Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Výkon elementu HTML5 Canvas

Bakalářská práce

Autor: Hnik Jaromír

Studijní obor: Aplikovaná informatika

Vedoucí bakalářské práce: Ing. Zdeněk Mlčoch

Hradec Králové Srpen 2016

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 19.08.2016 Jaromír Hnik

Poděkování:

Děkuji tímto vedoucímu bakalářské práce, kterým byl pan Ing. Zdeněk Mlčoch za jeho metodické směrování a užitečné rady při tvorbě této práce.

Anotace

Cílem této práce je nalézt optimální programové konstrukce pro práci s HTML5 Canvas elementem. V této práci je otestováno vykreslování elementů v prvku Canvas v různých alternativách webových prohlížečů a jejich dopad na systém. Dosažené výsledky jsou zhodnoceny a porovnány. Na závěr práce je zhodnocen celkový přínos tohoto prvku do reálného světa.

Annotation

Title: Performance of HTML5 Canvas Element

The aim of this work is to find the optimal design patterns for working with HTML5 Canvas element. In this work is tested the rendering of elements in the Canvas in different web browsers alternatives and their impact on the system. The achieved results are evaluated and compared. At the end of work will be evaluate the overall contribution of this element into the real world.

Obsah

1 Úvod

Jako téma bakalářské práce byla vybrána studie elementu HTML5 Canvas. Toto téma bylo vybráno především pro obeznámení čtenáře s tímto, v dnešní době často využívaným prvkem. Dále pak pro rozbor výkonu tohoto elementu, kterýmž je v dnešní době vysoce ovlivňováno vykreslování v prohlížečích webového obsahu. Zprvu bude čtenář zanesen do počátků a začínajících vývojů prvku HTML a prvků grafických. Bude rozvedeno kde, kdy a proč vlastně se tento prvek začal tak masivně využívat. Čtenář bude dále seznámen s mnohými základními historickými milníky v prvku HTML, přičemž budou rozvedeny především verze HTML, HTML5 a v neposlední řadě element HTML5 Canvas, na který je tato práce zaměřena. Dále je nezbytnou součástí práce zavést čtenáře do problematiky měření výkonu vykreslování ve webových prohlížečích. Budou proto vytvořeny testy, na kterých bude rozvedeno, jaký dopad tento výkon má ve zmiňovaných prohlížečích na systém. Další část bude zaměřena na navrhnutí konstrukcí pro optimalizaci těchto dopadů na celkové zatížení systému. Celá práce bude shrnuta závěrem, kde budou zhodnoceny a porovnány dopady výkonů v jednotlivých prohlížečích na systém, jejich výhody, nevýhody a autorovo zvolení nejlepší konstrukce.

2 Představení a vývoj webových a grafických prvků

Chronologické nahlédnutí do historie vývoje webových struktur. Seznámení s nejdůležitějšími milníky a zásadními změnami.

2.1 Vznik

V počátcích 90. let vznikla myšlenka o sdílení volitelného obsahu koncových zařízení neboli počítačů mezi sebou navzájem. Tuto myšlenku propagoval Tim Berners-Lee, nejvýznamnější postava pro prvopočátky internetu tak, jak jej známe dnes. Tim Berners-Lee se podílel na projektu „WWW“ který se zabýval právě touto komunikací. Prvotní vizí však nebylo propojit zařízení globálně, nýbrž sdílet výsledky výzkumů v CERNu v rámci lokální sítě.

2.2 Prvopočátky struktury HTML

Prvotní návrh již zamýšlel strukturování a členění kódu dle logických funkcí. První verze HTML byla popsána dokumentem HTML Tags, kde se již poprvé objevili prvotní konstrukce pro členění dokumentů, včetně prvotních grafických funkcí a sice vkládání obrázků, grafická úprava textu a práce s grafickými prvky. Prvotní verze také nebyla operačně flexibilní, aby koncový uživatel (v té době pouze člen výzkumu CERNu) mohl tyto principy využívat, bylo nutné být vlastníkem operačního systému NextStep, který umožňoval základní editaci a prohlížení webových stránek. To způsobovalo neznalost HTML struktury pro informační techniky, což by v globálním měřítku nebylo zcela žádoucí.

2.3 Etapní vývoj

Vzhledem k expanznímu využívání tohoto jazyka mimo CERN vzrůstaly také požadavky uživatelů a tak se začala tvořit struktura kódu, jak jej známe dnes. Přibývaly nové prvky s ohledem na zachování kompatibility. Vzniká tak verze HTML 2.0. Tato verze jazyka je rozšířena o práci s formuláři. Vyskytlo se zde jedno z prvních kontrolování výsledného vzhledu webu. Nástavbou na tuto platformu byla verze HTML+, která však nebyla takovým skokem vpřed, jak se očekávalo, neboť poskytovala pouze rozšíření o tabulkové prvky. [1]

2.4 Rozšíření HTML

Mezi první verzi jazyka, která se rozšířila globálně, se dá považovat verze HTML3. Tato verze přinesla mnoho unikátních změn, které zůstaly využívány dodnes. Je však nutno podotknout, že pro tehdejší dobu, byla tato nástavba poměrně složitá. V polovině 90. let totiž neexistoval nikdo, kdo by tuto změnu dokázal implementovat, kromě konsorcia W3C, které tento vývoj následně koordinovalo. Vytvořena byla totiž verze HTML 3.2, která nabízela to, co předchozí verze nikoliv. Tuto verzi podporovalo velké množství prohlížečů webového obsahu a tak bylo umožněno právě požadovanému globálnímu rozšíření. Jedno z nejpodstatnějších rozšíření byla podpora Java appletů. Vznikla tak možnost psát webové stránky dynamicky, což mělo za následek i to, že konsorcium W3C doporučovalo Java applety používat každému tvůrci webových stránek, kvůli následnému zajištění kompatibility napříč globálním spektrem prohlížečů webového obsahu.

2.5 Konec 90. let

Toto období bylo pro současnou podobu webu zřejmě jedno z nejdůležitějších, neboť v roce 1997 podporovala verze HTML 3.2 již tvoření obsahu pomocí rámců a čím dál více se využívalo klientských skriptovacích jazyků jako je JavaScript. Kvůli globálnímu rozšíření se zavedla také podpora vícejazyčných webů. Toto ustanovení vedlo k zavedení standartu HTML 4.0, které de facto zastavilo vývoj jazyka, neboť si nikdo nedokázal představit, co více by mohl potřebovat. V roce 1999 byla vydána opravná verze HTML 4.01, která vývoj HTML jazyka na celých 7 let zastavila. Po tuto nečinnou dobu byl zpracováván standart XHTML, který zůstal až do dnešní doby a byl založen na principech XML, který se okamžitě stal standardním formátem pro ukládání a výměnu dat. Tato technologie se zdála jako ta pravá, pro expanzi vývoje dalších standardů webových stránek. Podporováno v této verzi bylo tisknutí pomocí ořezání „tagů“ a formátování. Opak byl však pravdou, čas a postupná práce informačních techniků s tímto jazykem ukázala, že XHTML nepřináší nijak výrazné vylepšení oproti verzi HTML 4.01. Po tuto dobu se tak využívalo čtvrté verze jazyka a zdálo se, že vše bude stačit na několik let dopředu.

Tehdejší tvoření webových stránek se rozdělovalo do dvou můstků. Jedni, kteří využívali HTML verze 4.01 a druzí, kteří věřili, že v XHTML je potenciál a využívali právě jej. V roce 2007 se hlavní tvůrci těchto dvou skupin (W3C a WHATWG) spojili a začali vyvíjet zcela novou verzi jazyka a sice HTML 5.

2.6 HTML 5

Na začátek je nutno podotknout, že tato verze není dokončena a že je v dnešní době sice často využívána, nikoliv však v plné míře. Celkové dokončení tohoto rozšíření je odhadováno až po roce 2020. V dnešní době mají implementovanou podporu této verze téměř všechny přední webové prohlížeče. HTML5 má již mnoho stabilních bodů, o které se v dnešní době autoři webových stránek již opírají, mnoho funkcí je však zatím čistě experimentálních.

Obr. 1 - Standardizované logo HTML5

2.6.1 Nové konstrukce v HTML5

Obecně HTML5 přináší mnoho radikálních rozšíření. Celá idea této nástavby je přiblížit se vysoce kvalitním programovacím jazykům pomocí konstrukcí. Tímto je myšleno především to, že je třeba odlišovat vstupní typy událostí, jako jsou:

- color, range, search
- date, datetime, datetime-local, month, time, week
- url, tel, email, number

Lze si povšimnout implementace především logických struktur časových událostí a personálních údajů. HTML5 přineslo zároveň nové strukturální elementy, jako jsou article, dialog, mark, aside a mnoho dalších. Dále pak elementy pro média jako audio, video, track a source. [2]

2.7 Grafické prvky

Pro práci s grafikou ve verzi HTML5 se využívá dvou základních elementů, a sice <svg> a <canvas>. Prvkem canvas se bude práce dále zaobírat, je však nutné pro porozumění vykreslování zmínit i druhý prvek, který má také své přednosti.

2.7.1 Element SVG

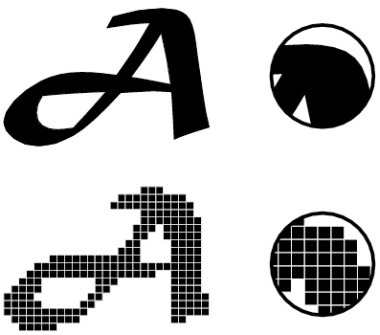
SVG, neboli *Scalable Vector Graphics* vznikl již s používáním struktury XML. Tento element přenáší vektorovou informaci a rozšířil se tak mohutně, že se z něho stal většinový standard pro přenos vektorové grafiky. Tento element nachází využití především při tvorbě uživatelského rozhraní. Je tedy zřejmé, že se konkrétně v HTML struktuře jedná o interaktivní prvek.

2.7.2 Vektorová grafika

Vektorová grafika je dána funkčním předpisem a je spojitá pro všechna měřítka. Znamená to tedy, že pomocí vektorové grafiky lze přenášet geometrické útvary. Výhodou tohoto principu přenášení grafických údajů je především to, že při zvětšování objektu nedochází ke ztrátě detailu, neboli rozostření.

