> definuje tvar tzv. *ending shape* radiálneho gradientu. Tento parameter môže byť zadaný hodnotou *circle* alebo *ellipse* a v prípade vynechania tohoto parametru bude mať gradient elipsový tvar.

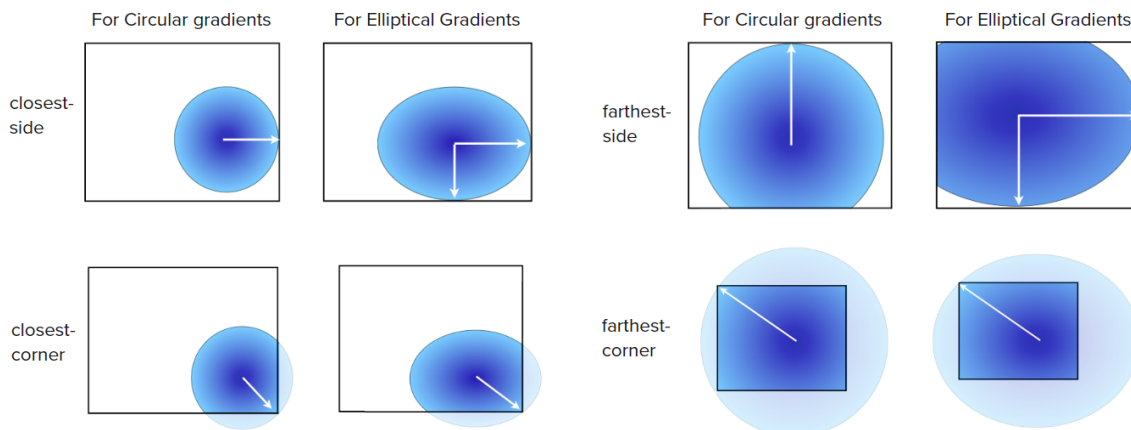
<size> určuje rádius konečného tvaru radiálneho gradientu. Ak je tvar gradientu kruhový, <size> môže byť zadaný jednou hodnotou v tvare čísla spolu s konkrétnou dĺžkovou jednotkou, ktorá reprezentuje rádius (polomer) kruhu. Ak je tvar elipsový, tento parameter môže obsahovať dve hodnoty v tvare čísla alebo v percentách, pričom prvá hodnota určuje horizontálny rádius a druhá vertikálny rádius elipsy. Ak nie je definovaný konečný tvar gradientu, tak je možné podľa počtu hodnôt zadaných v <size> určiť tento tvar. Parameter <size> môžeme zadať aj pomocou nasledujúcich reťazcov (*keyword*):

- **closest-side** - Ak má ending shape kruhový tvar, tak jeho rádius bude mať hodnotu reprezentujúcu vzdialenosť od stredu gradientu do najbližšej hrany rámčeka elementu, do ktorého je aplikovaný daný gradient. Ak má tvar elipsy, tak jeho rádiusy odpovedajú vzdialenosti od stredu gradientu do jeho najbližšej hrany v horizontálnom aj vertikálnom smere.
- **farthest-side** - podobne ako pri zadaní closest-side avšak rádius je daný vzdialenosťou od stredu do jeho najvzdialenejšej hrany.
- **closest-corner** - u radiálneho gradientu s kruhovým útvarom bude jeho rádius rovný vzdialenosti od stredu gradientu do najbližšieho rohu rámčeka elementu. Pri elipsovom útvare bude horizontálny rádius daný touto vzdialenosťou a pomer horizontálneho a vertikálneho rádiusu tejto elipsy je rovný pomeru rádiusov elipsy vzniknutej v prípade zadania keyword closest-side.
- **farthest-corner** - má podobný význam ako closest-corner avšak kruhový radiálny gradient bude mať rádius rovný vzdialenosti od stredu do najvzdialenejšieho rohu elementu. U elipsového tvaru podobne ako u closest-corner bude pomer jeho rádiusov rovný pomeru rádiusov elipsy pri zadaní farthest-side.

Ak tento parameter nie je zadaný vo funkcií, bude mať implicitnú hodnotu farthest-corner.

<position> definuje pozíciu, kde sa nachádza stred radiálneho gradientu v rámci elementu. Tento parameter posunie počiatočný bod, ktorý sa nachádza v ľavom hornom rohu elementu, o zadané hodnoty v smere osi x doprava a v smere osi y nadol. <position> môže obsahovať najviac dve hodnoty, pričom prvá hodnota reprezentuje veľkosť posunutia počiatočného bodu na osi x a druhá hodnota definuje veľkosť posunutia na osi y. Hodnoty

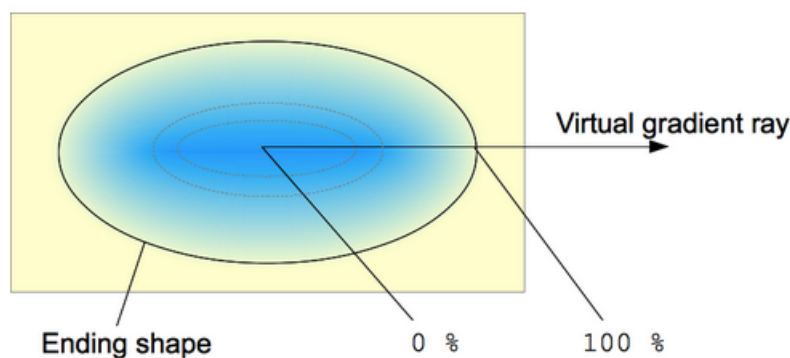
⁴tento obrázok je tvorený z obrázkov dostupných na stránke <https://www.webdirections.org/blog/css3-radial-gradients/>



Obr. 6.2: Náčrt významu jednotlivého keyword v parametre `<size>`.⁴

parametru môžeme zadať v číslach alebo percentách, pričom 100% u prvej hodnoty reprezentuje šírku elementu a u druhej reprezentuje výšku. Ak tento parameter nie je definovaný, tak bude mať implicitné hodnoty 50% 50% (center elementu).

`<color-stop>` je jediný povinný parameter vo funkcií `radial-gradient()`, ktorý je definovaný dvoma alebo viacerými farbami, pomocou ktorých sa vykresľuje gradient. Každá farba v `<color-stop>` môže byť zadaná spolu s hodnotou v percentách, ktorá reprezentuje pozíciu tejto farby na tzv. *virtual gradient ray* (obrázok 6.3), konkrétne na rádiuoch tvoriacich ending shape. Ak pri farbách nie sú definované príslušné hodnoty pre rozmiestnenie farieb, tak sa automaticky dopočítajú podľa špecifického spôsobu, ktorý je popísaný v odseku [Vypočítanie bodov definujúcich hrany a rohy rámčeka](#).



Obr. 6.3: Radial gradient a jej komponenty.⁵

Syntax lineárneho gradientu

`linear-gradient([[<angle> | to <side-or-corner>],] ? <color-stop> [, <color-stop>]+)`

Prvý parameter tejto funkcie je nepovinný a definuje smer priamky tzv. gradient line (obrázok 4.4), na ktorej sa potom rozmiestia farby lineárneho gradientu. Tento parameter je

⁵použitý obrázok je dostupný na stránke <https://developer.mozilla.org/en-US/docs/Web/CSS/radial-gradient>

zadaný hodnotou `<angle>` alebo `<side-or-corner>`, pričom `<angle>` reprezentuje hodnoty v stupňoch (pri 0deg je gradientline zvislý a ending point sa nachádza smerom hore, pri kladných hodnotách sa točí v smere hodinových ručičiek a pri záporných sa točí proti smeru) a `<side-or-corner>` reprezentuje nasledujúce kľúčové slová: *left* (je ekvivalentné s 270deg), *right* (90deg), *top* (0deg), *bottom* (180deg), *top left*, *top right*, *bottom left*, *bottom right*. Pri zadaní kľúčového slova definujúceho roh elementu prebieha prevod tohto slova do uhlu pomocou knižnice CSSBox. Vyššie uvedené kľúčové slová pre rohy budú v tom poradí reprezentované nasledujúcimi uhlami: 315deg, 45deg, 135deg, 225deg. Avšak tento prevod je správny iba v prípade elementu so štvorcovým útvarom.

