VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D HRA VE WEBGL – INBALL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ JURČÍK
AUTHOR

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D HRA VE WEBGL - INBALL

3D GAME USING WEBGL - INBALL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ JURČÍK
AUTHOR

VEDOUCÍ PRÁCE Ing. ISTVÁN SZENTANDRÁSI
SUPERVISOR

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá tématem tvorby aplikací v prostředí webových prohlížečů, zvláště počítačových her. Na začátku jsou popsány různé metody tvorby her v prohlížečích. Následuje část věnující se frameworkům pro WebGL a fyzice ve hrách. Cílem práce je návrh a implementace 3D počítačové hry Inball založené na hrách typu Neverball a Ballance. Aplikace je založená na technologii WebGL, používající knihovnu Three.js.

Abstract

This bachelor thesis outlines the possibilities of making 3D applications in web-browsers, focusing on computer games. First, different methods of creating games inside web-browsers are briefly introduced. Next section contains a description of WebGL's frameworks and physics in games. The aim of this work was to design and implement a 3D browser game called Inball based on games like Neverball and Ballance. The application is based on WebGL technology using Three.js of frameworks encapsulating WebGL.

Klíčová slova

WebGL, OpenGL, detekce kolizí, Three.js, Ammo.js, GLSL, grafická pipeline, vykreslování do textury

Keywords

WebGL, OpenGL, detection collision, Three.js, Ammo.js, GLSL, graphic’s pipeline, render to texture

Citace

Lukáš Jurčík: 3D hra ve WebGL - Inball, bakalářská práce, Brno, FIT VUT v Brně, 2012

3D hra ve WebGL - Inball

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Istvána Szentandrásiho.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Jurčík
16. května 2012

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Ing. Istvánovi Szentandrásimu za jeho odborné rady, ochotu a čas, který mi věnoval při vzniku této práce.

Obsah

Obsah 1
1 Úvod 2
2 Teorie 3
2.1 OpenGL ES 3
2.2 WebGL 4
2.3 Hry ve webových prohlížečích 5
2.4 Detekce kolizí 6
3 Návrh hry 9
3.1 WebGL frameworky 9
3.2 Koncept hry 10
3.3 Shadery 14
3.4 Herní fyzika 19
3.5 Pohyb herní kuličky 22
4 Implementace a výsledky 23
4.1 Použité knihovny a nástroje 23
4.2 Grafické rozhraní 23
4.3 Popis modulů 25
4.4 Testování a výsledky 28
5 Závěr 30
Příloha A 33
Příloha B 35
Příloha C 36

1 Úvod

S vývojem webových standardů dochází k pronikání 3D technologií i do internetových prohlížečů, jejichž doménou byl dlouhou dobu pouze dvourozměrný prostor. Jednou z těchto technologií je i WebGL, které umožňuje tvorbu 3D počítačových her, vizualizaci grafů, matematických nebo fyzikálních struktur. WebGL je multiplatformní technologie založená na OpenGL. Používá nízko úrovňový přístup. Pro snadnější práci a přístup k WebGL vznikla řada frameworků, jimž bude věnována kapitola 3.1.

Cílem této práce je předvést technologii WebGL v podobě 3D počítačové hry s názvem Inball, která je spustitelná v rámci webového prohlížeče bez nutnosti instalace podpůrných doplňků a pro zrychlení zobrazených dat využívá akceleraci grafické karty. Díky této vlastnosti mohou vznikat propracované 3D hry s vizuální stránkou, která byla doposud doménou počítačových her na stolních počítačích. Jedná se o arkádovou hru založenou na hrách typu Ballance a Neverball. Hráč ve hře Inball ovládá kuličku, kterou se snaží dostat do cílového místa v co nejkratším čase. Dostupných je několik různých herních režimů.

Tato práce, v kapitole 2 věnované teorii, seznámí čtenáře s technologií tvorby grafiky v prostředí internetového prohlížeče – WebGL. Osvětlí, jak tato technologie funguje, jaké je její využití v praxi a jak se s ní pracuje. Další část dokumentu je zaměřena na tvorbu her v prohlížeči, jaké máme obecně možnosti vytváření her, poté se zaměřením na WebGL. Srovnává výhody a nevýhody různých technologií. Dále jsou probrány metody řešení detekce kolizí a následuje část, která se zabývá návrhem grafických efektů vytvořených vlastními shadery. V kapitole 3 zaměřené na návrh hry budou probrány frameworky pro WebGL. Následující část se věnuje problematice herní fyziky a použití různých fyzikálních knihoven. Hlavní část se zabývá návrhem konkrétní počítačové hry Inball a poslední část, kapitola 4, je zaměřena na implementaci samotné hry, testování funkčnosti na různých počítačích a porovnání vlivu různých prohlížečů na běh aplikace.

2 Teorie

V teoretické části se představí OpenGL a WebGL, které vychází z prvně jmenované technologie. Na programovatelné grafické pipeline se popíše fungování a princip obou těchto technologií. Další část je věnována tvorbě aplikací, především her ve webových prohlížečích. Kromě WebGL jsou zmíněné další nástroje Flash a Silverlight a jejich odlišnosti. Poslední část této kapitoly vysvětluje pojem detekce kolizí a popisuje různé možnosti řešení a implementace.

2.1 OpenGL ES

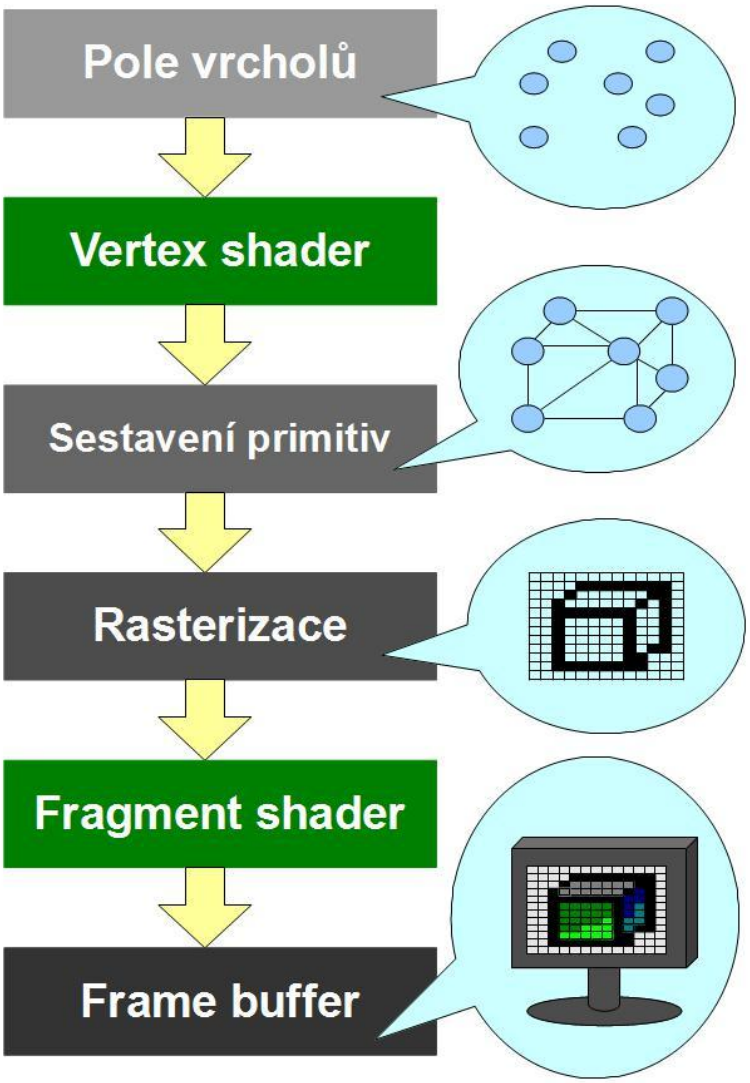
Na desktopových systémech existují dvě programové rozhraní pro tvorbu 3D aplikací, DirectX a OpenGL. DirectX je spjatý s firmou Microsoft, tudíž tuto technologii můžeme nalézt v zařízeních, pro která tato firma dodává svůj software. Nejčastěji jsou to stolní počítače s operačním systémem Windows. Většina 3D počítačových her na tomto systému je vytvořena právě technologií DirectX. OpenGL je multiplatformní rozhraní. Používáno je na systémech typu Unix, Mac a Windows.

OpenGL ES[1] je odnoží OpenGL. Jedná se o rozhraní pro programování 3D aplikací zaměřené na přenosná zařízení jako jsou mobilní telefony, tablety, PDA atd. OpenGL ES existuje ve třech specifikacích, poslední 2.0 vychází z OpenGL 2.0 a implementuje programovatelnou grafickou pipeline.

2.1.1 Grafická pipeline

Na Obrázku 2-1 je znázorněna programovatelná grafická pipeline OpenGL ES 2.0. Jedná se o schéma operací prováděných při zpracování obrazu. Nejde o detailní popis, znázorněny jsou pouze nejvýznamnější části.

Pole vrcholů definující objekty ve scéně jsou předány **Vertex Shaderu**. To je program, který počítá pro každý vrchol pozici ve scéně a přiřazuje mu další atributy popisující jeho vlastnosti. Vrcholy jsou seskupeny do jednotlivých základních geometrických primitiv (bod, úsečka nebo trojúhelník). Z těchto primitiv jsou složeny výsledné objekty. **Rasterizací** se převede scéna na 2D obrázek a odstraní se části, které nejsou viditelné. **Fragment Shader** přidá každému pixelu barvu nebo texturu a hloubku. Z **Frame bufferu** je obraz poslán na monitor. Výsledný obraz nemusí být uložen do frame bufferu, ale může se dále upravovat v jiném bufferu. Vertex a fragment shader jsou uživatelem programovatelné jednotky. Pomocí nich je možné implementovat osvětlení, stínování a další efekty.



Obrázek 2-1: OpenGL ES 2.0 programovatelná grafická pipeline.

2.2 WebGL

WebGL (Web-based Graphics Library) je multiplatformní knihovna umožňující zobrazovat grafiku ve webových prohlížečích. K vykreslování je použita grafická karta počítače. Díky tomu je celý proces rychlý a můžeme zobrazovat i složité 3D scény. Tato technologie je založena na OpenGL ES 2.0. K zápisu kódu je použit jazyk JavaScript, který patří v oblasti webových technologií k zavedeným a používaným programovacím jazykům. To napomáhá této technologii k tomu, aby se mohla více rozšířit bez nároků na programátory učit se další programovací jazyk. K použití WebGL, ať už pouze zobrazení, nebo vyvíjení aplikací není potřeba instalovat žádné doplňující knihovny nebo programy. Je nutné mít pouze grafickou kartu, která podporuje akceleraci 3D grafiky a OpenGL, dále webový prohlížeč s podporou HTML5. Tuto podmínku splňuje většina moderních prohlížečů, přesto některé z nich WebGL nepodporují. Důvodem je, že tato technologie je zatím relativně nová, stále se vyvíjí a některé firmy ji ať již z důvodu potencionálních bezpečnostních rizik, politiky dané firmy nebo jiných důvodů nepodporují nebo v základním nastavení je podpora vypnuta. Z nejrozšířenějších internetových prohlížečů není WebGL podporováno u Internet Exploreru od firmy Microsoft.

Jelikož je WebGL nízko úrovňové API, vzniklo pro ni mnoho různých frameworků, které vývojářům usnadňují práci s touto knihovnou (viz kapitola 4). Pro použití této technologie je tedy nutné vlastnit již zmíněné softwarové vybavení (webový prohlížeč) a podporovanou grafickou kartu. Pro vývojáře je nutné znát programovací jazyk JavaScript a alespoň základy jazyka GLSL (OpenGL Shading Language) pro psaní shaderů.