2.7.3 Rastrová grafika

Rastrová grafika je udávána v malých, dále nedělitelných elementech neboli pixelech. Tyto pixely jsou jasně definovány barevným modelem (RGB) a usazeny do mřížky, neboli rastru. Rastrová grafika se využívá pro přenos grafických informací nedefinovaných předpisem, například digitální fotografie, přenos videa apod.



Obr. 2 - Rozdíl mezi vektorovou a rastrovou informací

3 Představení elementu Canvas

V této kapitole bude představeno několik stěžejních vlastností a využití elementu Canvas. Bude vyobrazena podpora prvku v dnešních prohlížečích a kódové ukázky vyobrazení základních grafických elementů do scény.

3.1 Základní specifikace

Jak již bylo zmíněno v předchozí kapitole, element Canvas je jeden ze dvou grafických elementů v notaci HTML5. Jedná se o párový tagový element, který má jasně definované parametry jako je výška a šířka plátna, kde samotné vykreslování probíhá uvnitř něj. Element Canvas pro své vykreslování využívá JavaScript, což umožňuje implementaci jeho metod, díky kterým lze v plátně vykreslit téměř cokoliv. Element Canvas tak oproti SVG elementu využívá primitivních funkcí na bázi JavaScriptu, namísto XML elementů.

„Canvas je element, který umožňuje využívat skripty s rastrovou scénou s určitými parametry, který umožňuje vykreslit grafické prvky, jako například grafy, herní grafiku či složitější obrazce, a to vše v reálném čase.“ [3]

3.2 Vytvoření grafického plátna

Pro vytvoření „plátna“ pro vykreslování prvků je nutno použít v HTML kódu párového elementu <canvas>, jemuž lze přiřadit kromě výšky a šířky plátna také platný identifikátor. Tento identifikátor slouží pro odkazování na vytvořený element z JavaScriptového kódu. V elementu lze využít tzv. kontextu, kde se přímo daný JavaScriptový kód vykonává. [4]

```
<canvas id="grafickePlatno" width="400 height="400">
   Doplňující prvky.
   Případný obsah pro weby nepodporující canvas element.
</canvas>
```

Obr. 3 - *Kód:* Vygenerování kostry grafického plátna

3.3 Práce s kontextem

Zadání identifikátoru samo o sobě je de facto nepodstatné. Potřebujeme-li však pracovat s operacemi, které se mají udávat ve vytvořeném plátně, pak je tento identifikátor nesmírně důležitý. Pomocí jeho reference se odkážeme na konkrétní plátno, kde provedeme vykreslení prvku pomocí zavolání konkrétní vykreslovací či animační funkce. Pro lépe přehledné kódování je však optimální si tento vytvořený element inicializovat do námi vytvořeného kontextu. Celý tento postup je založen na principiálních základech jazyků Java a C++. Přesouváme se tak ze psaní HTML tagů do funkční logiky programování a JavaScriptu. [4]

Vysvětleme si tento princip na následujícím příkladu, kde je vytvořena instance, do níž je zkopírována reference podle platného identifikátoru a bude postavena v prostoru 2D.

```
var element = document.getElementById(´grafickePlatno´);
var vytvorenyKontext = element.getContext(´2d´);
```

Obr. 4 - *Kód:* Vytvoření kontextu pomocí identifikátoru

3.4 Princip okamžité funkce skriptu

Při načítání stránky je cílem okamžité spuštění skriptu. To má na starost obstarat dotázání se na vytvořený Canvas a bez nutnosti dotazovat se serveru na potřebné běžící skripty ho rovnou spustí. Pro tento způsob inicializace slouží funkce, která vypadá a pracuje podobně jako odkazování pomocí tříd a jejich volání v jazyce Java. Pro korektní fungování je volání umístěno přímo do skriptu, kde se kód funkčně vykonává [4].

```
<script type="text/javascript">
    window.onload = function () {
     var element = document.getElementById(´grafickePlatno´);
     var vytvorenyKontext = element.getContext(´2d´);
     // Dodatečná práce s kontextem };
</script>
```

Obr. 5 - *Kód:* Inicializace spuštění po načtení stránky

3.5 Vykreslování prvků

V této podkapitole je představeno pár důležitých základních volání a odkazování se na vykreslení základních grafických prvků. Pro správnou funkci se musí předpokládat, že je již vytvořený Canvas element s parametry a přiřazeným kontextem. Kódy pro toto vykreslování jsou umístěny tam, kde je tento kontext již definován (Dodatečná práce s kontextem) viz Obr. 5.

3.5.1 Vykreslení čar

Pro vykreslení čáry je nutno uvažovat situaci, která odráží reálný svět. Představme si, že je třeba uchopit psací potřebu, přiložit ji na papír do bodu, kde potřebujeme začít čáru kreslit a tahem nepřerušeně kreslit do místa, kde je třeba kreslení ukončit. Přesně na tomto principu funguje i kreslení čar v kontextu uvnitř elementu Canvas. Je nutné nejprve určit počáteční souřadnice inicializací metody *moveTo* a tahem kreslit čáru pomocí metody *lineTo* do koncového bodu, který je rovněž určen souřadnicemi. Následně je nutné přiřadit čáře nějakou určitou barvu. Po vykreslení se čára na plátně zviditelní pomocí příkazu *stroke()*.

```
vytvorenyKontext.moveTo(pocatekX,pocatekY);
vytvorenyKontext.lineTo(konecX,konecY);
vytvorenyKontext.strokeStyle="green";
vytvorenyKontext.stroke();
```

Obr. 6 - *Kód:* Vykreslení čáry v kontextu

3.5.2 Vykreslení obdélníku a čtverce

Stejně jako u čáry je situace obdobná jako v reálném světě. Pro vykreslení obdélníku je využíván příkaz *rect()*. Je definován počátečními souřadnicemi, výškou a šířkou obdélníka. Budeme-li uvažovat situaci, že je potřeba vykreslit čtverec, pak jednoduše nastavíme parametry šířky a výšky na stejné hodnoty.

```
vytvorenyKontext.rect(pocatekX,pocatekY,sirka,vyska);
```

Obr. 7 - *Kód:* Vykreslení obdélníku nebo čtverce

3.5.3 Vykreslení kruhu

Na první pohled zcela jednoduchá situace, kde lze logickou dedukcí uvážit, že kruh má být vykreslen pomocí zadaného počátečního bodu a poloměru. V prvku Canvas však žádný příkaz `circle()` neexistuje. Je tak nutno uvažovat situaci jinou a sice že počáteční bod pomocí souřadnic definujeme i s poloměrem „r“, ale je nutné zadat i doplňující hodnoty. Na toto vykreslení se používá metoda `arc()`, která slouží pro vykreslení oblouku pomocí definovaného počátečního a koncového úhlu. Tato metoda je nakonec ještě doplněna o booleanovou hodnotu, zdali má být úhel vykreslení natočeno v pravém nebo levém směru.

Uvažujme tedy situaci, že je cílem vykreslit kruhový objekt. Ze znalostí základní školy je všem známo, že kruh svírá úhel 360°, nastavíme tedy počáteční úhel na 0° a konečný na 360° pomocí radiánů, takže hodnotu 2π.

```
vytvorenyKontext.arc(stredX,stredY,r,0,2*Math.PI,true);
```

Obr. 8 - *Kód:* Vykreslení kruhu s využitím úhlu

3.5.4 Vykreslení elipsy

Rovněž jako kruh i elipsa nemá žádnou speciální funkci. Vycházejme tak z doposud získaných znalostí. Intuitivně je známo, že pokud vykreslíme kruh a roztáhneme ho, vzniká nám elipsa. Přesně takový postup je nutné zapsat i do kontextu. Uvažujme však, že roztáhnutí neboli změna měřítka podle konstanty pomocí zavolání metody `scale()` se projeví globálně. Musíme tak uložit kontext pomocí zavolání funkce `save()`, po aplikování změn a vykreslení elipsy je žádoucí nastavit zpět defaultní situaci pomocí metody `restore()`.

```
vytvorenyKontext.save();

vytvorenyKontext.scale(kX,kY);

vytvorenyKontext.arc(stredX,stredY,r,0,2*Math.PI,true);

vytvorenyKontext.stroke();

vytvorenyKontext.restore();
```

Obr. 9 - *Kód:* Vykreslení elipsy s uložením a obnovením situace

3.6 Práce s prvky v elementu Canvas

Práce a editace prvků v Canvas elementu nabízí rozšířené možnosti, kdy můžeme vytvořené objekty vybarvovat, vyplňovat barevnými vzory a jiné. Dále pak do tohoto prvku můžeme rovněž vkládat elementy jako obrázky a text. V podkapitole je rovněž ukázána základní práce s editací textu, neboť pouhé vykreslení textu do dokumentu nestačí kvůli nedostatečně specifikujícím parametrům.