`<color-stop>` predstavuje povinný parameter tejto funkcie. Tento parameter obsahuje dve alebo viaceré farby, ktoré sú zadané pomocou špecifických slov (red, green atď.) alebo pomocou hexadecimálnych hodnôt. Tieto farby sa používajú pre vykreslenie lineárneho gradientu a každá farba môže byť zadaná spolu s hodnotou v percentách, ktorá určuje pozíciu danej farby na gradient line, konkrétne na úsečke medzi starting point a ending point (obrázok 4.4). 0% reprezentuje pozíciu v starting point, 100% reprezentuje ending point. Ak farba nemá zadanú hodnotu pre jej rozmiestnenie, tak sa táto hodnota dopočíta podľa spôsobu špecifikovaného v odseku [Vypočítanie bodov definujúcich hrany a rohy rámčeka](#).

Návrh implementácie radiálneho gradientu

Realizáciu vykreslenia radiálneho gradientu na PDF výstup vo platforme Java zabezpečuje metóda *shadingfill(PDShadingType3)*, ktorá je súčasťou triedy *PDPageContentStream* z knižnice *PDFBox*. Táto metóda vyfarbuje daný element so zadaným parametrom. Parameter tejto metódy - *PDShadingType3* je trieda z knižnice *PDFBox*, ktorá reprezentuje vykreslený radiálny gradient a je definovaná dátami, ktoré sú hodnotami parametrov *CSS3* funkcie *radial-gradient()*. Pre nastavenie týchto dát do triedy poskytuje trieda *PDShadingType3* niekoľko špecifických metód. Pomocou týchto metód ukladáme hodnoty čerpané z parametrov funkcie *radial-gradient()* zadanej na vstupe do triedy *PDShadingType3*. Tieto hodnoty môžeme získať pomocou knižnice *JStyleParser*, konkrétne použitím triedy *TermFunction.RadialGradient*, ktorá je podtriedou triedy *TermFunction* a ktorú získame použitím metódy z hlavnej triedy *Elementbox* reprezentujúcej elementy s radiálnym gradientom. V prípade zadania parametru `<size>` na vstup pomocou kľúčových slov je potrebné vypočítať rádiusy pre ending shape na základe vlastností parametru `<size>`.

Do *PDShadingType3* najprv uložíme x-ovú a y-ovú súradnicu bodu, ktorý definuje center radiálneho gradientu, okolo ktorého sa vykresľujú farby a ich prechod. Súradnice tohoto bodu sa nachádzajú implicitne v centre elementu a je možné ich získať z parametru `<position>`, ak je zadaný. Pre vykreslenie ending shape v tvare elipsy je použitý horizontálny a vertikálny rádius a pre kruhový tvar je potrebné použiť iba jeden rádius (polomer). Avšak trieda *PDShadingType3* obsahuje iba jeden parameter pre rádius, tzn., že dokáže reprezentovať iba kruhový tvar. Tento problém je možno vyriešiť tým, že do tejto triedy vložíme ten dlhší rádius z dvoch rádiusov elipsy a následne pred vykreslením gradientu do elementu prevedieme transformáciu s použitím *PDFBox* metódy *scale()*, pomocou ktorej zmenšíme gradient v smere kratšieho rádiusu na základe pomeru rádiusov elipsy. Ďalej je potrebné dopočítať chýbajúce hodnoty určujúce pozície jednotlivých farieb a následne uložiť tieto farby a ich pozície do *PDShadingType3*. Nakoniec sa vykresľuje gradient na výstup pomocou metódy *shadingfill(PDShadingType3)*.

Návrh implementácie lineárneho gradientu

Podobne ako u radiálneho gradientu existuje pre realizáciu lineárneho gradientu na PDF výstup iba jediná možnosť, a to je vytvorenie triedy *PDShadingType2*, ktorá tento gradient reprezentuje, a následne vyfarbiť pozadie daného elementu s inštanciou tejto triedy pomocou metódy *shadingfill(PDShadingType2)*.

Pre vykreslenie výsledného lineárneho gradientu je potrebné do triedy *PDShadingType2* uložiť hodnoty x-ovej a y-ovej súradnice počiatočného (starting point) a koncového bodu (ending point) definujúceho gradient line, zoznam použitých farieb a ich príslušné hodnoty reprezentujúce ich pozície na gradient line v tvare desatinných čísel (napr. 10% je ekvivalentné 0.1). Keďže súradnice počiatočného a koncového bodu nie sú súčasťou parametrov CSS3 funkcie *linear-gradient()*, tak ich musíme vypočítať na základe uhlu gradient line definovaného pomocou vstupného parametru. Na obrázku 4.4 môžeme vidieť, že smer gradient line je určený uhlom gradientu a podľa tohoto smeru sú určené dva najbližšie rohy pri každej strane gradient line, cez ktoré sú vedené kolmice na gradient line a následne tvoria dva priesečníky, ktoré sú počiatočným a koncovým bodom gradient line.

Spôsob vypočítania súradníc uvedených bodov je inšpirovaný spôsobom uvedeným v diplomovej práci, ktorá rieši vykreslenie gradientov na SVG výstup [6]. Tento spôsob spočíva v tom, že sa najprv vytvoria rovnice, ktoré definujú smernicový tvar gradient line a kolmíc. Následne sa vypočítavajú súradnice priesečníkov definovaných týmito rovnicami. Ostatné potrebné dáta pre vytvorenie triedy *PDShadingType3* sú dané vstupnými parametrami funkcie *linear-gradient()* a hodnoty týchto parametrov môžeme získať podobne ako u radiálneho gradientu pomocou metód tried dostupných v knižnici *JStyleParser* a *CSS-Box*. Jediný rozdiel je v tom, že namiesto triedy *TermFunction.RadialGradient* sa používa trieda *TermFunction.LinearGradient*. Ak je uhol zadaný pomocou jedného z kľúčových slov *<side-or-corner>*, tak nástroj *CSSBox* automaticky prevedie tieto kľúčové slová na hodnotu príslušného uhlu (napr. hodnota *45deg* je ekvivalentná *to top right*). V prípade, že nie sú zadané pri vstupe všetky žiadané hodnoty reprezentujúce pozície farieb, tak je nutné tieto hodnoty dopočítať podľa pravidiel uvedených v odseku nižšie.

Vypočítanie chýbajúcich hodnôt určujúcich pozície farieb u CSS3 gradientových funkciách

Spôsob vypočítania chýbajúcich hodnôt pre rozmiestnenie farieb je definovaný podľa nasledujúcich pravidiel:

- Ak nie je zadaná hodnota pozície pre prvú farbu, táto farba bude mať automaticky hodnotu 0%. Ak posledná farba nemá definovanú pozíciu, bude mať implicitne hodnotu 100%.
- Ak pre prvú farbu je zadaná hodnota pozície odlišná ako 0%, bude potrebné pre našu implementáciu vytvoriť duplikáciu tejto farby, ktorá bude mať hodnotu pozície 0%. To isté platí pre poslednú farbu s hodnotou pozície odlišnou ako 100%, v tom prípade bude mať pozíciu 100%. Pr. pri zadaní do *<color-stop>* *red 10%, blue 90%* bude potrebné previesť do nasledujúcej podoby - *red 0%, red 10%, blue 90%, blue 100%*, pričom tieto dve podoby sú ekvivalentné.
- Ak jedna alebo viac po sebe idúcich farieb okrem prvej a poslednej farby neobsahujú hodnoty určujúce ich rozmiestnenie na gradient line alebo na rádiusoch, tak tieto hodnoty sa vypočítajú nasledovne:

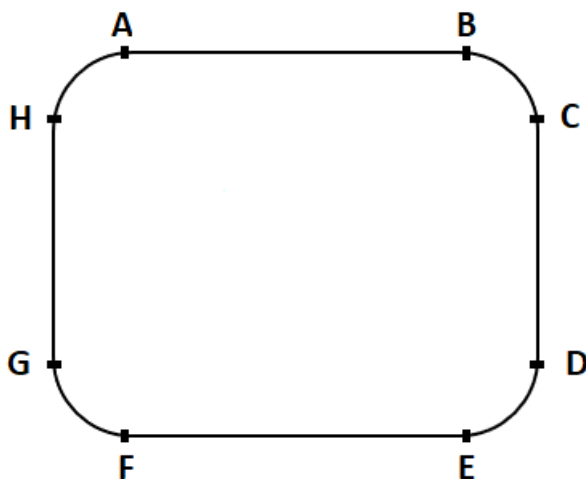
Nech je hodnota pozície farby, ktorá sa nachádza pred prvou farbou bez hodnoty, označená ako x . Nech y je hodnota pozície farby, ktorá sa nachádza za poslednou farbou bez hodnoty. Nech n je počet po sebe idúcich farieb bez zadaných pozícií a nech sú hodnoty týchto farieb označené ako a_1, a_2, \dots, a_n :

$$a_i = x + ((y - x) / (n + 1) \times i), \quad i = 1 \dots n$$

Pr. parameter `<color-stop>` so zadanou hodnotou red, yellow, green, blue 60%, orange, white 90% je prevedený pomocou vzorca do ekvivalentného tvaru red 0%, yellow 20%, green 40%, blue 60%, orange 75%, white 90%, white 100%.

6.3 Návrh renderovania border-radius

V knižnici PDFBox neexistuje žiadna trieda, ktorá poskytuje metódy pre jednoduchú realizáciu CSS3 border-radius vlastnosť na PDF výstup. Ostáva jediná možnosť, ako kresliť rámček elementu s okrúhlymi rohmi, a to kresliť rámček po častiach. Výsledný rámček s zaoblenými rohmi sa bude skladať zo štyroch úsečiek, ktoré definujú hrany elementu a zo štyroch kriviek, ktoré reprezentujú rohy. Každá z týchto úsečiek a kriviek je definovaná počiatočným a koncovým bodom. Detail výsledného rámčeka zloženého z úsečiek a kriviek a umiestnenie bodov, ktoré určujú tieto útvary je možné vidieť na obrázku 6.4.



Obr. 6.4: Náčrt rámčeka s zaoblenými rohmi a bodov, ktoré definujú jednotlivé časti rámčeka.

Vypočítanie bodov definujúcich hrany a rohy rámčeka

Pre zostrojenie úsečiek a kriviek tvoriacich hrany a rohy rámčeka je potrebné vypočítať súradnice ich krajných bodov, respektívne body A, B, C, D, E, F, G, H (obr. 6.4). Na obrázku 6.5 môžeme vidieť, že x-ová súradnica bodu A je posunutá od x-ovej súradnice ľavého horného rohu rámčeka elementu pred aplikovaním vlastnosti *border-radius* o veľkosť x -rádius a y-ová súradnica bodu A ostáva nezmenená. Podobne to platí aj pre ostatné body. Z toho vyplýva, že pre vypočítanie súradníc všetkých bodov uvedených vyššie je potrebné

získať súradnice jednotlivých rohov pôvodného rámčeka a x-rádus spolu s y-rádus, ktoré vytvárajú okrúhle rohy. Súradnice rohov a rádusov sa dajú ľahko získať pomocou funkcií dostupných v triede *Elementbox* z knižnice *CSSBox*, ktorá reprezentuje daný element a obsahuje všetky jeho informácie a CSS vlastnosti, ktoré sú na tento element aplikované.

Vytvorenie výsledného rámčeka s zaoblenými rohmi

Pre vytvorenie úsečiek AB, CD, EF, GH je vhodné použiť metódu *curveTo1(x1, y1, x3, y3)*, ktorá je súčasťou triedy *PDFPageContentStream* a ktorá slúži pre tvorenie krivky tzv. *cubic Bézier curves*⁶. Parametre tejto metódy reprezentujú súradnice kontrolných bodov, ktoré určujú smer krivky. Pre vytvorenie krivky v tvare úsečky je potrebné dosadiť do parametru x1 a y1 súradnice bodu, ktorý sa nachádza v strede počiatočného a konečného bodu určujúceho úsečky, do x3 a y3 dosadíme súradnice konečného bodu. Pred použitím tejto metódy je potrebné posunúť aktuálny bod kreslenia obsahu do počiatočného bodu úsečky pomocou metódy *moveto(x, y)*, pričom x a y sú súradnice tohoto bodu.

Pre vytvorenie zaoblených rohov HA, BC, DE, FG budeme používať metódu *curveTo(x1, y1, x2, y2, x3, y3)*. Táto metóda je podobná metóde *curveTo1*. Avšak narozdiel od tejto metódy má až tri kontrolné body ako parameter, pomocou ktorých dokáže kresliť zložitejšie krivky. Pre kreslenie rohu napr. BC budú súradnice kontrolných bodov (x1, y1, x2, y2, x3, y3) vypočítané nasledovne:

$$x_1 = x_B + bezier \times topright - x - radius, \quad bezier \doteq 0.552228474^7$$

$$y_1 = y_B$$

$$x_2 = x_C$$

$$y_2 = y_C + bezier \times topright - y - radius, \quad bezier \doteq 0.552228474^8$$

$$x_3 = x_C$$

$$y_3 = y_C$$

Podobným spôsobom sa vytvoria kontrolné body pre ostatné rohy rámčeka.

Po vytvorení všetkých rohov a úsečiek vyvoláme metódu *stroke()* z triedy *PDFPageContentStream*, ktorá kreslí vytvorené krivky na PDF výstup.

6.4 Návrh renderovania filter vlastností

CSS3 filter vlastnosť sa používa hlavne pre úpravy obrázkov. V našom projekte sú všetky obrázky reprezentované triedou *BufferedImage*, ktorá je potomkom abstraktnej triedy *Image* v jazyku Java. Táto trieda reprezentuje každý obrázok ako dvojrozmerné pole pixelov, pričom každý pixel je definovaný rozmerom (Raster) a modelom farieb (ColorModel), typicky *RGBA model*¹⁰. Pre realizáciu filteru vlastnosť musíme modifikovať jednotlivé farebné komponenty modelu *RGBA*(red, green, blue, alpha) každého pixelu na obrázku.

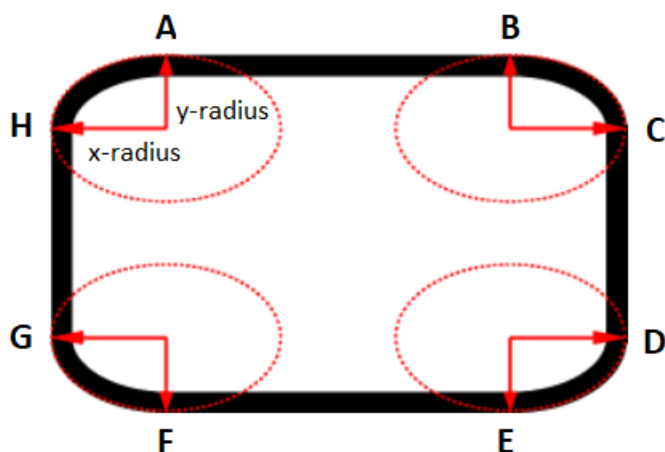
⁶<http://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node12.html>

⁷<https://nacho4d-nacho4d.blogspot.com/2011/05/bezier-paths-rounded-corners-rectangles.html>

⁸napriek tomu, že druhý kontrolný bod sa nachádza nad bodom C, sčítame y-ovú súradnicu kvôli opačnému súradnicovému systému v PDF.