WebGL může být použito pro různé vizualizace, zobrazení chemických a biologických struktur, vykreslení matematických funkcí, modelování 3D prostoru. Dále pro demonstrace fyzikálních jevů nebo pro počítačové hry. Hlavní výhodou je běh této technologie přímo v prohlížeči. Autor své dílo může nahrát na svůj server a nemusí svá data s sebou přenášet a starat se o jejich uschování. Dostane se k nim kdekoliv, kde je přístup na internet. Největší nevýhoda spočívá v dostupnosti zdrojových dat všem návštěvníkům stránek, kde je daná aplikace uložena a z toho plynoucí nevhodnost použití například pro komerční účely. To je obecně problém Javascriptu. Řešením je použití některého z řady nástrojů pro „zatemnění“, znečitelnění kódu. V angličtině je možné tyto nástroje nalézt pod pojmem Javascript Obfuscator. Tyto aplikace odstraní z kódu všechny bílé znaky, komentáře a názvy všech proměnných a procedur nahradí za libovolnou náhodnou sekvenci znaků, čísel tak, aby výsledek byl co nejméně čitelný.

2.2.1 WebGL grafická pipeline

WebGL je koncipováno tak, aby přímo přistupovalo ke grafické kartě. Díky tomu je výsledná aplikace velmi rychlá, protože grafiku nemusí počítat procesor, ale grafická karta, která je pro tyto operace optimalizována. Nevýhodou tohoto přístupu je celkový koncept, který je nízko úrovňový, tudíž více složitější a náročnější na seznámení a pochopení této technologie.

Při vytváření aplikace chceme vykreslovat jednotlivé obrazy, to se odehrává prostřednictvím volání daných funkcí pro vykreslení obsahu bufferů. V kapitole 2.1.1 byla vysvětlena grafická pipeline OpenGL ES 2.0. Jak bylo řečeno dříve na této technologii je WebGL založeno a nyní vysvětlím její použití.

Polem vrcholů obsahující nějaký objekt se musí naplnit buffer, který se vytvoří metodou `createBuffer`. Ten se dále musí navázat na daný typ bufferu (`bindBuffer`). Samotné naplnění daty obstará funkce `bufferData`. Dále je potřeba definovat, jak bude náš objekt vypadat pomocí vertex a fragment shaderu. Vykreslení dat obstará `drawArrays`, kde je nutné definovat typ geometrického primitiva, například pro trojúhelníky – `TRIANGLES` nebo útvary složené z napojených trojúhelníků – `TRIANGLES_STRIP`.

2.3 Hry ve webových prohlížečích

Hry v prohlížečích vynikají tím, že není potřeba instalovat žádné další aplikace nebo nástroje. Pouze stačí počkat, než se daná hra načte a můžeme hrát. Některé technologie ale vyžadují pro jejich používání mít nainstalovaný zásuvný modul v prohlížeči. Jedná se ale pouze o jednorázovou instalaci a poté případné aktualizace modulu. Některé technologie vyžadují pro vývoj aplikací dané vývojové prostředí.

Mezi metody vytváření webových her, které nepotřebují žádný zásuvný modul ani speciální vývojové prostředí patří hry na bázi HTML (XHTML) využívající kaskádové styly doplněné o javascriptový kód. Jedná se především o jednoduché 2D hry, které jsou převážně statické (tahové, logické). Díky použití WebGL dostáváme nástroj pro tvorbu 3D her, můžeme vytvářet dynamické

scény se spoustou objektů. Máme k dispozici stínování, osvětlení scény a další standardy používané běžně v 3D grafice.

Další možnost představuje sada nástrojů, které potřebují pro svou činnost na začátku nainstalovat zásuvný modul. Nejznámější a nejrozšířenější technologií pro tvorbu her v prohlížečích je FLASH od firmy Adobe. Hardwarová akcelerace je dostupná v Adobe Flash Player od verze 10. Aplikace se vytvářejí v prostředí Macromedia Flash Professional. Další nástroj je od firmy Microsoft, jedná se o Silverlight. Ten podporuje hardwarovou akceleraci od verze 5.

Hlavní výhodou WebGL je, že nevyžaduje instalaci žádných zásuvných modulů, které mohou představovat bezpečnostní riziko. Dále odpadá problém s aktualizacemi těchto plug-inů, případně problémů s nekompatibilitou mezi různými verzemi prohlížečů.

2.4 Detekce kolizí

Jedná se o termín sledující vzájemnou interakci mezi předměty, například aby hráč ve hře nemohl projít zdí, skrz předměty nebo aby při výstřelu ze zbraně nedošlo projití střely skrz stěnu. Proto musíme mít nějaký mechanismus, který nám tyto stavy zjistí, a my jsme tak mohli na ně adekvátně zareagovat.

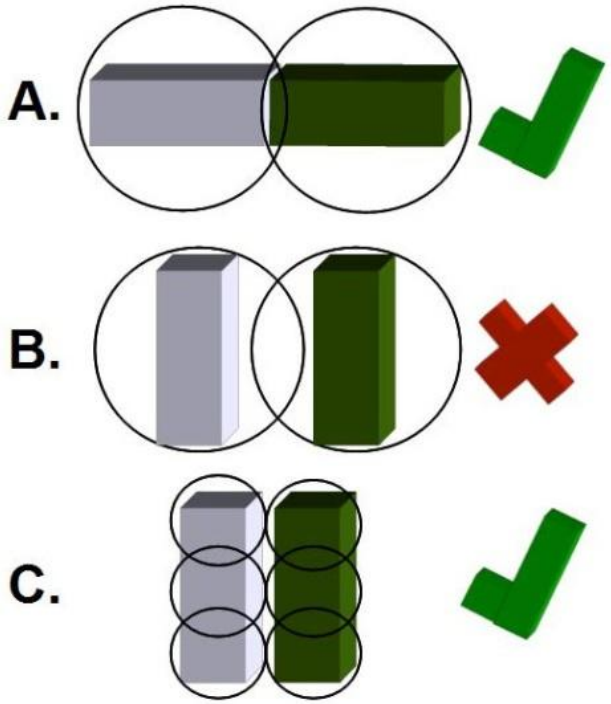
Detekci kolizí[4] můžeme rozdělit na dva druhy, *následnou* (detekce po kolizi) a *předcházející* (detekce před výskytem kolize). Lze se také setkat s označením *nespojitá* (discrete) a *nepřetržitá* (continuous). U nespojité detekce se v každém kroku zjišťuje protnutí jednotlivých těles. Zde je důležité zvolit vhodnou délku kroku výpočtu. Příliš malý krok může mít vliv na výkon počítače a může dojít ke zpomalení aplikace, naopak při velkém kroku může nastat situace, kdy malý předmět pohybující se velkou rychlostí proletí tenkou stěnou, aniž by byla detekována kolize. U nepřetržité detekce předvídáme kolize, počítá se trajektorie pohybu těles. Než se předměty ve scéně aktualizují, zavolá se příslušná metoda pro výpočet detekce a zjišťuje se, zda dojde ke kolizi. Tato metoda poskytuje kvalitní výsledky a zabraňuje stavům, které vznikali u předchozí metody volbou příliš dlouhého kroku.

Dále můžeme metody rozdělit na zjednodušující a přesné. První způsob objekty zjednodušuje a nebere v potaz přesný tvar objektů. Tyto metody jsou velmi rychlé, ale nevhodnou implementací mohou dosahovat špatných výsledků.

2.4.1 Průnik sfér

U této metody každé těleso počítá ze svého středu stejnou vzdálenost do všech směrů, tím se ocitne uvnitř jakéhosi pomyslného kruhu, který tvoří hranici tohoto předmětu. Vzdálenost je dána nejvzdálenějším bodem předmětu od jeho středu. Pokud jiný předmět naruší tento kruh, je detekována kolize. Tato metoda je velmi rychlá a snadná na implementaci. Její nevýhodou je malá kvalita detekce v závislosti na tvaru předmětů. Mějme předmět, který má tvar podobný písmenu I (Obrázek 2-2). Na dvou koncích se detekují kolize u vrcholů písmene, přesně s hranicí skutečného tvaru, ale jelikož hranice tvoří kruh, nachází se na bočních stranách slepé místo. Jiný předmět přibližující se z boku zaznamená kolizi s tímto předmětem, i když je ještě daleko od něj.

Tato metoda se dá vylepšit sférickým dělením objektu. V tomto případě se předmět rozdělí na menší části, tak aby pokrytí jednotlivých sfér bylo efektivnější. Tím se odstraní nevýhoda předešlé metody, ale vzniká další problém, rozhodnout jak mají být předměty děleny na sféry a implementovat tento algoritmus tak, aby pracoval rychle a vyčerpával co nejméně zdrojů (paměť a výkon počítače).

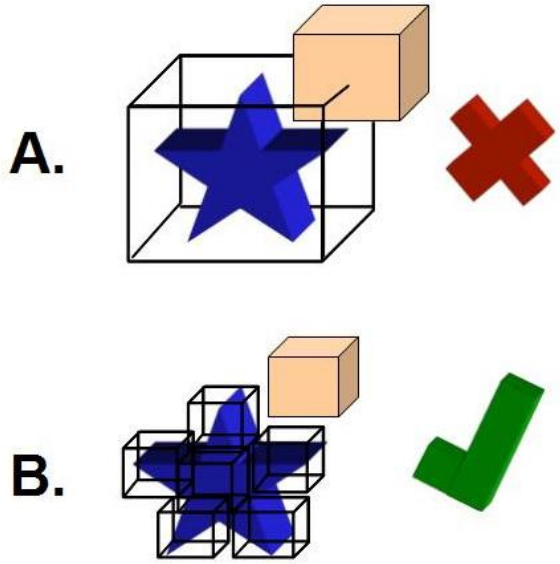


Obrázek 2-2: Detekce kolizí pomocí průniku sfér. A: správná detekce – tělesa jsou v kontaktu; B: chybná detekce – objekty se nedotýkají; C: tělesa jsou rozdělena na více sfér - nedochází k chybám jako v části B.

2.4.2 Obalení těles

Tato metoda u trojrozměrného předmětu rozšíří jeho prostorové souřadnice tak, že se předmět ocitne v pomyslné krabici. Oproti minulé metodě, tato respektuje rozdílné velikosti v různých osách a dosahuje tak lepších výsledků. Celkově patří tato metoda mezi rychlé a používané. Kvalita detekcí záleží na tvaru předmětu, například zubatý objekt, který má své výstupky těsně u sebe povede po jeho umístění do „krabice“ k dobrým výsledkům detekce, ale pokud by měl předmět výstupky daleko od sebe (Obrázek 2-3) a bylo jich málo, nastává opět problém, kdy druhý předmět ještě není plně u něj, a přesto se detekují kolize.

Tento problém se dá vyřešit podobně jako u minulé metody a to kubickým rozdělením předmětu na více částí (Obrázek 2-3B). Nevýhody jsou opět stejné jako u sférického dělení předmětu.



Obrázek 2-3: Detekce kolizí metodou obalení těles. A: celý objekt je obalen boxem, kolize je detekována i v případech, kdy se tělesa viditelně nedotýkají; B: kubické dělení objektu na menší oblasti.

