VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

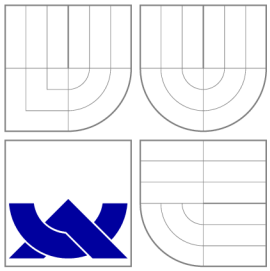
IMPLEMENTACE 3D LOGICKÉ HRY V JAVASCRIPTU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

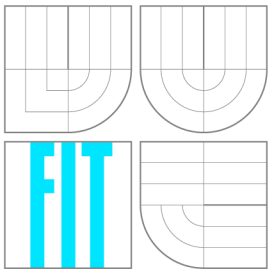
AUTOR PRÁCE
AUTHOR

MARTIN KNAPOVSKÝ

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

IMPLEMENTACE 3D LOGICKÉ HRY V JAVASCRIPTU

IMPLEMENTATION OF 3D LOGIC GAME IN JAVASCRIPT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN KNAPOVSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. VENDULA HRUBÁ
SUPERVISOR

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá vývojem logické webové hry, která vznikla na základě analýzy hry Berušky 2. Jsou zde stručně představeny technologie potřebně k vývoji a také teorie nutná k pochopení postupů použitých při implementaci. Výsledné řešení je podrobeno testům, jejichž výsledky jsou podkladem pro diskuzi obecné využitelnosti technologií, které se dnes v rámci webu nově objevují.

Abstract

This bachelor's thesis describes development of a web game based on analysis of Berušky 2 game. Technologies needed for development are briefly introduced as well as the theory necessary for understanding the methods used in implementation. The final solution was put through tests, results of which serve as a basis for discussion about overall usability of technologies newly appearing on the web scene.

Klíčová slova

JavaScript, WebGL, HTML5, Hra

Keywords

JavaScript, WebGL, HTML5, Game

Citace

Martin Knapovský: Implementace 3D logické hry v JavaScriptu, bakalářská práce, Brno, FIT VUT v Brně, 2012

Implementace 3D logické hry v JavaScriptu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením paní Ing. Venduly Hrubé. Další informace mi poskytl pan Ing. Martin Stránský ze společnosti Red Hat. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Knapovský
16. května 2012

Poděkování

Chtěl bych poděkovat paní Ing. Vendule Hrubé a panu Ing. Martinu Stránskému za jejich pomoc a rady při tvorbě této bakalářské práce.

Obsah

Kapitola 1

Úvod

V dnešní době již webové technologie pokročily natolik, že je možné vytvářet takový obsah, který je plnohodnotnou aplikací s uživatelským rozhraním, multimediálním obsahem, či možností komunikace jejích uživatelů. Největší výhodou těchto aplikací je však to, že jsou dostupné komukoliv s kompatibilním webovým prohlížečem. Multiplatformnost aplikací tak není řešena tím, že bychom vytvářeli pro každý typ operačního systému příslušné spustitelné soubory, ale stačí pouze otevřít prohlížeč, zadat adresu a následně s aplikací pracovat.

Cílem této práce bylo navrhnout a implementovat logickou webovou hru založenou na hře Berušky 2 a ověřit tak vhodnost použití nově se objevujících technologií k tomutu účelu.

Technologie, jejich historie a použití jsou popsány v kapitole 2. Ta se primárně zaměřuje na WebGL, avšak jsou zde obsaženy i informace ohledně technologie HTML5, programovacího jazyka JavaScript a reprezentaci dat ve formátu JSON. V této kapitole je také obsažena teorie nutná k pochopení principů použitých při implementaci hry. Kapitola 3 se zabývá analýzou původní hry Berušky 2, na jejímž základě je následně vypracován návrh implementované hry. Návrh webu, pomocí něhož je hra dostupná, je zmíněn pouze okrajově, jelikož není přímou součástí této práce. Implementační témata jsou obsažena v kapitole 4. Kapitola 6 pak shrnuje poznatky získané při vývoji hry, diskutuje rozdíly oproti hře původní a zkoumá použitelnost využitých technologií.

Kapitola 2

Použité technologie a teorie

V této kapitole jsou uvedeny informace týkající se technologií využitých při implementaci hry. Každá z nich je stručně představena a uvedena do souvislosti s touto prací. Mimo těchto informací kapitola také obsahuje teorii nutnou k pochopení postupů použitých při implementaci.

2.1 HTML5

HTML5[1] je připravovaným webovým standardem, který v první řadě vylepšuje podporu zobrazení multimediálního obsahu prostřednictvím webového prohlížeče. Co se týče doby od vydání předchozí specifikace HTML4.01[2], je nováčkem na poli webových technologií a jeho finální specifikace není v době psaní této práce stále dokončena. Vzhledem k tomu, že technologie zahrnuté v jeho specifikaci řeší mnohé praktické problémy, vývojáři webových prohlížečů již nyní jednotlivé prvky HTML5 implementují. Experimenty s prohlížeči tak dávají pracovním skupinám spravujícím tuto specifikaci zpětnou vazbu, na jejímž základě lze provést mnohá vylepšení. Významnou vlastností specifikace je zpětná kompatibilita s její předchozí verzí, což umožňuje zobrazovat nové typy dokumentů v prohlížečích, které tyto technologie doposud neimplementují. [5, 4, 3]

Nejprve bude zmíněn historický vývoj a rozdělení HTML5 technologií, poté bude uveden výčet některých nových syntaktických a sémantických elementů, za nímž bude následovat popis elementu `<canvas>`, který je použit při implementaci hry.

Historie

Roku 2004 představili Mozilla Foundation a Opera Sorfware na W3C konzorciu návrh, který odrážel současné požadavky na webové aplikace. Tento návrh se soustředil na technologie, které by byly zpětně kompatibilní s tehdejšími webovými prohlížeči a zahrnoval také počáteční návrh specifikace Web Forms 2.0. Hlasování konzorcia dopadlo v neprospěch návrhu, což vedlo k tomu, že pracovní skupina W3C nadále soustředila své prostředky na vývoj specifikace XHTML2. Požadavky na webové aplikace však zůstaly nevyslyšeny, a tudíž byla hlavními představiteli vývojářů webových prohlížečů vytvořena nová pracovní skupina s názvem Web Hypertext Application Technology Working Group (WHATWG). O pět let později, roku 2009, zanechala W3C snahy protlačit specifikaci XHTML2 a spojila

[1] http://www.w3.org/TR/html5/
[2] http://www.w3.org/TR/REC-html40/

se s pracovní skupinou WHATWG, aby své usilí zaměřily na vývoj společného standardu HTML5. [5, 4, 3]

Značné popularity HTML5 nabylo po roce 2010, kdy tehdejší výkonný ředitel firmy Apple, Steve Jobs, prohlásil [3], že na svých zařízeních nebudou nadále podporovat obsah zobrazovaný pomocí technologie Adobe Flash a zaměří se na podporu HTML5. Vzhledem k zastoupení této firmy na trhu začala valná většina velkých webových společností svůj obsah nabízet i pomocí této technologie.

V březnu roku 2011 vytyčila W3C cíle pro vývoj HTML5, kde určila rok 2014 jako cílový pro přijetí specifikace. Podmínky tohoto přijetí jsou 2 plně funkční implementace. Již dnes jsou mnohé implementace prvků stabilní a připraveny k použití. Jednou z nich je právě v implementaci hry využitý element `<canvas>`.

Rozdělení HTML5 technologií

Diagram technologií a jejich rozdělení je vyčleněn do přílohy A. V následujících odstavcích jsou tyto kategorie stručně popsány.

Oficiální W3C specifikace zahrnuje nové syntaktické a sémantické elementy, nové a vylepšené web widgety, podporu pro audio/video a také element `<canvas>`, který slouží k vykreslování grafiky pomocí JavaScriptu. Tato část zahrnuje většinu prvků HTML5, které jsou prohlížeči dobře podporovány.

Prvky původního návrhu patřily do specifikace, kterou připravila pracovní skupina WHATWG. Většina z nich vyžaduje ke své práci JavaScript a mezi ty nejdůležitější patří *Local Data Storage* a *Offline aplikace*.

Prky, které jsou často zahrnovány do HTML5 jsou například CSS3 a nástroje pro geolokaci.

Nové syntaktické a sémantické elementy HTML5

Nové sémantické elementy rozšiřují možnosti rozlišení úseků značkovaného textu. Na místě, kde byla typicky použita značka `<div>`, nebo `<span>` je nyní možné použít značky jako `<nav>` pro navigaci webu a dále pak například element `<footer>`, který označuje patičku dokumentu.

Video a **audio** data byla v předchozích verzích HTML vkládána do dokumentů pomocí elementu `<object>`. Ten umožňoval označit obsah interpretovaný zásuvným modulem prohlížeče, kterým byl pro přehrávání videa nejčastěji Adobe Flash. Ten však díky novým možnostem HTML5 a přístupu firem jako Apple postupně ztrácí své dominantní postavení.

[3] https://www.apple.com/hotnews/thoughts-on-flash/

Technologie HTML5 použité při implementaci

Použité technologie:

- Canvas API[4]
- WebGL
- DOM API (Součást HTML5 Microdata)
- Web Audio API

Z technologií HTML5 je při implementaci hry primárně využito **Canvas API**, které umožňuje prostřednictvím **WebGL** (podkapitola 2.5) přistupovat k elementu `<canvas>` a vykreslovat tak 3D grafiku. **DOM API** je rozhraní pro přístup k prvkům zobrazovaného dokumentu a slouží k modifikaci jejich obsahu, struktury, nebo stylu, kterým jsou zobrazovány. **Web Audio API** je rozhraním elementu `<audio>`, které slouží k přehrávání herní hudby. Původní návrh hry zahrnoval také využití technologie **localStorage**, která umožňuje uchovávat herní data na straně klienta bez nutnosti jejich načítaní při obnovení webové stránky. Vzhledem k tomu, že dnešní prohlížeče mají omezení na množství uložených dat[5], nenalézá tato technologie v implementaci smysluplné využití. Na diagramu A je také vyznačen JavaScriptový framework **jQuery**, jehož je využito hlavně pro vytvoření různých animací webu.

HTML5 `<canvas>`

Základní HTML5 Canvas API obsahuje **2D** kontext, který umožňuje vykreslovat různé typy tvarů, text a také umožňuje zobrazovat obrazová data přímo do oblasti elementu `<canvas>`. Pro vykreslování je možné volit různé barvy, rotovat s prvky, zprůhledňovat je, či manipulovat s jednotlivými pixely tohoto elementu.

V souvislosti s WebGL mluvíme o **3D** kontextu, který umožňuje přímé vykreslování bitmapy, jejíž obsah je možné měnit pomocí JavaScriptu. U 3D kontextu je při každém volání vykreslovacího rozhraní plocha elementu zcela překreslena a je tedy programátorovou prací připravit vykreslovaný obraz před každým voláním. Tím se `<canvas>` odlišuje od technologií jako Flash, Silverlight, nebo SVG, které vykreslovaný obraz pouze aktualizují (umožňují například posouvat vykreslované objekty scény vůči své aktuální pozici), což na jednu stranu programátora odstiňuje od nízkoúrovňových operací, avšak na stranu druhou má programátor omezenou kontrolu nad výsledným vykresleným obrazem. Podrobnější informace o tom, jak vykreslování pracuje jsou součástí podkapitoly 2.5.

[4] V textu lze často nalézt pojem API. API, neboli Applicaton Programming Interface je, jak z názvu vyplývá, rozhraní pro programování aplikací. Jedná se o sbírku procedur, funkcí, či tříd, které může programátor při tvorbě aplikace využívat.

[5] http://dev-test.nemikor.com/web-storage/support-test/

2.2 Javascript

Většina webových stránek se dnes podobá plnohodnotným desktopovým aplikacím právě díky JavaScriptu. JavaScript je programovací jazyk, který umožňuje vylepšit zobrazovaný dokument prostřednictvím animací, interaktivitou, nebo dynamickými visuálními efekty. Jeho velkou výhodou je to, že stránky reagují na podněty uživatele okamžitě - při vyplňování formuláře, nebo v moment, kdy uživatel pohne myší na určitou pozici. Jeho zpracování je totiž oproti jiným skriptovacím jazykům používaným na webu prováděno na klientském zařízení (avšak nejen tam, jak bude uvedeno dále). V současné době je jeho masivní využití vidět v projektech jako Facebook[6], nebo například Google Maps[7].

Popularita tohoto jazyka roste a s ním i jeho využití. Nejen, že jeho interprety lze dnes nalézt ve všech webových prohlížečích na stolních počítačích, herních konzolích, chytrých telefonech či tabletech, ale je využit také pro popis grafických rozhraní desktopových aplikací, nebo jako jazyk použitý na straně serveru. Příklady využití JavaScriptu mimo web jsou uvedeny v následujícím přehledu.

- Doplňky pro prohlížeč Google Chrome, Opera, Safari
- Apple dashboard widgety
- JavaScript v pdf souborech
- Skriptování v Adobe Photoshop, Illustrator, InDesign
- Zpracování signálu v Max/MSP pro program Ableton Live
- Skriptování v Unity
- JavaScript v prostředí Javy
- Popis grafického rozhraní v knihovně Qt
- ...

Historie jazyka

Autorem jazyka je Brendan Eich z tehdejší společnosti Netscape. Jazyk byl nejdříve vyvíjen pod názvem *„Mocha, LiveScript“*, avšak s vydáním prohlížeče Netscape 2.0B3, jehož byl tento jazyk součástí, se název z marketingových důvodů změnil na JavaScript. Jeho velký úspěch vedl k tomu, že společnost Microsoft vytvořila jeho obdobu pro svůj prohlížeč Internet Explorer, kterou nazvala JScript. V roce 1997 byl jazyk standardizován pod názvem ECMAScript, ze kterého byly následně odvozeny i další implementace, jako je například ActionScript. Svého největšího rozmachu se tento jazyk dočkal s rozšířením technik, které se souhrnně označují jako AJAX (2.2).[2]

[6] www.facebook.com
[7] maps.google.com

Stručný popis jazyka JavaScript

JavaScript je multiplatformní, dynamický, slabě typovaný skriptovací jazyk. Podporuje různá paradigmata včetně objektového, čehož je v implementaci hry často využíváno. Syntaxe jazyka je ovlivněna konvencemi jazyků Java a C, principiálně je však jazyk vystavěn podobně jako jazyky Self a Scheme. Způsob, jakým jsou vytvářeny objekty, je zde oproti klasickým jazykům jako C++ nebo Java odlišný díky tomu, že je tento jazyk prototypově založený. Třídy jsou tak nahrazeny konstruktory objektů a dědičnost prototypováním.[2] Příklad vytvoření konstruktoru objektu a jeho následné použití je uveden ve zdrojovém kódu 2.1.

```
function TestObject(num){

    this.property1 = 1;
    this.property2 = this.property1;
    this.property3 = num;
}

var test = new TestObject(5);
alert(test.property1); // 1
alert(test.property2); // 1
alert(test.property3); // 5

test.property1 = 6;
alert(test.property1); // 6
```

Zdrojový kód 2.1: Příklad vytvoření objektu v JavaScriptu

JavaScript interaguje s webovou stránkou prostřednictvím dříve popsaného rozhraní DOM tak, že se naváže zpracování událostí vytvářených uživatelem na funkce, které je zpracovávají. Rozhraní DOM však není přesně standardizované a prohlížeče ho implementují různě. U některých prvků je tedy nutné rozlišovat, který prohlížeč JavaScript aktuálně interpretuje a dle toho volit, jakým způsobem se s nimi bude nakládat. Jako příklad je ve zdrojovém kódu 2.2 uvedeno zpracování události pohybu kolečka myši.

```
function handleWheel(event) {
    var delta = 0;

    // Zpracování události v Internet Exploreru
    if (!event){
       event = window.event;
    }

    // Internet Explorer, Opera, Chrome
    if (event.wheelDelta) {
        delta = event.wheelDelta / 120;
    }
    // Mozilla Firefox
    else if (event.detail) {
        delta = -event.detail / 3;
    }
    // ...
}
```

Zdrojový kód 2.2: Zpracování události pohybu kolečka myši

AJAX

AJAX, neboli Asynchronous JavaScript and XML, je souhrnné označení pro vzájemně propojené techniky používané na klientské straně webových aplikací. Tyto techniky umožňují asynchronně nahrát data ze serveru, což pro uživatele znamená, že aktuálně zobrazená webová stránka nemusí být znovu načtena a interakce s aplikací je tak mnohem přívětivější. Využívá se přitom `XMLHttpRequest` objektu, který je dostupný prostřednictvím JavaScriptu. Navzdory názvu, který vyzdvihuje přenos dat ve formátu XML, je možné použít i formáty jiné. Ve hře je těchto technik využito k načtení herních úrovní, které jsou reprezentovány soubory formátu JSON a následně pak potřebných textur a lightmap.