⁹použitý obrázok je modifikovaný z obrázku dostupného na stránke https://tympanus.net/codrops/css_reference/border-top-right-radius/

¹⁰https://www.w3schools.com/css/css3_colors.asp



Obr. 6.5: Náčrt spôsobu vykreslenia CSS3 vlastnosti `border-radius`⁹.

Návrh filter vlastnosti definovanej funkciou `brightness()` a `opacity()`

CCS3 funkciu `brightness()` a `opacity()` je možné v Java implementovať pomocou využitia triedy `RescaleOp` dostupnej v balíku `java.awt.image`. Je to trieda, ktorá modifikuje každý pixel nachádzajúci sa na obrázku podľa zadaných parametrov pri vytvorení jej konštruktoru. Konštruktor tejto triedy má nasledujúci tvar: `RescaleOp(float[]11 scaleFactors, float[] offsets, RenderingHints12 hints)`. Do parametru `scaleFactors` je potrebné zadať štyri desatinné hodnoty, pričom prvé tri hodnoty reprezentujú hodnoty funkcie `brightness()`, ktoré sa budú aplikovať na farebné komponenty `red`, `green`, `blue` a posledná hodnota zmení priehľadnosť (`opacity`) toho pixelu. Parametre `offsets` a `hints` nie sú pre náš účel potrebné a preto do `offsets` dosadíme štyri nulové hodnoty a `hints` nastavíme ako `null`. Na konci použijeme metódu `filter()` z triedy `RescaleOp`, ktorá aplikuje zmeny na pôvodný obrázok zadaný ako parameter a vráti výsledný obrázok na výstup. Pr.: pre vlastnosť `filter: brightness(150%)` nastavíme parameter `offsets` hodnotami 1.5, 1.5, 1.5, 1 a pre vlastnosť `filter: opacity(10%)` hodnotami 1, 1, 1, 0.1.

Návrh filter vlastnosti definovanej funkciou `invert()`

Realizácia funkcie `invert()` spočíva v tom, že sa prevrátia hodnoty farebných komponentov u každého pixelu obrázka. Ako prvé je potrebné získať farbu jednotlivého pixelu a následne hodnoty jej komponentov `red`, `green`, `blue`. Nové hodnoty týchto komponentov po prevrátení sa vypočítajú nasledovne:

$$newvalue = 255 \times invert - oldvalue$$

Hodnota 255 je maximálnou hodnotou, ktorú môže mať farebný komponent a reprezentuje bielu farbu. `Invert` je hodnota získaná z parametru CSS3 funkcie `invert()` a je potrebné túto hodnotu previesť na desatinné číslo. V prípade, že prekročí nová hodnota hranicu hodnôt pre komponent (od 0 do 255), tak je potrebné ju nastaviť na hodnotu najbližšej hranice.

¹¹<https://www.geeksforgeeks.org/arrays-in-java/>

¹²<https://docs.oracle.com/javase/7/docs/api/java/awt/RenderingHints.html>

Kapitola 7

Implementácia uvedených CSS3 vlastností

V tejto kapitole sa budeme venovať postupu implementácie jednotlivých CSS3 vlastností podľa ich návrhu rozobraných v kapitole 6. Ako prvú budeme popisovať štruktúru našej práce. Ďalej je popísaný implementačný postup pre vykreslenie pokročilých CSS3 vlastností na PDF výstup. Na konci sa budeme zaoberať implementáciou rozšírenia našej práce. Jedná sa o implementáciu priehľadnosti a funkcie *repeating-linear-gradient()* u gradientov a o implementáciu vlastnosti *list-style*¹ pre formát SVG, ktorá je doplnená do triedy SVGDOMRenderer v projekte CSSBoxSvg.

7.1 Štruktúra rozšírenia projektu CSSBoxPdf

Implementácia nášho rozšírenia do projektu CSSBoxPdf je rozdelená a uložená do štyroch nových tried a do triedy PDFRenderer, ktoré sú umiestnené v balíku *org.fit.cssbox.render*. Nové triedy *BorderRadius*, *Filter*, *LinearGradient* a *RadialGradient* reprezentujú štruktúry obsahujúce informácie a funkcií potrebné pre vykresľovanie príslušných CSS3 vlastností. Do už existujúcej triedy PDFRenderer je priamo zakomponovaný postup získania dát a realizácia transformačných funkcií a zároveň sú začlenené nové funkcie, ktoré vykresľujú tieto vlastnosti na PDF výstup.

7.2 Implementáci dvojdimenzionálnych transformačných vlastností

Pre vykreslenie transform vlastnosti je potrebné najprv zistiť pozíciu počiatočného bodu transformácie. Ak je na element aplikovaná vlastnosť *transform-origin*, tak táto pozícia je daná jej parametrami, inak sa implicitne nachádza v strede elementu. Následne musíme posunúť daný element na miesto počiatočného bodu pomocou vytvorením matice s hodnotami určujúcimi funkciu *translate()*. Ďalej je potrebné zistiť, aké funkcie zahrňuje CSS3 transform vlastnosť vo svojej deklarácii a parametre týchto funkcií. Typ transformačnej funkcie a hodnoty jej parametrov získame pomocou použitím triedy ElementBox a NodeData z knižnice CSSBox, konkrétne pomocou nasledujúceho kódu:

```
element.getStyle().getValue(TermList.class, "transform")
```

¹https://www.w3schools.com/cssref/pr_list-style.asp

element je inštanciou triedy `ElementBox`. Metóda `getValue()` vráti podľa vstupného CSS kódu jeden alebo viacero objektov typu `TermFunction`, ktorý obsahuje rozhrania reprezentujúce transformačné funkcie. Každé rozhranie reprezentuje práve jednu transformačnú funkciu a každé má definované špecifické metódy pre získanie hodnoty vstupných parametrov danej funkcie. Po získaní všetkých potrebných informácií o zadanej transform vlastnosti vytvoríme maticu zloženú z príslušných hodnôt. Následujúcim krokom je posunutie elementu na jeho pôvodné umiestnenie vytvorením matice reprezentujúcej funkcie `translate()`. Na konci zlúčime (angl. concatenate) všetky vytvorené matice s CTM v graphics state pomocou metódy `transform()` z triedy `PDPPageContentStream`. Ak nasledujúci element neobsahuje transformáciu alebo má zadanú odlišnú transform vlastnosť, tak musíme obnoviť graphics state do pôvodného stavu použitím metódy `restoreGraphicsState()`, aby nenastala situácia, že na daný element bude aplikovaná transform vlastnosť od predchádzajúceho elementu.

7.3 Implementácia lineárneho a radiálneho gradientu

Vykreslenie gradientov žiada mnoho dát, ktoré nie sú definované vo vstupných parametroch a ktoré sa dajú získať iba vlastným dopočítaním. V tomto dôsledku sa implementácia gradientov stáva najkomplikovanejším zo všetkých implementácií v rámci celej našej práce. Pre realizáciu gradientov boli vytvorené dve triedy `LinearGradient` a `RadialGradient`, pričom každá z týchto tried reprezentuje konkrétny gradient s rovnakým názvom. Vykreslenie gradientov na výstup zabezpečuje metóda `drawBgGrad`, ktorá bola implementovaná v triede `PDFRenderer`. Táto metóda na základe svojich parametrov vytvorí obdĺžnik reprezentujúci hranice elementu použitím `PDPPageContentStream` metódy `addRect()` a následne vyplní tento obdĺžnik výsledným gradientom pomocou metódy `shadingfill()`. Pred volaním metódy `shadingfill()` je volaná metóda `transform()`, ktorá slúži pre deformáciu kruhového útvaru na útvar elipsy u radiálneho gradientu. U lineárneho gradientu bude matica pre transformáciu nulová (nie je potrebná transformácia). Spôsob implementácie gradientov pomocou knižnice `PDFBox` je inšpirovaný projektom `pdfbox-graphics2d` ².