2.4.3 Přesná detekce

Předchozí metody objekt zjednodušovali uzavřením do prostoru definovaného jednoduchým tělesem. Přesná detekce počítá kolize přímo daných objektů. Princip je založen na matematickém základu, kdy zjišťujeme kolizi bodu a předmětu složeného z trojúhelníků (nebo jiného druhu geometrického primitiva). Nejdříve zjistíme, zda bod leží ve stejné rovině jako trojúhelník a poté jestli je uvnitř. Model ale může být složen z mnoha trojúhelníků a ve scéně může být takových objektů spousty. Tato metoda je tedy náročnější na výkon počítače, ale dosahuje nejlepších výsledků. Ke snížení nároků na počítač se používají různé optimalizace. Více o přesné detekci je možné dočíst se v práci M. C. Lina[4].

3 Návrh hry

Vytvořit komplexní 3D hru vyžaduje řadu znalostí, matematiky, fyziky a dalších věd, aby se mohl naprogramovat reálně chovající pohyb postavy nebo předmětu. Při tvorbě velkých her se na ní podílí celá řada lidí, kteří se specializují na jednotlivé části. Pokud chce nějakou hru vytvořit menší tým lidí nebo jeden člověk sám, tak mu to zabere mnohem delší dobu nebo může použít již vytvořené nástroje a upravit si je podle svých požadavků. Řeč bude o frameworcích, které některé činnosti a operace zapouzdřují a zjednodušují jejich použití. Tyto nástroje dále mohou implementovat například herní fyziku. Vývojář se tak nemusí starat detailně o celou problematiku.

3.1 WebGL frameworky

I když je WebGL ve vývoji již nějakou dobu, stále se jedná o relativně mladou technologii (prototyp vznikl v roce 2006, finální verze 1.0 byla vydána v březnu 2011). V době psaní této práce se na trhu dosud neobjevila žádná knižní publikace věnující se čistě této problematice. Pro seznámení se s touto technologií existuje na internetu spousta příkladů a tutoriálů. V neposlední řadě jako učební materiál poslouží i procházení kódu již hotových demo aplikací. Pro snadnější vstup do světa WebGL se také může postarat některý z frameworků.

Jak už bylo zmíněno, WebGL je nízkoúrovňová technologie. Pro vytvoření i jen primitivní aplikace je potřeba se postarat o řadu věcí. Tento přístup může začátečníky o tuto problematiku odradit od další práce. Existuje řada frameworků, které se snaží tyto nedostatky odstranit a ulehčit tak programování. Použít můžeme například Three.js[1] (Jedná se o javascriptovou 3D knihovnu, která je zaměřena na jednoduchost řešení. Vhodná pro začátečníky s WebGL), PhiloGL[2] nebo GLGE[3]. Pro každý je možné nalézt řadu demo příkladů, k dispozici jsou dokumentace, takže začít pracovat s některým z těchto nástrojů není nijak složité.

3.1.1 Knihovna Three.js

Pro implementaci hry jsem se rozhodl pracovat s knihovnou Three.js (verze 47 – 14.1.2012). Z hlediska funkcionality nabízejí všechny frameworky obdobné prvky. Three.js jsem si vybral díky široké škále hotových demo příkladů, které předvádějí jednotlivé vlastnosti knihovny. Další výhodou jsou rozsáhlé diskuze k těmto příkladům, kde jsou řešeny podrobnosti případně objevené chyby a nedostatky. Hlavní nevýhodou je zatím neúplná dokumentace knihovny.

3.1.2 Modelování objektů

Jak bylo vysvětleno v kapitole o fungování WebGL, každý objekt (model) ve scéně je složen z jednoduchých primitiv. Vytvářet modely ručně a ukládat do pole indexy jednotlivých vrcholů je velmi nepraktické a vhodné maximálně jako ukázka v demo příkladu. Nabízí se otázka, v čem takový model vytvořit a následně v jakém formátu jej uložit. Formát uložení dat je podstatný z hlediska použití v programu. Abychom nemuseli soubor s uloženým modelem ručně dekódovat, bylo by výhodné použít takový, se kterým si kód v javascriptu poradí. Řešením je formát JSON[4]. Pro jehož zpracování máme vhodné funkce přímo v daném programovacím jazyku. Ale vraťme se k první

[1] https://github.com/mrdoob/three.js
[2] http://www.senchalabs.org/philogl/
[3] http://www.glge.org/
[4] http://www.json.org/

otázce, v čem model vytvořit. Tyto služby nám poskytují speciální modelovací nástroje, jmenujme například komerční program 3ds Max[1] nebo opensource řešení program Blender[2]. Tyto programy sice mívají své vlastní proprietární formáty pro uložení dat, ale zároveň umožňují export do dalších formátů. Pokud daný nástroj neumí export dat přímo do JSON, lze na internetu nalézt řadu nástrojů, které převod z běžných formátů těchto programů zvládají.

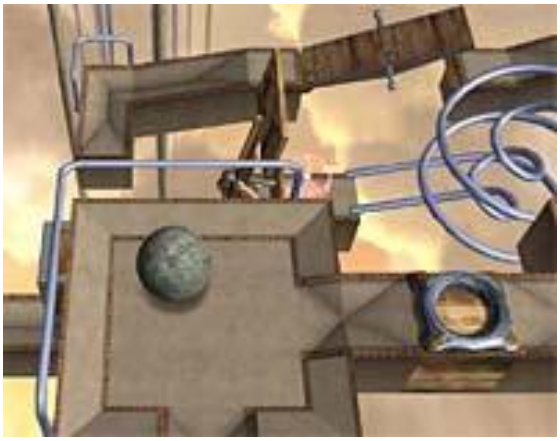
Knihovna Three.js nabízí načítání více formátů dat pro uložení modelů. Kromě zmíněného Json je to Collada nebo lze načítat data v binární podobě.

3.2 Koncept hry

Po seznámení se s technologií WebGL se nyní zaměřím na stěžejní část práce a to návrh 3D hry Inball, která bude implementována jmenovanou technologií. Jedná se o hru žánru arkáda a plošinovka založenou na hrách typu Neverball[3] (Obrázek 3-1)[4] a Ballance[5] (Obrázek 3-2)[6]. Hráč ve hře ovládá kuličku pomocí klávesových šipek a jeho cílem je ji dostat daným prostředím do stanoveného cíle. Přesný cíl hry je dán v závislosti na zvoleném herním módu. U hry Neverball hráč neovládá přímo kuličku, ale ovládá ji nepřímo nakláněním prostředí.

Obrázek 3-1: Screenshot ze hry Neverball.

[1] http://usa.autodesk.com/3ds-max/
[2] http://www.blender.org/
[3] http://neverball.org/
[4] Zdroj: http://neverball.org/screenshots.php
[5] Původní stránky hry již neexistují, následující odkaz je na webový archiv: http://web.archive.org/web/20071011004707/www.ballance.org/news.htm
[6] Zdroj: http://web.archive.org/web/20071011005133/http://www.ballance.org/game.htm



Obrázek 3-2: Screenshot ze hry Ballance.

3.2.1 Struktura a typy hracího pole

Herní prostředí bude několika druhů, od labyrintu, kdy půjde především o co nejrychlejší nalezení správné cesty skrz, s možností v některých místech výběru z více cest (některé cesty budou slepé uličky, jiné se postupně napojí na správný směr, ale po delší dráze než při výběru správné cesty na křižovatce).

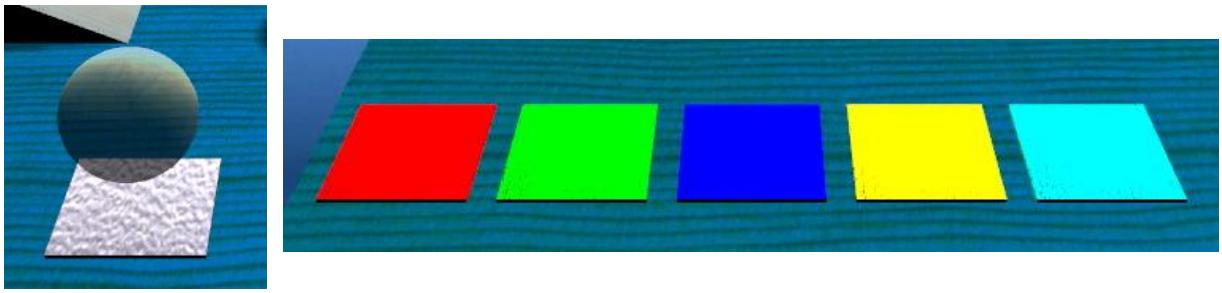
V dalším typu prostředí půjde pro změnu o přesnost vedení kuličky po dráze. Cesta se bude skládat z úzkých částí, po jejichž krajích nebude žádná zábrana. Vypadnutím míčku z dráhy do prostoru kolem dojde ke ztrátě herního života. Po vyčerpání všech životů hra končí. Na začátku hráč získává 3 životy. Na některých místech dráhy budou ukryty bonusové životy.

Prostředí bude v některých částech interaktivní. Například po rovných úsecích bude následovat ostré zabočení dráhy a v místě změny směru bude pohyblivá nebo za určitých okolností rozbitelná překážka. Bude se jednat o druh zábradlí nebo tenké zdi. Tyto části budou od zbytku okolí rozeznatelné odlišnou barvou nebo tvarem. Bude záležet na intenzitě střetu kuličky a předmětu (vliv rychlosti míčku a jeho typu, viz typy kuliček).

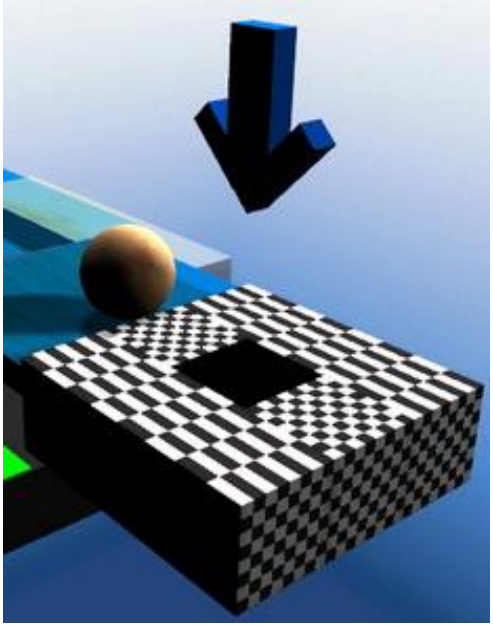
Dále se budou v prostředích vyskytovat speciální pole, přidávající bonusové skóre a měnící efekt zobrazení nebo přidávající život (Obrázek 3-3).

Speciální typy polí:

- **bonusový život:** přidá hráči *+1* život. Pole sestává ze základny v podobě čtverce a průhledné malé kuličky, která kmitá nad tímto polem.
- **bonus skóre:** + *500* a změna efektu podle barvy políčka (více o efektech v kapitole 3.3):
  - červená – Toon Shader verze 1
  - zelená – Toon Shader verze 2
  - modrá – Invertování barev
  - žlutá – odstíny šedé
  - azurově modrá – rozmazání obrazu
- **cílové místo** (Obrázek 3-4): cílové místo se nachází v každém herním kole právě jedno. Je ohraničeno poli, které mají texturu černobílé šachovnice značící cíl. Konec levelu nastává zapadnutím kuličky do černého středu a zobrazením informační hlášky. Pro snazší nalezení cíle se nad jeho středem nachází modrá šipka.



Obrázek 3-3: Speciální typy polí. Vlevo pole s průhlednou kuličkou – bonusový život. Barevná pole pro změnu efektu a přidání bonusového skóre: červená – toon shader v1, zelená – toon shader v2, modrá – invertování barev, žlutá – odstíny šedé a azurově modrá – rozmazání obrazu.