3.6.1 Vyplňování prvku barvou

Práce s metodikou je obdobná jako principy v jazyce Java. Nastavením hexadecimální hodnoty je alokována požadovaná barva pomocí nastavením parametru v metodě `fillStyle()`, pod tím uvažujeme hodnoty ve tvaru například 0xFF0058 a zavolání metody `fill()` z kontextu má na starost vyplnění objektu nastavenou barvou.

```
vytvorenyKontext.fillStyle = [0xFF0058];

vytvorenyKontext.fill();
```

Obr. 10 - *Kód:* Vyplnění objektu konstantní barvou

Pozn.: *Pro využití RGB modelu bude metoda využívat předpisu* `fillStyle = 'rgb(257,32,150)'`. *Rozdíl ve výkonu by byl pozorovatelný pouze při využití modelu RGBA, tedy s uvažováním průhlednosti.*

3.6.2 Vyplňování gradientním přechodem

Tento princip využití spočívá v tom, že se mezi zvolenými barvami vytvoří přechod. K tomuto využití slouží metoda `createLinearGradient()` a pomocné metody `addColorStop()`.

```
var grid = vytvorenyKontext.createLinearGradient(0,0,170,0);

grid.addColorStop(0,"black"); grid.addColorStop(1,"white");

vytvorenyKontext.fillStyle = grid;
```

Obr. 11 - *Kód:* Vytvořený gradientní černobílý přechod

3.6.3 Vyplnění kontextu texturou

Budeme-li uvažovat situaci, kdy potřebujeme zajistit vyplnění kontextu souvislým vzorem, neboli texturou musíme uvažovat následovně. Nejprve je nutno reflektovat situaci z reálného světa, neboli stanovení, čím chceme kontext vyplňovat, v tomto případě obrázek. V kontextu na to slouží proměnná „img“, kam je obrázek uložen. Následně z kontextu zavoláme metodu specifickou pro toto uplatnění a sice `createPattern()` kde vstupními parametry bude právě tento obrázek a druhým parametrem bude definice výskytu. Tato definice umožňuje zvolení, zdali je cílem obrázek opakovat ve všech osách, pouze podle jedné, nebo jestli naopak texturu opakovat vůbec nechceme (repeat-y, repeat-x, no-repeat, repeat). Pokud je takto nadefinováno vyplnění, musíme nadefinovat i to, co chceme vyplnit. V tomto případě použijeme již vytvořený kruh, viz kapitola 3.5.3. Následující postup uvažujme obdobně jako při vyplňování barvou. [5]

```
var img = document.getElementById("nazevObrazku");

var pattern = vytvorenyKontext.createPattern(img,"repeat-x");

vytvorenyKontext.arc(stredX,stredY,r,0,2*Math.PI,true);

vytvorenyKontext.fillStyle = pattern;

vytvorenyKontext.fill();
```

Obr. 12 - *Kód:* Namapování textury do vykresleného kruhu

3.6.4 Vkládání obrázku

Princip vkládání obrázků je téměř shodný s principem vkládání čar. Jde tedy o pouhé vložení požadovaného obrázku na určité souřadnice. Předpokladem však je, že do dokumentu musíme tento požadovaný obrázek nahrát. Častou chybou bývá, že se obrázky nezobrazují ihned po načtení stránky. My však z výše nabytých znalostí víme, že pokud využijeme princip okamžitého spuštění skriptu, tento problém bude odladěn. Jako v syntaxi HTML a Java uvažujme, že vytvoříme nový obrázek pomocí metody `new Image()` a tomuto prvku přiřadíme cestu k souboru (požadovanému obrázku). Poté můžeme obrázek vykreslit, avšak v metodě `onload` a pomocí metody `drawImage()`, kde vstupními parametry bude právě vytvořený obrázek, souřadnice x a y, výška a šířka obrázku. [6]

```
var img = new Image();

img.src = "cesta k souboru\obrazek.gif";

img.onload = function () {

vytvorenyKontext.drawImage(img,souradniceX,souradniceY,
pozadovanaSirka,pozadovanaVyska);

}
```

Obr. 13 - *Kód:* Princip vložení obrázku do kontextu

Pozn.: *Načtení obrázku lze docílit i pomocí ID, kde si obrázek vložíme do dokumentu a poté ho pomocí metody* `getElementById()` *přiřadíme do proměnné img. Další postup zůstává neměnný.*

3.6.5 Vkládání textu

Úvodem je nutno zdůraznit pár základních věcí. Vkládání textu do prvku neslouží tak, jak by uživatel očekával. Jde o to, že vložený text se sice jeví jako text, ale ve skutečnosti je to obrázek vložený do kontextu. Zadávání textu probíhá pomocí vložení atributu textového řetězce, metoda `fillText()` si však tento obrázek převede do rastrové grafiky a vykreslí ho, nikoliv vypíše. Upravování se však jeví tak, že se skutečně jedná o text, neboť pomocí příkazu `font` lze nastavit výšku, styl a řež písma. Pomocí již známého příkazu `fillStyle` lze nastavit také barvu. Z tohoto příkazu lze vydedukovat, že se doopravdy jedná o vykreslení prvku nikoliv vypsání, neboť tato metoda se používá u všech předešlých grafických entit.

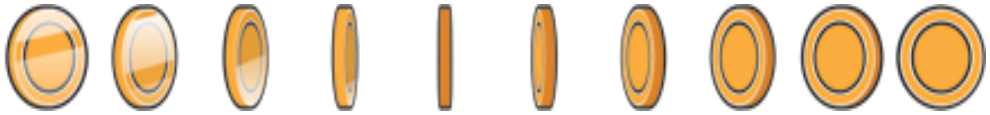
Ukažme si tedy příklad, kdy bude vykreslen text do kontextu, nastavíme mu řez, styl a velikost písma a barvu.

```
vytvorenyKontext.fillText("Testovaci text",x,y);

vytvorenyKontext.font = "bold 18px Times New Roman";

vytvorenyKontext.fillStyle = "0xFF0058";
```

Obr. 14 - *Kód:* Vytvoření textu a jeho nastavení

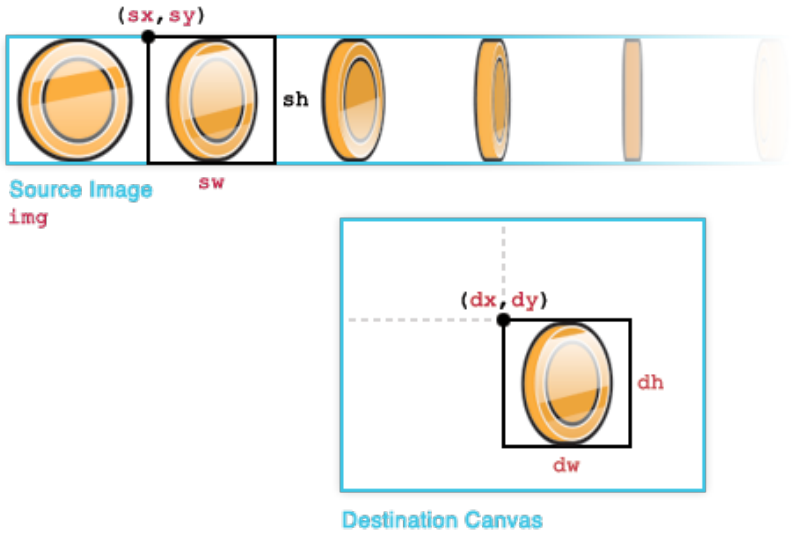
3.7 Animace obrazu v Canvas elementu

Jelikož už známe metody pro přidání obrázků je na místě se zamyslet, jak tyto znalosti implementovat do více použitelných situací. Jelikož celá práce s Canvas elementem spočívá ve využití JavaScriptu není nepředstavitelné vložený obrázek animovat. Princip animace však nespočívá v animování obrázku, ale v matematické úpravě celku. K animování se musí obrázek skládat ze všech pozic, které chceme vyobrazit, viz Obr.15.



Obr. 15 - Celkový obrázek poskládaný z dílčích částí

Na obrázku si lze povšimnout, že nyní již budeme schopni pomocí různých matematických operací poskládat animaci tak, aby byla vykreslována vždy jen určitá část obrázku. Je tak nutno vykreslit obrázek vždy pouze v určité části pomocí metody `drawImage(img, sx, sy, sw, sh, dx, dy, dw, dh)`, označení „s“ nám signalizuje source a „d“ destination. Tyto rozměry si lze uvědomit pomocí přiloženého obrázku.



Obr. 16 - Znázornění získání rozměrů do metody drawImage()

Tímto funkčním principem lze dosáhnout ořezávání obrázku dle vymezených hranic. Princip animace tedy probíhá ve smyčce, kde se bude Canvas neustále resetovat a překreslovat vymezenými hranicemi, tím bude dosáhnuto efektu animace. [7].

4 Metodika testování a hypotézy

Cílem této práce je vyhodnocení dopadu výkonu elementu Canvas na systém. Jeho výkon bude měřen a vyhodnocován. Základními prvky, kterými se práce bude zabývat je měření výkonu v počtu zobrazovaných snímků za sekundu, dále pak dopad zatížení na operační paměť a procesor. Testování výkonu proběhne v několika různých hladinách odhadovaného zatížení, a sice vykreslení primitivních objektů, textů, obrazů a animace. Tento výkon bude porovnán mezi známými prohlížeči webového obsahu. Metodika optimalizace nastíní, kde a jak by se dalo využít optimalizačních konstrukcí.

4.1 Stanovení hypotéz

Z předešlého odstavce lze usoudit, co budeme testovat a porovnávat. Stanovme tedy předpoklady pro testování, které se práce bude pokoušet potvrdit.

- Rozdíl testování výkonu v jednotlivých prohlížečích bude velmi znatelný, předpokladem je, že nejlepší by měl být Google Chrome a nejhorší Microsoft Edge.
- Vykreslování textů by mělo být více náročné než vykreslení obrázku, neboť dle získaných znalostí víme, že text se sice převádí na rastrový obraz, ale fonty jsou nativně vektorové, tudíž musí dojít k rasterizaci.
- Rozdíl dopadu výkonu by se měl velmi lišit podle HW sestavy, na které bude výkon testován (stolní PC, notebook střední třídy a základní kancelářský notebook).
- S výsledky testů by mělo být možno potvrdit teoretické znalosti, které v práci byly zmíněny.
- Vykreslení animací pomocí Canvas elementu by mělo být optimalizováno pro prohlížeče tak, aby se hranice vykreslovaných snímků za vteřinu pohybovala okolo 60, což je maximum, co dokáže lidské oko zpracovat.

5 Vytvoření testů a Canvas pravidla

Vytváření testů je prováděno pro různé webové prohlížeče pomocí vytvoření statické stránky v HTML formě, která bude do konzole vypisovat počet snímků za vteřinu. Důležité je sledování dopadu výkonu na systém. Toho je dosáhnuto opakováním testu v cyklu 5000 opakování. Testy jsou sestavné tak, aby bylo rozpoznatelné, který princip vykreslování má největší dopad na zátěž systému a který dosáhne nejvyšší snímkové frekvence. Zátěž operační paměti a procesoru je měřena přes monitorování aktivit systému.