Trieda `LinearGradient`

Reprezentuje lineárny gradient a obsahuje metódy pre vytvorenie výsledného gradientu. V tejto triede sú uložené ako atribúty súradnice počiatočného a konečného bodu definujúceho gradient line typu `double`, zoznam farieb v tvare poľa objektov typu `Color` a zoznam ich pozícií na gradient line v tvare poľa desatinných čísel (`float[]`). Proces vypočítania počiatočného a koncového bodu prebieha v metóde `createGradLinePoints()` a táto metóda je podobná metóde `setAngleDeg()` v projekte `CSSBoxSvg`, ktorá slúži k rovnakému účelu avšak sa používa pre renderovanie lineárneho gradientu na SVG výstup. V dôsledku opačného súradnicového systému v PDF sa niektoré výpočty rohov a smerníc vo funkcií `createGradLinePoints()` odlišujú od výpočtov v `setAngleDeg()`. Ďalej sa v tejto triede nachádza metóda `createColorStopsLength()`, pomocou ktorej sa vypočítajú chýbajúce hodnoty určujúce pozície farieb pomocou pravidiel uvedených v návrhu gradientov. Vytvorenie výsledného gradientu má na starosť metóda `createLinearGrad()`, ktorá postupne ukladá získané dáta do triedy `PDS shadingType2` pomocou príslušných metód tejto triedy. Najprv sa ukladajú súradnice bodov do objektu `CosArray` ³ a následne sa tento objekt ukladá do triedy pomocou `PDFBox` me-

²<https://github.com/rototor/pdfbox-graphics2d>

³<https://pdfbox.apache.org/docs/1.8.10/javadocs/org/apache/pdfbox/cos/COSArray.html>

tódy *setCoords()*. Následne je potrebné vytvoriť triedu `PDFFunctionType3`⁴, ktorá bude obsahovať farby gradientu a ich pozície. Pre vytvorenie tejto triedy boli implementované metódy *buildType3Function()* a *buildType2Functions()*. Na konci sa táto trieda uloží do triedy `PDShadingType2` použitím metódy *setFunction()*.

Trieda `RadialGradient`

Táto trieda bola vytvorená pre uloženie atribútov a metód slúžiacich k vytvoreniu výsledného radiálneho gradientu. Medzi atribúty potrebné pre vykreslenie radiálneho gradientu patria súradnice stredného bodu, okolo ktorého sa vytvorí daný gradient, tvar gradientu (je uložený ako reťazec typu `String`), rádiusy pre vytvorenie tvaru gradientu, pole farieb typu `Color` a pole ich príslušných pozícií na rádiusoch. V tejto triede sú implementované nasledujúce tri metódy, ktoré slúžia pre uloženie hodnôt získaných zo vstupných parametrov CSS3 funkcie `radial-gradient()` do vyššie uvedených atribútov:

- **`setShape()`** - táto metóda slúži k uloženiu útvaru gradientu zadaného v parametre `<shape>` u radiálnych funkcií. Hodnotu tohoto parametru môžeme dostať použitím *getShape()*.
- **`setGradientCenter()`** - pomocou tejto metódy môžeme uložiť súradnice stredného bodu gradientu, ktoré sú zadané v parametre `<position>`. Tento parameter je možné získať volaním *getPosition()*.
- **`setRadiusFromSizeValue()`** - použitím tejto metódy sa uložia hodnoty rádiusov, ktoré sú zadané ako číselné hodnoty v parametre `<size>`. V prípade, že rádiusy sú zadané pomocou jedného z kľúčových slov, tak je potrebné získať toto kľúčové slovo pomocou metódy *getSizeIdent()* a následne zvolať implementovanú metódu *setRadiusFromSizeIdent()* pre vypočítanie rádiusov na základe zadaného slova. *getShape()*, *getSize()*, *getSizeIdent()* a *getPosition()* sú metódy definované v triede `TermFunction.RadialGradient` v knižnici `JStyleParser` a tieto metódy vrátia hodnoty ako objekty typu `Term`. Pre ich ďalšie spracovanie je potrebné tieto hodnoty previesť do numerických dátových typov (`int`, `double`⁵). Toto prevedenie zabezpečuje trieda `CSSDecoder` z knižnice `CSSBox`.

Vypočítanie príslušných hodnôt rádiusov zadaných pomocou kľúčových slov prebieha tak, že sa najprv vypočítajú vzdialenosti od stredného bodu do každej strany a do každého rohu elementu. Ďalej sa podľa zadaného kľúčového slova vyberú príslušné vzdialenosti, pričom tieto hodnoty vzdialenosti určujú rádiusy gradientu (napr. pre kľúčové slovo `closest-side` pre gradient s elipsovým útvarom sú vybrané vzdialenosti od stredného bodu do najbližšej strany v horizontálnom a vertikálnom smere a tieto vzdialenosti určujú x-rádus a y-rádus výslednej elipsy). Vypočítanie rádiusov podľa kľúčových slov `closest-corner` a `farthest-corner` pre elipsový gradient je o niečo komplikovanejšie. Najprv sa podľa daného slova vyberie vzdialenosť od stredného bodu do jeho najbližšieho/najvzdialenejšieho rohu, ktorá bude x-rádus elipsy. Následne sa vypočíta y-rádus na základe pomeru x-rádusu ku tomuto rádiuse. Tento pomer je rovný pomeru rádiusov elipsy v prípade zadania `closest-side/farthest-side`.

⁴<https://pdfbox.apache.org/docs/2.0.12/javadocs/org/apache/pdfbox/pdmodel/common/function/PDFFunctionType3.html>

⁵<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Ďalej je do tejto triedy implementovaná metóda *createTransformForEllipseGradient()*, ktorá vytvorí maticu pre transformáciu kruhového útvaru gradientu na elipsový útvar, keďže v PDF je možné vytvoriť iba kruhové gradienty. Transformácia sa bude skladať z funkcie *scale()* a *translate()*. Ak je x-rádus väčší, tak šírka gradientu sa nezmení a jej výška bude zmenšená o pomer x-rádus/y-rádus a opačne. Pred zmenšením rozmeru gradientu je potrebné posúvať gradient pomocou *translate()* tak, aby sa po zmenšení jeho rozmeru pomocou *scale()* gradient nachádzal na svojom pôvodnom mieste.

Vytvorenie výsledného radiálneho gradientu podľa atribútov triedy má na starosť metóda *createRadialGrad*, ktorá funguje na rovnakom princípe ako metóda *createLinearGrad()* u lineárneho gradientu.

7.4 Implementácia filtrových vlastností

Realizácia filtrových funkcií je implementovaná v triede *Filter*. Je to trieda, ktorá je vytvorená pre uchovanie hodnôt parametrov od zadaných filtrových funkcií a má definované špecifické metódy pre úpravu obrázkov na základe týchto hodnôt. Konštruktor tejto triedy má nasledujúci tvar:

```
Filter(String[] filterType, float invert, float grayscale, float opacity, float brightness)
```

Parameter *filterType* obsahuje jeden alebo viac reťazcov v prípade zadania viac funkcií pri vstupe, pričom každý reťazec reprezentuje názov konkrétnej filtrovej funkcie a bude mať jednu z týchto hodnôt: *brightness*, *opacity*, *invert*, *grayscale*⁶ alebo implicitne (prázdny reťazec). Ostatné parametre konštruktoru reprezentujú hodnoty príslušných filtrových funkcií s rovnakým názvom, pričom *brightness* a *opacity* majú implicitnú hodnotu 1, *invert* a *grayscale* majú hodnotu 0. Typ filtrovej funkcie a jeho hodnotu je možné získať rovnakým spôsobom ako pri transformačných funkciách. Jediný rozdiel je iba v tom, že namiesto vstupnej hodnoty *transform* nahradíme reťazcom *filter*.