Obrázek 3-4: Cílové místo hry. Level končí spadnutím kuličky do černého místa pod šipkou.

3.2.2 Druhy hracích kuliček

Hra bude obsahovat 3 druhy kuliček (Obrázek 3-5):

- **dřevěná** (výchozí typ)
- **kovová**
- **skleněná**

Kuličky se liší svojí pohyblivostí a váhou. Skleněná kulička je nejpohyblivější a nejlehčí, reaguje okamžitě na změnu směru a dosahuje nejvyšší rychlosti. Její nevýhodou je nízká váha, díky které disponuje malou silou při kontaktu s pohyblivými předměty. Díky této skutečnosti, pak s těmito předměty nezvládne téměř pohnout. Kovová kulička je opak skleněné. Je nejpomalejší a manipulace s ní je neohrabaná. Zrychlení a zpomalení se odehrává se zpožděním, stejně tak změny směru pohybu. Posledním typem je dřevěná kulička. Jde o výchozí kuličku, jejíž vlastnosti odpovídají kombinaci předešlých kuliček. Tato kulička je rychlejší než kovová, i když nedosahuje maximální rychlosti jako skleněná. Na změnu pohybu reaguje jen s malým zpožděním. Změnu typu kuličky může hráč

provádět libovolně v průběhu hry, pouze s krátkým časovým odstupem od předchozí změny, aby se zabránilo možnému podvádění hráče.

Nyní popíši několik situací, kdy má smysl provést změnu kuličky a jaké to má výhody. Kvůli své váze kovová kulička není schopná překonat většinu nakloněných plošin umístěných ve hře, pouze ty, které ve své blízkosti mají jinou nakloněnou plošinu, kterou může využít pro získání rychlosti a setrvačnosti. Přínos této kuličky je v situacích, kdy je potřeba pohnout pohyblivými předměty a zároveň není prostor pro nabrání rychlosti nebo dostatek herního času pro použití jiného typu. Skleněná kulička není zase příliš vhodná na úseky, kde je potřeba provádět pohyb s velkou přesností, např. úzké římsy nad propastí. Všechny levely jsou koncipovány tak, aby šly dohrát výchozí tj. dřevěnou kuličkou. Pro ostatní typy toto pravidlo neplatí pro všechna herní kola.

Obrázek 3-5: Typy herních kuliček. Zleva dřevěná, kovová a skleněná

3.2.3 Herní módy

Hra bude obsahovat několik herních módů:

- **Classic**
- **Best Time**
- **Survival**

První mód - klasické hraní, kde se herní čas po dokončení hry přepočítává na skóre. Cílem je nahrát co nejvyšší skóre a projít všemi úrovněmi.

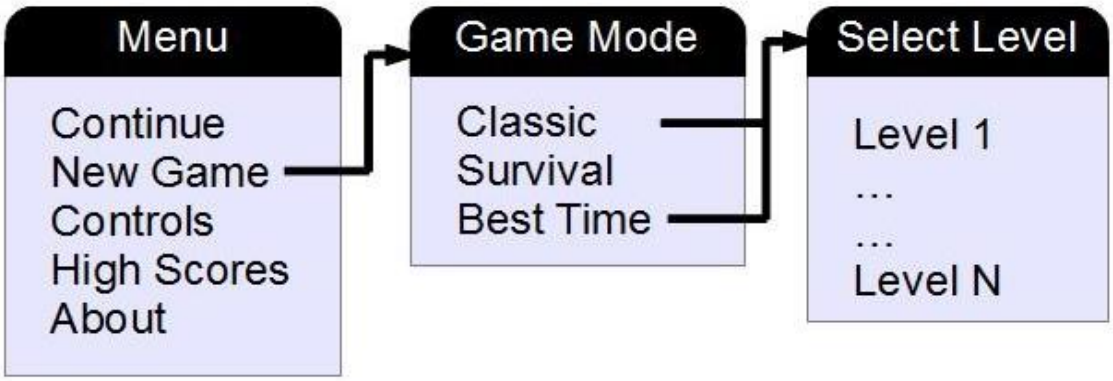
Dalším herním módem je *best time*. Cílem je projít danou úrovní v co nejkratším čase. Hodnota best time se uchovává pro každý level. Dalším typem je *survival mode*, hráč začíná od prvního kola a na začátku je mu přidělen určitý časový limit. V tomto limitu musí dojít do cíle. Pokud mu nějaký čas zůstane, přičítá se mu k časovému limitu pro další level jako bonusový čas. Na konci každého kola se zbývající čas přepočítává na skóre. Cílem je projít všechna kola před vypršením časového limitu a nahrát co nejvyšší skóre. Pokud hráči v průběhu kola dojde čas, končí celá hra bez ohledu na zbývající počet herních životů.

3.2.4 Hlavní nabídka

Po spuštění aplikace se nejprve spustí hlavní nabídka hry (Obrázek 3-6). Tlačítko *Continue* slouží pro navrácení se zpět do hry. Při novém spuštění aplikace, kdy ještě nebyla žádná hra vytvořena, je toto tlačítko deaktivované. Tlačítko *Controls* zobrazí nové okno popisující ovládání hry s popisem jednotlivých kláves. Položka *About* zobrazí základní informace o aplikaci a autorovi. Tlačítko *High*

Scores zobrazí tabulku nejvyššího dosaženého skóre v podobě jména a hodnoty skóre u herních módů *Classic* a *Survival* a časového údaje pro mód *Best Time*.

Stisknutím tlačítka *New Game* dojde k zobrazení okna s výběrem herního módu (viz kapitola 3.2.3). Při výběru módu *Survival* se ihned spustí nová hra. Předchozí rozehraná hra se ztratí a nelze se k ní vrátit. Po výběru zbývajících dvou herních módů se zobrazí nové okno s výběrem levelu. Herních kol je plánováno 4 – 6.



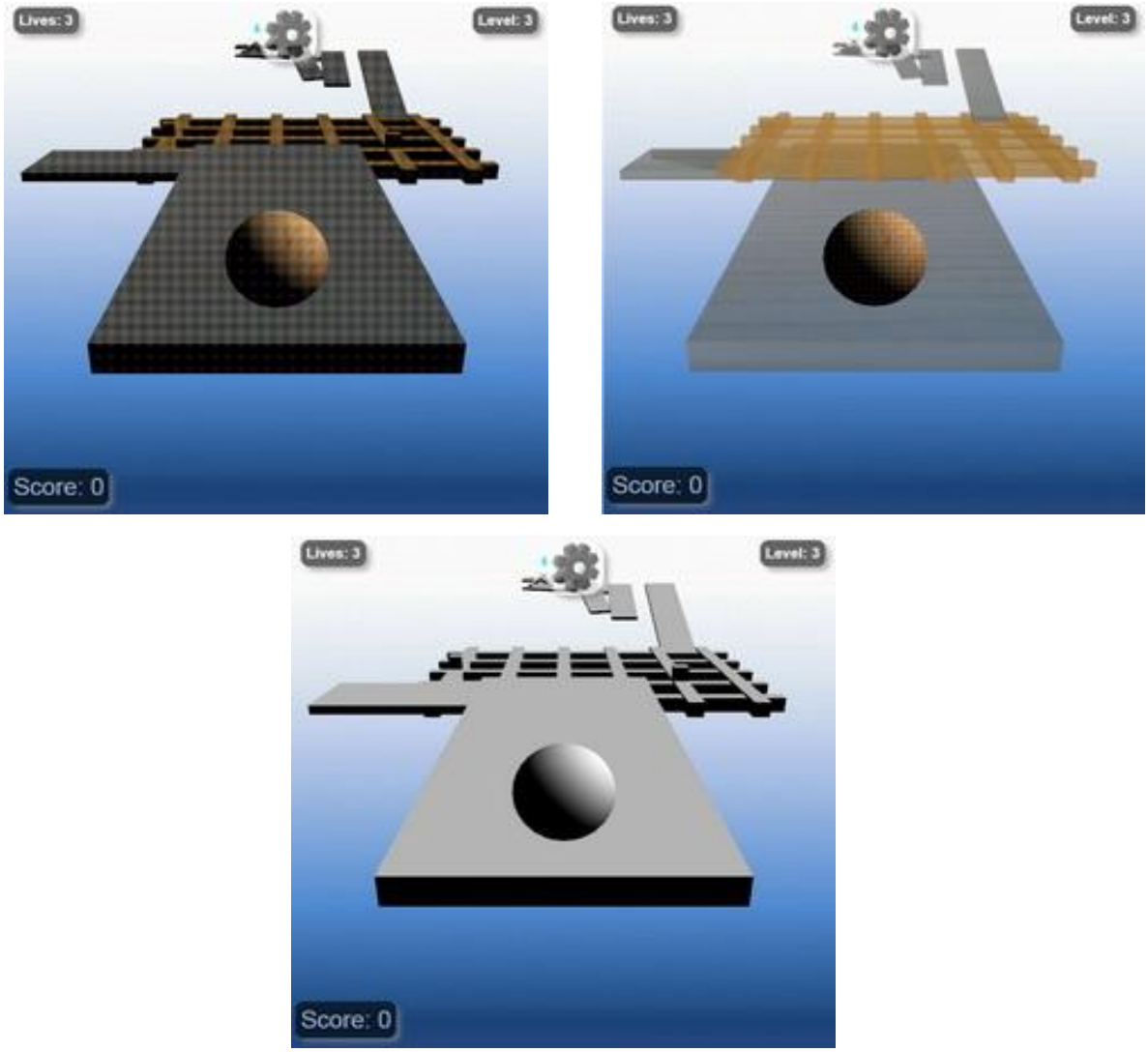
Obrázek 3-6: Struktura hlavní nabídky.

3.3 Shadery

V části popisující OpenGL grafickou pipeline byly zmíněny shadery, konkrétně fragment a vertex shader. Knihovna Three.js dovoluje používat jak vestavěné, tak i vlastní shadery. V aplikaci jsou použity obě možnosti. Vestavěné pro efekt environment mapping, osvětlení a pro vytvoření pozadí hry. Vlastní shadery jsou použity dvěma způsoby. První slouží ke změně materiálů objektů, druhý ke změně celé scény tzv. Postprocessing. V obou případech jsou použity pouze jednoduché vertex shadery, které plní pouze základní činnost, předávají fragment shaderům hodnoty normál vrcholů a souřadnice jednotlivých bodů. Hlavní činnost se odehrává ve fragment shaderech.

3.3.1 Shadery pro změnu materiálu objektu

U těchto efektů se nevyužívají informace o současném materiálu daného objektu, ale pracuje se s novou barvou nebo texturou. Zde nelze použít osvětlení, které řeší knihovna Three.js. U efektu 2 (Obrázek 3-7) není žádné osvětlení ani stíny, kromě hrací kuličky, která používá efekt 1. Zbývající efekty používají falešné stíny[16]. Stíny se počítají skalárním součinem pozice světelného zdroje a normály daného bodu. U herní kuličky je tento proces fixní, stín se neaktualizuje se změnou rotace. Efekt 1 dále upravuje barvu jednotlivých bodů podle jejich souřadnic a hodnoty času, tím je dosaženo efektu pohyblivých částí na povrchu předmětů.