JavaScript Frameworky

JavaScriptové frameworky slouží k urychlenému vývoji webových aplikací. Vytvářejí jakousi vrstvu nad základními prostředky jazyka a poskytují k nim tak zjednodušený přístup. Frameworků pro jazyk JavaScript existuje celá řada, avšak pro implementaci bylo využito jednoho z nejznámějších - jQuery.

jQuery umožňuje přistupovat k prvkům DOM tak, že se rozdíly mezi prohlížeči (jako třeba právě zpracování události kolečka myši) stírají a k prvkům je tak přistupováno jednotnou formou. Za základě událostí generovaných uživatelem je také umožněno animovat vzhled prvků dokumentu - měnit jejich pozice, průhlednost, či například vrstvu, ve které se mají zobrazit. Toho je ve hře využito velmi často, avšak vzhledem k tomu, že se jedná o animace webu samotného, který není přímou součástí této práce, nebude jejich popisu věnováno mnoho prostoru. Příklad animace průhlednosti prvku je uveden ve zdrojovém kódu 2.3.

```
1 // Animace průhlednosti z aktuální hodnoty na hodnotu 0.0 za čas 1000 ms.
2 // Po dokončení animace je zavolána anonymní funkce měnící vrstvu,
3 // ve které se prvek zobrazuje.
4 $("#item").animate({'opacity':'0.0'}, 1000, function(){
5     $("#item").css({'z-index':'-1'});
6 });
```

Zdrojový kód 2.3: jQuery Animace

2.3 JSON

JSON, neboli JavaScript Object Notation je otevřený standard určený pro výměnu dat. Jeho notace je odvozena od JavaScriptu, což oproti dalšímu rozšířenému formátu pro přenos dat - XML, usnadňuje práci s daty v tomto programovacím jazyku. XML dokáže svým značkováním lépe popsat kontext toho, co přenáší, avšak za cenu většího objemu dat. Proto je také JSON chápán jako odlehčená verze XML. JSON není použit pouze v programovacím jazyku JavaScript, avšak v dnešní době můžeme nalézt jeho syntaktické analyzátory i v knihovnách mnohých dalších jazyků.[2]

JSON je založen na dvou strukturách:

- Kolekce párů název/hodnota. Ta bývá v rozličných jazycích realizována jako objekt, záznam (record), struktura (struct), slovník (dictionary), hash tabulka, klíčový seznam (keyed list), nebo asociativní pole.

- Tříděný seznam hodnot. Ten je ve většině jazyků realizován jako pole, vektor, seznam (list) nebo posloupnost (sequence).

Na následujících diagramech je znázorněna JSON syntaxe zápisu objektu (Diagram 2.1), pole (Diagram 2.2) a hodnoty (Diagram 2.3).

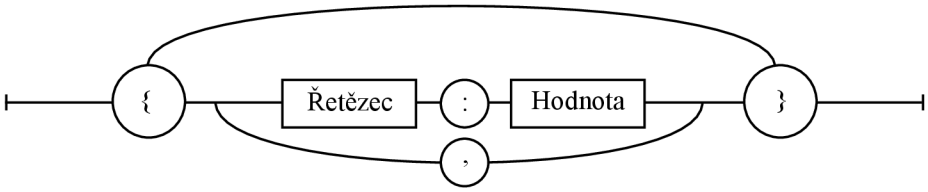


Diagram 2.1: JSON Objekt

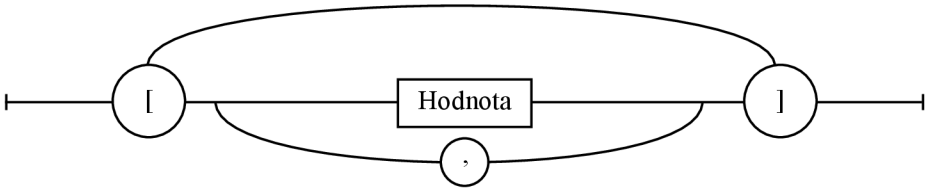


Diagram 2.2: JSON Pole

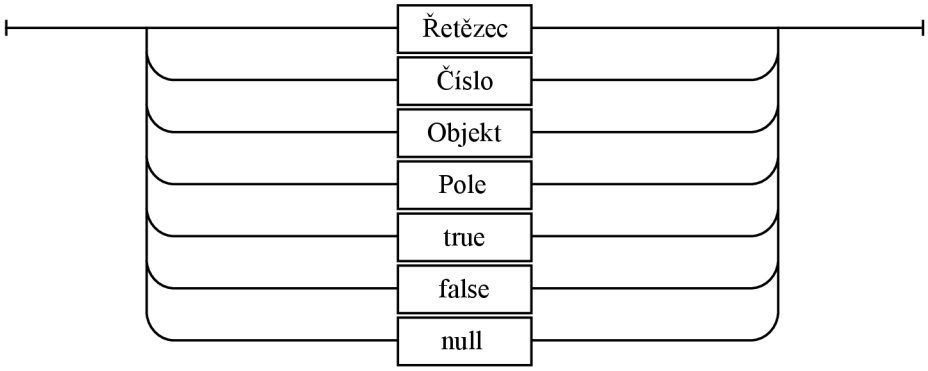


Diagram 2.3: JSON Hodnota

JSON je ve hře použit pro reprezentaci herních úrovní. Vzhledem k tomu, že při použití jakéhokoliv jiného formátu by bylo potřeba na vstupu provádět jeho parsování, což by hru zbytečně zpomalovalo, je tento formát v podstatě jediným vhodným kandidátem.

2.4 Lineární algebra pro 3D grafiku

Lineární algebra je ta část matematiky, která se zabývá vektory a maticemi. Pro porozumění 3D grafice a WebGL je dobré mít alespoň základní znalosti v tomto oboru. Vzhledem k rozsahu této práce není možné zmínit vše, a proto jsou pojmy jako souřadný systém, vektory a jejich skalární a vektorový součin považovány za znalost čtenáře. Následující text se soustředí na homogenní souřadnice, matice a transformace ve 3D grafice.

Homogenní souřadnice

V 3D prostoru je možné bod[8] definovat pomocí tří souřadnic. To však může být matoucí v tom, že body a vektory jsou definovány stejným způsobem. S homogenními souřadnicemi přidáváme čtvrtou souřadnici, která je značena jako w. Pro vektory je pak $w = 0$, a pokud je $w! = 0$, pak homogenní souřadnice definují bod. Homogenní bod lze převést na tříprvkový bod vydělením všech souřadnic souřadnicí homogenní. Pro převod tříprvkového bodu na bod homogenní pak stačí přidat jako homogenní souřadnici hodnotu 1. Homogenní souřadnice se používají z toho důvodu, že ve 3D grafice jsou nejčastější operací různé transformace, které jsou popsány pomocí 4×4 matic. Abychom mohli bod pomocí těchto matic transformovat, je nutné, aby byl bod popsán právě čtyřmi souřadnicemi.

Matice

Matice je složena z řádků a sloupců. Elementy uvnitř matice se nazývají prvky matice a dle počtu řádků a sloupců rozlišujeme různé její dimenze. Nejčastěji používaným typem jsou ve WebGL matice se čtyřmi řádky a čtyřmi sloupci. Tyto pak označujeme jako 4×4 matice.

$$M = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \tag{2.1}$$

Maticím s jedním sloupcem $(m \times 1)$ se také jinak říká sloupcový vektor a maticím s jedním řádkem řádkový vektor $(1 \times m)$.

$$V = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{2.2}$$

$$V = \begin{bmatrix} v_0 & v_1 & v_2 & v_3 \end{bmatrix} \tag{2.3}$$

Součet a rozdíl matic

Součet a rozdíl je možný, pokud mají matice stejné dimenze. Součet, nebo rozdíl pak probíhá prvek po prvku.

$$A = \begin{bmatrix} 1 & 5 & 3 \\ 4 & 4 & 1 \end{bmatrix} \tag{2.4}$$

$$B = \begin{bmatrix} 5 & 3 & 3 \\ 2 & 5 & 2 \end{bmatrix} \tag{2.5}$$

$$A + B = \begin{bmatrix} 6 & 8 & 6 \\ 6 & 9 & 3 \end{bmatrix} \tag{2.6}$$

[8] Ve 3D grafice nazývaný *vertex*

Násobení matic
Násobení matic je ve 3D grafice velmi důležitou operací. Definice násobení je taková, že dvě matice mohou být vzájemně vynásobeny pouze tehdy, když se počet sloupců první matice rovná počtu řádků matice druhé. Výsledná matice má pak počet řádků rovný matici první a počet sloupců matici druhé.

$$[m \times p][p \times n] = [m \times n] \tag{2.7}$$

Prvky ij výsledné matice vzniknou skalárním vynásobením řádku i matice $\mathbf{A}$ a sloupce j matice $\mathbf{B}$.

$$M = \begin{bmatrix} -3 & 1 \\ -2 & 2 \\ -4 & 5 \end{bmatrix} \tag{2.8}$$

$$N = \begin{bmatrix} -4 & 3 \\ 3 & 5 \end{bmatrix} \tag{2.9}$$

$$M \times N = \begin{bmatrix} (-3) \times (-4) + 1 \times 3 & (-3) \times 3 + 1 \times 5 \\ (-2) \times (-4) + 2 \times 3 & (-2) \times 3 + 2 \times 5 \\ (-4) \times (-4) + 5 \times 3 & (-4) \times 3 + 5 \times 5 \end{bmatrix} = \begin{bmatrix} 15 & -4 \\ 14 & 4 \\ 31 & 13 \end{bmatrix} \tag{2.10}$$

Matice identity a inverzní matice
Matice identity je takovou maticí, že pokud s ní vynásobíme jakoukoliv matici jinou, tak získáme opět tu samou.

$$M \times I = I \times M = M \tag{2.11}$$

Matice identity je vždy čtvercová (stejný počet sloupců jako řádků), na své diagonále má prvky rovny 1 a mimo diagonálu 0.

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

Nyní, když víme, co je to matice identity, můžeme přistoupit k definici matice inverzní. Pro jakékoliv číslo x (kromě 0) nalezneme číslo $1/x$ (zapisováno také jako x^{-1}), které při vynásobení touto inverzní hodnotou dává jako svůj výsledek hodnotu 1. Podobně je definována i matice inverzní, kterou označujeme jako M^{-1}.

$$M \times M^{-1} = M^{-1} \times M = 1 \tag{2.13}$$

Je důležité poznamenat, že pouze čtvercové matice mají matici inverzní.

Transponovaná matice

Transponovaná matice vznikne prohozením svých řádků se sloupci. Matice je definována pro jakékoliv dimenze a označuje se jako M^T. Vzhledem k tomu, že ve WebGL se používají matice 4×4, jsou i zde příklady uvedeny v těchto dimenzích.

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \tag{2.14}$$

$$M^T = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix} \tag{2.15}$$

Transformace

Transformace je operace, která na svém vstupu přijímá jakousi entitu, jakou je například bod, nebo vektor, a nějakým způsobem ji modifikuje. Speciálním typem transformace je pak transformace lineární, což je zobrazení f z jednoho vektorového prostoru do druhého $f : V \to W$, které zachovává vektorové operace sčítání a násobení skalárem. Název *lineární* je odvozen z faktu, že grafem obecného lineárního zobrazení z reálných čísel do reálných čísel je přímka.

Mějme dva vektory u, v a transformaci reprezentovanou pomocí funkce f. Pak lineární transformací je operace, která splňuje následující podmínky:

$$f(u) + f(v) = f(u + v) \qquad (aditivita) \tag{2.16}$$

$$kf(u) = f(ku) \qquad (homogenita) \tag{2.17}$$

Mezi lineární transformace patří například posunutí (translace), rotace, změna měřítka (scaling), či zkosení (shearing). Násobením transformačních matic lze vytvářet ze základních transformací transformace komplexní. Při násobení však musíme dávat pozor na pořadí, ve kterém jsou matice násobeny. Násobení matic totiž není komutativní operací. Jakákoliv transformace bodu, nebo vektoru v 3D prostoru může být vyjádřena pomocí 4×4 matice.

$$Mv = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} m_{00}v_0 & m_{01}v_1 & m_{02}v_2 & m_{03}v_3 \\ m_{10}v_0 & m_{11}v_1 & m_{12}v_2 & m_{13}v_3 \\ m_{20}v_0 & m_{21}v_1 & m_{22}v_2 & m_{23}v_3 \\ m_{30}v_0 & m_{31}v_1 & m_{32}v_2 & m_{33}v_3 \end{bmatrix} \tag{2.18}$$

Translace
Translace je lineární transformací, která posouvá bod v prostoru. Translační matice vypadá následovně:

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

Uvedená translační matice posouvá bod s ofsetem, který je reprezentován pomocí vektoru (t_x, t_y, t_z). Diagram 2.4 znázorňuje translaci bodů krychle násobením s následující maticí:

$$T(4, 5, 0) = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

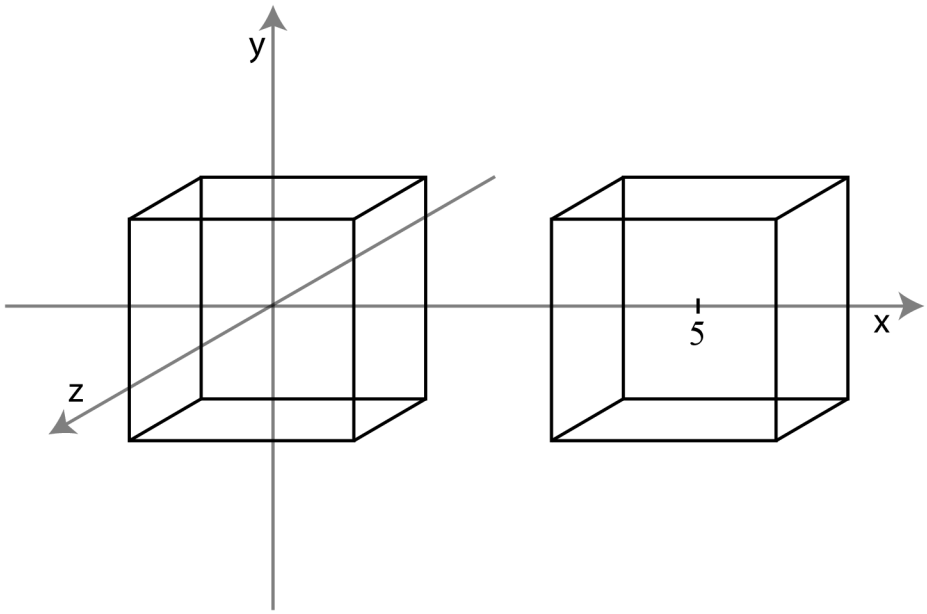


Diagram 2.4: Translace

Rotace

Tato transformace rotuje bod, či vektor o zadaný úhel kolem počátku souřadného systému $[0, 0, 0]$. Běžně se používají rotační matice pro rotaci kolem osy[9] x, y a z.

[9]Rotace se řídí tzv. pravidlem pravé ruky. Palec směřuje v kladném směru osy a zbylé zatočené prsty ruky naznačují směr kladné rotace.

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.21}$$

$$R_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.22}$$

$$R_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

Diagram 2.5 znázorňuje rotaci bodů krychle kolem počátku soustavy souřadnic s využitím této transformační matice:

$$R_x(30°) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(30°) & -\sin(30°) & 0 \\ 0 & \sin(30°) & \cos(30°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

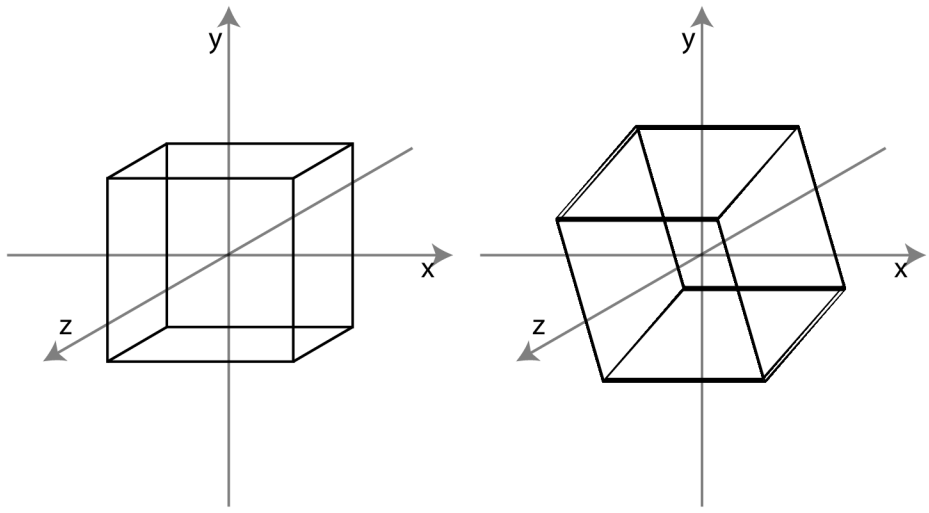


Diagram 2.5: Rotace

Dalšími transformacemi používanými ve 3D grafice je tzv. scaling, který zvětšuje, či zmenšuje objekt a pak tzv. shearing, který dokáže objekt zkosit dle dané osy. Tyto dvě transformace však nejsou při implementaci hry využity, a tudíž zde nebudou zmíněny.