Kľúčovou metódou triedy *Filter* je metóda *filterImg()*, ktorá na základe typu filtrovej funkcie reprezentovaného pomocou premennej *filterType* volá príslušnú metódu, ktorá nastavuje grafický efekt na vstupný obrázok podľa jeho filtrovej vlastnosti a vracia výsledný obrázok po úprave. Realizáciu filter funkcie *brightness()* a *opacity()* zaisťuje metóda *setBrightOpacImg()* a realizáciu funkcie *invert()* zaisťuje *invertImg()*. Tieto metódy sú implementované na základe návrhov popísaných v podkapitole 6.4. Pri vstupnej filter vlastnosti s deklaráciou *invert(50%)* nahradí CSS celý obrázok sivou farbou. Tento detail je implementovaný v metóde *invertImg()* tak, že do farebných komponentov *red*, *green*, *blue* každého pixelu na obrázku dosadíme hodnotu 128 (model *rgb(128, 128, 128)* reprezentuje sivú farbu).

V triede *Filter* je implementovaná aj realizácia funkcie *grayscale()* pomocou metódy *grayScaleImg(BufferedImage img)*, ktorá avšak nie je plne funkčná. Funguje iba pri zadaní *grayscale()* s hodnotou 100%.

⁶reprezentuje funkciu *grayscale()*, ktorá je popísaná tu: https://www.w3schools.com/cssref/css3_pr_filter.asp

7.5 Implementácia vlastnosti border-radius

Pre implementáciu vykreslenia elementu ohraničeného rámčekom s okrúhlymi rohmi je tvorená trieda *BorderRadius*, ktorá slúži pre uloženie hodnoty reprezentujúcej x-rádus a y-rádus pre vytvorenie okrúhle rohy. Hodnoty rádiusov každého rohu rámčeka je možné získať pomocou použitím rovnakej funkcie uvedenej v popise implementácie transform vlastnosti. Avšak rozdiel je v tom, že ako posledný parameter tejto funkcie je zadaný jeden z refazcov, ktoré reprezentujú rohy rámčeka: *border-top-left-radius*, *border-top-right-radius*, *border-bottom-left-radius*, *border-bottom-right-radius*. Ak pri vstupe nie sú zadané hodnoty pre všetky rohy, tak rádiusy týchto rohov budú mať implicitnú hodnotu 0. Získané hodnoty rádiusov reprezentované triedou *TermList* sú uložené do triedy *BorderRadius* pomocou volania metódy *setCornerRadius()*. Táto funkcia okrem prevedenia hodnôt v tvare *TermList* do tvaru desatinných čísel pre ich ďalšie použitie tiež nastavuje tieto hodnoty podľa špecifických pravidiel:

- Ak je x-rádus väčší ako hodnota polovice šírky rámčeka, tak do x-rádus je priradená táto hodnota.
- Ak je y-rádus väčší ako hodnota polovice výšky rámčeka, tak do y-rádus je priradená táto hodnota.
- Ak x-rádus aj y-rádus konkrétneho rohu prekročia maximálne dosiahnuteľné hodnoty (polovica šírky pre x a polovica výšky pre y), tak rádus s menšou hodnotou bude mať príslušnú maximálnu hodnotu a druhý rádus bude mať rovnakú hodnotu. Pr.: ľavý roh s x-rádus s hodnotou 150px, y-rádus je 100px, šírka rámčeka je 120px a výška 100px. Keďže x-rádus je väčší ako polovica šírky a y-rádus je väčší ako polovica výšky a zároveň y-rádus je menší ako x-rádus, tak y-rádus bude mať novú hodnotu 50px (polovica výšky) a x-rádus tiež bude mať hodnotu 50px.

Pomocou získaných rádiusov vypočítame jednotlivé body určujúce tvar rámčeka, ktoré boli uvedené v návrhu v podkapitole 6.3. Následne sa podľa týchto bodov postupne vykreslia úsečky tvoriace rámčeka elementu volaním našej funkcie *drawBorderRadius()*. Vzhľadom na to, že každú stranu rámčeka je možné definovať rôznou farbou a šírkou pomocou CSS vlastnosti *border*, táto funkcia pred vykreslením každej úsečky najprv posunie *PDPageContentStream* na počiatkový bod tejto úsečky pomocou metódy *moveto()[3]* a následne nastaví jej šírku a farbu podľa hodnoty CSS vlastnosti *border*. Pre nastavenie šírky sa používa metóda *setLineWidth()[3]* a pre nastavenie farby *setStrokeColor()[3]*. Na konci sa volaním metódy *stroke()[3]* vytvorí výsledná úsečka. Toto riešenie spôsobuje jeden problém, ktorý vzniká v prípade, že je element definovaný s pozadím. Keďže *PDPageContentStream* neustále posúvame na rôzne pozície a vytvoríme rámček po častiach tak na konci procesu vytvorenia rámčeka *PDPageContentStream* iba zachová tvar poslednej úsečky a nie celý rámček. Preto sa vyplnenie pozadia pomocou metódy *fill()[3]* aplikuje iba na poslednú úsečku. Riešenie tohoto problému spočíva v tom, že pred vykreslením výsledného rámčeka vytvoríme ďalší rámček s rovnakým tvarom a ktorý sa bude nachádzať vo vnútri výsledného rámčeka (o 1 pixel do vnútra). Šírku tohoto rámčeka nastavíme hodnotou 1 pixelu a jeho farba bude mať rovnakú hodnotu ako pozadie elementu. Keďže u tohoto rámčeka nie je potrebné nastaviť pre každú úsečku rôzne hodnoty šírky a farby, tak do tohoto rámčeka môžeme vyplniť pozadie volaním metódy *fill()*. Pre nastavenie pozadia používame metódu *setNonStrokeColor()[3]*.

7.6 Rozšírenie implemetácie

V našom projekte sú implementované aj niektoré rozšírenia, ktoré sú mimo rozsahu našej práce. V tejto podkapitole sa postupne budeme zaoberať týmito implementáciami.

Rozšírená implementácia priehľadnosti gradientov a funkcie `repeating-linear-gradient()`

Pre vytvorenie priehľadnosti gradientov na webových stránkach sa používa CSS funkcia `rgba()`⁷ pre definovanie farieb v parametre `<color-stop>` (napr. `linear-gradient(to right, rgba(255,0,0,0), rgba(255,0,0,1))`). Posledná hodnota v tejto funkcii reprezentuje stupeň priehľadnosti a má hodnotu v desatinnom tvare v rozsahu od 0 do 1, pričom 0 reprezentuje úplnú priehľadnosť farby a 1 reprezentuje nulovú priehľadnosť (farba sa nezmení). Túto vlastnosť je možné realizovať tak, že sa vytvoria nové farby, ktoré sú ekvivalentné pôvodným farbám so zadanou priehľadnosťou. Komponenty nových farieb (red_{new} , $green_{new}$, $blue_{new}$) sa vypočítajú nasledovne:

$$\begin{aligned}red_{new} &= red_{old} \times alpha + (1 - alpha) \times red_{background} \\green_{new} &= green_{old} \times alpha + (1 - alpha) \times green_{background} \\blue_{new} &= blue_{old} \times alpha + (1 - alpha) \times blue_{background}\end{aligned}$$

pričom $alpha$ je hodnota priehľadnosti, new je nová farba, old reprezentuje pôvodnú farbu a $background$ reprezentuje farbu na pozadí (v rámci našej práce je táto farba vždy biela, čo znamená, že vykreslenie priehľadnosti funguje iba vtedy, keď je gradient aplikovaný na element bez definovaného iného pozadia). Proces vypočítania nových farieb na základe hodnoty priehľadnosti je implementovaný v metóde `buildType2Functions()` v oboch triedach reprezentujúcich gradienty v našom projekte. Táto metóda ukladá farby gradientov do objektu typu `CosArray`, ktorý sa potom ukladá do výslednej triedy `PDShadingType3`.