Obrázek 3-7: Shader efekty pro změnu materiálů. Zleva nahoře efekt 1, efekt 2 a dole falešné světlo

3.3.2 Vykreslování scény do textury

Při těchto efektech se neupravují zvlášť jednotlivé předměty, ale celá scéna se zpracuje najednou prostřednictvím jednoho fragment shaderu daného efektu. Je vytvořena nová scéna, která obsahuje jediný objekt, plátno, které má stejnou velikost jako okno aplikace a je umístěno těsně před kamerou. Texturu plátna tvoří původní scéna. Tím vzniká dojem, že se díváme pořád na stejnou scénu. Nejedná se přímo o techniku post processingu, ale metodu podobnou deferred shadingu[17].

Při mapování textury na plátno dochází k převrácení obrazu. Změnu je možné provést nastavením mapování textury prostřednictvím frameworku nebo pomocí vertex shaderu. V aplikaci je tento problém řešen druhým způsobem. Následuje popis jednolitých efektů.

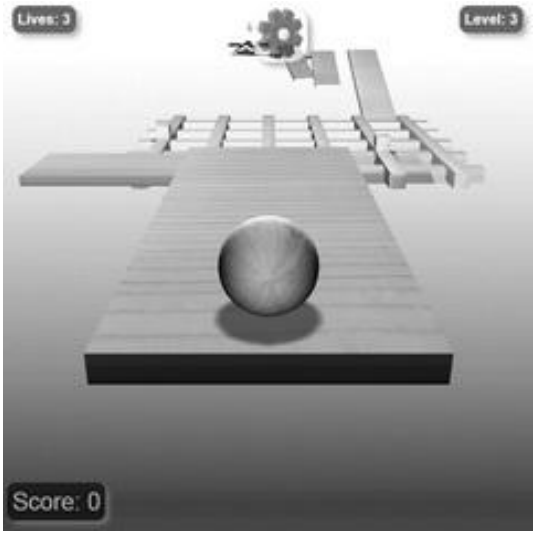
Odstíny šedé

Pro každý bod obrazu je vypočítána intenzita podle empirického vzorce 3-1[6], která je přiřazena všem barevným složkám.

$$I = 0{,}299R + 0{,}587G + 0{,}144B \tag{3-1}$$

kde I … výsledná intenzita

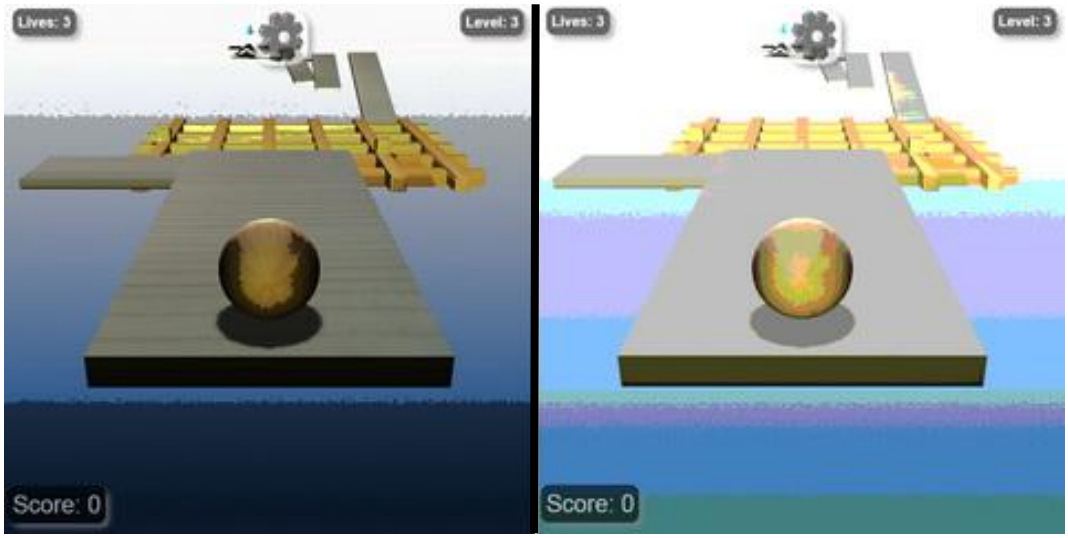
R, G, B … jednotlivé barevné složky – červená, zelená a modrá
Na Obrázku 3-8 je vidět výsledek po převodu scény do odstínů šedé.



Obrázek 3-8: Odstíny šedé.

Toon shader

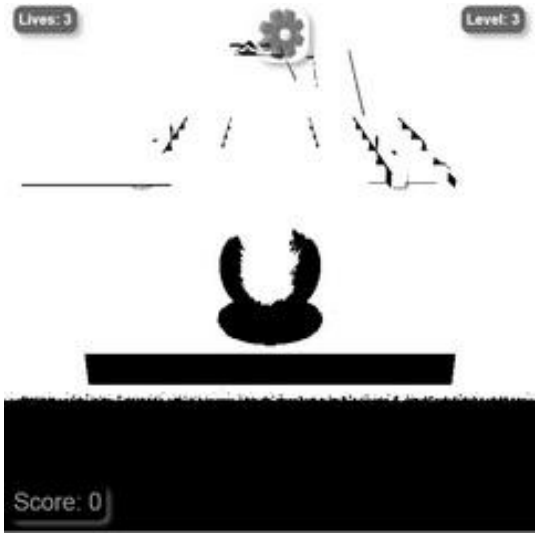
Jedná se o efekt podobný komiksovému vzhledu[18], při kterém je omezen celkový počet zobrazených barev. Komiksový vzhled mívá navíc zvýrazněné hrany černou barvou. U prvního efektu jsou všechny tři barevné složky redukovány současně a nastaveny na stejnou hodnotu podle barevné intenzity (vzorec 3-1). Jsou použity 4 prahové hodnoty intenzity: 0.95, 0.5, 0.05 a hodnoty menší. V druhém případě je každá barevná složka redukována zvlášť. Není zde použita jako prahová veličina intenzita, ale příslušná barevná složka, která používá 5 prahových hodnot: 0.8, 0.75, 0.5, 0.25 a čísla menší. Na Obrázku 3-9 je možné vidět rozdíl mezi oběma efekty. Verze 1 používá menší barevnou paletu, to se projevilo ztrátou některých detailů, které splynuly s okolím.



Obrázek 3-9: Toon Shader. Zleva verze 1, 2.

Černo bílý obraz

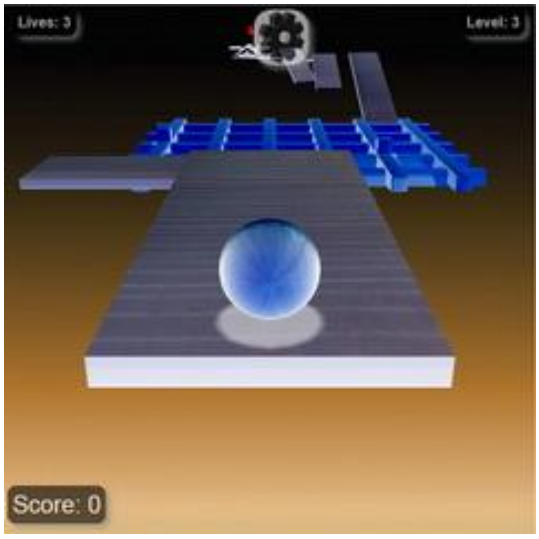
Zde je barevná paleta redukována na 2 barvy – bílou a černou. Stejně jako u odstínů šedé se spočítá pro každý bod intenzita a podle prahové hodnoty, která je nastavena na 0.5 se určí výsledná barva. Efekt je znázorněn na Obrázku 3-10. Z obrázku je vidět, že některé objekty nejsou téměř viditelné nebo jenom některé části. Pro praktické využití ve hře, je tento efekt spíše nevhodný. Vylepšení může představovat například použití náhodného rozptýlení nebo ditheringu[6].



Obrázek 3-10: Černo bílý obraz.

Invertované barvy

Tento efekt provede u všech bodů změnu každé barevné složky za její číselný doplněk. Výsledek je zobrazen na Obrázku 3-11. Pro některé barvy, například hnědou, představuje barevný doplněk barva modrá. To na první pohled jednoznačně neuvádí, zda je tato změna správná a jedná se o barvu z opačného spektra. Pro potvrzení správnosti uvažuji černou barvu, jejíž doplněk je barva bílá a obráceně. Zde je změna jasná u stínů, které byly černé, a naopak horní část pozadí byla v původní scéně bílá.



Obrázek 3-11: Invertované barvy.

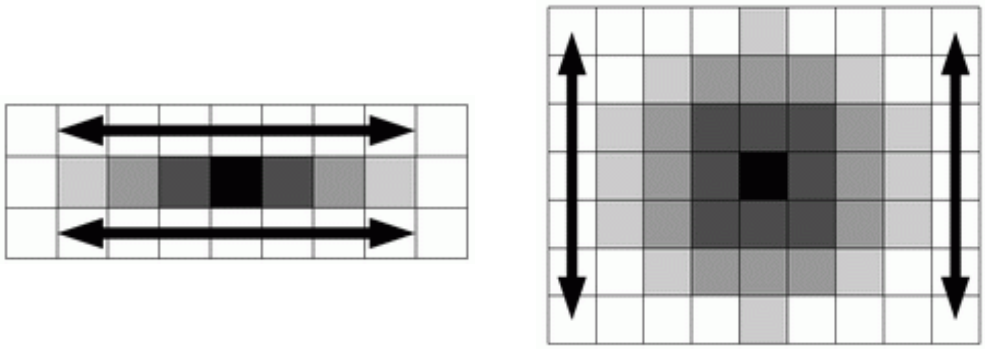
Rozmazání obrazu

Tento efekt je založen na Gaussově rozložení. Je aplikován na mřížce 7 krát 7 bodů. K prostřednímu bodu mřížky se postupně přičítají hodnoty barev okolních bodů vynásobené konstantou danou Gaussovou funkcí. Výpočet konstant se provede podle vzorce 3-2:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3-2}$$

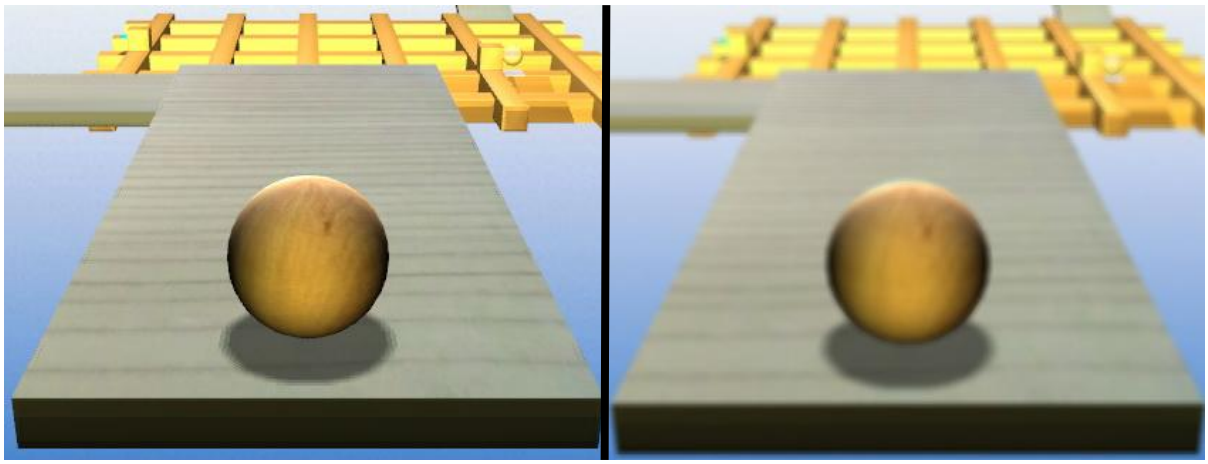
kde σ je směrodatná odchylka a μ – střední hodnota. Mí a sigma tvoří parametry Normálního rozložení. Sigma udává, jak velké bude rozmazání. Při implementaci jsem vycházel z matematického základu $\sigma = 3$, $\mu = 0$. Před vykreslením barvy je ještě nutné vydělit ji součtem jednotlivých vah.