2.5 WebGL

WebGL je nízkoúrovňové aplikační rozhraní pro zobrazení pokročilé 3D grafiky na webu. Je založeno na OpenGL ES 2.0 a umožňuje programátorovi použít hardwarově akcelerované[10] vykreslování obrazu v kontextu HTML a JavaScriptu. Vykreslovací plochou, která je zde použita je HTML5 `<canvas>` element a jeho `webgl`, resp. `experimental-webgl` kontext. [6]

Historie WebGL

První experimenty s 3D grafikou v `<canvas>` elementu prováděl Vladimir Vukićević ze společnosti Mozilla. Výsledkem jeho pokusů se stal prototyp, který nazval Canvas 3D. V roce 2009 vytvořila Khronos Group novou pracovní skupinu pro WebGL, která byla složena z hlavních tvůrců webových prohlížečů včetně firem jako Apple, Google, Mozilla a Opera. Khronos Group je neziskovou organizací, která vytváří a spravuje otevřené standardy a aplikační rozhraní. Byla založena roku 2000 a mimo jiné stojí i za standardy jako OpenGL, či výše zmíněným OpenGL ES, které je primárně určeno pro vestavěné systémy. [1]

Finální specifikace WebGL 1.0[11] byla vydána v březnu roku 2011 a její implementaci můžeme nalézt v prohlížečích jako Google Chrome, Mozilla Firefox, Safari a v počátečních fázích implementace v prohlížeči Opera. V případě Microsoft Internet Exploreru je situace poněkud odlišná. Microsoft neohlásil žádný záměr v podpoře WebGL ve svém prohlížeči. Uživatelé, kteří chtějí WebGL používat, jsou tedy nuceni přejít k jinému prohlížeči. [1]

Výhody WebGL

V dobách, kdy web jako takový začínal, byl jeho obsah vytvořen ze statických dokumentů. Hlavní prací webových prohlížečů tehdy bylo takovýto obsah získat a zobrazit. V průběhu času však webové technologie značně pokročily a nyní tak jejich prostřednictvím můžeme přistupovat k plnohodnotným webovým aplikacím s bohatým uživatelským rozhraním a audiovizuálním obsahem. Tyto aplikace se nyní stávají alternativou k aplikacím nativním. [1]

Jejich hlavními výhodami jsou:

- Rychlá **dostupnost** a jednoduchá **distribuce** mezi mnoho uživatelů.
- **Snadná údržba a aktualizace aplikací.** Pokud je v aplikaci nalezena chyba, nebo pokud chceme rozšířit její funkcionalitu, pak vše, co je potřeba, je aktualizovat aplikaci na serveru a uživatelé mají ihned přístup k její nové verzi.
- **Multiplatformnost aplikací.** Vše, co uživatel potřebuje je kompatibilní webový prohlížeč schopný zobrazit námi definovaný obsah.

Oproti nativním aplikacím nejsou ty webové obsahem tak bohaté, avšak s příchodem HTML5 začíná tento rozdíl mizet. Prostřednictvím WebGL je nyní možné zobrazovat hardwarově akcelerovanou grafiku přímo uvnitř prohlížeče. Je tak možné vytvořit 3D hry, nebo pokročilé 3D grafické aplikace a zároveň těžit z výhod webu, jak byli popsány výše. Technologie WebGL má navíc i další výhody, mezi něž patří například:

- WebGL je otevřený standard, který může používat každý bez poplatků.

[10] K hardwarové akceleraci je nutno vlastnit GPU s podporou minimálně shader modelu 2.0. V opačném případě lze obraz vykreslovat softwarově.

[11] http://www.khronos.org/registry/webgl/specs/1.0/

- WebGL využívá přímo grafický hardware, což znamená, že jsou aplikace rychlé.
- Vzhledem k tomu, že je WebGL založeno na OpenGL ES, je možné vytvořené aplikace spouštět i na mnohých moderních mobilních zařízeních.

Grafická API

Existují dva fundamentální přístupy, které jsou u grafických aplikačních rozhraní využity. Jsou jimi:

- immediate-mode API a
- retained-mode API,

přičemž WebGL používá první ze zmíněných přístupů.

Immediate-mode API

U tohoto typu rozhraní je celá scéna překreslena s každým snímkem bez ohledu na to, zda byla změněna, či nikoliv. Grafická knihovna která zprostředkovává rozhraní programátorovi neukládá žádnou interní reprezentaci scény, která má být vykreslena. Reprezentaci scény je tak nutné uchovávat v paměti vlastní aplikace. Tento přístup je vysoce flexibilní a poskytuje programátorovi větší úroveň kontroly nad výsledným vykreslovaným obrazem. Na druhou stranu však tento přístup vyžaduje ze strany programátora více úsilí oproti přístupu, který si popíšeme nyní.

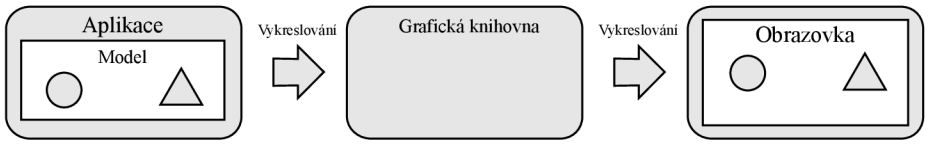


Diagram 2.6: Immediate-mode API

Retained-mode API

U tohoto přístupu je interní model a veškeré vykreslované objekty scény obsažen v grafické knihovně, která toto rozhraní programátorovi nabízí. Programátor využívá přístupových metod k rozhraní a knihovna sama rozhoduje o tom, zda má být scéna a s ní její interní reprezentace aktualizována, či nikoliv. To znamená, že aplikace, která rozhraní využívá, nemusí v každém vykreslovaném snímku scénu překreslovat. Tento přístup je v mnohých ohledech pro programátora méně náročný a používá se například pro vykreslování vektorové grafiky (SVG).

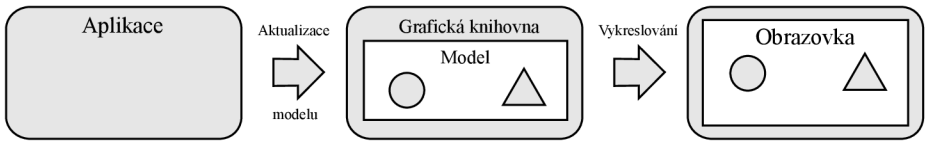


Diagram 2.7: Retained-mode API

Tvorba obrazu v grafickém hardwaru

WebGL je nízkoúrovňové rozhraní a pracuje tak velmi blízko grafického hardwaru. K pochopení fundamentálních konceptů použitých při implementaci hry je tedy nejprve potřeba alespoň okrajově osvětlit, jakým způsobem grafický hardware pracuje.

Na diagramu 2.8 je zjednodušeně znázorněn vztah grafického hardwaru k ostatním částem systému.

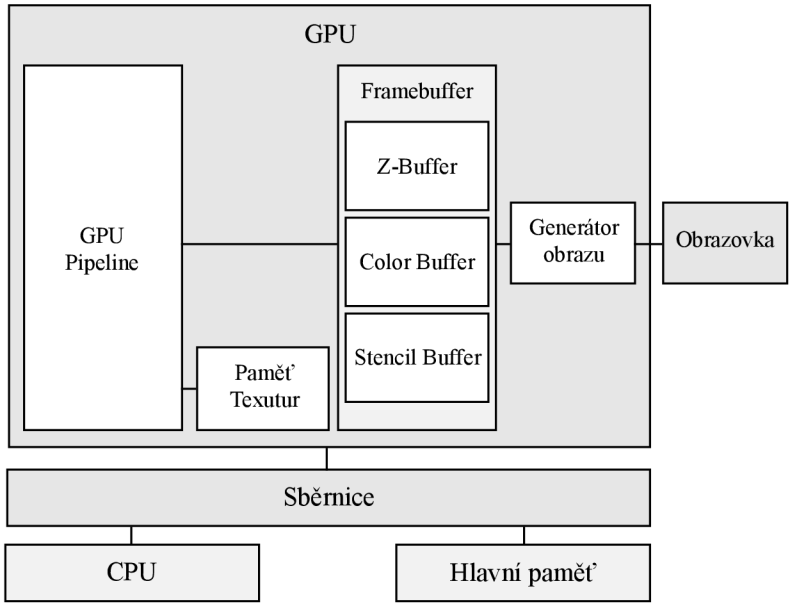


Diagram 2.8: Vztah grafického hardwaru k ostatním částem systému

GPU

GPU, neboli také Graphics Processing Unit, je dedikované zařízení, které je přímo navrženo pro zobrazení grafiky. Architektura GPU je vysoce paralelizovaná a provádí operace s grafickými daty velmi rychle. Zpracování dat probíhá typicky zřetězeně v několika úrovních.

Framebuffer

Framebuffer je místem, kde je uložen výsledek operací provedených na GPU. Je to paměť, která obsahuje informace nutné pro zobrazení výsledného obrazu na zobrazovací zařízení. V jednoduchém grafickém systému může být fyzická paměť framebufferu součástí hlavní operační paměti, avšak u moderních systémů je tato paměť alokována ve speciální rychlé grafické paměti na GPU. Framebuffer se typicky skládá ze tří odlišných *subbufferů*.

- Color Buffer
- Z-Buffer
- Stencil Buffer

Color Buffer

Color buffer představuje dvourozměrné pole, jehož prvky jsou výsledné barvy pixelů obrazu. Každý z těchto prvků obsahuje informaci o výsledné barvě v RGB, či RGBA formátu.

Každý z barevných kanálů má alokován určitý počet bitů a navíc může být obsažena informace o *alpha* kanálu, který určuje viditelnost daného pixelu. Celkový počet bitů použitých pro jeden pixel je označován jako barevná hloubka (color depth).

Varianty barevné hloubky:

- 16 bitů na pixel
- 24 bitů na pixel
- 32 bitů na pixel

Barevná hloubka 16 bitů se často používá na menších zařízeních. Je zde použito systému RGB565, který zohledňuje zvýšenou citlivost lidského oka na zelenou barvu. Rozložení jednotlivých barevných kanálů tedy není rovnoměrné a je alokováno 5 bitů pro červenou barvu, 6 bitů pro barvu zelenou a 5 bitů pro barvu modrou. Barevná hloubka 24 bitů má pak pro každou barvu alokováno po osmi bitech a v případě barevné hloubky 32 bitů je dalších 8 bitů alokováno pro alpha kanál.

Z-Buffer
Jak již bylo uvedeno v předchozím odstavci, color buffer typicky obsahuje barvy pixelů výsledného obrazu. V zobrazované scéně jsou však některé vykreslované objekty překryty jinými a pixely, které těmto zakrytým objektům náleží, by tak neměly být viditelné. Toho je docíleno pomocí z-bufferu, který má stejný počet prvků jako color buffer, avšak není zde uložena barva, nýbrž vzdálenost pozorovatele od nejbližšího objektu scény. Ta následně rozhoduje o tom, jaký objekt má být v daném pixelu vykreslen a který nikoliv. V souvislosti s implementací je tento buffer využíván pouze v případě, že objekty scény nejsou zprůhledněny.

Stencil Buffer
Jako doplněk ke dříve popsaným bufferům je na moderním grafickém hardwaru přítomen i stencil buffer, který určuje to, kam má být aktuálně zpracovávaný objekt scény vykreslen v rámci color bufferu. V praxi je tento buffer využit například pro vykreslování stínů. Stíny nejsou v implementaci použity, proto se již dále stencil bufferem nebudeme zabývat.

Grafická pipeline

Webové aplikace jsou typicky složeny z HTML, CSS a JavaScriptových souborů, které jsou interpretovány a zobrazovány prohlížečem. Aplikace využívající WebGL navíc obsahuje zdrojový kód shaderů a data, která reprezentují 3D scénu. JavaScript využívá rozhraní WebGL, aby předal této knihovně zdrojové kódy programovatelných součástí grafické pipeline a data, která reprezentují vykreslovaný 3D model. Poté, co jsou data knihovně předána, je výsledek vykreslen do tzv. *drawing bufferu*, což je ve své podstatě framebuffer pro WebGL. Stejně tak jako framebuffer obsahuje color, stencil a z-buffer, avšak jeho obsah je nejdříve spojen se zbytkem webové stránky a teprve poté končí ve framebufferu fyzickém. V implementaci hry je využito 32 bitové varianty drawing bufferu a je tak tedy možné využít alpha kanálu na prolínání vykreslované 3D grafiky se svým okolím, resp. se zbytkem webové stránky. Grafická pipeline je zobrazena na diagramu 2.9.

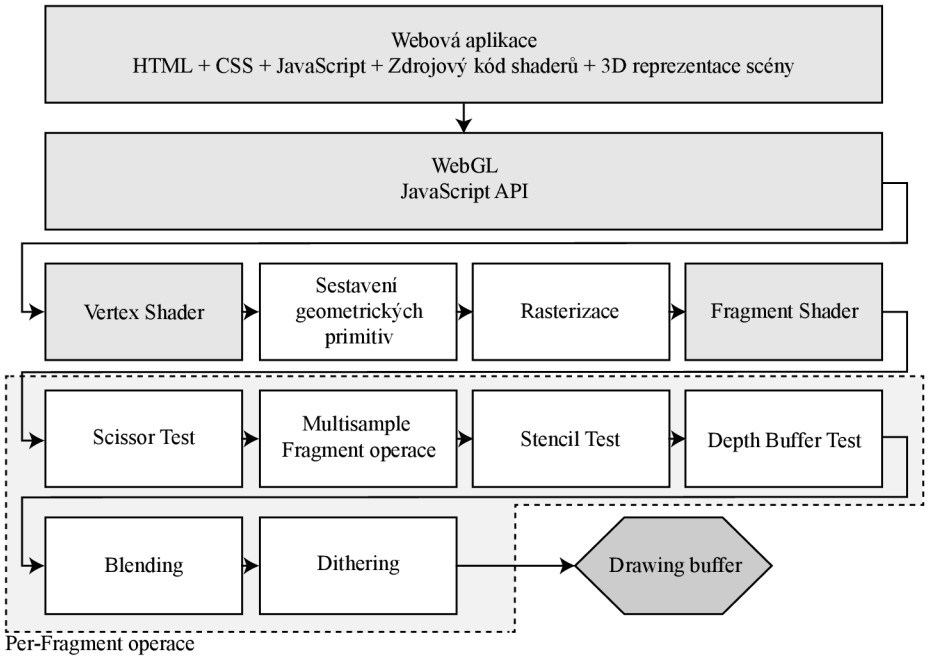


Diagram 2.9: GPU pipeline

K tomu, abychom získali realistickou 3D scénu nestačí pouze vykreslit objekty na určité pozice. Musíme také vzít v úvahu, jak budou objekty vypadat, pokud na ně bude dopadat světlo ze světelných zdrojů scény. Obecně se technice, která se používá ve spojitosti s působením světla na různé typy materiálů říká *shading*. Ve WebGL je shading rozdělen do dvou částí, jejichž chování je možné naprogramovat.

Programovatelnými součástmi grafické pipeline jsou:

- Vertex Shader
- Fragment Shader

Vertex Shader
První částí grafické pipeline, do které vstupují data předaná WebGL knihovně, je vertex shader. Jak již jeho název napovídá, provádí shading jednotlivých vertexů vykreslované scény. Než však samotný proces shadingu započne, je nutno vertexy transformovat a umístit tak objekt, jemuž vertexy náleží, na požadovanou pozici. Toho je docíleno pomocí transformačních matic, které jsou popsány v podkapitole 2.4. Vertex shader používá následující vstupy:

- Zdrojový kód vertex shaderu, který je zapsán v OpenGL ES Shading Language (GLSL ES).
- Atributy, které definují data specifická každému vertexu. Typicky je to jeho pozice, normála a barva.
- Tzv. *uniform* proměnné, které jsou pro všechny vertexy objektu konstantní a lze je změnit až po dokončení operace vykreslování. Jedná se o transformační matice a dále pak například o pozice, na kterých se nachází světelné zdroje.