Implementácia vykreslenia funkcie `repeating-linear-gradient()`⁸ je zahrnutá v metóde `createColorStopsLength()` v triede `LinearGradient`. Pre realizáciu tejto funkcie je potrebné najprv zistiť dĺžku gradient line, na ktorý sa umiestnia farby gradientu a následne získať hodnotu určujúcu pozíciu poslednej zadanej farby. Dĺžku gradient line získame nasledujúcim spôsobom: $Math.hypot(Math.abs(y_2 - y_1), Math.abs(x_2 - x_1))$, pričom x_1 a y_1 sú súradnice počiatočného bodu, x_2 a y_2 sú súradnice koncového bodu definujúceho gradient line. Podľa uvedenej dĺžky a hodnoty sa môžeme dozvedieť, koľkokrát sa opakuje zadaná postupnosť farieb na gradient line. Na základe toho vytvoríme nový zoznam obsahujúci potrebný počet duplikovaných postupností farieb. Ak pozícia poslednej farby v novom zozname nedosiahne hodnotu 100% (koniec gradient line), tak je potrebné pridať ďalšie farby v postupnosti podľa ich poradia dovtedy, kým nedosiahneme pozíciu 100%. Na konci sa používa nový zoznam farieb a ich príslušných pozícií na gradient line pre vykreslenie výsledného lineárneho gradientu.

Rozšírená implementácia vlastnosti `list-style` na SVG výstup

Pomocou CSS vlastnosti `list-style` je možné nastaviť rôzne vzhľady odrážok zoznamu. Implementácia tejto vlastnosti je uložená do triedy `SVGDOMRenderer` a je rozdelená do dvoch metód - `writeMarkerImage()` a `writeBullet()`.

⁷https://www.w3schools.com/css/css3_gradients.asp

⁸<https://developer.mozilla.org/en-US/docs/Web/CSS/repeating-linear-gradient>

Metóda `writeMarkerImage()` slúži pre vykreslenie odrážok zadaných v tvare obrázka (použitím vlastnosti `list-style-image`). Použitím metódy `getMarkerImage()` z triedy `ListItemBox`⁹ môžeme získať obrázok zadaný v deklarácii vlastnosti `list-style-image`. Následne sa určujú súradnice počiatočného bodu pre vykreslenie tohto obrázku na základe jeho výšky a šírky. Na konci sa ukladá tento obrázok spolu s jeho atribútmi do DOM stromu, podľa ktorého sa na konci programu generujú SVG kódy odpovedajúce jednotlivým uzlom stromu.

Metóda `writeBullet()` umožňuje vykresliť odrážky definované vlastnosťou `list-style-type` s hodnotou `circle`, `disc`, `square` alebo s textom. Ak sú odrážky definované ako geometrické útvary, tak je potrebné vytvoriť SVG element reprezentujúci dané útvary a uložiť do tohto elementu príslušné atribúty pre vykreslenie útvaru. Ak sú odrážky zadané ako text, tak tento text sa dá získať pomocou metódy `getMarkerText()` z triedy `ListItemBox`. Štýl tohto textu je rovnaký ako štýl textu daného zoznamu a je možné ho získať pomocou použitia metód v triede `VisualContext` z knižnice `CSSBox`, ktorá reprezentuje kontext daného zoznamu.

Implementácia `list-style` vlastnosti vychádzala z implementácie danej vlastnosti v triede `SVGRenderer`, ktorá je starou verziou pre generovanie SVG výstupu.

⁹trieda z `CSSBox`, ktorá reprezentuje HTML zoznam.

Kapitola 8

Testovanie

V tejto kapitole sa budeme zaoberať procesom testovania funkcionality našej práce, ktorý prebiehal už počas vytvorenia implementácie. Najprv budeme popisovať proces testovania na vlastných vytvorených HTML súboroch¹, pričom každý súbor slúži pre testovanie jednej z implementovaných CSS3 vlastností. Na konci sa budeme venovať zhodnoteniu procesu testovania a možnostiam rozšírenia tejto práce v budúcnosti.

8.1 Testovanie na vytvorených HTML súboroch

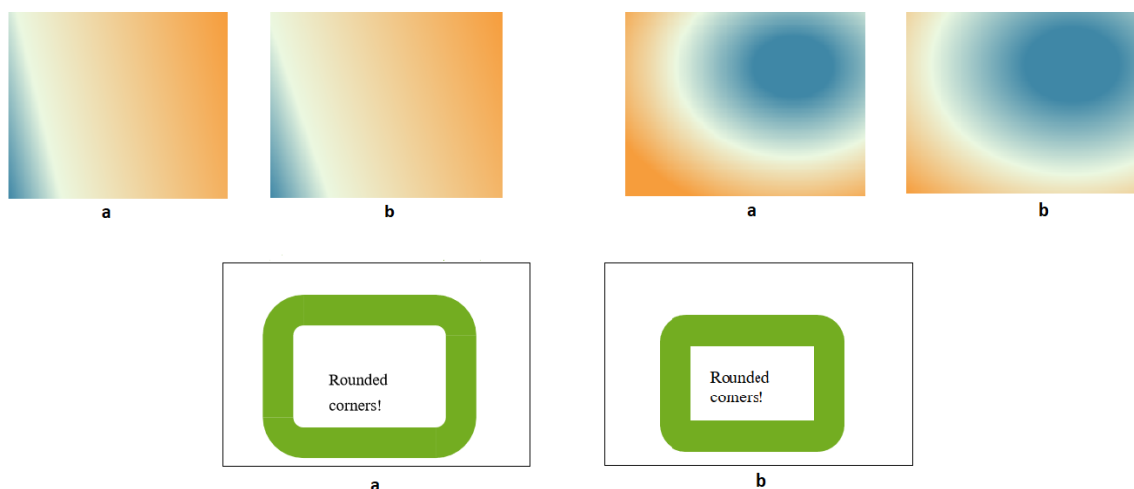
V každom HTML súbore je vytvorených niekoľko HTML elementov, ktoré majú jednoduchý obdĺžnikový tvar a každý z týchto elementov má zadaný určitý typ CSS3 vlastnosti. Testovanie funkcionality našej práce sa zameriava na kontrolu zhodnosti výstupu vo formáte PDF, ktorý bol generovaným pomocou nášho projektu a zobrazený na prehliadači *Chrome*, s HTML výstupom generovaného prehliadačom *Mozilla Firefox*.

Výsledky vykreslenia CSS3 filter funkcií na PDF výstup sa zhodujú s zobrazeniami HTML súborov na prehliadači. Výstupy 2D transformácie, lineárneho gradientu, radiálneho gradientu a border-radius sú v určitých prípadoch o trochu odlišné od HTML výstupov. Transformačné vlastnosti sa zhodujú iba v prípade zadania rozmeru stránky 1200x600. Pri zadaní iného rozmeru stránky dochádza k zlému umiestneniu elementu (trochu sa posunie hore), pričom táto chyba spôsobí projekt CSSBoxPdf pri vykresľovaní elementu na PDF. Výsledok renderovania lineárneho gradientu sa odlišuje od HTML súboru vtedy, ak je zadaný šikmý gradient line (napr. s uhlom 70deg). Je to z dôvodu, že sa nedajú úplne presne vypočítať súradnice počiatočného a koncového bodu podľa návrhu v podkapitole 6.2, avšak výstup realizovaný pomocou tohoto návrhu sa najviac podobá HTML výstupu. Radiálne gradienty v tvare elipsy so zadanými hodnotami rádiusov pomocou kľúčového slova *closest-corner* alebo *farthest-corner* sú v PDF výstupe o trochu menšie ako v HTML súboroch. Vo výstupoch vlastnosti border-radius nastáva rozdiel oproti zobrazeniu HTML v prehliadači vtedy, ak je zadaná hrúbka rámčeka veľkou hodnotou. Jedná sa o zlé umiestnenie rámčeka avšak po mnohých pokusoch sa nepodarilo túto chybu odstrániť. Na základe týchto testovaní počas implementácie prebiehalo mnoho úprav časti kódu a návrhu, avšak nakoniec sa bohužiaľ nepodarilo tieto odlišnosti odstrániť. Všetky uvedené rozdiely je možné vidieť na obrázku 8.1.