5.1 Vytvoření testu pro vykreslení čáry

S využitím předchozích znalostí z teoretické části jsme již schopni sestavit jednoduchý kód, který umožňuje v elementu Canvas vykreslit čáru. Pro různá zatížení upravíme metodu *drawLine()* o parametr *sirka*, který umožnuje definovat tloušťku čáry.

```
<canvas id="grafPlatno" width="400" height="400"></canvas>
<script>
  var element = document.getElementById('grafPlatno');
  function drawLine(sirka){
   var vytvorenyKontext = element.getContext('2d');
   vytvorenyKontext.lineWidth = sirka;
   vytvorenyKontext.strokeStyle = "green";
   vytvorenyKontext.beginPath();
   vytvorenyKontext.moveTo(100, 90);
   vytvorenyKontext.lineTo(200, 90);
   vytvorenyKontext.stroke();
   vytvorenyKontext.drawLine(10); }
</script>
```

Obr. 17 - *Kód:* Příprava kódu pro vykreslení čáry s parametrem šířky

5.2 Vytvoření testu pro vypsání textu

Před vytvořením testu je nutné zopakovat, že vytváření textu v Canvas elementu je dosti diskutabilní, neboť nejde o vytvoření textového řetězce, ale o vygenerování obrazu s textovým řetězcem. Na příkladu je rovněž možno demonstrovat obarvení textu gradientním vyplněním.

```
<canvas id="grafPlatno" width="400" height="400"></canvas>
<script>
  var element = document.getElementById('grafPlatno');
  function fillText(text, x, y){
   var vytvorenyKontext = element.getContext('2d');
   vytvorenyKontext.font = "18px Times New Roman";
   var gradient = vytvorenyKontext.createLinearGradient(0,0,
   element.width,0);
   gradient.addColorStop("0","blue");
   gradient.addColorStop("0.5","red");
   gradient.addColorStop("1.0","black");
   vytvorenyKontext.fillStyle = gradient;
   vytvorenyKontext.fillText("Testovaci Text",10,10);
   }
</script>
```

Obr. 18 - *Kód:* Příprava kódu pro vypsání textu

Pro představu si ukažme, jak vytvořený text v kódu výše popsaném bude vypadat s využitím gradientního vyplnění.

Obr. 19 - Vytvořený testovací text s gradientní výplní

5.3 Vytvoření testu pro vykreslení obrázku

Pro vykreslování obrázku bude zhotoveno více testů, kvůli demonstraci jednotlivých úskalí a náročností metody. Porovnáno bude základní vykreslení objektu, transformace objektu a v neposlední řadě vykreslení externího obrázku s dostatečnou složitostí, kde je demonstrováno, jak užitečné je využívat externího zdroje.

5.3.1 Vytvoření testu vykreslení základního objektu

Pro zachycení rozdílů výkonů zvolíme nejprve triviální algoritmus pro vykreslení pomocí metody *drawImage()*.

```
<canvas id="grafPlatno" width="400" height="400"></canvas>
<script>
  var element = document.getElementById('grafPlatno');
  function drawImage(img, x, y){
    var vytvorenyCanvas = document.createElement('canvas');
    var vytvorenyKontext = element.getContext('2d');
    vytvorenyCanvas.width = 360;
    vytvorenyCanvas.height = 360;
    vytvorenyKontext.fillStyle = 'green';
    vytvorenyKontext.fillRect(10, 10, 200, 200);
    var img = new Image();
    img.src = vytvorenyCanvas.toDataUrl();
    vytvorenyKontext.drawImage(img, 10, 10);
  }
</script>
```

Obr. 20 - *Kód:* Vytvoření testu pro triviální vykreslení objektu

***Pozn.**: U tohoto konkrétního příkladu budeme metodu `drawImage()` volat pouze se třemi parametry, a sice obrázkem a souřadnicemi x a y, na nichž chceme obrázek vykreslit.*

Nyní vytvoříme obdobnou metodu, avšak s námi již známými atributy, které byli objasněny v teoretické části. Následující upravená metoda je rozšířena pouze o parametry výška a šířka, kterými můžeme objekt předefinovat na cíleně požadované rozměry.

```
function drawImage(img, x, y, sirka, vyska){
      Stejný kód jako v předchozím příkladu.
}
```

Obr. 21 - *Kód:* Úprava metody drawImage()

5.3.2 Vytvoření testu transformace objektu

Abychom mohli obrázek editovat, konkrétně zvětšovat nebo zmenšovat, musíme předpis funkce `drawImage()` opět předefinovat a sice doplněním o vstupní parametry zvětšení šířky a výšky. Ty v osách „x“ nebo „y“ násobí původně definovaný obrázek.

```
<canvas id="grafPlatno" width="400" height="400"></canvas>
<image id="image" width="400" height="400" src="odkaz">
<script>
   var element = document.getElementById('grafPlatno');
   var vytvorenyKontext = element.getContext('2d');
  function drawImage(img, x, y, sirka, vyska, sirkaX, vyskaX){
   var img = document.getElementById('image');
   sirka = img.width * sirkaX;
   vyska = img.height * vyskaX;
   vytvorenyKontext.drawImage(img, 10, 10, 200, 200, 2, 2);
  }
</script>
```

Obr. 22 - *Kód:* Vytvoření testu pro transformaci obrázku

Ačkoliv byl v předchozí kapitole vytvořen objekt dvěma různými metodami, stále se jednalo o obrázek vytvoření nějakým funkčním předpisem, neboli zavoláním určité metody, konkrétně `fillRect()`. Z toho důvodu by rozdíl výkonu nebyl příliš znatelný a právě z toho důvodu byl nyní použit obrázek dostupný přes externí odkaz. Rovněž bylo demonstrováno alternativní přidání obrázku pomocí HTML syntaxe, která se také dodržuje.

***Pozn**.: Pro docílení transformace je nutno uvažovat matematických znalostí. A sice, že zavolání metody* `drawImage(img, x, y, sirka, vyska, 2, 2)` *vytvoří dvakráte zvětšený obrázek apod.*

5.4 Vytvoření testu pro vykreslení animace

Vytvoření testu pro znázornění animace může mít dvojí průběh. Zaprvé bude znázorněno, jak vytvořit animaci pomocí parametrově statického zadání objektu a zadruhé pomocí předdefinovaného objektu a posouváním zobrazovaného celku *(viz. 3.7)*.

5.4.1 Vykreslení animace objektu definovaného pomocí předpisu

Tento princip využívá matematického zadání objektu a implementuje metodu, která umožňuje pomocí cyklu docílit efektu animace tohoto objektu. Metoda, kterou tento princip využívá se nazývá `requestAnimationFrame()`. Tato metoda se volá přímo na `window`, aneb vytvořené okno, ve kterém je požadováno animace dosáhnout. Tato metoda však sama o sobě nepostačuje. Je nutno také uvažovat, že tento proces je žádoucí opakovat. Aby bylo fungování korektní, vytváříme i metodu `animace()`, která nám zajistí průběh animace v opakovacím intervalu. Tuto metodu je důležité implementovat tak, aby obsluhovala rychlost animace, pomocí využívání současného času a jeho přinásobením s konstantou rychlosti. Dále pak řešení hranic plátna, aby nedošlo k animaci takové, kdy objekt „vyletí“ z okna a nebude již znatelné, že animace stále probíhá. Tím by se zatěžoval systém bez ohledu na to, že uživatel by už nepoznal, že proces stále běží.

***Pozn**.: Pro zjednodušení je využito již nabitých znalostí. Animaci pomocí tohoto principu lze provádět s jakýmkoli objektem, který je popsán a zadán staticky. Vytvoření objektu, se kterým lze uvažovat právě takovouto animaci je popsáno v kapitole 3.5.*

```
<canvas id="grafPlatno" width="400" height="400"></canvas>
<script>
  var element = document.getElementById('grafPlatno');
  var vytvorenyKontext = element.getContext('2d');
  var cas = new Date().getTime();
   function animace () {
     vytvorenyKontext.clearRect(0, 0, 400, 400);
     var delta = (new Date()).getTime() - cas;
     var rychlost = 10;
     var noveX = (rychlost * novyCas) / konstanta;
     vytvorenyKontext.fillRect(noveX, 100, 30, 30);
     window.requestAnimationFrame(animace);
   }
   window.requestAnimationFrame(animace);
</script>
```

Obr. 23 - *Kód:* Princip vytvoření animace objektu zadaného předpisem

5.4.2 Vykreslení animace objektu stávajícího se z obrázku

Pro vykreslení animace pevně daného objektu, který má určité grafické zpracování musíme uvažovat, že jelikož se objekt nedá matematicky popsat, potřebujeme všechna grafická vyobrazení natočení tak, aby animace vypadala věruhodně. Využito je nabitých znalostí z kapitoly 3.7 a tento princip je v kódu níže implementován.

Pro lepší pochopení si ukažme, jak bude animace probíhat, a sice, že je přepínáno zobrazení mezi čísly, kde se ve výsledku bude zobrazovat pouze 1 výřez daného čtverce s číslem.

Obr. 24 - Připravený obrázek pro výřezy a dosáhnutí efektu animace

```
<canvas id="grafPlatno" width="400" height="400"></canvas>
<script>
 var element = document.getElementById('grafPlatno');
 var img = new Image();
 img.src = "cesta k souboru\ctverce_cisla.gif";
 function sprite(objekt){
  var pom = {}, index = 0, pocet = 0, delka = delka || 0;
  pom.context = objekt.context;
  pom.image = objekt.image;
  pom.width = objekt.width;
  pom.height = objekt.height;
  pom.loop = objekt.loop;
  pom.update = function(){
   počet += 1;
    if (počet > delka){
     pocet = 0;
     if (index < pocetSnimku - 1) {
      index += 1;
    } else {
     index = 0;
 }}};
  pom.render = function(){
   pom.context.drawImage(pom.image,
    index * pom.width / pocetSnimku, 0,
    pom.width / pocetSnimku,  pom.height, 0, 0,
    pom.width / pocetSnimku, pom.height)};
 return pom;
 }
```

```
 var vysledek = sprite({
  vytvorenyKontext = element.getContext(´2d´);
  width = height / velikost;
  image = img;
 });

 function vyslednaAnimace(){
  window.requestAnimationFrame(vyslednaAminace);
  vysledek.update();
  vysledek.render();
}
</script>
```

Obr. 25 - *Kód:* Celkové sestavení animace pro posun v obrázku

Obr. 26 - Výsledné zobrazení jednoho výřezu v obrázku

Celá animace se tedy stává z přepočítání výřezů obrázku pro jednotlivé části pomocí metody `update()` a jejich vykreslování pomocí metody `render()`. Je nutno uvažovat, že se jedná o čtvercový výřez, tudíž toto překreslování není vidět, pokud by se však jednalo o složitější rotace, jako případ kulatých mincí z kapitoly 3.7, bylo by optimální před každým renderováním plátno mazat pomocí metody `clear()`.