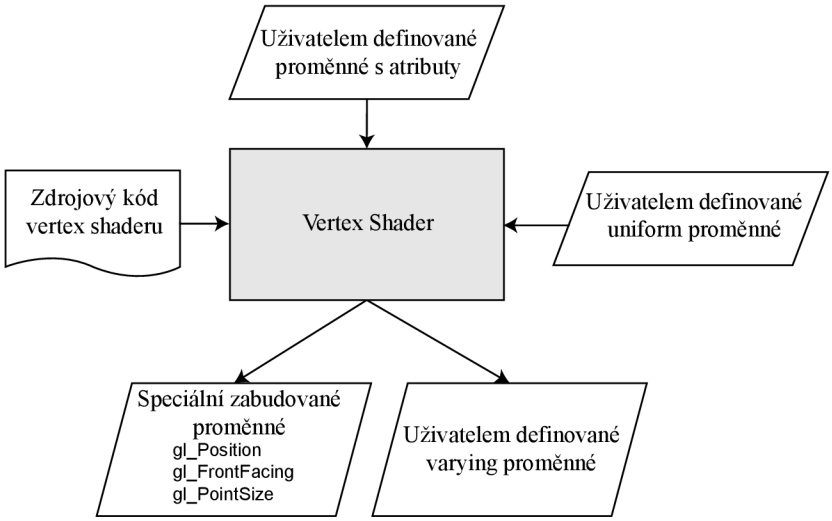


Diagram 2.10: Vertex Shader

Na diagramu 2.10 jsou pak jestě znázorněny zabudované a uživatelky definované *varying* proměnné. Využití varying proměnných je v delegaci informací z vertex shaderu do fragment shaderu. Jednou z nejdůležitějších speciálních zabudovaných proměnných je `gl_Position`, která po práci vertex shaderu udává pozici, na které se vertex nachází. Popis vertex shaderu probíhá pomocí jazyka GLSL, který je svou syntaxí podobný programovacímu jazyku C. Příklad takového popisu je uveden ve zdrojovém kódu 2.4

```
1  // Deklarace atributů vertexu
2  // Vektor pozice vertexu (XYZ)
3  attribute vec3 aVertexPos;
4  // Barva vertexu (RGBA)
5  attribute vec4 aVertexColor;
6
7  // Uniform proměnné
8  // Model-View Matice (4x4)
9  uniform mat4 uMVMatrix;
10 // Projekční matice (4x4)
11 uniform mat4 uPMatrix;
12
13 // Deklarace varying proměnné obsahující výstupní barvu vertexu,
14 // která je vstupem pro fragment shader.
15 varying vec4 vColor;
16
17 void main() {
18   // Transformace vertexu projekční a model-view maticí, kde
19   // gl_Position udává jeho výslednou pozici ve scéně.
20   gl_Position = UPMatrix * uMVMatrix * vec4(aVertexPos, 1.0);
21
22   // Barva vertexu je poslána dále do fragment shaderu  .
23   vColor = aVertexColor;
24 }
```

Zdrojový kód 2.4: Ukázka zdrojového kódu GLSL pro vertex shader

Sestavení primitiv
V tomto kroku jsou sestaveny jednotlivé vertexy, které prošly skrze vertex shader, do geometrických primitiv, jakými jsou například trojúhelníky či hrany. Následně je potřeba rozhodnout o tom, zda je sestavený objekt v regionu, který je aktuálně viditelný na obrazovce. Tento region je označován jako *frustrum* a je představován komolým jehlanem s obdélníkovou, či čtvercovou podstavou. Objekt který je uvnitř frustra je předán ke zpracování dalším částem grafické pipeline. Objekty, které jsou kompletně mimo frustrum, se odstraní, a ty, které jsou vně pouze částečně, budou oříznuty. Frustrum je znázorněno na diagramu 2.11

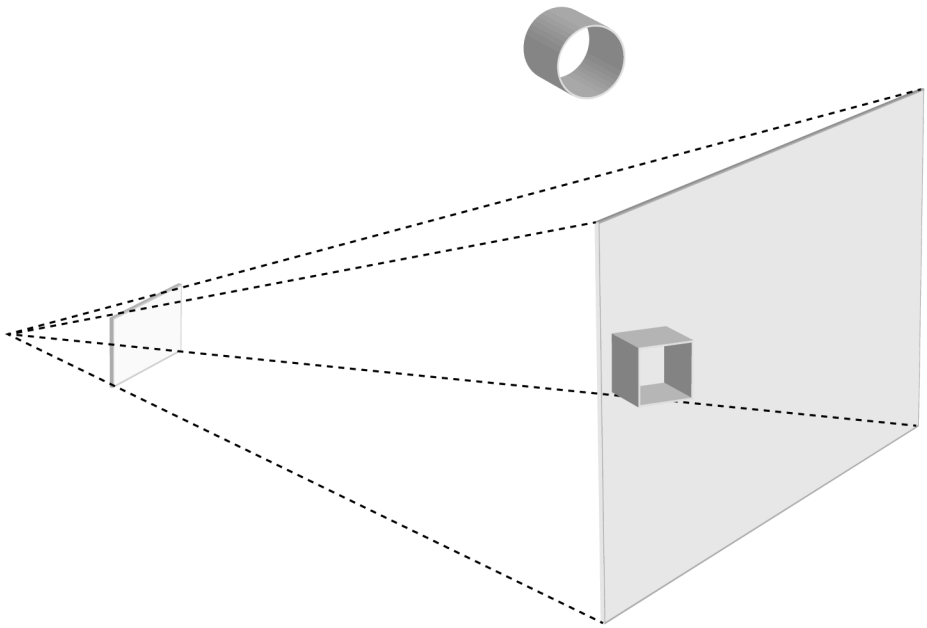


Diagram 2.11: Frustrum

Rasterizace
Dalším krokem je převod primitiv na fragmenty (diagram 2.12), pod kterými si můžeme představit jednotlivé pixely obrazovky. Fragmenty dále putují do fragment shaderu.

Fragment shader
Fragment shader je druhou programovatelnou součástí grafické pipeline. Jak již bylo zmíněno, fragment odpovídá pixelu, avšak ne všechny fragmenty se pixely stanou. Fragmenty totiž mohou být odstraněny v dalších částech pipeline. Fragment shader se svými vstupy a výstupy je znázorněn na diagramu 2.13. Vstupem fragment shaderu jsou:

- Zdrojový kód fragment shaderu v jazyku OpenGL ES Shading Language.
- Speciální zabudované proměnné, mezi něž patří například `gl_PointCoord`.
- Varying proměnné, které byli poslány skrze vertex shader.
- Uniform proměnné, které obsahují konstanty společné všem fragmentům vykreslovaného objektu.

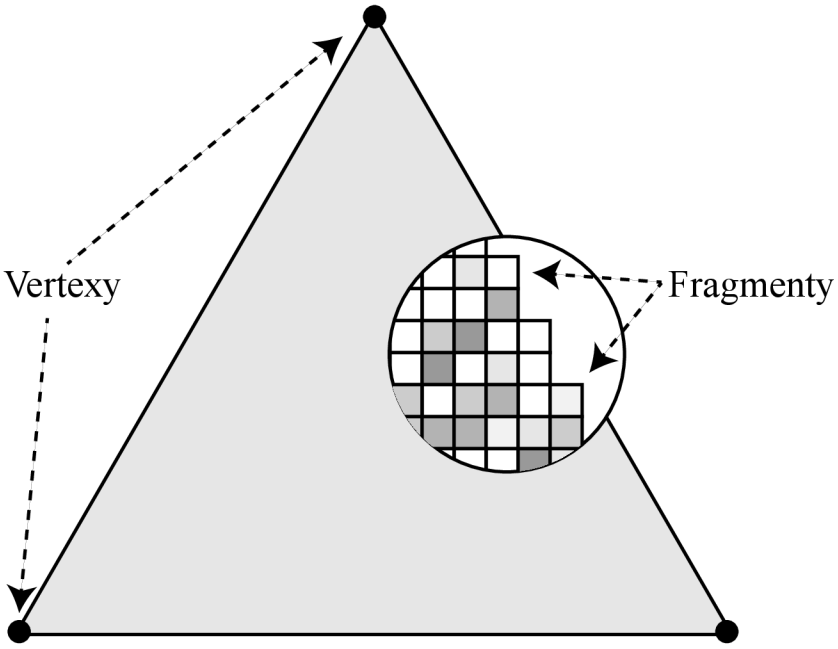


Diagram 2.12: Fragmenty

- Samplery, což jsou speciální uniform proměnné určené pro texturování.

Jak již bylo zmíněno dříve, varying proměnné slouží k posílání informací skrze vertex shader. Obecně však platí, že objekt má více fragmentů než vertexů. Obsah varying proměnných, který je zaslán skrze vertex shader, je lineárně interpolován pro každý fragment.

Výsledek práce fragment shaderu je zapsán do zabudované proměnné `gl_FragColor`, která následně obsahuje výslednou barvu daného fragmentu. Ve zdrojovém kódu 2.5 je uveden příklad GLSL programu fragment shaderu, který pro každý fragment získá lineárně interpolovanou hodnotu s jeho barvou a uloží ji jako svůj výstup.

```
1 varying ver4 vColor;
2 void main(){
3   gl_FragColor = vColor;
4 }
```

Zdrojový kód 2.5: Příklad jednoduchého GLSL programu fragment shaderu

Per-Fragment operace

Každý fragment, který projde fragment shaderem, je postoupen do dalšího bloku pipeline, která se skládá z několika částí provádějící tzv. per-fragment operace. Každá z operací může ovlivnit výsledný pixel v drawing bufferu, avšak v implementaci je využito pouze blendingu a depth buffer testu. Zbylé části jsou uvedeny pro úplnost popisu pipeline.

Scissor test

Scissor test určuje, zda je zpracovávaný fragment uvnitř pravoúhlého rovnoběžníku, který je definován jedním bodem, výškou a šírkou. Fragmenty mimo tento rovnoběžník jsou zahozeny, ostatní pokračují dále v cestě grafickou pipeline.

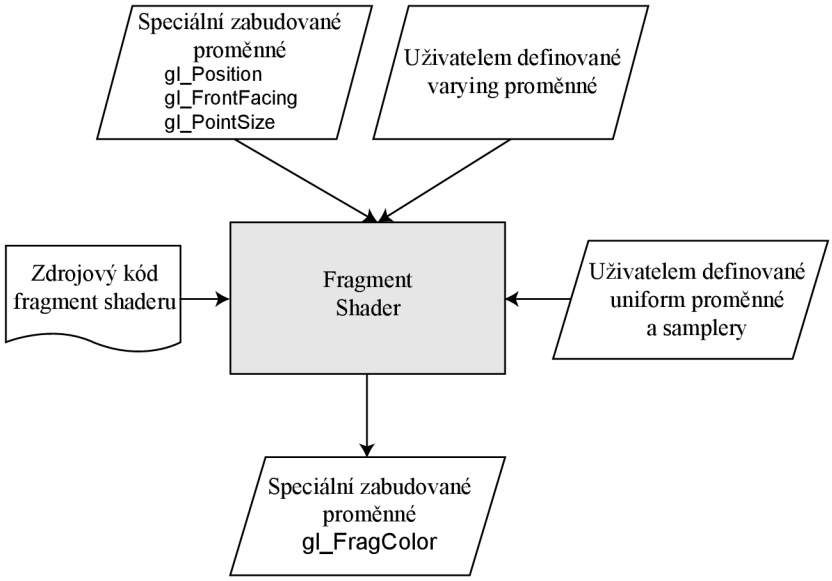


Diagram 2.13: Fragment Shader

Multisample fragment operace
Tato část pipeline modifikuje aplha kanál fragmentu, čímž je docíleno vyhlazení hran vykreslovaných objektů. Tato technika se obecně označuje jako *anti-aliasing*.

Stencil test
Zde se fragment porovnává s nastavenou referenční hodnotou. Na základě výsledku porovnání je fragment opět zahozen, nebo postoupen dále.

Depth buffer test
Vzhledem k tomu, že se v téměř každé vykreslované scéně objekty překrývají, je nutné do color bufferu vykreslovat pouze ty objekty, které jsou viditelné. Tento test ve spolupráci s depth bufferem rozhoduje o tom, zda fragment vykreslovat, či nikoliv.

Blending
V dalším kroku označovaným jako blending jsou kombinovány barvy fragmentů, které jsou momentálně vykresleny do color bufferu na odpovídající pozici. Toho je v implementaci využito pro vykreslování průhledných objektů scény.

Dithering
Posledním krokem před vykreslením do color drawing bufferu je tzv. dithering. Vzhledem k tomu, že color buffer má omezený počet bitů k reprezentaci každé z barev, dithering tyto složí tak, že vytvoří iluzi toho, že máme k dispozici barev více.

Kapitola 3

Koncepce hry

Návrh implementované hry zakládá na analýze hry Berušky 2, která byla provedena metodou *černé skříňky*[1]. Hra je nejprve stručně představena a následně jsou uvedeny výsledky herní analýzy, na kterých zakládá implementace 4

3.1 Berušky 2

Berušky 2, neboli také Berušky 3D, jsou pokračováním logické hry vývojářského týmu Anakreon[2], jehož členem je pan Ing. Martin Stránský, který byl konzultantem této bakalářské práce. Oproti své první, volně dostupné verzi, bylo toto pokračování od počátku vytvářeno jako komerční produkt. Hra byla od roku 2004 distribuována v České republice a některých dalších zemích společností Cinemax. V březnu roku 2011 byla část této hry uvolněna pod open-source licencí a je dále vyvíjena panem Stránským. První verze hry Berušky je svou koncepcí velmi podobná hře Sokoban [3]. Druhá verze hry se svou koncepcí příliš od prvního dílu neodlišuje, avšak do hry přibyly nové herní prvky, a co je hlavní, hra je kompletně ve 3D. Každá úroveň této hry je logickou hříčkou, která ke svému řešení vyžaduje volbu správného plánu a dávku trpělivosti. Každá z berušek má schopnost před sebou tlačit bedny a používat herní předměty, čímž vytváří cestu k cíli, avšak pro jeho dosažení je často důležité, aby spolu berušky vzájemně spolupracovali.

Původní hra distribuovaná firmou Cinemax obsahuje celkově 160 herních úrovní včetně 20 tutoriálů a 45 jednodušších úrovní, které jsou určeny pro mladší hráče a trénink. Hlavní součástí hry je pak *Beruščí cesta*, která obsahuje celkem 95 úrovní rozdělených do 9 epizod odehrávajících se v různých prostředích. Open-source verze hry pak obsahuje tutoriály, některé tréninkové úrovně a 3 z úrovní Beruščí cesty.

3.2 Herní pole

Herní pole je ve tvaru krychle či kvádru a je rozděleno na jednotlivé pozice, ve kterých jsou umístěny jeho prvky (diagram 3.1). Nikdy nemůže dojít k situaci, že by se herní prvek včetně berušek vyskytl mimo herní pole. Po prvcích jako jsou bedny, či zeď je možné se volně pohybovat, avšak beruška nikdy nemůže z vyšší herní pozice seskočit dolů.

[1] Na základě akcí uživatele byla zkoumána reakce hry s tím, že princip vnitřní implementace zůstal utajen.
[2] www.anakreon.cz
[3] https://en.wikipedia.org/wiki/Sokoban

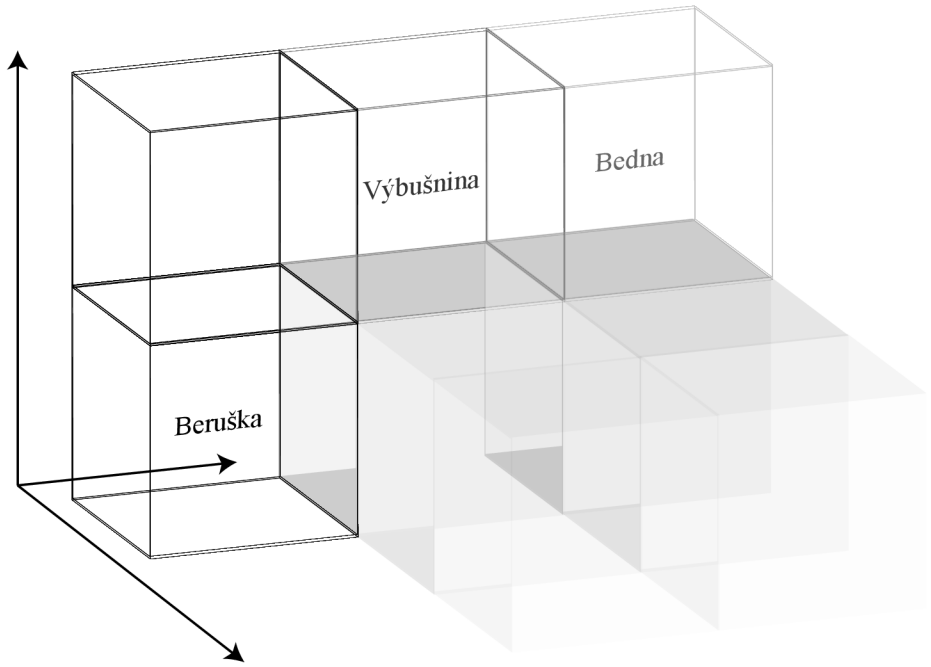


Diagram 3.1: Herní pole a jeho prvky

3.3 Prvky herního pole

Jak již bylo uvedeno, v rámci herního pole jsou umístěny prvky, jejichž rozmístění představuje samotnou logickou hříčku, kterou hráč řeší. Prvky se od sebe liší nejen funkčností a vzhledem, ale také tím, že některé z nich jsou umístěny staticky (hráč s nimi nemůže pohnout) a některé z nich jsou dynamické (bedny, výbušniny, ...). Následuje přehled prvků herního pole a jejich stručný popis.

Beruška
V každé herní úrovni se vyskytuje 1-5 berušek, které jsou ovládány hráčem. Berušky mohou pohybovat bednami, či výbušninami za účelem vytvoření cesty k východu. Každá z nich má inventář, který obsahuje předměty, které beruška při svém pohybu herním polem získala.

Zeď
Zeď je jedním ze statických prvků, který má ve svém herním poli stálou pozici a nelze ho nijak odstranit. Beruška se po zdech může volně pohybovat.

Východ
Cílem je dostat všechny berušky do východu. Stejně jako zeď je i východ umístěn stále na stejné pozici.

Bedna
Bedna je základním herním prvkem, který beruška tlačí před sebou a vytváří tak cestu k cíli. Počet přesouvaných beden je závislý na aktuální síle berušky (podkapitola 3.4) a je možné je odstranit pomocí výbušniny. Bedny lze také tlačit po šikmé podlaze a dostat je

tak do vyšší, či nižší úrovně herního pole. Podle váhy pak rozlišujeme bedny na lehké a těžké.