Pro tento efekt je obvyklé zpracování na dvakrát, nejdříve se provede horizontální rozmazání a poté vertikální na již rozpracovaný obraz s horizontálním rozmazáním, jak znázorňuje Obrázek 3-12.



Obrázek 3-12: Horizontální a vertikální rozmazání bodu.

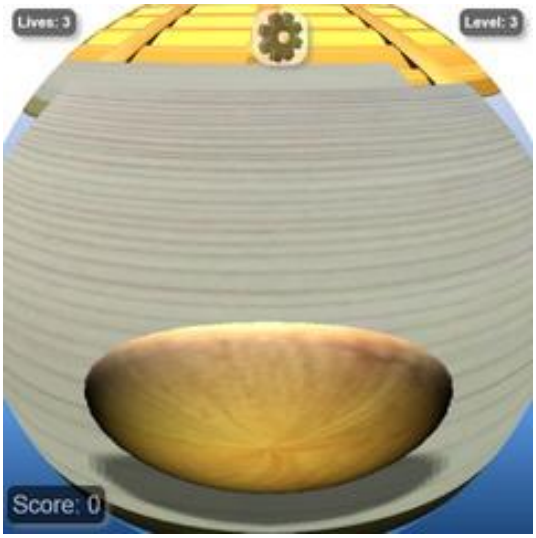
Důvod pro rozdělení do dvou částí je v efektivitě zpracování. Na mřížce 7x7 při dvoufázovém průchodu je nutné pro každý bod obrazu zpracovat 14 vzorků, při jednofázovém zpracování 49 vzorků. Aplikace je pro jednoduchost implementována jednoprůchodovým rozmazáním. Výsledný efekt je na Obrázku 3-13 společně s normálním obrazem hry pro porovnání vlivu rozmazání.



Obrázek 3-13: Porovnání normálního a rozmazaného obrazu.

Rybí oko

V předchozích případech byla scéna vykreslována do rovinného útvaru. Zde je scéna v podobě textury mapována na sférický objekt. Při pohybu herní kuličky do stran tak vzniká dojem, že se koule s namapovanou scénou otáčí. Na rozdíl od předchozích efektů post processingu, je u rybího pohledu kladen důraz na rozlišení textury. V našem případě vykreslované scény. Jelikož textura musí být potažena přes celou plochu koule, musí být větší, aby nedocházelo k aliasingu. Efekt je znázorněn na Obrázku 3-14.



Obrázek 3-14: Rybí pohled.

3.4 Herní fyzika

Použití frameworku Three.js řeší pouze část problematiky tvorby hry, grafickou podobu. Dále je potřeba věnovat se herní fyzice, zvláště řešení detekce kolizí a pohybu předmětů (odrazy, zrychlení, zpomalení atd.). V obou případech je možné naprogramovat si potřebné funkce vlastnoručně, to považuji dokonce za nejlepší variantu. Autor pak přesně ví, co jeho knihovna umí a jak s ní zacházet efektivně. Nevýhodou je nutnost nastudovat teoretické znalosti z dané problematiky dopodrobna. Poté naimplementovat, odzkoušet a odladit. Řešením umožňující se plně věnovat vlastnímu vývoji hry ve WebGL, která je náplní této práce, představuje použití výše zmiňované knihovny a dále knihovny řešící herní fyziku.

3.4.1 Physics.js, Microphysics.js

Aplikace používá pro výpočet herní fyziky knihovnu Ammo.js. Tato knihovna nebyla mou první volbou. V první fázi vývoje jsem použil knihovnu physics.js[1] a THREEx.microphysics.js[2], která prvně zmiňovanou knihovnu zapouzdřuje a usnadňuje vzájemné použití s knihovnou Three.js. Velkou výhodou a zároveň v některých ohledech nevýhodou je malá velikost knihovny. Obsahuje jenom ty nejnutnější věci a díky tomu pracuje rychle. Kolize umí počítat pouze pro tělesa typu koule a kostka (kvádr). Dále knihovna zvládá pohyb těles a řeší reakce na střety s dalšími předměty, odrazy, pohyb jinými předměty atd. Důvod proč jsem tuto knihovnu nakonec nepoužil, byl ten, že knihovna

[1] http://learningthreejs.com/blog/2011/10/17/lets-make-a-3d-game-microphysics-js/
[2] autor: http://twitter.com/#!/pyalot

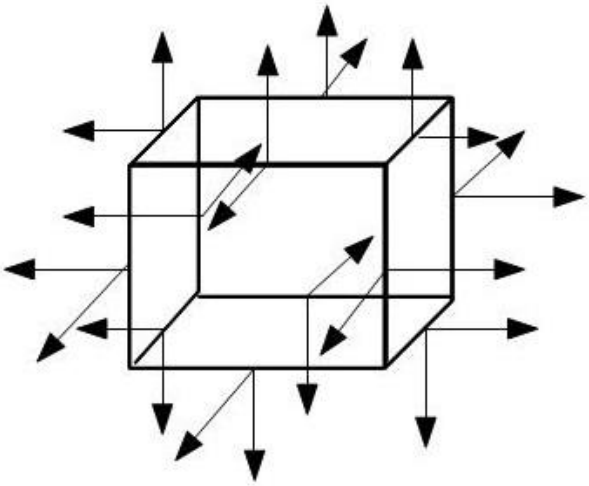
neobsahuje nástroje umožňující rotaci předmětů. Herní level by tak nemohl obsahovat žádné nakloněné roviny a design prostředí by byl příliš jednoduchý.

3.4.2 JigLibJS, JigLibJS2

Další mou volbou byli knihovny JigLibJS[1] a JigLibJS2[2]. Obě vychází z knihovny JigLib napsané v jazyce C++. Druhá jmenovaná knihovna není přímý port z originální knihovny, ale z JigLibFlash, která je v jazyce Flash. Obě knihovny si tak jsou podobné, většina funkcí má podobné nebo stejné jméno. Jelikož je knihovna Three.js stále ve vývoji, nepoužil jsem tyto fyzikální knihovny z důvodu problémů s novější verzí frameworku (verze 47).

3.4.3 THREE.Ray

V další fázi jsem zkoušel řešit detekci kolizí vlastnoručně pomocí prostředků dostupných přímo v knihovně Three.js, použitím paprsků. Pomocí vektorů se nastaví paprsek a kontroluje se jeho protnutí s ostatními objekty ve scéně. Celý předmět poté pracuje s několika paprsky, vyšší počet představuje kvalitnější detekci. Toto řešení je celkem jednoduché pro tvar předmětu krychle, která má všechny hrany rovné. Pro kouli je nutné zvolit mnohem vyšší počet paprsků a správně je nastavit. Kromě kolizí hra vyžaduje i řešení pohybu těles, hrací koule a případných dalších interaktivních předmětů. Proto jsem opustil variantu vlastní detekce kolizí a hledal vhodnou fyzikální knihovnu. Na Obrázku 3-15 je zobrazena krychle využívající 24 paprsků.



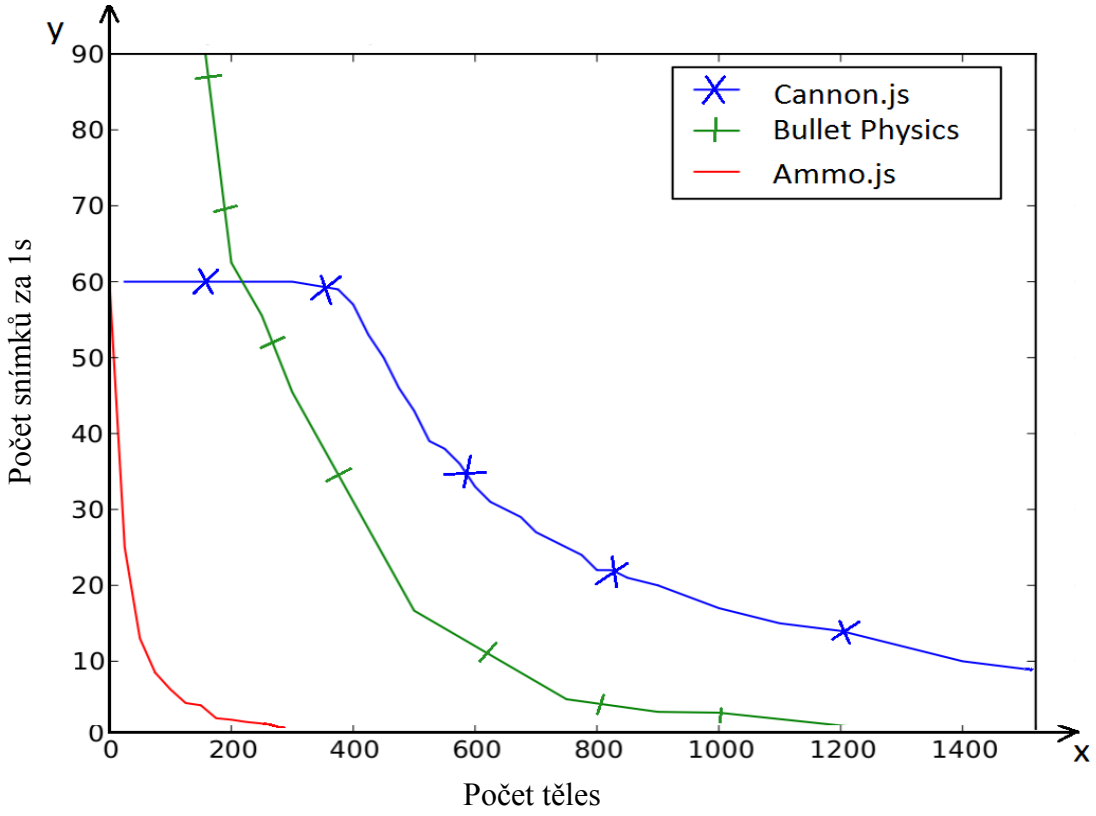
Obrázek 3-15: Detekce kolizí pomocí paprsků.

3.4.4 Ammo.js

Nakonec jsem zvolil Ammo.js[3]. Pro tuto knihovnu neexistuje zatím žádná dokumentace, ale při práci může pomoci řada demo příkladů nebo dokumentace knihovny, ze které je ammo.js portována - Bullet Physics[4]. Slabší stránkou Ammo.js je výkon. Největší váhu na tom má použitý programovací

[1] http://www.jiglibjs.org/
[2] https://github.com/bartdeboer/JigLibJS2
[3] https://github.com/kripken/ammo.js/
[4] http://bulletphysics.org/wordpress/

jazyk Javascript. Porovnání s jinými knihovnami je na Obrázku 3-16[1]. Cannon.js[2] je kompaktní malá knihovna zaměřena především na rychlost. Obsahuje pouze nejnutnější funkce. Pro potřeby hry Inball je tato knihovna s jejími možnostmi nedostatečná. Jelikož hra není koncipována tak, aby obsahovala velké množství pohybujících se objektů v jeden okamžik, nepředstavuje omezení knihovny Ammo.js velký problém. Tento nedostatek je dále kompenzován možnostmi knihovny, které nabízí. Například již zmíněné jednoduché rozlišení statických a dynamických předmětů podle jejich hmotnosti, řešení odrazů, tření, udělení hybnosti předmětům, jednoduché použití kolize detekcí atd.