5.5 Vyrovnávací paměť Cache

V oblasti počítačové informatiky je paměť cache nepostradatelnou věcí. Jde o hardwarovou nebo softwarovou realizaci podpůrné paměti, která umožňuje uchovávat data pro rychlejší pozdější přístup. Vzhledem k zaměření práce se zabýváme optimalizací načítání webového obsahu, neboli webovou cache. Typické chování webové cache je, že ukládá předchozí odpovědi ze serverů. Tím je myšleno, že se ukládají celé načtené stránky nebo grafické objekty. Výhodou „cachování" je především to, že na straně klienta nedochází k načítání většího množství dat a na starších strojích může být rozdíl v načítání vysoce znatelný. Na straně serveru pak, že nedochází ke zbytečně veliké zátěži. Konkrétní příklad užití webové cache je proxy server, který cachuje data ze serveru směrem ke klientovi. V dnešní době si nativně cache zpracovává každý prohlížeč sám, jsou však případy, kdy stránka nemůže být cachována a je tak třeba umožnit cachování každému návštěvníkovi.

Zobrazený kód udává defaultní čas, po kterém se cachované soubory vymažou z uživatelské cache. Konkrétní podtypy souborů (css, jpg, png, ...) mají nastavenou speciální dobu, po kterou budou uložena. V tomto případě je to 2592000, což značí dobu ve vteřinách, a sice jeden měsíc. [8]

5.5.1 Cachování obrázků pomocí CSS

Další alternativou, jak cachovat soubory (v tomto případě grafické prvky a obrázky) je cachování pouze pomocí CSS kaskádových stylů, tzn. bez využití JavaScriptu. Výhodou tohoto řešení je, že mnoho prohlížečů nestahuje obrázky, které jsou nativně skryty pomocí metod *visibility:hidden* nebo *display:none*. Další problém se naskýtá na stránkách optimalizovaných k tisku. Vzhledem ke cachování se potřebujeme tomuto vyhnout a přepsat tak metody přes „override", kvůli předejití situace, že prvek nebude nacachován. Je také optimální každý prvek načíst rovnou, ale s rozměry například 1x1.

```
<div id="cache_pamet">
 <img src="zdroj/obrazek1" width="1" height="1" />
 <img src="zdroj/obrazek2" width="1" height="1" />
</div>
```

Obr. 27 - *Kód:* Vytvoření grafických prvků se zanedbatelnými rozměry

Za předpokladu, že takto vytvoříme prvky kvůli načítání, můžeme pomocí CSS upravovat jejich chování a především přepsat metody `media screen` a `media print`.

```
@media screen {
 div #cache_pamet {
  position: absolute;
  top: -9999px;
  left: -9999px;
 }
 div #cache_pamet img{
  display: block;
 }}
@media print {
 div #cache_pamet, cache_pamet img {
  visibility: hidden;
  display: none;
 }
}
```

Obr. 28 - *Kód:* Override metod media screen a print kvůli cache

Tento způsob umožní načtení všech obrázků, které potřebujeme. Následnými úpravami lze s obrázkem manipulovat, transformovat ho a následně zobrazit, jak je potřeba.

5.5.2 Paměťová optimalizace Flyweight

Kromě paměti Cache lze optimalizovat výkon a paměťovou náročnost i objektem Flyweight. Jeho funkcí je sdílení paměti pro podobné objekty. Typické využití lze rozpoznat při využití opakovaného vzoru. Tento objekt se postará o to, že objekt bude načten do paměti pouze jednou a matematickými operacemi bude opakován či modifikován. Praktický příklad lze vysvětlit při použití písem. Namísto paměťového zatížení každého znaku pro daný dokument lze využít funkčního předpisu a uchovávat tak pouze informace jako je samotný znak a jeho pozice. To poskytuje při rozsáhlém dokumentu úsporu několika tisíců bytů pro konkrétní znak. Tento algoritmus lze aplikovat téměř v každém programovacím jazyce. [9]

***Pozn.**: Typickým využitím v problematice vykreslování do Canvasu lze uvažovat zjednodušení kalkulace barevných přechodů pomocí předpřipravených gradientů, neboť pokud budou tyto data načteny v paměti ušetří se vypočítávání pro každý konkrétní přechod mezi počtem požadovaných barev.*

```
public class UrcityObjekt {
    private string _objekt;
    public UrcityObjekt(string objekt) {
        _objekt = objekt;
    }
    public string Objekt {
        get { return _objekt; }
    }
    public override bool Equals(object obj) {
        if (ReferenceEquals(null, obj)) return false;
        return obj is UrcityObjekt && Equals((UrcityObjekt)obj);
    }
    public bool Equals(UrcityObjekt jiny) {
        return string.Equals(_objekt, jiny._objekt);
    }
    public override int GetHashCode() {
        return (_objekt != null ? _objekt.GetHashCode() : 0);
    }
    public static bool operator ==(UrcityObjekt a,UrcityObjekt b)
    {return Equals(a, b); }
    public static bool operator !=(UrcityObjekt a,UrcityObjekt b)
    {return  !Equals(a, b; }
}
```

Obr. 29 - *Kód:* Ukázka implementace Flyweight v C#

5.6 Dynamické vykreslování na Canvas

Dalším ulehčením práce s prvkem Canvas je využití dynamického překreslování. Pokud uvažujeme situaci, kdy máme Canvas plátno, na kterém se mění jen nějaká část, pak lze využít vykreslování Canvasu na Canvas. Tento princip umožňuje vykreslit Canvas na Canvas jako obrázek a ulehčit tak paměťovou náročnost. Nejprve je nutné stávající Canvas uložit jako URL data, která se překódují na PNG formát pomocí metody `toDataURL()` a následovně jeho načtení na jiný Canvas.

```
var dataURL = canvas.toDataURL();

document.getElementById('canvasImg').src = dataURL;
```

Obr. 30 - *Kód:* Ukázka načtení Canvas jako obrázku

5.7 Přemazávání plochy Canvas

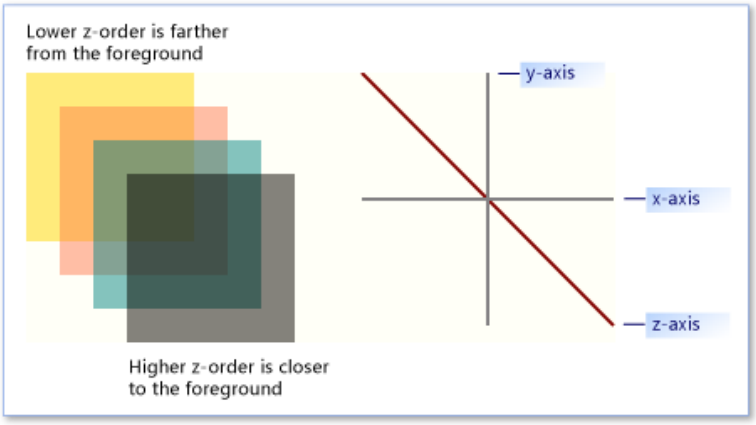
V rámci optimalizace je nutno uvážit rovněž logiku úspory samotným tvořením elementů v Canvasu. Na tvorbě určitých elementů není kromě snahy tvořit efektivně moc co ušetřit, avšak naskýtá se otázka, jak být efektivní v opakovaném vykreslování. Obecné vymazání kreslícího pole se dá dosáhnout pomocí zavolání funkce metody `clearRect(x, y, sirka_Platna, vyska_Platna)`. Ze znalostí však víme, že lidské oko je schopné vnímat pouze 60 obrázků za vteřinu nám vzniká problém při rychlých animacích, neboť na 1 snímek náleží 16 milisekund. Tím pádem nám při plynulé animaci vzniká možnost vnímání přemazání celé plochy.

Mnohem efektivnější je tedy uvažovat situaci, že se bude přemazávat pouze určitý výřez plátna, neboť většinou (záleží na konkrétním případě) se plátno stává z dynamických ale i statických částí. Stejnou metodou se pak dá dosáhnout vymazání právě zmíněného výřezu, zadáme-li jeho x a y souřadnici a výšku a šířku plochy, kterou budeme chtít vymazat. Toto uvažování přináší velikou optimalizaci pro systém a snížení celkového zatížení. Lze to pozorovat na zvýšení FPS (obrázky za vteřinu).

Dále je důležité zmínit, že přemazávání po částech je daleko optimálnější z hlediska toho, že mazání celé canvas vrstvy obstarává grafická karta a už tento dotaz na externí operaci celý proces zpomaluje.

5.8 Využití Z-ové souřadnice v Canvasu

Mezi další ulehčení tvorby s Canvas elementem lze považovat využití Z-indexu. Jde o to, že se nachází několik vrstev Canvas nad sebou a řadí se podle integerové hodnoty Z-indexu, přičemž „0“ je defaultní hodnota, pokud je hodnota vyšší, znamená to, že objekt (prvek vytvořený v Canvasu) je blíže k pozorovateli. Konkrétní hodnota se nastavuje (v jazyce XAML) do proměnné `Canvas.ZIndex`. Do této proměnné lze přidávat hodnoty v kladném i záporném rozsahu celých čísel, podle požadovaného umístění na celkovém plátně. V jazyce JavaScript se využívá stejného principu, avšak tato hodnota se nastavuje ve tvaru `hodnota = object["Canvas.ZIndex"].`



Obr. 31 - Znázornění Z-indexu v Canvas plátně

***Pozn.**: Pro Obr.31 je přiložen následující kód, který obstarává toto celkové vyobrazení. [10]*

```
<Canvas Canvas.Left="200">
  <Rectangle Canvas.ZIndex="2"  Fill="Maroon" Canvas.Top="20"
   Canvas.Left="20" Height="100" Width="100" />
  <Rectangle Canvas.ZIndex="1" Fill="LightBlue" Canvas.Top="40"
   Canvas.Left="40" Height="100" Width="100" />
  <Rectangle Canvas.ZIndex="0" Fill="Teal" Canvas.Top="60"
   Canvas.Left="60" Height="100" Width="100" />
</Canvas>
```

Obr. 32 - *Kód:* Použití Z-indexu v jazyce XAML

6 Testování prvku Canvas v konfiguracích

V předchozí kapitole bylo vytvořeno několik testů díky kterým bude vytvářeno několik posuzovacích hodnot. Záměrem této kapitoly je, aby byly jednotlivé testy provedeny v několika konfiguracích, co se výkonu počítače týče a zároveň pro každou sestavu provést testy pro různé prohlížeče webového obsahu.