Výbušnina
Výbušnina je prvkem, který se po většinu času chová jako obyčejná bedna, avšak pokud je „natlačena“ na některou z beden, pak dojde k výbuchu. Bližší popis toho, jak výbuchy probíhají, je v podkapitole 3.6.

Kámen
Kámen představuje překážku, kterou nelze posunout, avšak je možné ho odstranit pomocí krompáče, který může beruška nalézt při průchodu herním polem.

Voda
Voda je pro berušku další překážkou. Pokud je v dané herní úrovni voda, pak musí beruška s největší pravděpodobností pod vodní hladinu, kde získá potřebný předmět. Někdy se dokonce pod vodní hladinou nachází i východ z úrovně, avšak v každém případě beruška potřebuje šnorchl, aby se mohla potopit. Ten může stejně jako krompáč získat při průchodu herním polem.

Krompáč
Krompáč je předmět, který se používá pro odstranění kamene. Maximální počet krompáčů, který může mít jedna beruška v inventáři, je 4. Po použití je krompáč odstraněn z inventáře.

Šnorchl
Beruška ho potřebuje, aby se mohla potopit pod vodní hladinu. Pokud beruška tento předmět nemá ve svém inventáři, pak je zamezeno jakémukoliv hernímu kroku, kterým by se beruška mohla pod hladinu dostat.

Závaží
Závaží dvojnásobně zvyšuje váhu berušky. Toho se využívá v situacích, kdy potřebujeme, aby se pod beruškou propadla podlaha.

Hormonální vitamín
Pokud beruška získá hormonální vitamín, pak získá dvojnásobnou sílu, což ji následně umožňuje před sebou tlačit více beden.

Bortící se podlaha
Občas se hře vyskytuje i podlaha, která se propadá pod vahou, která je nad ní naskladněna. Bližší informace o vahách herních prvků jsou uvedeny v podkapitole 3.4.

Šikmina
Šikmina je posledním z herních prvků. Umožňuje berušce sestoupit, či vystoupit z/do vyšší úrovně herního pole. Zároveň je možné po šikmině pohybovat bednami a výbušninami.

Prvek	Váha
Beruška	1
Lehká bedna	1
Těžká bedna	2
Výbušná bedna	2

Tabulka 3.1: Váhy dynamických prvků

3.4 Váhy herních prvků a síla berušky

Ve hře hraje velkou roli váha, která je přiřazena každému z prvků. Ta rozhoduje o tom, zda je beruška schopná posunout prvky, které se před ní nacházejí, a také o tom, zda se pod nimi nepropadne podlaha. Základní síla berušky, resp. to, kolik váhy před sebou může tlačit, jsou 2 váhové jednotky. S hormonálním vitamínem ve svém inventáři je pak síla berušky navýšena na 4 váhové jednotky. Bortící se podlaha nad sebou udrží pouze 1 váhovou jednotku a může na ni tedy být natlačena lehká bedna, nebo si na ni může stoupnout beruška, která ve svém inventáři nemá závaží. Při překročení váhy se podlaha propadne a objekty, které na ní byly naskládány, změní svou vertikální pozici tak, aby pod sebou měly podklad. Zajímají nás pouze váhy prvků, se kterými je možné ve hře pohybovat. Přehled dynamických prvků a jejich vah je uveden v tabulce 3.4.

3.5 Posuvy beden

Bedny jsou ve hře na různých pozicích a často jsou umístěny za sebou. O tom, zda beruška může provést posuv jedné či více beden rozhoduje více faktorů.

- Aktuální beruščina síla.
- Obsah pozice za poslední posouvanou bednou.
- Součet vah posouvaných beden.

Pokud uvažujeme posuv jedné jediné bedny, pak je situace jednoduchá. Zjistí se obsah pozice, kam má být bedna posunuta, a pokud je tato pozice prázdná, pak se posun provede. Při posuvu více beden najednou je potřeba spočítat souhrnnou váhu posouvaných beden a určit obsah pozice, na kterou bude posunuta od berušky nejvzdálenější z nich. Jakmile má beruška dostatečnou sílu k posunu a konečná pozice je prázdná, pak je posun proveden. V opačném případě zůstává beruška i bedny na svém původním místě.

Je také důležité zdůraznit, že bedny mohou být posunuty do míst, kde pod sebou nemají žádný podklad. V takových situacích je pozice těchto beden upravena tak, aby pod sebou měly nějaký prvek.

3.6 Výbušné bedny

Výbušnými bednami lze ve většině situací normálně pohybovat. Změna nastává v případě, kdy je před výbušninou bedna obyčejná. V takové situaci je na ni výbušná bedna „nasunuta“ a dochází k výbuchu. Při výbuchu jsou obě bedny odstraněny a je upravena vertikální pozice

prvků, které se nad nimi před výbuchem nacházely. Posouvanými prvky mohou být další výbušné bedny a ty při posunu vždy odstraní obyčejné, které se nacházejí pod nimi.

Na diagramu 3.2 je ukázka komplexního výbuchu. Váha beden zde značně přesahuje beruščinu sílu, avšak za výbušnou bednou do které beruška tlačí je bedna obyčejná. Dojde tedy k výbuchu, odstranění beden a „pádu“ těch, které se nad nimi nacházely. Při pádu jsou odstraňovány bedny, které nad sebou mají výbušninu a zbude pouze jediná. Ta se nakonec bude nacházet přímo před beruškou, která zůstává na své původní pozici.

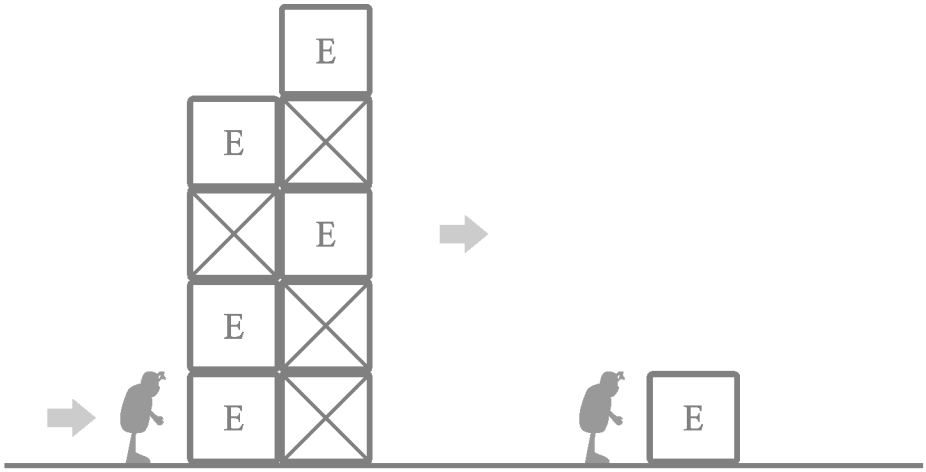


Diagram 3.2: Ukázka komplexního výbuchu

3.7 Ovládání

Berušky jsou vybírány pomocí kláves 1 až 5 a je s nimi pohybováno pomocí šipek. Důležité je ovládání kamery, která rotuje kolem momentálně vybrané berušky pohybem myši na okraj herní obrazovky. Pokud nemá uživatel přímou viditelnost na vybranou berušku, pak může objekty, které se mezi ním a beruškou nacházejí, nechat zprůhlednit pomocí mezerníku. Tlačítkem Enter je pak pozice kamery přesunuta nad herní pole.

3.8 Vykreslování

Informace týkající se technologie vykreslování jsou uvedeny v následujícím odstavci. Jejich přítomnost nechť čtenář bere pouze jako zajímavost, jelikož implementace hry, která je součástí této práce, probíhala nezávisle na hře původní. Jediným převzatým materiálem byla, jak se čtenář dále dozví, data pro zobrazení herní úrovně.

V původní hře je celá scéna organizována jako strom hierarchických OBB obálek. Při vykreslování je pak strom procházen a je zjišťována viditelnost jednotlivých obálek. Osvětlovací model je kombinovaný z per-vertex shadingu a lightmap. Hra obsahuje také mnohé animace, které jsou u berušek realizovány jako objektové a pro zbytek objektů scény jako mesh animace. Aktuálně vyvíjená open-source varianta hry obsahuje také například zrcadlový rendering, kreslení odlesků, halo efekty, anisotropické filtrování textur, bump-mapping, komprimované textury a mip-mapping. Informace o uvedených technologiích vykreslování zde nebudou z důvodu omezeného rozsahu této práce uvedeny.

3.9 Návrh webu

Implementace této části nebyla přímou součástí této práce, a proto jejímu návrhu nebude věnován velký prostor. Návrh webu je možné vidět na diagramu B.1. Web je rozdělen na 3 části a jejich popis se nachází v následujících odstavcích.

Menu

Pomocí menu se volí kontexty obrazovky (viz. dále).

Obrazovka

Obrazovka je primárně určena k zobrazení aktuálně načtené herní úrovně. Pro ucelenost webu je však obrazovka využita i k zobrazení dalších informací a lze tedy rozpoznávat její jednotlivé kontexty. Těmi jsou:

- Hra
- Informace o hře
- Návod na hraní hry
- Popis ovládání hry

Při volbě herního kontextu se v rámci obrazovky zobrazí samotný element `<canvas>`, do kterého je vykreslována zvolená herní úroveň. V levém dolním rohu je zobrazen inventář aktuálně vybrané berušky a v rohu pravém je možné ovládat přehrávání herní hudby. Reakce na události vytvářené hráčem jsou zpracovány a o některých z nich je zobrazena notifikace, která tak dává hráči zpětnou vazbu.

Výběr herních úrovní

Tato část umožňuje uživateli vybrat z dostupných herních úrovní, které jsou následně vykreslovány do herního kontextu obrazovky. Mezi jednotlivými bloky pro výběr úrovně lze přecházet poklikem na šipky. Při jednom pokliku jsou úrovně posunuty o jeden blok a při pokliku dvojitém pak o bloky 4. Výběr by neměl obtěžovat hráče v okamžiku, kdy není potřeba. Proto je možné ho otevřít/skrýt pomocí tlačítka, které je umístěno ve spodní části obrazovky.

Kapitola 4

Implementace

V kapitole 3 byla analyzována hra Berušky 2, která se stala vzorem pro hru implementovanou v rámci této práce. Způsob, jakým je hra implementována, je uveden na diagramu 4.1. V následující podkapitolách budou popsány jeho jednotlivé části a tím i objasněna herní architektura.

TODO...jak byla testovana funkcnost

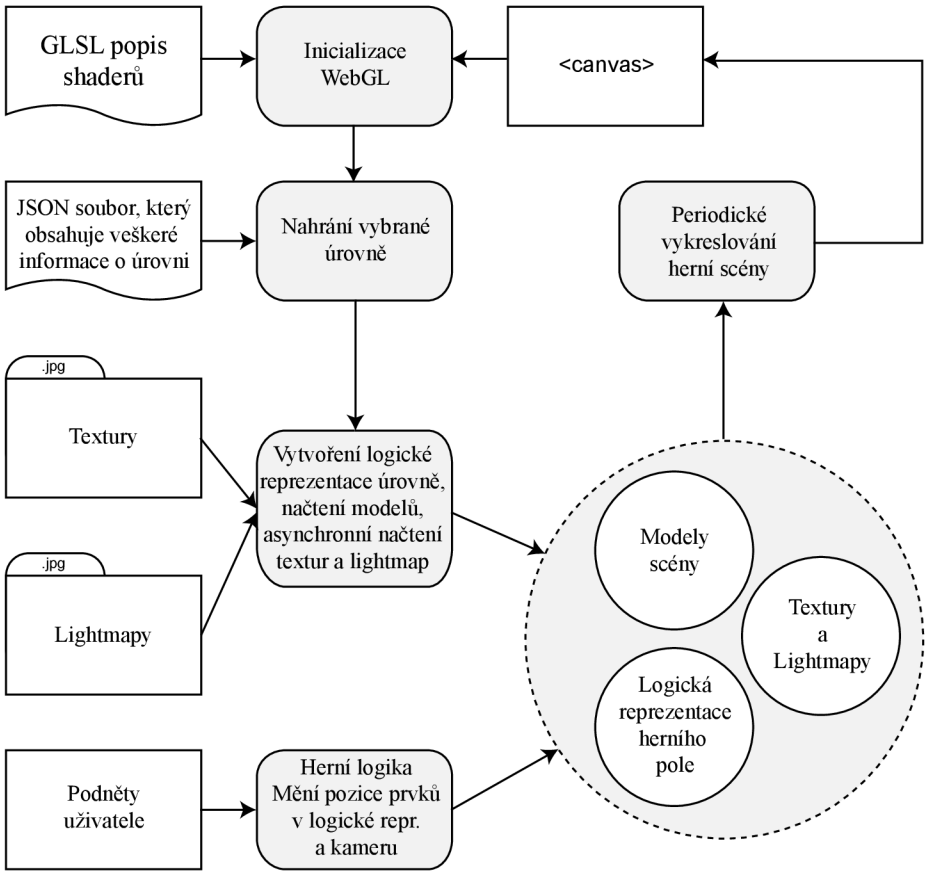


Diagram 4.1: Architektura hry Berušky 2 WebGL

4.1 Inicializace

Inicializace elementu `<canvas>` je prvním krokem, který je potřeba vykonat. Tím, že získáme referenci na jeho `webgl` kontext, můžeme přejít k dalšímu kroku, který nastaví vertex a fragment shadery. Popis funkčnosti shaderů je proveden pomocí jazyka GLSL, jehož ukázky byly uvedeny u popisu grafické pipeline (2.5). Zdrojové kódy jsou obsaženy v HTML dokumentu uvnitř těchto elemtentů:

- Vertex Shader
  `<script id="per-fragment-lighting-vs" type="x-shader/x-vertex">`
- Fragment Shader
  `<script id="per-fragment-lighting-fs" type="x-shader/x-fragment">`

V implementaci je hra inicializována pomocí funkce `webGLStart()`, která mimo jiné nastavuje způsob zpracování událostí vytvářených uživatelem.

4.2 Nahrání vybrané úrovně

Po inicializaci je přistoupeno k nahrání úrovně vybrané hráčem. Každá z úrovní je představována samostatným JSON souborem, který je asynchronně nahrán (2.2) ze serveru.

JSON soubor

Tento soubor obsahuje kompletní informace, které jsou potřebné pro zobrazení dané herní úrovně. Jedná se tak o hlavní součást celé hry, bez které by nemohla fungovat. Soubor vzniká exportem potřebných informací z původní hry a jeho hlavní součásti jsou popsány v následujících odstavcích.

Informace o materiálech

Materiály jsou použity k otexturování modelů. Rozdíl mezi materiálem a texturou je takový, že materiál se obecně může skládat z více textur, které se pak vzájemně prolínají. Lze tak mít například materiál, který vznikne složením z textur zdi a mechu. Výhoda je v tom, že složitější textury lze složit z jednodušších a není tedy potřeba uchovávat nadbytečná data. Každý z materiálů má v souboru jména textur, ze kterých se skládá, a také své unikátní jméno, pomocí něhož se následně objekty scény na tento materiál odkazují. Ukázka popisu materiálu je uvedena ve zdrojovém kódu 4.1.

```
1  {
2    "type" : "material",
3    "name" : "256_p-d1-256",
4    "transparent" : "0",
5    "z_buffer_mask" : "1",
6    "z_buffer_test" : "1",
7    "backface_culling" : "1",
8    "diffuse_color" : "1",
9    "specular_color" : "0",
10   "textures" : [ "s1_0013.jpg" ]
11 }
```

Zdrojový kód 4.1: Objekt `material` vstupního souboru JSON

Obálky objektů herní scény

Málokterý ze složitějších objektů je složen pouze z jednoho modelu. Ve chvíli, kdy celkový model objektu rozdělíme na části, je možné tyto části samostatně transformovat či měnit materiál, který používají. Avšak v okamžik, kdy chceme nějakým způsobem transformovat celý objekt, je vhodné mít všechny modely objektu v jakési obálce. Tato obálka má v souboru své identifikační číslo, které slouží k identifikaci všech jejích modelů. Identifikační číslo je potřebné pouze tehdy, pokud potřebujeme s obálkou transformace provádět a je tedy využito pouze u dynamických objektů herní plochy.

Modely

Modely jsou v souboru reprezentovány strukturami, které obsahují informace o poloze modelů v rámci scény, o jejich barvě či například o materiálech, které jsou jim přiřazeny. Některé z modelů mají také další textury s předpočítanými stíny pro realističtější zobrazení scény - lightmapy.