Obrázek 3-16: Porovnání fyzikálních knihoven.

Použití Ammo.js

Objekty ve scéně jsou vykresleny pomocí WebGL využívající Framework Three.js. Pokud chceme, aby na některé objekty působilo okolí, případně gravitace, musí se jim vytvořit odpovídající fyzikální model. Objekty tak existují dvakrát. Za prvé v obrazové podobě a poté fyzikální, která není zobrazena, ale odpovídá viditelnému předmětu. Zobrazené předměty se následně při překreslování scény aktualizují s jejich fyzikálními protějšky a dostáváme tak výsledný dynamický obraz. Pro rozlišení objektů, které tvoří nepohyblivé části levelu a pohyblivé se používá jejich hmotnost. Přiřazením nulové hmotnosti předmětu zařídíme, že na něj nebude působit gravitace ani jiná tělesa.

Při aktualizaci dynamických předmětů s jejich fyzikálními modely se kromě jejich pozice upravuje i jejich natočení. Three.js pro rotace používá v základním nastavení Eulerovu metodu, ale Ammo.js používá kvaterniony. Proto je nutné ve frameworku změnit toto nastavení.

[1] převzato z článku: http://granular.cs.umu.se/browserphysics/?p=729
[2] http://schteppe.github.com/cannon.js/

3.5 Pohyb herní kuličky

Pohyb hráčovy kuličky může být řešen dvěma způsoby. Přímo, kdy pohyb vychází přímo od samotné kuličky jako by byla malé autíčko, které má svůj motor a samo se uvádí v pohyb. Nebo může být ovládána nepřímo jinými předměty jako v kulečníku pomocí tága nebo jako v již zmíněné hře Neverball nakláněním okolního prostředí.

3.5.1 Přímý pohyb

Jako první možnost ve fázi návrhu hry a později implementace jsem zvolil přímé ovládání kuličky. Využitím knihovny Ammo.js a její funkce `setLinearVelocity`, se danému objektu (herní kuličce) určí směr pohybu pomocí vektoru. Nastavení rychlosti se dosáhne vynásobením tohoto vektoru požadovaným číslem.

Nevýhodou tohoto řešení je nekonečné trvání pohybu tělesa. Na kuličku přestane působit gravitace a pohybuje se pořád stejnou rychlostí. Řešením gravitace je zadávat vektor pro pohyb i pro směr dolů. Pro zrychlení a zpomalení bylo nutné uchovávat hodnotu naposledy použité rychlosti a aktuální směr pohybu kuličky a hodnotu rychlosti průběžně měnit. Dále bylo nutné řešit přechod pohybu do jiného směru, aby se nejdříve kulička plynule zastavila ve stávajícím směru a poté zrychlovala v nově zadané trajektorii. Dalším problémem byla nemožnost měnit plynule směr pohybu. Funkce pro nastavení pohybu nedovoluje nastavit šikmý pohyb pod různými úhly, ale pouze v hodnotě 45 stupňů v každém kvadrantu.

Tímto řešením nebylo dosaženo očekávaných výsledků a to hlavně v plynulém pohybu v rozmezí 360 stupňů. Použitím funkce `setLinearVelocity`, přestanou na dané těleso působit ostatní fyzikální vlivy řešené knihovnou Ammo.js, gravitace, odrazy a působení ostatních interaktivních předmětů. Kvůli těmto důvodům jsem přešel k nepřímému pohybování kuličky.

3.5.2 Nepřímý pohyb

U tohoto typu řešení je nutné zvolit vhodnou formu udílení pohybu kuličce. V situaci, kdy mezi hráčovou kuličkou a předmětem, který udělí kouli rychlost a směr pohybu, stojí další interaktivní předmět nebo objekt tvořící prostředí levelu, se tento předmět udělující pohyb ke kuličce vůbec nedostane skrze překážku anebo bude pohybovat jiným interaktivním předmětem.

Pokud se interaktivní předmět, označme jej A, přemístí tak, aby alespoň částečně byl uvnitř jiného objektu B, dojde k reakci srovnatelné, jako by předmět A narazil v rychlosti do předmětu B. Této vlastnosti jsem využil při implementaci. Při stisku klávesy se vytvoří nový objekt, který se umístí dovnitř herní kuličky tak, aby byl celý v ní a nemohl ovlivňovat jiné předměty v blízkém okolí. Tento objekt se umístí kousek od středu kuličky, do opačného směru než je stisknuta klávesa. Vyvolaná reakce poté rozpohybuje kuličku směrem podle stisknuté klávesy.

Na rozměrech pomocného objektu udělujícímu pohyb kuličky nezáleží, velikost vyvolané reakce tj. rychlost kuličky ovlivňují pouze hmotnosti obou předmětů. Pomocný objekt je vytvořen pouze ve fyzikální podobě, grafické vykreslení ve scéně nemá žádný účel, vyjma fáze testování a ověřování, že je celý objekt skryt uvnitř kuličky. Objekt je vymazán ze scény ihned po udělení pohybu kuličce, nečeká se na puštění klávesy, protože při delším stisku by se kulička přesunula na jiné místo a mezi jiným interaktivním a tímto pomocným předmět by mohlo dojít ke kolizi.

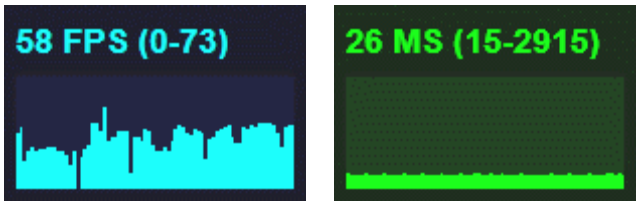
4 Implementace a výsledky

Hlavní implementační jazyk aplikace je JavaScript, ve kterém je napsána většina aplikace, herní logika, ovládání a zobrazení hry. Tímto jazykem jsou také napsány všechny externí knihovny použité ve hře. Dalšími implementačními nástroji je značkovací jazyk HTML a kaskádové styly CSS pro definici vzhledu ovládacích prvků a menu. Pro psaní shaderů je využit jazyk GLSL.

Aplikace je vyvíjena a testována pro internetové prohlížeče Firefox a Chrome. V ostatních prohlížečích i přes podporu WebGL není zaručena úplná funkčnost hry.

4.1 Použité knihovny a nástroje

Kromě frameworku WebGL, Three.js (kapitola 3.1.1) a fyzikální knihovny Ammo.js (kap. 3.4.4) dále při implementaci používám následující knihovny. Stats.js[1] pro zobrazení údajů počtu vykreslených snímků za vteřinu (fps) nebo po přepnutí do druhého režimu hodnotu udávající čas v milisekundách potřebný pro vykreslení jednoho snímku (Obrázek 4-1). Dále knihovnu Detector.js[2] pro kontrolu podpory WebGL webovým prohlížečem a grafickou kartou. Poslední převzatou knihovnou je webgl-utils[3]. Z té je použita pouze funkce `requestAnimFrame`, která slouží k zastavení vykreslování při skrytí aplikace. Důvodem je úspora výpočetního výkonu počítače v době nepoužívání hry.



Obrázek 4-1: Screenshot zobrazení počtu fps a ms.

4.2 Grafické rozhraní

Grafické rozhraní hry má za cíl vhodně informovat hráče o průběhu a vývoji hry prostřednictvím prvků ukazujících aktuální herní level, počet životů a dosažené skóre. Dále jsou to různé informativní hlášení, která jsou popsána níže a hlavní nabídka hry sloužící k lepší orientaci v možnostech použití a nastavení hry.

Hlavní nabídka a ovládací prvky hry jsou vytvořeny pomocí html a css. Vzhled je definován kaskádovými styly v souboru *style.css*. Okno s hrou a hlavní nabídkou jsou dva elementy typu *div* za sebou. Oba mají rozměry 600 krát 600 bodů. Rozměry byly zvoleny s ohledem na běžné rozlišení obrazovek (především notebooků) tak, aby se zobrazila celá aplikace na výšku bez nutnosti stránku rolovat. Navíc byla k tomuto rozměru zvolena určitá rezerva. Režim přes celou obrazovku nebyl zvolen z důvodu testů na demo příkladech, které dovolovaly přepínat zobrazení v okně a fullscreen. Při běhu přes celou obrazovku aplikace byly náročnější na výkon počítače. Proto z důvodu hratelnosti i na slabších počítačích je hra vytvořena v okenním režimu.

[1] http://github.com/mrdoob/stats.js
[2] autoři: http://alteredqualia.com/ a http://mrdoob.com/
[3] https://cvs.khronos.org/svn/repos/registry/trunk/public/webgl/sdk/demos/common/webgl-utils.js

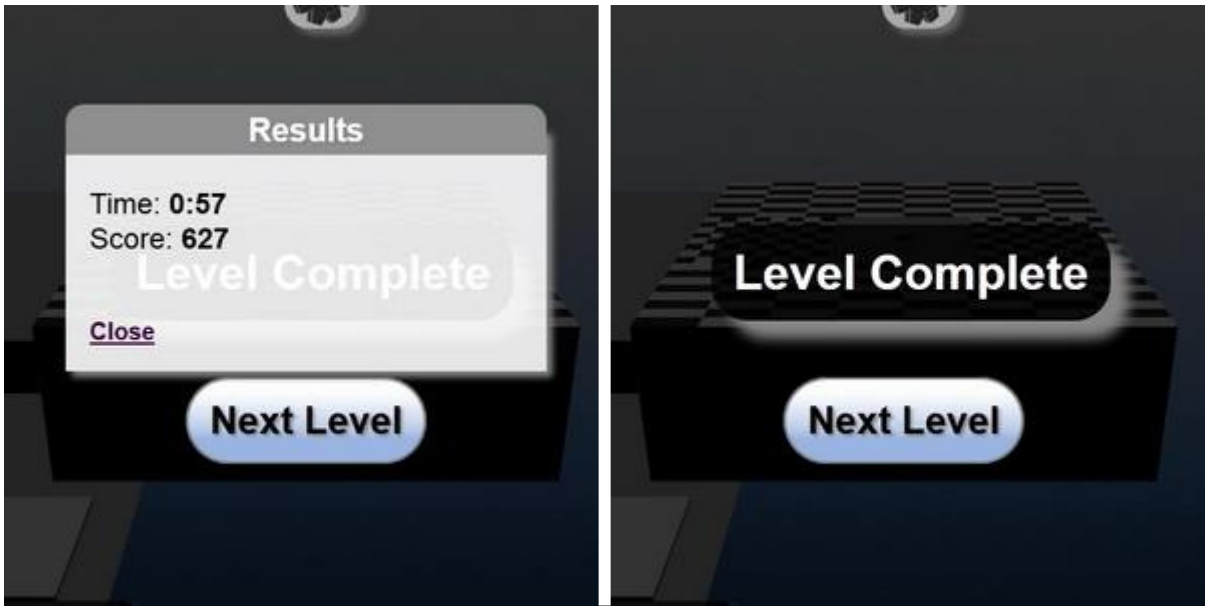
Menu ani hra nejsou nikdy zobrazeny současně. Jeden z těchto prvků je vždy skrytý. Na Obrázku 4-2 je zobrazeno menu hry v základním zobrazení a také se zapnutou podnabídkou. Všechny podnabídky jsou založeny na stejném stylu jako na obrázku. Jednotlivá tlačítka jsou popsána v kapitole 3.2.4. Neaktivní tlačítka mají aktivovanou průhlednost. *Continue* je neaktivní dokud není poprvé spuštěna libovolná hra. *High Scores* není ve hře implementováno.