6.1 Konfigurace počítačových sestav

Pro docílení toho, aby byly rozdíly testů znatelnější budou použity tři počítače. Prvním z nich bude stolní sestava s osmi-jádrovým procesorem výrobce AMD, taktovaným na 3,8 GHz a externí grafickou kartou nVidia, která poskytuje 1 GB vlastní operační paměti a zároveň další 4 GB paměti sdílené. Stolní počítač rovněž disponuje operační paměti (RAM) o velikosti 12 GB.

Druhým použitým počítačem bude přenosný notebook HP Pavilion, který je určený pro nenáročné kancelářské práce a disponuje procesorem Intel i3, integrovaným grafickým čipem a 4 GB operační paměti.

Posledním použitým počítačem bude MacBook Pro od firmy Apple, označovaným jakožto vysoce výkonným počítačem pro složitější práci. Tento model disponuje procesorem Intel i5, taktovaným na 2,7 GHz, integrovaným grafickým čipem s dedikovanou operační pamětí o velikosti 1,536 GB a 8 GB vlastní operační paměti.

6.2 Použití webových prohlížečů

V dnešní době je několik zástupců webových prohlížečů. Na počítačových sestavách budou testy vykonávány na pěti jejich nejznámějšími a nejrozšířenějšími zástupci. Mezi tyto zástupce patří Google Chrome, Mozilla Firefox, Opera, Safari a Microsoft Edge.

Google Chrome je považován za nejlepší webový prohlížeč, avšak v poslední době se prý potýká s problémem využití operační paměti. Mozilla Firefox a Opera by měla být v testování zhruba na stejné úrovni. Safari je prohlížeč od Apple, tudíž se předpokládá nejefektivnější fungování na MacBooku a naposledy Microsoft Edge, který si firma Microsoft chválí, že dokáže být stejně praktický, jako jeho konkurence..

6.3 Princip optimálního testování

Do testů, které již byli vytvořeny musíme implementovat rovněž korektní logiku toho, abychom byli schopni optimisticky posoudit jednotlivé rozdíly. Jak již bylo zmíněno, testy jsou prováděny v cyklech. To má za účel jediné, a sice, že pokud na nejsilnější konfiguraci budeme vykreslovat tolikrát, aby nebyla překročena hranice 60 snímků za vteřinu, pak lze teprve tyto naměřené hodnoty porovnávat.

Zároveň musíme uvažovat korektní algoritmus pro vypočítání snímků za vteřinu. Znalosti, které musíme uvažovat vycházejí z předpisu, že počet snímků za vteřinu se rovná 1 / (rozdíl času / 1000 ms). V našem případě je implementace tohoto algoritmu `1 / ((Date.now() - startTime) / 1000))`. Tato implementace nám umožňuje korektně zachytit počet vykreslovaných snímků v celém opakování.

Musíme si také uvědomit fakt, že zatěžování procesoru není optimální snímat jako maximální zatížení v momentu spuštění. Vytížení procesoru v plném rozsahu na 100% není skutečné zatížení, které vzniká díky vykreslování do Canvas elementu. Je proto snímáno zatížení, které způsobuje proces webového prohlížeče a zapisovány jsou tedy hodnoty, které z této části procesor zatěžují.

Zatěžování operační paměti se liší podle testovaného webového prohlížeče. Některé prohlížeče si spouštějí vlastní procesy pro jednotlivá okna, naopak některé prohlížeče pracují jako jeden proces. Mezi první zmíněná patří prohlížeče Google Chrome, Opera a Safari. Naopak ty, které pracují jakožto jeden proces jsou Mozilla Firefox a Microsoft Edge.

6.4 Provádění testů

V táto části jsou strukturovány výsledky testů pro jednotlivé sekce, které byly zmíněny výše. Výsledky jsou zapisovány do přehledných tabulek pro každou počítačovou konfiguraci, kde je snadno odlišitelné, co která položka znamená.

6.4.1 Zátěž systému při vykreslování čar

STOLNÍ POČÍTAČ	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	53%	37%	47%	/	58%
RAM	53,2 MB	53,3 MB	140,3 MB	/	55,3 MB
FPS | %	**19,05** / *32%*	**59,88** / *100%*	**36,60** / *61%*	/	**22,95** / *38%*

Obr. 33 - *Test*: Vykreslení čar pro Stolní PC (5 milionů px)

HP PAVILION	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	44 %	29 %	30 %	/	49 %
RAM	61,9 MB	54,2 MB	156,3 MB	/	47,3 MB
FPS | %	**13,32** / *26%*	**51,22** / *100%*	**26,04** / *50%*	/	**15,50** / *30%*

Obr. 34 - *Test*: Vykreslení čar pro HP Pavilion (5 milionů px)

MACBOOK PRO	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	26 %	/	21 %	15 %	32 %
RAM	41,9 MB	/	239,9 MB	40,2 MB	74,2 MB
FPS | %	**44,94** / *76%*	/	**32,19** / *54%*	**59,24** / *100%*	**44,23** / *74%*

Obr. 35 - *Test*: Vykreslení čar pro MacBook Pro (5 milionů px)

6.4.2 Zátěž systému při vykreslování textu

STOLNÍ POČÍTAČ	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	38 %	32 %	35 %	/	41 %
RAM	81,8 MB	131,2 MB	156,8 MB	/	56,3 MB
FPS | %	**14,52** / *96%*	**0,91** / *6%*	**4,77** / *32%*	/	**15,08** / *100%*

Obr. 36 - *Test*: Vykreslení textu pro Stolní PC (14 znaků, velikost 18 px)

HP PAVILION	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	38 %	27 %	33 %	/	34 %
RAM	72,3 MB	74,2 MB	153,9 MB	/	46,6 MB
FPS | %	**12,11** / *96%*	**0,83** / *6%*	**4,84** / *38%*	/	**12,63** / *100%*

Obr. 37 - *Test*: Vykreslení textu pro HP Pavilion (14 znaků, velikost 18 px)

MACBOOK PRO	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	28 %	/	24 %	35 %	42 %
RAM	36,3 MB	/	241,9 MB	26,2 MB	74,2 MB
FPS | %	**27,97** / *95%*	/	**15,49** / *52%*	**0,23** / *1%*	**29,58** / *100%*

Obr. 38 - *Test*: Vykreslení textu pro MacBook Pro (14 znaků, velikost 18 px)

6.4.3 Zátěž systému při vykreslování základního objektu

STOLNÍ POČÍTAČ	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	42 %	23 %	32 %	/	39 %
RAM	168,1 MB	69,2 MB	145,3 MB	/	145,8 MB
FPS | %	**11,99** / *99%*	**10,63** / *88%*	**11,29** / *94%*	/	**12,03** / *100%*

Obr. 39 - *Test*: Vykreslení základního objektu pro Stolní PC (5 milionů px)

HP PAVILION	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	11 %	7 %	9 %	/	15 %
RAM	154,2 MB	55,9 MB	158,3 MB	/	141,5 MB
FPS | %	**1,89** / *75%*	**2,51** / *100%*	**1,82** / *73%*	/	**1,98** / *79%*

Obr. 40 - *Test*: Vykreslení základního objektu pro HP Pavilion (5 milionů px)

MACBOOK PRO	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	25 %	/	12 %	16 %	22 %
RAM	105,1 MB	/	271,3 MB	45,9 MB	79,7 MB
FPS | %	**6,03** / *98%*	/	**5,81** / *94%*	**5,35** / *87%*	**6,16** / *100%*

Obr. 41 - *Test*: Vykreslení základního objektu pro MacBook Pro (5 milionů px)

6.4.4 Zátěž systému při transformaci objektu

STOLNÍ POČÍTAČ	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	58 %	39 %	49 %	/	57 %
RAM	34,9 MB	60,3 MB	147,5 MB	/	45,1 MB
FPS | %	**34,92** / *77%*	**45,25** / *100%*	**34,70** / *76%*	/	**32,62** / *72%*

Obr. 42 - *Test*: Transformace objektu pro Stolní PC (rozlišení 220 × 277)

HP PAVILION	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	23 %	14 %	15 %	/	25 %
RAM	62,6 MB	56,8 MB	155,6 MB	/	41,7 MB
FPS | %	**14,08** / *100%*	**11,51** / *82%*	**12,95** / *89%*	/	**13,68** / *97%*

Obr. 43 - *Test*: Transformace objektu pro HP Pavilion (rozlišení 220 × 277)

MACBOOK PRO	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	16 %	/	17 %	19 %	21 %
RAM	37,8 MB	/	263,7 MB	48,9 MB	33,6 MB
FPS | %	**27,18** / *62%*	/	**24,89** / *56%*	**44,14** / *100%*	**27,33** / *62%*

Obr. 44 - *Test*: Transformace objektu pro MacBook Pro (rozlišení 220 × 277)

6.4.5 Zátěž systému při vykreslování animace

STOLNÍ POČÍTAČ	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	22 %	20 %	25 %	/	22 %
RAM	82,2 MB	50,6 MB	135,6 MB	/	40,3 MB
FPS | %	**60,67**	**60,75**	**60,02**	/	**60,39**

Obr. 45 - *Test*: Vykreslení animace pro Stolní PC (rychlost animace 10)

HP PAVILION	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	13 %	8 %	12 %	/	15 %
RAM	64,7 MB	47,2 MB	152,9 MB	/	64,5 MB
FPS | %	**60,68**	**60,64**	**59,92**	/	**60,46**

Obr. 46 - *Test*: Vykreslení animace pro HP Pavilion (rychlost animace 10)

MACBOOK PRO	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
CPU	16 %	/	21 %	12 %	17 %
RAM	17,3 MB	/	252,1 MB	17,2 MB	17,9 MB
FPS | %	**59,63**	/	**60,49**	**59,94**	**60,46**

Obr. 47 - *Test*: Vykreslení animace pro MacBook Pro (rychlost animace 10)

6.5 Vyhodnocení testů

Předešlým testováním bylo zjištěno několik překvapivých faktů, které je potřeba vysvětlit a okomentovat. Globálně však bylo zjištěno, že zatěžování procesoru je nejvyšší na konfiguraci s procesorem od firmy AMD. Lze tak usoudit, že procesor od tohoto výrobce je nejvíce náchylný na zatěžování i přes fakt, že je oproti oběma zbylým procesorům od výrobce Intel výše taktovaný a má více jader. Dokázali jsme tak fakt, že na počtu jader, či frekvenci tolik nezáleží, na čem naopak záleží je počet vykonávaných operací za sekundu.