Ve zdrojovém kódu 4.2 je uveden zjednodušený popis obálky, která obsahuje 1 model. Identifikační číslo `2` znamená, že se jedná o dynamický prvek scény (pokud by se jednalo o statický prvek, pak by číslo bylo `-1`). Pomocí položky `material` se model v tomto případě odkazuje na materiál, který byl popsán ve zdrojovém kódu 4.1. Trojice prvků v poli `vertexPositions` vždy udává pozici vertexu ve scéně. O tom, které vertexy náleží geometrickým primitivám, ze kterých je model složen, rozhoduje pole `indices`. Zbylé položky pak obsahují informace potřebné pro správné namapování materiálů a lightmap.

```
{
  "type" : "geometry_container",
  "name" : "exit.b2m",
  "container_id" : "2",
  "poly_id" : "17",
  "prvek" : "1",
  "geometry_objects" : [
   {
    "name" : "exit.b2m",
    "material" : "256_p-d1-256",
    "vertexPositions" :
        [-27.586000,2.008000,-34.421001,-27.586000,0.008000,-36.421001,
         47.704437,85.772964,6.669896,130.704437,85.772964,-3.330104],
    "vertexNormals" :
        [-1.000000,0.000000,0.000000,-1.000000,0.000000,0.000000],
    "vertexTextureCoords0" :
        [1.000000,1.000000,0.000000,0.000000,1.000000,0.000000],
    "vertexTextureCoords1" :
        [1.000000,1.000000,0.000000,0.000000,1.000000,0.000000],
    "vertexTextureCoordsLight" :
        [0.140625,0.328125,0.171875,0.328125,0.156250,0.359375],
    "indices" : [0,1,2,3,0,5]
   }
  ]
}
```

Zdrojový kód 4.2: Objekt `geometry_container` vstupního souboru JSON

Logická reprezentace herního pole

Obálky modelů neobsahují žádnou informaci o tom, jaký typ objektu ve scéně představují. Z tohoto důvodu je nutné rozlišit to, co je vykreslováno na obrazovku, a s čím pracuje

Prvek	itemClass	itemSubclass
Beruška	1	0
Zeď	2	0
Východ	4	0
Bedna	5	0
Těžká bedna	5	0
Lehká bedna	5	1
Výbušnina	6	0
Kámen	7	0
Voda	12	0
Šnorchl	13	0
Hormonální vitamín	13	5
Závaží	13	7
Krompáč	13	8
Bortící se podlaha	15	0
Šikmina	19	0

Tabulka 4.1: Třídy a podtřídy prvků herního pole

logika hry. Jak již bylo uvedeno v kapitole 3, herní pole je krychlové či kvádrové a je složeno z jednotlivých pozic, na kterých se mohou nacházet herní prvky. Je to právě logická reprezentace herního pole, která obsahuje informace o tom, který prvek se na které pozici nachází. Každý prvek logické reprezentace herního pole má opět své identifikační číslo, pomocí kterého se odkazuje na obálku modelu, který určuje jeho vzhled.

Ve zdrojovém kódu 4.2 je uveden popis herního pole o velikosti 6×6 a výšce 8. Položka `level_start` udává pozici, na kterou mají být přesunuty dynamické objekty, které jsou normálně umístěny v počátku souřadného systému scény. Herní pole zde obsahuje pouze jeden prvek, jehož typ je určen položkami `class` a `subclass` (identifikátory všech prvků jsou uvedeny v tabulce 4.2). Logický prvek se v tomto příkladu pomocí `container_id` odkazuje na model, který byl popsán ve zdrojovém kódu 4.2.

```
{
  "type" : "level_logic",
  "logic_level_size" : [ 6, 8, 6],
  "level_start" : [ 21.586000, -1.992000, -46.421001],
  "item_size" : "2",
  "level_items_num" : "1",
  "level_items" : [{
    "name" : "Exit - teleport - 1",
    "guid" : "4000",
    "class" : "4",
    "subclass" : "-1",
    "position" : [ 0, 5, 1 ],
    "rotation" : "0",
    "container_id" : "2"
  }]
}
```

Jak si mohl pozorný čtenář všimnout, čísla, která označují typ prvku, nejdou sekvenčně za sebou. Je to způsobeno tím, že původní návrh hry Berušky 2 (desktopové verze) obsahoval více typů herních prvků, než jich bylo nakonec implementováno.

4.3 Vytvoření vnitřní reprezentace herní úrovně

Každé struktuře vyskytující se v JSON souboru (popsáno v 4.2) přísluší odpovídající objekty a po jeho asynchronním načtení je soubor zpracován pomocí funkce `handleLoadedJSON`. Ta tento soubor sekvenčně prochází, rozpoznává jeho jednotlivé struktury a následně pomocí JavaScriptových konstruktorů vytváří odpovídající objekty, se kterými hra pracuje. V následujících odstavcích je uveden přehled těchto konstruktorů s jejich stručným popisem.

`Material`
Při vytváření objektu tímto konstruktorem jsou ze serveru asynchronně (2.2) nahrány textury, které materiál využívá. Z těch jsou následně vytvořeny texturovací buffery, které jsou nahrány do grafické paměti. Tím, že jsou do této paměti umístěny, je pak dosaženo rychlejšího vykreslování celé herní scény. Buffery jsou nastaveny tak, že pokud je textura oproti své původní velikosti zvětšována (upscaling), pak se použije lineární filtr, který na základě okolních pixelů dopočítá lineární interpolací barvu pixelu mezi nimi. Pro textury, které jsou naopak zmenšovány je vygenerována mipmapa, ze které se pak vybírá nejvhodnější verze textury. Všechny materiály jsou uchovávány v asociativním poli `materials`, kde jednotlivé klíče tohoto pole jsou samotné názvy materiálů.

Ve zdrojovém kódu 4.3 je uveden příklad nahrání texturovacího bufferu vzniklého načtením první textury materiálu uvedeného ve zdrojovém kódu 4.1 do grafické paměti.

```
1  // ...
2  // Textura je zrcadlově obrácena kolem osy Y
3  // WebGL používá jiný souřadný systém
4  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
5  // Nastavení aktuálně zpracovávaného bufferu textury
6  gl.bindTexture(gl.TEXTURE_2D, materials["256_p-d1-256"].buffers[0]);
7  // Nahrání textury do grafické paměti
8  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
9                that.textures[temp].image);
10 // Nastavení filtru, kterým bude textura zvětšována
11 gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
12 // Nastavení filtru, kterým bude textura zmenšována
13 gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
14                   gl.LINEAR_MIPMAP_NEAREST);
15 // Vygenerování mipmapy
16 gl.generateMipmap(gl.TEXTURE_2D);
17 // ...
```

Zdrojový kód 4.3: Příklad nahrání textury do grafické paměti

`Lightmap`
Objekt vytvořený tímto konstruktorem je až na pár detailů stejný jako objekt konstruktoru `Material`. Lightmapy jsou taktéž načítány asynchronně a ukládány do paměti grafické karty, avšak vzhledem k tomu, že se z původní hry nepodařilo všechny lightmapy vyexportovat, je jejich použití vypnuto.

`GeometryContainer`
Tento konstruktor vytváří obálku jednotlivých modelů herní scény. Podle toho, zda obálka obsahuje identifikační číslo, rozlišujeme mezi obálkami dynamických a statických objektů scény a umísťujeme je do odpovídajících polí. Pro dynamické obálky je to asociativní pole

`dynamicItems`, jehož klíči jsou identifikátory obálek a pro statické obálky je to pole staticItems, kde jsou obálky seřazeny tak, jak byli načítány ze vstupního souboru. Každá z obálek ve svém poli `objects` uchovává modely, které jí náleží.

`GeometryObject`
Objekt vytvořený tímto konstruktorem obsahuje veškeré informace spojené s vykreslováním modelu. Důležité je to, že jsou zde uloženy buffery, do kterých jsou nahrány informace o pozicích vertexů, směrech jejich normál a dále pak například o souřadnicích potřebných pro správné namapování materiálu. Ve zdrojovém kódu 4.4 je uveden příklad nahrání pozic vertexů ze struktury, která v JSON souboru reprezentuje načítaný model.

```
1 // Vytvoření vertex position bufferu
2 this.vertexPositionBuffer = gl.createBuffer();
3 gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexPositionBuffer);
4 // Načtení dat z JSON souboru
5 gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(model.vertexPositions),
6               gl.STATIC_DRAW);
7 this.vertexPositionBuffer.itemSize = 3;
8 this.vertexPositionBuffer.numItems = model.vertexPositions.length / 3;
```

Zdrojový kód 4.4: Příklad nahrání pozic vertexů do bufferu

`Logic`
Objekt tohoto konstruktoru obsahuje veškeré informace spojené s herním polem a také veškerou herní logiku. Prvky herního pole jsou reprezentovány objekty konstruktoru `LogicItem`. Vzhledem ke svému rozsahu je tento objekt popsán samostatně v podkapitole 4.4.

`LogicItem`
Objekty představují jednotlivé herní prvky, které obsahují informace o jejich typu, pozici a aktuálním natočení. Každý z prvků má své unikátní identifikační číslo, které slouží k propojení s obálkou jeho modelu.

4.4 Herní logika a podněty uživatele

V implementované hře je každý podnět uživatele zpracováván herní logikou, která je implementována na základě analýzy hry Berušky 2 provedené v kapitole 3. Pro vyhodnocení podnětu uživatele je volána funkce `gameStep()`. Ta představuje herní krok, ve kterém je vždy zjištěna pozice aktuálně zvolené berušky a dle jejího natočení se pak určí pozice, na kterou hodlá přejít. Podle toho, jaký typ prvku se na následující pozici nachází, jsou vykonávány akce, jejichž popis je uveden v následujícím přehledu.

- **Nic** - pokud je beruška v nejnižší úrovní herního pole, pak je krok proveden. Pokud ne, pak je nejdříve zkontrolován obsah pozice, která je pod místem, kam hodlá beruška přejít.
  - Pod místem je *šikmina*. Pokud je šikmina správně natočena, pak se ještě zkontroluje, zda nevede šikmina pod vodní hladinu, kde by beruška potřebovala šnorchl. Při splnění všech podmínek je krok následně proveden.
  - Pod místem je *bortící se podlaha*. Ta se propadne pouze v případě, že má beruška ve svém inventáři závaží.

  - Pod beruškou je jiná *beruška* - přechod se neprovede.
  - Pokud zde není *žádná pevná plocha*, na které by beruška mohla stát (zeď, bedna, východ, výbušnina či kámen), pak se krok neprovede.

- **Beruška** - krok se neprovede, jinou beruškou nelze přímo pohnout.
- **Zeď** - krok se neprovede, zeď je statickým prvkem herního pole.
- **Východ** - beruška opouští herní pole. Pokud je to beruška poslední, pak končí hra.
- **Bedna** - zjistí se celková váha všech posouvaných beden a pokud je nižší nebo rovna beruščině síle, pak se bedna/bedny v daném směru posouvají. Pokud pod sebou posunutá bedna nemá podklad, pak je její pozice upravena tak, aby ho pod sebou měla.
- **Výbušnina** - zjistí se, co se nachází na pozici, kam by měla být výbušnina posunuta. Pokud je na následující pozici bedna, pak je výbušnina i bedna odstraněna a beruška zůstává na své původní pozici. Pokud za výbušninou není bedna, pak se stejně jako bedna posune.
- **Kámen** - prohledá se inventář berušky a pokud obsahuje krompáč, pak je kámen odstraněn s tím, že beruška zůstává na své původní pozici. V opačném případě se krok neprovede.
- **Šnorchl** - beruška může vlastnit pouze jeden. Pokud ho tedy ještě ve svém inventáři nemá, pak se šnorchl odstraní z herního pole, přidá se do beruščina inventáře a ta samotná je posunuta na pozici, kde se šnorchl nacházel.
- **Hormonální vitamín** - situace je obdobná jako u šnorchlu.
- **Závaží** - opět stejná situace jako u šnorchlu.
- **Krompáč** - krompáč se odstraní a přidá se do inventáře pouze v případě, že v něm má beruška místo. Maximální počet krompáčů, který může mít beruška v jeden okamžik v inventáři je 4.
- **Bortící se podlaha** - může se nacházet i před beruškou, avšak krok který by vedl k tomu, že by beruška byla pod podlahou se neprovede.
- **Šikmina** - zde opět záleží na natočení šikminy. Vzhledem k tomu, že se nad šikminou může nacházet jakýkoliv jiný herní prvek, je krok při správném natočení šikminy rozdělen na 2 fáze. Nejprve je beruška pro potřeby výpočtu přesunuta nad šikminu a následně je krok prováděn z tohoto umístění.

Je také důležité připomenout, že při každém posunu prvků je potřeba zkontrolovat, zda se posunem nedostaly do místa, kde by levitovaly ve vzduchu. Pozice se musí upravit s tím, že pokud se v nově vzniklém sloupci prvků nachází některé výbušné bedny, pak při úpravě vertikální polohy sloupce výbušné bedny odstraňují normální bedny, které mají pod sebou. Stejně tak je potřeba kontrolovat, zda se v nově vzniklém sloupci prvků nenachází bortící se podlaha. Pokud je váha nad bortící se podlahou vyšší jak 1, pak je bortící se podlaha odstraněna a sloupec objektů je vertikálně posunut.

Kategorie	Zdrojový kód
Práce s beruškami	4.5
Práce s inventářem	4.6
Získávání reference na prvky	4.7
Výpočet váhy prvků	4.8
Posun prvků	4.9
Odstraňování prvků	4.10
Herní krok	4.11

Tabulka 4.2: Kategorie funkcí herní logiky

Objekt herní logiky

Při vytvoření tohoto objektu jsou z JSON souboru načteny potřebné informace a podle nich jsou pak dopočítány ty zbylé. Prvky herního pole se stejně jako obálky modelů (4.3) dělí na statické a dynamické a jsou uloženy v odpovídající polích `staticItems` a `dynamicItems` tohoto objektu (neplést s poli určenými pro obálky modelů, které jsou uloženy v globálních polích se stejnými názvy). Statické prvky jsou indexovány pomocí čísla, které je odvozeno z pořadí při jejich načítání. Dynamické prvky jsou naopak indexovány svým identifikačním číslem.

Jak již bylo uvedeno v 4.3, tento objekt obsahuje veškerou herní logiku v podobě funkcí, které jsou rozděleny do několika kategorií. Kategorie a odpovídající komentované přehledy jsou uvedeny v tabulce 4.4.

```
1  /**
2  * Vybere berušku s daným ID.
3  * @param id ID berušky, která má být vybrána
4  */
5  function selectBug(id) {/*...*/}
6  /**
7  * Vybere následující berušku.
8  */
9  function selectNextBug() {/*...*/}
10 /**
11 * Odstraní berušku z hracího pole.
12 * Po odstranění poslední berušky končí hra.
13 * @param id ID berušky, která má být odstraněna
14 */
15 function removeBug(id){/*...*/}
16 /**
17 * Posune berušku na zadanou pozici a navíc
18 * zjistí, zda se pod beruškou nenacházelo propadlo.
19 * Pokud ano, pak se zjistí aktuální váha nad propadlem
20 * a propadlo se případně odstraní.
21 * @param id Identifikační číslo berušky
22 * @param position Pozice, na kterou má být beruška přesunuta
23 */
24 function moveBug(id, position){/*...*/}
```

Zdrojový kód 4.5: Funkce pro práci s beruškami

```
/**
* Ověří, zda se v inventáři berušky nachází daný předmět.
* @param bugID Identifikační číslo berušky
* @param itemSubclass Podtřída vyhledávaného předmětu
* @return -1 pokud nebyl předmět nalezen, nebo pozici předmětu v inventáři
*/
function inventoryContains(bugID, itemSubclass){/*...*/}
/**
* Přidá předmět do beruščina inventáře.
* @param bugID Identifikační číslo berušky
* @param itemSubclass Podtřída přidávaného předmětu
*/
function inventoryAppend(bugID, itemSubclass){/*...*/}
/**
* Odebere předmět z beruščina inventáře.
* @param bugID Identifikační číslo berušky
* @param itemPosition Pozice odebíraného předmětu v inventáři
*/
function removeFromInventory(bugID, itemPosition){/*...*/}
```

Zdrojový kód 4.6: Funkce pro práci s beruščiným inventářem

```
/**
* Získá referenci na prvek hracího pole, který se nachází na dané pozici.
* @param position Pozice prvku
*/
function getObjectOnPosition(position) {/*...*/}
/**
* Získá referenci na prvek hracího pole s daným ID.
* @param id Identifikátor prvku.
*/
function getObjectByID(id) {/*...*/}
/**
* Získá pozici prvku se zadaným id.
* @param id Identifikátor prvku
*/
function getPositionOfObject(id) {/*...*/}
```

Zdrojový kód 4.7: Funkce pro získávání reference na prvky herního pole

```
/**
* Získá váhu prvku na zadané pozici.
* Jakmile se jedná o statický prvek, pak jeho váha 1000.
* @param position Pozice objektu
*/
function getWeightOfObject(position) {/*...*/}
/**
* Vypočítá váhu sloupce od zadané pozice nahoru.
* @param position Pozice, od které má výpočet probíhat
*/
function getWeightOfColumn(position){/*...*/}
/**
* Sečte váhu herních prvků v daném směru. Pracuje rekurzivně.
* @param direction Směr - up, right, down, left
* @param position Pozice od, které má být výpočet proveden
*         obvykle pozice, na následující pozice berušky
*/
function countWeight(direction, position){/*...*/}
```