Testovanie funkčnosti renderovania CSS3 vlastností by bolo vhodné vykonať aj na webových stránkach. Avšak je veľmi ťažké nájsť webové stránky, ktoré používajú žiadané CSS3

¹tieto súbory sa nachádzajú v zložke *test* na priloženom CD

vlastnosti. Z toho dôvodu boli vybrané iba niektoré stránky W3schools², ktoré popisujú návod na použitie týchto vlastností. Počas testovania sa na týchto stránkach neobjavili žiadne ďalšie nedostatky ohľadom renderovania CSS3 vlastností.



Obr. 8.1: Prehľad odlišnosti medzi vlastnosti zobrazené WebVectorom (a) a vlastnosti zobrazené prehliadačom Mozilla Firefox (b).

8.2 Zhodnotenie testovania a možnosť rozšírenia práce

Okrem uvedených odlišností sa počas výsledného testovania neobjavili skoro žiadne ďalšie nedostatky ohľadom vykreslenia CSS3 vlastností. Pri testovaní na webových stránkach bolo objavených niekoľko nezhôd v rámci umiestnenia elementov a vykreslenia ich obsahu. Avšak tieto nezhody sú väčšinou zapríčinené buď knižnicou CSSBox alebo projektom CSSBoxPdf pri generovaní PDF výstupu.

Renderovanie HTML dokumentu spolu s jeho CSS vlastnosťami je veľmi široký okruh, keďže jazyk CSS je veľmi rozsiahly a stále sa vyvíja. Preto existuje ešte mnoho vlastností, ktoré zatiaľ nie sú podporované v tomto projekte a ktoré by mohli byť v budúcnosti implementované do tohoto projektu. Ďalej by bolo dobré implementovať podporu pre vytvorenie *hyperlink*³ v PDF dokumente alebo vloženie ďalších interaktívnych prvkov ako video, komentáre atď.

²napr. stránka https://www.w3schools.com/css/css3_gradients.asp

³<https://en.wikipedia.org/wiki/Hyperlink>

Kapitola 9

Záver

Cieľom tejto bakalárskej práce bolo doplniť do aplikácie WebVector podporu generovať výstup vo formáte PDF a podporu vykresliť pokročilé CSS3 vlastnosti. Pre generovanie PDF výstupu bol vybraný projekt CSSBoxPDF[9], ktorý dokáže docela presne vykresliť zadané webové stránky až na pár nepresností pri umiestnení elementov. Vykreslenie CSS3 vlastností boli tým pádom implementované do tohoto projektu. Na platforme Java existuje mnoho knižníc, pomocou ktorých sa dá vytvoriť PDF dokument. Preto je potrebné pred vypracovaním projektu vybrať najvhodnejšiu knižnicu. Keďže projekt CSSBoxPdf používa knižnicu Apache PDFBox pre vykreslenie výsledného PDF dokumentu, tak je vhodné túto knižnicu použiť aj pre realizáciu CSS3 vlastností. Z mnohých CSS3 vlastností boli vybrané a implementované tieto: *transform*, *filter*, *border-radius* a gradientové funkcie *linear-gradient()* a *radial-gradient()*. Výsledky renderovania týchto uvedených vlastností sa takmer zhodujú so vzorovými výstupmi generovanými prehliadačom Mozilla Firefox, pričom v niektorých prípadoch vzniknú drobné nepresnosti oproti vzorového výstupu. Prípady, v ktorých tieto nepresnosti nastanú, boli uvedené v podkapitole 8.1.

Behom práce bolo nutné naštudovať mnoho technológií a informácií, ktoré sú potrebné pre vytvorenie výsledného projektu. Konkrétne sa jednalo o vlastnosti a štruktúru dokumentu vo formáte PDF, funkčnosti nástroja CSSBox a jeho súvisiacich projektov, tvorenie PDF dokumentu s využitím knižnice Apache PDFBox a podobne. Keďže knižnica CSSBox je veľmi rozsiahla a súvisí aj s inými knižnicami, tak pre pochopenie jej činnosti a funkcionality musela prebiehať pomerne dlhá doba študovania detailov tejto knižnice. Počas študovania a vypracovania práce som sa dozvedel o mnohých užitočných informáciách o formáte PDF a o jazyku CSS, ktoré by sa dali využiť v budúcnosti.

Literatúra

- [1] Adobe Corp.: *PDF Reference sixth edition*. November 2006, [Online; navštíveno 14.04.2019].
URL https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf
- [2] Andreas Lehmkuhler and his team: *Apache PDFBox*. [Online; navštíveno 22.04.2019].
URL <https://pdfbox.apache.org/index.html>
- [3] Ben Litchfield: *Class PDPageContentStream*. [Online; navštíveno 06.05.2019].
URL <https://pdfbox.apache.org/docs/2.0.8/javadocs/org/apache/pdfbox/pdmodel/PDPageContentStream.html>
- [4] *All About Images*. [Online; navštíveno 12.04.2019].
URL <https://guides.lib.umich.edu/c.php?g=282942&p=1885352>
- [5] *Raster Images vs. Vector Graphics*. [Online; navštíveno 12.04.2019].
URL <https://www.printcnx.com/resources-and-support/additional-resources/raster-images-vs-vector-graphics/>
- [6] Ing. Martin Šafář: *Transformace dokumentů HTML na vektorovou grafiku SVG*. Faculty of Information Technology, VUT v Brně, 2016, [Online; navštíveno 10.03.2019].
URL <http://www.fit.vutbr.cz/study/DP/DP.php.cs?id=18251&file=t>
- [7] Ing. Radek Burget Ph.D: *CSSBox*. Faculty of Information Technology VUT v Brně, [Online; navštíveno 23.04.2019].
URL <http://cssbox.sourceforge.net/manual/>
- [8] Ing. Radek Burget Ph.D. and Karel Piwko and Jan Svercl: *jStyleParser*. Faculty of Information Technology VUT v Brně, [Online; navštíveno 23.04.2019].
URL <http://cssbox.sourceforge.net/jstyleparser/manual.php>
- [9] Ing. Zbyněk Červinka: *Generování PDF dokumentů z webových stránek*. Faculty of Information Technology, VUT v Brně, 2015, [Online; navštíveno 23.04.2019].
URL <http://www.fit.vutbr.cz/study/DP/BP.php.cs?id=17079&file=t>
- [10] iText Group: *iText*. [Online; navštíveno 22.04.2019].
URL <https://itextpdf.com/en>
- [11] Stefano Chizzolini: *PDF Clown*. [Online; navštíveno 22.04.2019].
URL <https://pdfclown.org/overview/>

- [12] W3C Group: CSS Tutorial. [Online; navštíveno 23.04.2019].
URL <https://www.w3schools.com/css/>
- [13] W3C Group: *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. August 2011,
[Online; navštíveno 14.04.2019].
URL <https://www.w3.org/TR/SVG11/>
- [14] W3C Group: *CSS Image Values and Replaced Content Module Level 3*. Apríl 2012,
[Online; navštíveno 27.04.2019].
URL <https://www.w3.org/TR/css3-images/>
- [15] Zoltan Hawryluk: *The CSS3 matrix() Transform for the Mathematically Challenged*.
Január 2011, [Online; navštíveno 25.04.2019].
URL <http://www.useragentman.com/blog/2011/01/07/css3-matrix-transform-for-the-mathematically-challenged/>

Príloha A

Obsah CD

- **zložka source-code** - zdrojové kódy aplikácie WebVector a jej súviciace projekty.
 - **CSSBox**
 - **CSSBoxPdf** - nachádzajú sa tam zdrojové kódy našej práce
 - **CSSBoxSvg**
 - **JStyleParser**
 - **WebVector**
- **zložka test** - nachádzajú sa tam všetky súbory v HTML jazyku, s ktorými bola testovaná výsledná práca.
- **zložka result** - nachádzajú sa tam PDF výstupy generované aplikáciou WebVector s zadaným rozmerom stránky 1200x600.
- **readme.txt** - inštrukcia pre inštaláciu a spustenie práce.