6.5.1 Vyhodnocení testu vykreslování čar

Zprvu zcela triviální test velice překvapil. Bylo zjištěno, že operace, u které by se dalo předpokládat, že jeho výsledek bude celkem srovnatelný na všech konfiguracích a sice, že vykreslovaný počet snímků za vteřinu se bude pohybovat kolem maximální hranice se vyskytly větší rozdíly. Zjištěno bylo, že nejlepší v testu dopadl MacBook a vykreslování v Safari. Na platformě Windows byl počet snímků za sekundu v prohlížečích značně nižší, zhruba až o 50%. Jedinou výjimkou byl však prohlížeč Microsoft Edge, který se kolem maxima pohyboval. Co se zátěže na systém týká, zprvu míněná „nejsilnější“ konfigurace a sice stolní počítač měl druhou největší zátěž na procesor, což se od tak triviální operace nečekalo.

6.5.2 Vyhodnocení testu vykreslování textu

Tento test objevil nečekaná úskalí dvou prohlížečů a to Safari a Microsoft Edge. Zatímco v prohlížečích, které jsou multiplatformní (Google Chrome, Mozilla Firefox a Opera) se počet vykreslovaných snímků za vteřinu pohybuje okolo podobných hodnot, tak tyto dva prohlížeče naprosto zaostávají a dostávají se pod hranici jednoho snímku za vteřinu. Tímto pozorováním lze usoudit, že se výkon v operačním systému OS X dostává na téměř dvojnásobné hodnoty a zatížení systému je, co se procesoru a operační paměti menší. Lze tak říci, že v těchto triviálních vykreslovacích metodách je tento operační systém lépe optimalizován, neboť jeho výkon je značně vyšší než u stolního počítače, který by s těmito operacemi neměl mít problémy.

6.5.3 Vyhodnocení testu vykreslování objektu

V tomto testu bylo demonstrováno vykreslování obrázku, aby nedošlo k možnosti zkreslování výsledků, byl vytvořen objekt o stejných rozměrech, jako vytvořená čára, kterou jsme testovali výše. Jde tedy o stejné pixelové zatížení a je možné tyto výsledky posoudit objektivně. Jak se z nabitých znalostí dalo očekávat, dochází k tomu, že vykreslování objektu (vytvořeného jako obrázek) je náročnější než vykreslování již zmíněných čar. V tomto testu lze také pozorovat vyšší paměťovou náročnost celé operace, ve srovnání s ostatními testy jde dokonce o nejvyšší náročnost. Dochází totiž k opakovanému načítání vytvořeného objektu a to paměť logicky zatěžuje. Při omylném pokusu, kdy bylo vytvoření nové instance obrázku uvnitř cyklu dokonce docházelo k zatížení operační paměti v řádech gigabajtů.

6.5.4 Vyhodnocení testu pro transformaci objektu

Zde bylo názorně ukázáno, jak načtení obrázku z externího (internetového) zdroje dokáže ulehčit celý proces a efektivně zvýšit počet vykreslovaných snímků za vteřinu. Díky tomuto usnadnění se frekvence vykreslení obrázku za vteřinu (stejně jako v testu pro vykreslení objektu) dostává na násobné hodnoty. Nastává zde i úspora zátěže celého systému. Jakožto i v testu vykreslování čar i zde nastává situace, kde pro platformu Microsoft Windows tuto operaci zvládne nejlépe jeho nativní webový prohlížeč Microsoft Edge a pro operační systém OS X to je Safari.

6.5.5 Vyhodnocení testu animace

Zde nás především zajímalo vytížení procesoru. Překvapivým zjištěním bylo, že proces animace nezatěžuje procesor tak, jak by se mohlo očekávat. Ve všech třech konfiguracích i ve všech prohlížečích se zatížení procesoru pohybuje kolem hodnoty 20 %, což je často méně, než při sledování videa na internetové stránce. V tomto testu byla rovněž potvrzena hypotéza, že dnešní prohlížeče webového obsahu jsou optimalizovány pro vykreslování dynamických částí uvnitř stránek, které se právě přes animaci uvnitř Canvas elementu dá dosáhnout. Potvrzuje to především fakt, že všechny testy na třech naprosto výkonnostně odlišných konfiguracích proběhly tak, že se animace skutečně promítala s frekvencí okolo 60 snímků za vteřinu.

6.6 Srovnávací testy

V testech byl nastíněn dopad na systém při vykreslování plochy 1000px, otázkou ale je, co se stane dále, pokud budou upraveny některé z parametrů metod. Bylo zjištěno, že vykreslování čáry je náročnější, než by se očekávalo, zkusme proto, jak se změní rychlost vykreslování snímků za vteřinu, pokud zvětšíme délku čáry. Tím zároveň zvětšíme počet vykreslovaných pixelů a to je rozdíl, který nás zajímá.

MACBOOK PRO	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
FPS 1000px	**44,94** / *76%*	/	**32,19** / *54%*	**59,24** / *100%*	**44,23** / *74%*
FPS 9000px	**37,52** / *97%*	/	**28,12** / *73%*	**38,46** / *100%*	**36,92** / *96%*

Obr. 48 - *Test:* Porovnání počtu FPS při změně vykreslovaných pixelů

Tímto testem lze pozorovat, že při zvýšení velikosti vykreslované plochy dochází k úbytku počtu vykreslovaných snímků za vteřinu. Je to způsobeno větší zátěží na systém, neboli nutnosti rozsáhlejší kalkulace výpočtů pro vykreslení.

Další rozdíl by mohl být pozorovatelný při změně velikosti vykreslovaného objektu. Víme, že v předchozím testu byl v jednom cyklu vykreslován obraz o velikosti 1000 pixelů, otázkou tedy zůstává, jak ubyde výkon, pokud zvětšíme obrázek k vykreslení.

MACBOOK PRO	*Google Chrome*	*Microsoft Edge*	*Mozilla Firefox*	*Safari*	*Opera*
FPS 1000px	**6,03** / *98%*	/	**5,81** / *94%*	**5,35** / *87%*	**6,16** / *100%*
FPS 9000px	**5,97** / *99%*	/	**6,02** / *100%*	**5,31** / *88%*	**5,93**/ *98%*

Obr. 49 - *Test:* Porovnání výkonu FPS při vykreslování objektů s různou velikostí

Jak je z výsledku patrné, změna velikosti nám výkon moc nezmění, neboť jakmile se objekt načte, pak už jde o stejně náročnou operaci nehledě na rozměry.

7 Metody optimalizace

Navržení různých optimalizačních konstrukcí pro zefektivnění výkonnostního dopadu na systém.

7.1 Úspora při vykreslování obrázků

Jak jsme si mohli povšimnout ve výše testovaném vykreslování obrázku, který byl vytvořen jako Canvas element, následně převeden do obrázku a ten vykreslen, tak frekvence vykreslování nebyla nijak oslnivá. Je proto nutné zamyslet se, jak toto vykreslování optimalizovat jinak, než s voláním vykreslování obrázku z externího zdroje. Jak jsme si mohli povšimnout, metodu `drawImage()` voláme s parametrem „img“, což je de facto canvas. Možnou optimalizací je, že místo vykreslování s voláním tohoto parametru budeme volat přímo Canvas, který je převáděn do obrázkové podoby. Tato optimalizace však nabývá největšího smyslu, pokud voláme metodu `drawImage()` bez parametrů šířky a výšky. Toto nepatrné upravení chování přináší celkové zrychlení, neboť se celá operace provádí nativně bez nutnosti dalšího zásahu.

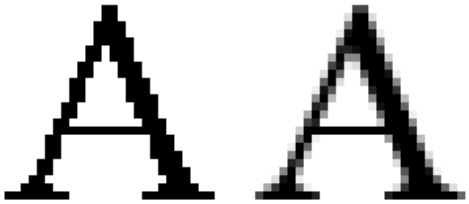
Další možnou alternativou, jak vykreslování obrázků, které sami definujeme zjednodušit, je použití WebGL. Toto JavaScriptové API je oproti Canvas elementu o mnoho rychlejší. Jeho správné použití je však diskutabilní. Sice nabízí zrychlení vykreslování až o 70 %, ale na starších konfiguracích počítačových strojů může být také o mnoho pomalejší, nežli obyčejný Canvas.

7.1.1 WebGL

Je na místě ujasnit si také, co to vlastně WebGL je. Jak již bylo zmíněno, je to JavaScriptové API, které využívá kódu v JavaScriptu, kde se vykonává funkční logika, ale celá výhoda tohoto prvku je v tzv. „shader“ kódu. Jde o tu část kódu, která se dá vykonávat čistě na grafické kartě. Celý tento princip je obdobný jako u OpenGL, ze kterého právě WebGL vychází. Je také nutno říci, že se nejedná o zcela odlišný prvek. WebGL elementu Canvas využívá pro vykreslování statických částí uvnitř obrazů. Největší smysl využití má WebGL ve 3D grafice.