Zdrojový kód 4.8: Funkce pro výpočet váhy objektů

```
/**
* Posune prvek herního pole a s ním i všechny
* prvky, které se nachází nad ním.
* Jakmile je posun ukončen, jsou upraveny vekrtikální pozice
* prvků tak, aby pod sebou měli podklad.
function moveObject(direction, tempObject){/*...*/}
/**
* Rekurzivně posouvá prvky herníno pole.
* K posuvu využívá funkci moveObject a je volána
* teprve tehdy, když už je jisté, že prvky lze
* posunout - nejdříve se počítá váha posouvaného bloku
* @param direction Směr posunu - up, right, down, left
* @param position Pozice, od které má být posuv proveden
*                 Obvykle následující pozice berušky
*/
function moveAction(direction, position){/*...*/}
```

Zdrojový kód 4.9: Funkce pro posun objektů

```
/**
* Odstranní prvek herního pole.
* Nejprve je odstraněn model prvku a poté jeho záznam v logické reprezentaci
    .
* @param item Reference na prvek, který má být odstraněn
*/
function removeItem{/*...*/}
```

Zdrojový kód 4.10: Funkce pro odstraňování objektů

```
/**
* Veškerý pohyb ve hře je zprostředkován pomocí této funkce.
* Vypočítá následující pozici berušky, určí typ předmětu na této
* pozici a podle typu se rozhoduje, jaké kroky provést.
*
* Tato funkce přijímá jeden z parametrů.
* Buď je jím směr, kterým se aktuálně vybraná beruška má vydat,
* nebo je to přímo pozice, na kterou hodlá beruška přejít.
* Přímé pozice je využito například u šikminy, kde beruška nemění
* svou pozici pouze o 1 krok, avšak je nutné berušku posunout nad/pod
* šikminu.
* @param direction Směr, kterým se vybraná beruška vydává.
* @param newBugPositionIn Pozice, na kterou se beruška chystá jít.
*/
function gameStep(direction, newBugPositionIn) {/*...*/}
```

Zdrojový kód 4.11: Funkce herního kroku gameStep

Klávesa	Akce klávesy
↑	Pohyb aktuálně vybrané berušky - provádění herního kroku
←	Natočení berušky o 90°vlevo
→	Natočení berušky o 90°vpravo
↓	Otočení berušky o 180°
1	Výběr berušky 1
2	Výběr berušky 2
3	Výběr berušky 3
4	Výběr berušky 4
5	Výběr berušky 5
m	Změna vykreslovacího módu
n	Přepíná mezi vykreslováním celé scény a samotného herního pole
c	Změna typu kamery
x	Zapíná/vypíná zobrazení lightmap
l	Zapíná/vypíná použití bodového světla
f	Přepíná mezi zobrazením přes celou obrazovku a normálním zobrazením
o	Zapíná/vypíná průhlednost objektů
s	Zapíná/vypíná zobrazení odlesků
r	Restartuje herní úroveň
Enter	Přepíná mezi horním pohledem a pohledem normálním

Tabulka 4.3: Ovládání hry pomocí klávesnice

4.5 Ovladání

Hra je ovládána pomocí klávesnice a myši. Při stisku jakékoliv klávesy je zjištěn její kód, kterým je identifikována v rámci JavaScriptu a dle tohoto kódu je prováděna příslušná akce.

Existují dva způsoby, kterými lze reagovat na stisknuté klávesy:

- Reagovat ihned na událost stisku klávesy,
- Pravidelně kontrolovat aktuálně stisknuté klávesy a provádět příslušné operace.

Pro uživatele je rozdíl takový, že pokud drží klávesu, tak v prvním případě již po první reakci nenásledují žádné další. V případě druhém se pak akce opakovaně provádějí do té doby, dokud uživatel klávesu drží, avšak za tu cenu, že nemusí být zachyceny všechny stisky kláves. Ze spolehlivostních důvodů je tedy v implementaci zvolen první způsob reakce.

Klávesy a odpovídající akce na jejich stisk jsou uvedeny v tabulce 4.5.

Myší je ovládána kamera. Tlačítko c slouží k přepínání různých typů kamery, kterými jsou:

Kamera má více možností zobrazení:

- Kamera s bodem otáčení kolem středu herního pole
- Kamera s bodem otáčení kolem aktuálně vybrané berušky
- Pohled berušky
- Pohled na záda berušky

U prvních dvou typů zobrazení je kamera ovládána pomocí myši tak, jak možné vidět na diagramu B.1. Herní obrazovka je rozdělena na oblasti, které jsou citlivé na kurzor myši. Najetím kurzoru myši je následně změněn úhel rotace kamery a její elevace. U zbylých dvou typů kamery je rotace a elevace pevně určena aktuální pozicí a rotací berušky. WebGL nemá přímou podporu pro kameru. Výsledný pohled nevzniká tedy tak, že by se transformovala pozice a natočení kamery v rámci scény, avšak je vytvořen tak, že kamera je umístěna na pevné pozici a pohybuje se celou scénou. To, jakým způsoben je toto implementováno, je uvedeno v podkapitole 4.6.

4.6 Vykreslování

Vykreslování probíhá periodicky s tím, že modely scény jsou při vykreslování transformovány na základě aktuálních informací, které se nacházejí v logické reprezentaci herní úrovně. Původní hra obsahuje animace prvků herního pole a svého okolí. V implementované hře animace nejsou obsaženy a všechny transformace objektů scény jsou tedy prováděny ihned po dokončení herního kroku. Jedná se o zjednodušení, které bylo odsouhlaseno již při zadávání projektu.

Vykreslování obstarává funkce `drawScene()`. Na počátku této funkce je vždy vyčištěn color a depth buffer a následně je přistoupeno k vytvoření projekční matice a matice kamery

Projekční matice je nastavena na úhel projekce 45°. Je však možné ho po nastavení proměnné `useProjection` měnit tlačítky  a  . Reset projekční matice se pak provádí pomocí tlačítka  . Další nastavení hry je uvedeno v podkapitole 4.7.

Matice kamery je sestavena na základě jejího aktuálně zvoleného typu. Vytvoření této matice se skládá z několika kroků:

- Vytvoření 4×4 matice identity `cameraMatrix`
- Translace této matice v osách X a Z - určí se střed otáčení kamery
- Rotace kolem osy Y - aplikuje rotaci vlevo, nebo vpravo
- Rotace kolem osy X - aplikuje aktuální elevaci
- Translace kolem osy Z - posune kameru od středu otáčení.

Jelikož není maticové násobení komutativní operací, je nutné provádět transformace v tomto pořadí. Po sestavení matice kamery je vzhledem k tomu, že pohybujeme scénou a ne kamerou, vytvořena matice k ní inverzní. Tou je vynásobena tzv. *model-view* matice, kterou jsou následně násobeny všechny vertexy, které se ve scéně nacházejí. Vykreslování se řídí následujícím algoritmem.

```
for all (obálka in pole_obálek) do
Ulož model-view matici
    if ((frustrum culling) AND (obálka není viditelná ve frustru kamery)) then
     continue
    end if
    if ((obálka není součástí herního pole) && (není zobrazena celá scéna)) then
     continue
    end if
    for all (model in obálka) do
        if ((frustrum culling) AND (model je ve frustru kamery)) then
          continue
        end if
        if ((model má průhledný materiál) OR (je zapnut blending)) then
          přidej objekt do pole blendedObjects, continue
        end if
        if (zapnuto použití textur) then
          Nahraj texturu používanou modelem do pipeline
        end if
        if ((zapnuto použití lightmap) AND (model má lightmapu)) then
          Nahraj lightmapu používanou modelem do pipeline
        end if

        if (dynamický prvek scény) then
          Proveď trasnsformace model-view matice dle logické reprezentace
        end if
        Nahraj buffer s vertexy do pipeline
        Nahraj buffer s normálami vertexů do pipeline
        Nahraj model-view matici do pipeline
        Vykresli modely za použití aktuálně zvoleného typu vykreslování

    end for
Nahraj uloženou model-view matici
end for
```

Algoritmem jsou procházeny jednotlivé obálky statických či dynamických objektů a je testováno, zda náleží do oblasti viditelné pozorovateli. Pokud není ani jeden bod obálky v této oblasti, pak nejsou její modely vykreslovány. Se zapnutým vykreslováním celé scény je další test přeskočen, avšak v případě, že tomu tak není, je testováno, zda je obálka prvkem herního pole. V dalším cyklu se již procházejí jednotlivé modely obálky, které jsou opět testovány na svou viditelnost. Jakmile jsou viditelné, pak je potřeba získat informaci o tom, zda není jejich materiál částečně průhledný, nebo zda nebyli zprůhledněny všechny vykreslované objekty. Pokud se tedy jedná o model s průhledným materiálem, pak je jeho vykreslování odloženo na pozdější dobu. Jakmile jsou splněny další podmínky, pak jsou nahrány textury/lightmapy a je přistoupeno k samotnému vykreslování. Model-view matice, která vznikla složením z invertované matice kamery a transformací, které byly provedeny na základě informací v logické reprezentaci scény, je nahrána do grafické pipeline a s ní i vertexy modelu a jejich normály. Modely jsou následně vykresleny tak, jak bylo popsáno v podkapitole 2.5.

V implementované hře jsou obálky modelů rozděleny na statické a dynamické, takže vykreslování probíhá ve více samostatných cyklech. Navíc je ještě při zobrazení samotného herního pole vykreslována podlaha. Při zobrazení modelů s průhledným materiálem dojde k odloženému vykreslování modelů v samostatném cyklu. Jednotlivé vykreslovací cykly jdou tedy v tomto pořadí:

1. Vykreslení podlahy herního pole
Podlaha herního pole je vykreslována v případě, že je zobrazeno pouze samotné hrací pole (klávesa [n]). Umístění vertexů a pozice textury podlahy jsou vypočítány vždy při vytvoření objektu s logickou reprezentací úrovně.

2. Vykreslení statických objektů
Statické objekty jsou uchovány v poli `staticItems` a není u nich potřeba provádět transformaci model-view matice, jelikož se vůči scéně nacházejí stále na stejné pozici.

3. Vykreslení dynamických objektů
Dynamické objekty jsou umístěny v asociativním poli `dynamicItems` a jsou umisťovány na pozice, které odpovídající jejich pozici v logické reprezentaci scény. Výsledná translace v jednotlivých osách je vypočítána podle následujícího vztahu.

$$translace_{xzy} = start_{xyz} + pozice_{xyz} * velikostPozice \tag{4.1}$$

Proměnná $start_{xyz}$ je místo, na kterém se nachází pozice $[0, 0, 0]$ herního pole. Proměnná $pozice_{xyz}$ udává pozici herního pole, kde se nachází aktuálně vykreslovaný model a $velikostPozice$ je konstantou, která udává, jak velká je jedna pozice herního pole v souřadném systému scény.

4. Vykreslení průhledných objektů
Některé z herních úrovní obsahují modely, které mají průhledné materiály. Tyto modely musí být vykresleny až po modelech, které průhledné nejsou. Vykreslování průhledných objektů se dále komplikuje tím, že k tomu, abychom byli schopni zkombinovat barvy materiálů a dosáhli tak efektu průhlednosti, musíme modely nejdříve seřadit podle jejich vzdálenosti od pozorovatele. Jako první jsou vykresleny modely nejvzdálenější a nakonec ty, které jsou k pozorovateli nejblíže. Při načítání jednotlivých modelů z JSON souboru jsou také dopočítávány jejich středy, které jsou právě při tomto řazení využity. Je důležité si uvědomit, že seřazení modelů musí proběhnout při každém volání vykreslovací funkce, a tudíž je následný propad v rychlosti vykreslování poměrně znatelný. Barvy materiálů jsou pak kombinovány v části grafické pipeline, která byla popsána v podkapitole 2.5. Průhlednost všech objektů scény lze zapnout pomocí klávesy [o] .

Proměnná	Použití
`drawOnlyGameField`	Zapnutí vykreslení celé scény
`useOpacity`	Zprůhlednění všech objektů scény
`opacityLevel`	Nastavení úrovně průhlednosti objektů scény
`useTextures`	Zobrazení textur
`useLightmaps`	Zobrazení lightmap
`useLightning`	Zapnutí osvětlení
`useSpecular`	Zobrazení odlesků
`useTopCamera`	Přepne na kameru kolmou herní pole
`paintSelectedBugRed`	Beruška zčervená po jejím vybrání
`drawEnvelopes`	Vykreslení obálek dynamických objektů scény
`drawFloor`	Vykreslení podlahy
`useNotifications`	Zobrazení notifikací
`showFPSinConsole`	Zobrazí FPS v konzoli prohlížeče
`useProjection`	Zapnutí možnosti změny úhlu projekce
`log`	Zobrazení debugovacích informací v konzoli prohlížeče
`renderMode`	Nastavení typu vykreslování
`cameraMode`	Nastavení typu kamery

Tabulka 4.4: Některá z herních nastavení

Osvětlení
Výsledný obraz je také určen osvětlením z různých světelných zdrojů, které se ve scéně nacházejí. Stejně tak jako chybí podpora pro práci s kamerou, chybí ve WebGL i podpora osvětlení. Veškeré výpočty jsou tedy prováděny „ručně“, a to přímo v shaderech grafické karty. V implementaci je využito phongova osvětlovacího modelu, který oproti osvětlovacímu modelu původní hry používá per-fragment shading[1]. Využito je pouze jednoho dynamického bodového světla, které je umístěno přímo nad herním polem. Implementace jako taková je připravena na využití více světel, avšak při navýšení jejich počtu exponenciálně klesá výkonnost vykreslovaní. Osvětlení lze zapnout pomocí klávesy [l] a odlesky pomocí klávesy [s]. Vzhledem k omezenému rozsahu práce se již nebudeme osvětlením dále zabývat.

4.7 Nastavení hry

Některé způsoby nastavení již byli uvedeny v předchozím textu. V tabulce 4.7 je uveden přehled nejdůležitějších proměnných, které mění způsob, jakým se chová vykreslování herní scény. Kompletní přehled je obsažen ve vygenerované dokumentaci, která je součástí přiloženého DVD.

[1]Intenzita osvětlení fragmentu nevzniká interpolací intenzit osvětlení vertexů, ale je počítána pro každý fragment zvlášť. Je tak dosaženo více realistického typu zobrazení scény.

Kapitola 5

Testování

Tato kapitola se primárně zabývá testováním implementované hry. Jeho výsledky jsou důležité pro následnou diskuzi ohledně využitelnosti nově objevujících webových technologií. K tomu, abychom mohli takovou diskuzi provádět, je nejprve potřeba porovnat výsledné řešení s původní hrou a představit tak výhody/nevýhody webu nativních aplikací.

Porovnání her

Implementovaná hra zakládá na hře Berušky 2. Herní koncept zůstal zachován, avšak způsob, jakým je řešeno vykreslování, je odlišný. Původní hra je technologicky mnohem vyspělejší (viz. 3.8). Obsahuje animace, řeší viditelnost objektů scény pomocí hierarchických OBB obálek a využívá mnoho dalších moderních technologií. Hra je oproti implementované i výrazně rychlejší. Při rozlišení $1920x1080$ dosahuje hra rychlosti 52 snímků za sekundu a při rozlišení $1024x768$ 124 snímků za sekundu [1]. Implementované řešení má tedy v oblasti vykreslování mnoho prostoru ke zdokonalení a optimalizaci. Implementovaná hra však vítězí v dostupnosti. Je možné ji bez instalace začít ihned používat na většině dnešních operačních systémů. Byla ověřena funkčnost i na operačním systému Android. Hru na tomto systému však není možné ovládat, jelikož nepřijímá dotykové události. Výslednou podobu hry Berušky 2 WebGL v prohlížeči Firefox 12 je možné vidět na obrázku C.1.

Testování

Hra byla testována v prohlížečích Mozilla Firefox a Google Chrome na operačním systému Microsoft Windows 7. Testy byly provedeny na dvou různých strojích s rozdílnou hardwarovou konfigurací.

- Intel C2D T7100, GPU Intel x3100
- Intel C2D T5700, GPU ATI Radeon 4330

Rozdílnost těchto strojů tkví hlavně v instalované grafické kartě. Intel x3100 neumí používat shader model 2.0, který je nutností pro hardwarovou akceleraci WebGL. Obraz je tedy vykreslován softwarově. Hra byla vždy testována se zapnutým zobrazením celé scény a následně se zobrazením samostatného herního pole. Každý z testů je navíc proveden pro

[1] Intel x3100

různé velikosti drawing bufferu 2.5 a různé vykreslovací módy. Hodnoty v polích tabulek vždy udávají vykreslené snímky za sekundu[2].

Intel x3100

	896 × 504		1920 × 979	
	Herní pole	**Celá scéna**	**Herní pole**	**Celá scéna**
Plné zobrazení	15	3	7	2
Žádné odlesky	15	3	7	2
Bez textur	16	3	7	2
Bez osvětlení	16	3	7	2
Bez textur a osvětlení	17	3	7	3

Tabulka 5.1: Windows 7, Intel x3100, Firefox 11.0

	896 × 504		1920 × 979	
	Herní pole	**Celá scéna**	**Herní pole**	**Celá scéna**
Plné zobrazení	51	17	35	13
Žádné odlesky	51	17	35	13
Bez textur	51	17	35	13
Bez osvětlení	51	17	48	17
Bez textur a osvětlení	56	18	50	17

Tabulka 5.2: Windows 7, Intel x3100, Chrome 19.0.1084.46

ATI Radeon 4330

	896 × 504		1920 × 979	
	Herní pole	**Celá scéna**	**Herní pole**	**Celá scéna**
Plné zobrazení	60	37	44	32
Žádné odlesky	60	37	44	32
Bez textur	60	39	51	38
Bez osvětlení	60	40	44	32
Bez textur a osvětlení	60	41	43	40

Tabulka 5.3: Windows 7, Chrome 19.0.1084.46, ATI Radeon 4330

Rozdíl softwarového a hardwarového vykreslování obrazu není příliš znatelný. Není ani vidět příliš velký rozdíl v rychlosti vykreslování různých typů zobrazení. Rozdíl je však znatelný v rychlosti prohlížečů Firefox 11 a Chrome 19. Google Chrome je obecně známý tím, že obsahuje velice rychlý interpret JavaScriptového kódu. Bylo také provedeno testování

[2] FPS

	896 × 504		1920 × 979	
	Herní pole	**Celá scéna**	**Herní pole**	**Celá scéna**
Plné zobrazení	46	4	31	4
Žádné odlesky	46	4	31	4
Bez textur	47	4	33	4
Bez osvětlení	49	4	32	4
Bez textur a osvětlení	50	41	34	4

Tabulka 5.4: Windows 7, Firefox 11, ATI Radeon 4330

v prohlížeči Google Chrome 9, což je první verze tohoto prohlížeče, která podporovala technologii WebGL. Obraz nebyl vykreslován rychlostí vyšší než 2 snímky za sekundu a lze tedy u tohoto prohlížeče vidět výrazný posun k lepšímu.

Z poznatků, které byly při testech získány je možné vidět velikou závislost rychlosti vykreslování na rychlosti zpracování JavaScriptového kódu. I když se rychlost interpretace JavaScriptového kódu neustále zvyšuje, není tento jazyk stále určen pro zpracování velkého množství dat. Pro optimalizaci WebGL aplikací je tedy potřeba přenechat co největší část práce na grafickém hardwaru.

Na prvním uvedeném systému byla také testována byla také rychlost načítání herní úrovně. Původní hra načetla úroveň za 9 sekund. Herní úroveň hry implementované v této práci se v prohlížeči Chrome 19 načítá 30 milisekund. Z toho se 20 milisekund načítá JSON soubor a 10 milisekund se načítají potřebné textury. V této situaci však byly veškeré herní soubory uloženy lokálně. Doba potřebná k zobrazení úrovně ze serveru je výrazně vyšší. JSON soubor obsahuje mnoho informací a jeho průměrná velikost se pohybuje kolem 6 MB. Jsou však i takové herní úrovně, které s texturami zabírají kolem 15 MB dat. Čtenář si jistě dokáže představit, jak dlouhou dobu načítání takového množství dat trvá.

Kapitola 6

Závěr

V rámci této bakalářské práce byla analyzována hra Berušky 2, na jejímž základě byl následně vytvořen návrh a implementace webové hry Berušky 2 WebGL. K vývoji bylo využito moderních webových technologií, které byly stručně představeny a uvedeny do souvislosti s jejich použitím. Veškerá funkčnost implementovaného řešení byla popsána pomocí jazyků JavaScript a GLSL pro popis shaderů grafické karty. Implementované řešení bylo podrobeno testům, jejichž výsledky byly diskutovány s konzultantem této práce - Ing. Martinem Stránským. Je důležité zdůraznit, že řešení rozšiřuje zadání práce o různé možnosti nastavení vykreslování, implementaci webu, přehrávání herní hudby a mnohá vylepšení v podobě animací, které vylepšují celkový dojem z celé aplikace.

Po konzultaci s panem Ing. Martinem Stránským byly použité technologie celkově zhodnoceny jako vhodné pro vývoj aplikací jako je ta, jejíž implementace byla součástí této práce. Pro komerční využití je však potřeba, aby byli nejprve odstraněny některé problémy, které jsou spojené s tím, že web a jeho prostředky nebyly pro tyto účely vytvořeny. WebGL například používá hardwarové akcelerace vykreslování, avšak aplikace využívající toto rozhraní stále nedosahují rychlosti aplikací desktopových. To, co podle provedených testů WebGL *brzdí*, je výpočetní rychlost JavaScriptu. JavaScript není určen pro operace s velkým množstvím dat. Interprety tohoto jazyka jsou však dnes středem pozornosti a lze tedy předpokládat, že tento problém bude postupně mizet.

Existuje velké množství technik sloužících k zobrazení realisticky vypadající herní scény. Tato hra některé z nich implementuje, avšak stále zbývá mnoho prostoru pro optimalizace a vylepšení. Jednou z optimalizací, se kterou se hra v brzké době setká, je tzv. frustrum culling, pro nějž je hra již částečně připravena. Další je redukce přenášených dat. Pokud by to technologie umožnily, pak by bylo pro hráče jistě přívětivé, kdyby si mohl hru spustit z lokálního úložiště a nebyl tak závislý na datovém připojení. Pokud by se technologie vyhovující tomuto účelu objevila, pak bude její využití jednou z priorit. Při konzultacích s panem Stránským byla také navržena mnohá vylepšení v tom, jakým způsobem je hra ovládána. Tato vylepšení s ním budou dále diskutována.

Literatura

[1] Anyuru, A.: *Professional WebGL Programming*. John Wiley & Sons, 2012.

[2] Flanagan, D.: *JavaScript: the definitive guide*. O'Reilly Media, 2006.

[3] Lubbers, P.; Salim, F.; Albers, B.: *Pro HTML5 Programming*. Apress, 2011.

[4] MacDonald, M.: *HTML5: The Missing Manual*. O'Reilly Media, Inc., 2011.

[5] Pilgrim, M.: *HTML5: up and running*. Oreilly & Associates Inc, 2010.

[6] Seidelin, J.: *Html5 Games: Creating Fun with Html5, Css3, and Webgl*. Wiley, 2011.

Příloha A

Diagram HTML 5 API

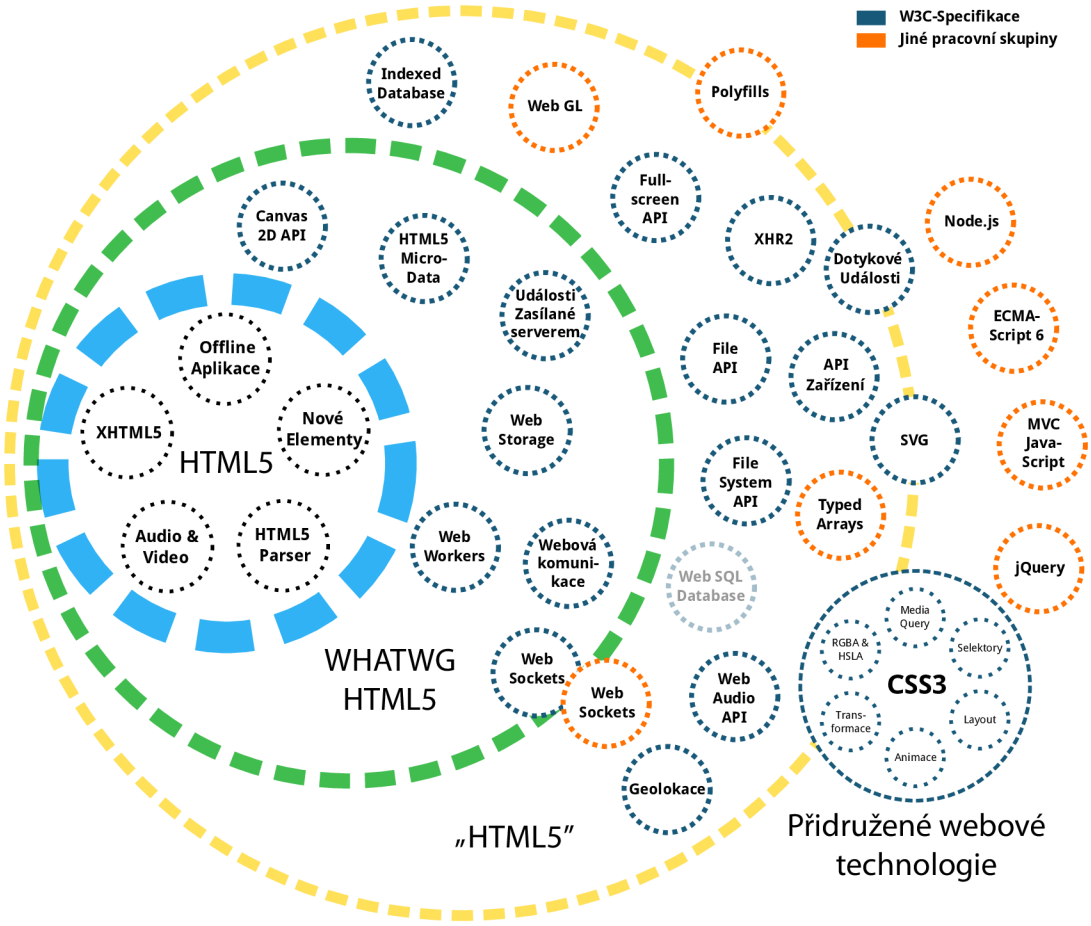


Diagram A.1: HTML 5 API

Příloha B

Diagram návrhu webu

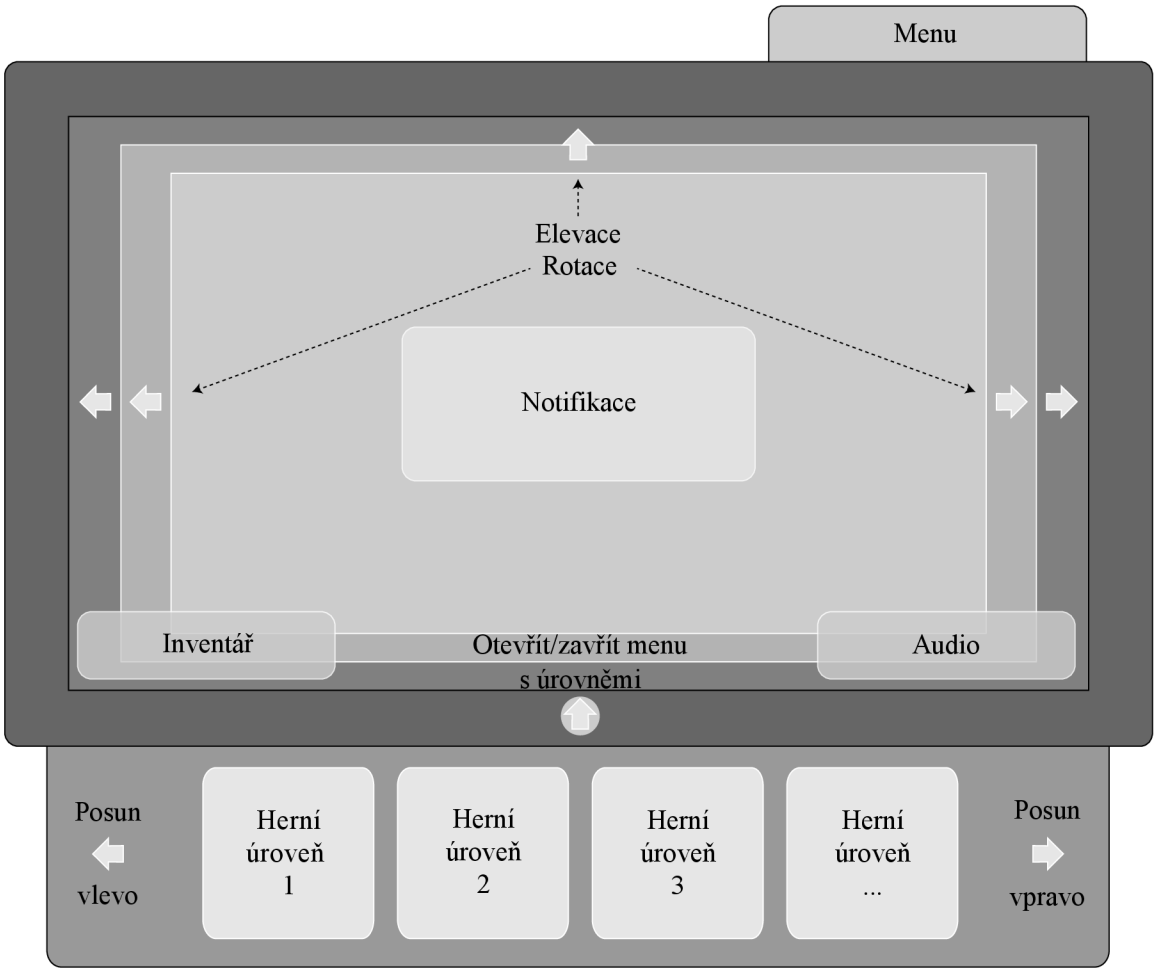


Diagram B.1: Návrh webu

Příloha C

Podoba hry Berušky 2 WebGL

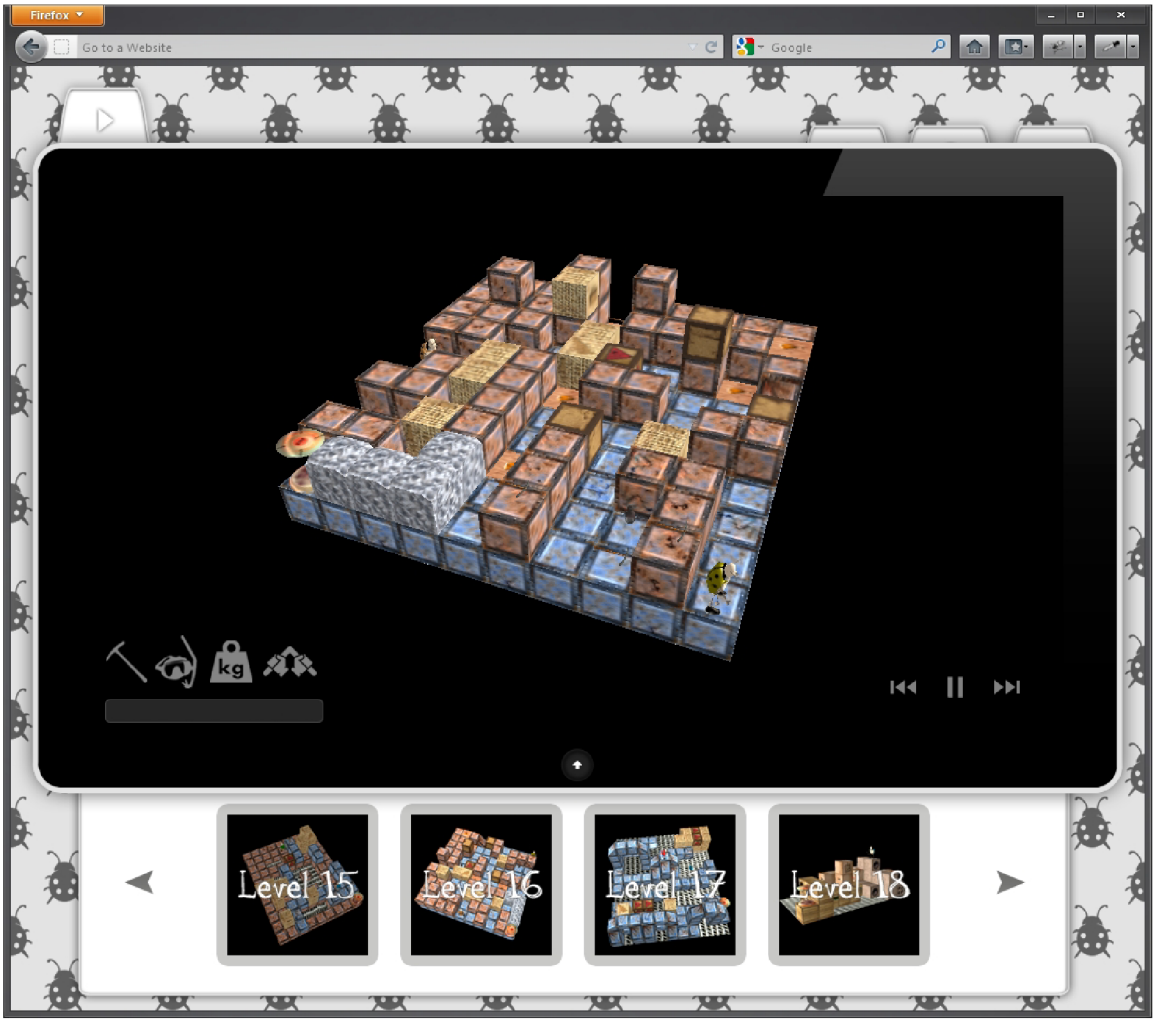


Diagram C.1: Berušky 2 WebGL v prohlížeči Firefox 12

Příloha D

Obsah DVD

Cesta	**Popis**
`./audio`	Herní hudba
`./css`	Kaskádové styly
`./doc`	Vygenerovaná HTML dokumentace
`./graphics`	Zdrojové soubory grafiky
`./img`	Webová grafika
`./js`	Implementace hry a soubory frameworků
`./levels`	JSON soubory herních úrovní
`./lightmaps`	Lightmapy
`./textures`	Textury
`./index.html`	HTML dokument hry s GLSL popisem shaderů

Tabulka D.1: Obsah DVD