7.2 Řešení antialiasingu v Canvas elementu

Toto řešení záleží na situaci použití. K nutnosti řešit tento jev se dostáváme, pokud je cílem vykreslit do Canvas elementu statický prvek, který by měl být následnými úpravami zvětšován. Pro objasnění je nutno zmínit, co to vlastně alias je. Je to jev, kdy se spojitá informace převádí na diskrétní, neboli nespojitou, díky vzorkovací frekvenci. Laicky řečeno dochází k rozrastrování prvku, např. čáry. Celý jev se děje díky neoptimálnímu vzorkování, díky tomu jsou tak vidět pixely a čára se nejeví jako rovná. Řešení tohoto jevu se nazývá antialiasing. Jde o vzorkování s dvojnásobnou frekvencí než maximální. Dochází tak k vjemu, že okolo čáry se vytvářejí pixely s poloviční intenzitou barvy.



Obr. 50 - Příklad jevu aliasing a jeho řešení antialiasing

Řešení tohoto jevu v Canvas elementu je takové, že nativně je antialiasing zapnutý. Jsou však případy, kdy je třeba ho udělat více znatelný. Toho dosáhneme logickou úvahou tak, že přetransformujeme vytvořené plátno v Canvas elementu poloviční distancí pomocí funkce `vytvorenyKontext.translate(0.5, 0.5)`. Jsou však také situace, kdy je optimální antialiasing vypnout. Je však nutno říci, že následující metoda funguje pouze v případě obrázků, na kterých vypíná umělé vyhlazování, nevypíná však celkový antialiasing v celém plátně Canvas elementu. Toto vypnutí lze dosáhnout pomocí metody `vytvorenyKontext.imageSmoothingEnabled = false`.

7.3 Řešení vykreslování textu

Řešením pro toto úskalí jsou CSS3 kaskádové styly. Jelikož bylo zjištěno, že vykreslování textu přes Canvas není úplně optimální, je efektivnější text na plátnu připsat přes HTML. Zbavujeme se tak mnoha výpočtů a tím se ulehčuje zátěži systému. Tato optimalizace je efektivnější, neboť se text již nepřevádí do rastrového obrazu a navíc umožňuje využívat trendů, jako je rozpoznávání textu a následovné hledání v článcích.

7.4 Zefektivnění běhu celoobrazovkových aplikací

V situaci reálné praxe se často můžeme setkat s nutností vytvořit souvislou funkční aplikaci přímo v prohlížeči. Možností jak toto vytvořit je mnoho. Naprogramovat ji přímo v HTML, vložit jako externí a nebo s využitím JavaScriptu. Nás však zajímá více otázka vykreslování takovéto aplikace. Pokud by se celá aplikace měla kreslit přes Canvas na celoobrazovkový režim, bylo by to velice paměťově i provozně náročné. Možností, jak toto vykreslování zefektivnit je, že pokud vytvoříme okno, kde kód probíhá v plátně s malými rozměry (např. 640x480), pak můžeme pomocí kaskádových stylů zvětšit celé plátno s aplikací. Dochází zde sice ke ztrátě detailu, nicméně běh celé aplikace je několikanásobně výkonnější a to je pro optimální fungování důležitější.

Pokud by však bylo cílem provozovat aplikaci s nejvyšším detailem a programátor nedbal na hladký průběh, dalo by se nastavit celoobrazovkové zobrazení následujícími metodami. Pro nastavení šířky plátna zvolit `canvas.width = window.innerWidth` a pro nastavení výšky `canvas.height = window.innerHeight.`

7.5 Kreslení na vrstvách plátna

Jak již bylo naznačeno v kapitole přemazávání, tak vykreslování v několika vrstvách s využitím Z-indexu přináší vysokou úsporu nejen při změnách dynamického obsahu, ale rovněž, co se týče zátěže systému. Vzhledem k tomu, že se jedná o menší plátna s jednoduchými operaci, pak výpočetní složitost není taková, aby výrazně dopadala na výkon systému.

7.6 Následky při nesprávném tvoření

Pokud se pravidly a zásadami pro správné tvoření webového obsahu nebude vývojář, či programátor řídit, pak se může stát, že při zatěžování metod bude docházet k prudším úbytkům vykreslovaných snímků za vteřinu a mnohem vyššímu zatěžování systému. Tento jev by byl pozorovatelný především na animaci, kde by se animace „trhala“.

8 Závěr

V dnešní době je mnoho možností, jak tvořit strukturální a smysluplný obsah webových stránek. Často je však zapotřebí uvažovat tak, aby stránka, kterou tvůrce dělá dostávala smysl nejen po obsahové stránce, ale zároveň po stránce tvůrčí. Práce objasnila několik stěžejních prvků pro tvorbu grafiky ve webových stránkách. Specializace byla především na element Canvas a částečně bylo objasněno, jak a kdy se tento element používá. Práce rovněž potvrdila předem stanovené hypotézy. Testováním a měřením výkonů bylo dokázáno, že výkon se skutečně mnohdy znatelně liší podle HW výbavy dané konfigurace a daného prohlížeče. Velkým překvapením bylo, že prohlížeč nativně obsažený v operačním systému Windows a sice Microsoft Edge si vedl v testech velice dobře, ba dokonce nad rámec očekávání. Nedá se přesně specificky určit, který prohlížeč je nejlepší. Kdybych měl osobně doporučit jeden, pak bych si vybral Operu pro operační systém Windows, tento prohlížeč nikde nestrádal, pokaždé se držel kolem maximálních hodnot a i když v nějakém testu nebyl nejvýkonnější, neměl hrubé nedostatky s nějakou ze zvolených metod. Pro operační systém OS X bych osobně zvolit Safari, jeho optimalizace průběhů metod je vysoce překvapivá a kromě vykreslování textu nikde nezaostával. Co se vykreslování textových řetězců týče, rozhodně se vyplatí, řešit vykreslování textu zvlášť přes HTML tagy na Canvas plátno přes jiný Canvas s jinou hodnotou v Z-indexu. Práce měla přinést čtenáři široký rozhled o dostupných konstrukcích v elementu Canvas a seznámit ho s jeho dopadem na zátěž systému a především plynulostí počtu vykreslovaných snímků za vteřinu, tak, aby tuto problematiku pochopil i laik. Tento prvek není jenom o tom, kreslit obrazce do plátna, ale dá se využít i na tvorbu komplexnějších aplikací, kde umožňuje programátorovi trochu volnosti. Osobně si tak myslím, že Canvas je velikým přínosem, co se vykreslování obsahu webových stránek týče.

9 Seznam použité literatury

[1] KOSEK, Jiří (překl.). Historie HTML. In: *HTML Guru* [online]. Praha, 2013 [cit. 2016-01-22]. Dostupné z: http://htmlguru.cz/uvod-historie.html

[2] W3, Schools. HTML5 New Elements. In: *W3 Schools: THE WORLD'S LARGEST WEB DEVELOPER SITE* [online]. USA: -, 2013 [cit. 2016-01-24]. Dostupné z: http://www.w3schools.com/html/html5_new_elements.asp

[3] W3, Schools. HTML5 Canvas. In: *W3 Schools: THE WORLD'S LARGEST WEB DEVELOPER SITE* [online]. USA: -, 2013 [cit. 2016-01-24]. Dostupné z: http://www.w3schools.com/html/html5_canvas.asp

[4] GEARY, David M. Core HTML5 canvas: graphics, animation, and game development. Upper Saddle River, NJ: Prentice Hall, 2012, xxv, 723 p. ISBN 9780132761611.

[5] W3, Schools. HTML canvas createPattern() Method. In: *W3 Schools: THE WORLD'S LARGEST WEB DEVELOPER SITE* [online]. USA: -, 2013 [cit. 2016-01-24]. Dostupné z: http://www.w3schools.com/tags/canvas_createpattern.asp

[6] W3, Schools. HTML canvas drawImage() Method. In: *W3 Schools: THE WORLD'S LARGEST WEB DEVELOPER SITE* [online]. USA: -, 2013 [cit. 2016-01-24]. Dostupné z: http://www.w3schools.com/tags/canvas_drawimage.asp

[7] MALONE, William. Canvas SpriteAnimation. In:*WilliamMalone: Canvas Guide* [online]. USA: -, 2014 [cit. 2016-02-21]. Dostupné z: http://www.williammalone.com/articles/create-html5-canvas-javascript-sprite-animation/

[8] ERLICH, Tomáš. Cache na webu. In: Tomáš Erlich [online]. [cit. 2016-08-09]. Dostupné z: http://tomaserlich.cz/kesovani-souboru-pomoci-htaccess/

[9] GAMMA, Erich, Richard HELM, Ralph JOHNSON a John VLISSIDES. Design Patterns. 1. USA, 1994. ISBN 0-201-63361-2.

[10] MICROSOFT. Canvas.ZIndex: Developer Network. In: Microsoft Documentation [online]. 2015 [cit. 2016-08-12]. Dostupné z: https://msdn.microsoft.com/en-us/library/bb980110(v=vs.95).aspx

10 Seznam obrázků

11 Rejstřík použitých názvů

12 Zadání bakalářské práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Hnik Jaromír	Veská 114, Sezemice - Veská	I1301289

TÉMA ČESKY:

Výkon elementu HTML5 Canvas

TÉMA ANGLICKY:

Performance of HTML5 Canvas element

VEDOUCÍ PRÁCE:

Ing. Zdeněk Mlčoch - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Student vyhledá dostupné optimalizace a změří jejich dopad na výkon v rozšířených prohlížečích.

Osnova:
1.Úvod a seznámení s grafickými a webovými prvky.
2.Představení prvku Canvas.
3.Metodika testování a optimalizací.
4.Vytvoření testů.
5.Testování prvků Canvas v prohlížečích.
6.Zhodnocení optimalizace.
7.Závěr.
8.Zdroje

Cíl práce:
Cílem této práce bude nalézt optimální programové konstrukce pro práci s HTML5 canvas elementem. V této práci bude otestováno vykreslování elementů v prvku Canvas v různých alternativách webových prohlížečů a jejich dopad na systém. Další část práce se bude zaobírat optimalizací tohoto dopadu. Dosažené výsledky budou zhodnoceny a porovnány. Na závěr práce bude zhodnocen celkový přínos těchto dvou prvků do reálného světa.

SEZNAM DOPORUČENÉ LITERATURY:

Core HTML5 Canvas: Graphics, Animation, and Game Development (Core Series) ISBN-13: 978-0132761611