Obrázek 4-2: Hlavní nabídka hry. Vpravo s otevřenou nabídkou pro výběr levelu.

V okně se hrou se trvale nachází 3 informační prvky sdělující aktuální herní level, počet životů a aktuální skóre. Dále jeden interaktivní prvek (v podobě ozubeného kolečka) sloužící pro přepínání do hlavní nabídky (do menu se lze přepnout i pomocí klávesové zkratky). Tyto prvky jsou částečně průhledné, aby nepůsobili příliš rušivě při hraní, ale po najetí na kterýkoli z nich se průhlednost deaktivuje. Tlačítko pro vstup do menu se dále změní z černého ozubeného kolečka na bílé. Tyto ovládací prvky jsou vidět na Obrázku 4-3.

Obrázek 4-3: Grafické prvky v okně hry. Nahoře zleva: aktuální level, počet životů a skóre hry. Dole tlačítko pro přechod do menu. Černé - výchozí zobrazení, bílé – po najetí kurzoru myši.

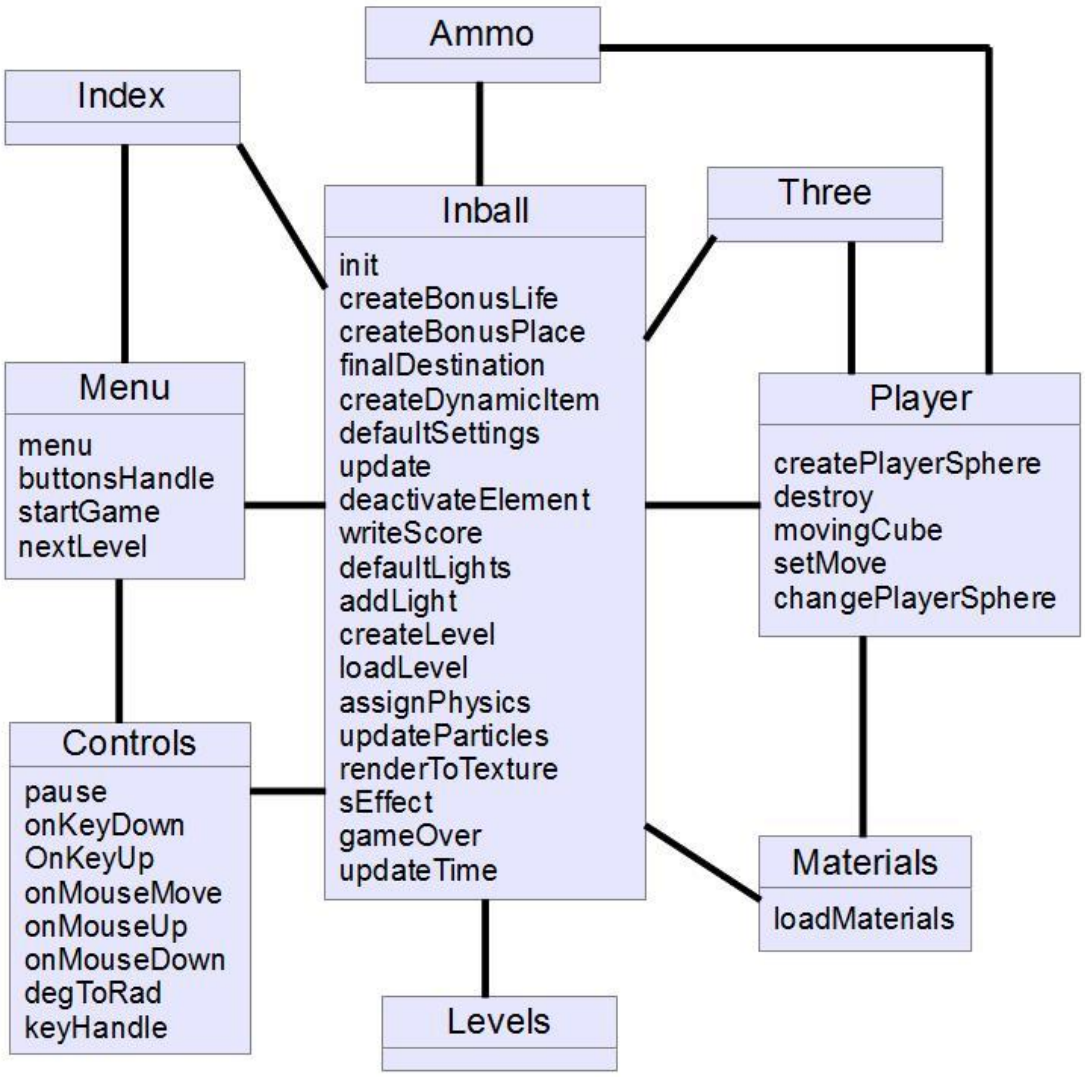
Kromě těchto prvků jsou v okně se hrou ještě další, které se zobrazí při ukončení hry. Při neúspěšném ukončení se zobrazí prvek s nápisem *Game Over*. Při dokončení levelu *Level Complete* a při dokončení celé hry *The End*. Po dohrání každé hry se zobrazí okno informující o skončené hře (dosažené skóre a čas). Pokud dohraný level není posledním ve hře, zpřístupní se tlačítko umožňující spustit další kolo. Zobrazením těchto prvků hra přechází do zamčeného stavu, kdy nelze již manipulovat s herní kuličkou. Přes okno hry se zobrazí černé poloprůhledné plátno a zapne se shader efekt falešného osvětlení (viz kapitola 3.3.1). Znázornění těchto prvků, které se zobrazují při ukončení hry je na Obrázku 4-4.



Obrázek 4-4: Screenshoty ukazující grafické prvky zobrazené při ukončení hry. Vlevo se zapnutou výsledkovou nabídkou.

4.3 Popis modulů

Část hry psaná v JavaSriptu, je pro větší čitelnost rozdělena do několika modulů, které obsahují tematicky podobné funkce. Předchozí kapitola věnující se grafické podobě hry patřila souborům *style.css* a *index.html*. Druhý jmenovaný obsahuje kromě prvků definující menu a ovládací prvky hry všechny fragment a vertex shadery použité v aplikaci. Tento soubor zároveň slouží ke spuštění hry. Provádí test kompatibility prohlížeče a grafické karty na WebGL. A zároveň inicializuje globální proměnné, jejichž prostřednictvím se řídí chod celé hry. Na Obrázku 4-5 jsou znázorněny vazby mezi jednotlivými moduly hry. Následující část popisuje jednotlivé moduly.



Obrázek 4-5: Zjednodušený diagram vazeb mezi moduly hry Inball.

Controls

Knihovna zabezpečuje ovládání hry, obsluhu událostí od klávesnice a myši. Dále zajišťuje činnost herní pauzy. Ta se dá spustit přímo ve hře danou klávesovou zkratkou nebo se spouští automaticky při vstupu do hlavní nabídky hry. Během pauzy se zablokuje ovládání kuličky a přestane se aktualizovat překreslování scény. Také je zastaveno počítání herního času. Dojde ke ztmavení okna hry a zobrazení informační hlášky o přerušení hry. Při navrácení z menu zpět do hry se pauza automaticky vypíná.

Inball

Tato knihovna tvoří jádro aplikace. Řídí činnost ostatních modulů. Vytváří a aktualizuje scénu, ovládá aktivaci a deaktivaci efektů. Dále načítá data uložená v modulu *Levels*, která zpracuje a vloží do scény. Jedna část těchto dat jsou speciální bonusová pole. Pro detekci kontaktu herní kuličky a těchto polí není použita fyzikální knihovna, ale pozice polí. Není pro ně tudíž ani vytvářen fyzikální model. Další část, která se načítá z dat levelu, jsou zdroje osvětlení. Framework Three.js implementuje následující typy světel: point, ambient, directional a spot light. U všech je možné

zapnout, aby vrhaly stíny, ale výsledkem je značné zvýšení zátěže počítače. Pouze u typu *Spot Light* jsou nároky malé, a proto v aplikaci vrhá stíny pouze tento typ světla. S použití jiného druhu osvětlení tak dochází k situaci, kdy kulička najednou přestane vrhat stín na okolní objekty i přes to, že na ni dopadá světlo.

Levels

V konečné podobě hra obsahuje 5 herních prostředí. Pro popis a uložení informací o jednotlivých levelech je použit formát JSON. Zpracování jednotlivých dat provádí hlavní modul *Inball*. Parametry objektů jsou proměnlivé v závislosti na jeho typu. Přehled těchto parametrů znázorňuje Tabulka 4-1.

Vlastnost	Hodnota	Popis
object	cube	objekt typu kostka/kvádr tvořící pevné nebo pohyblivé části levelu
	finish	cílové místo
	life	políčko s bonusovým životem
	toon1	speciální pole pro změnu efektu - komiksový vzhled v1
	toon2	speciální pole pro změnu efektu - komiksový vzhled v2
	invert	speciální pole pro změnu efektu - invertování barev
	blur	speciální pole pro změnu efektu - rozmazání obrazu
	greyscale	speciální pole pro změnu efektu - odstíny šedé
	spot	světelný zdroj typu Spot Light
type	-2	zdroj světla
	-1	cílové místo
	0	objekty, které mají stejnou texturu po celé ploše
	1	objekty, které mohou mít na každé straně jinou texturu
	2	interaktivní předměty
	3	bonusové políčko
	4	bonusový život
sizeX	xx	rozměry objektu
sizeY	xx	
sizeZ	xx	
material	xx	číslo udávající použitý materiál
posX	xx	pozice objektu ve scéně
posY	xx	
posZ	xx	
rotX	xx	natočení objektu ve scéně
rotY	xx	
rotZ	xx	
targetX	xx	souřadnice místa, kam má směřovat dopad paprsků světla
targetY	xx	
targetZ	xx	

Tabulka 4-1: Přehled a popis parametrů použitých pro uložení dat levelů. Hodnota xx značí libovolné celé číslo.

Materials

Tento modul obsahuje většinu materiálů použitých ve hře. Materiály definují barvu objektů, texturu, průhlednost, stínování, lesk apod. Jsou rozděleny do tří skupin. První typ materiálů slouží pro běžné zobrazení, kde použité efekty např. stíny řeší knihovna Three.js. Zbývající skupiny používají vlastní fragment a vertex shadery. Druhý typ materiálů je použit pro efekty změny materiálů (Kapitola 3.3.1). Poslední, třetí typ slouží pro efekty post processingu (Kapitola 3.3.2).

Menu

Tato část se stará o hlavní nabídku hry, její zobrazení a skrytí při přechodu zpět do hry. Dále obsluhu jednotlivých tlačítek (popsáno v Kapitole 3.2.4) a nastavení řídících proměnných při spouštění nové hry nebo přechodu do dalšího herního kola.

Player

Modul *Player* provádí operace týkající se herní kuličky. Její vytvoření, odstranění a změnu typu. Ve hře jsou 3 druhy kuliček a všechny se vytvoří hned při spuštění hry. Při změně se starý typ skryje a nový se objeví na pozici předchozího typu, ale o něco výše. Plynulá změna kuličky není použita jednak kvůli vizuálnímu podnětu, upozorňujícímu že došlo k nějaké změně na obrazovce a poté k zamezení stavů podvádění. Z důvodu, aby hráč nemohl používat celou dobu hry např. skleněnou kuličku, která je nejrychlejší, ale obtížně pohybuje s jinými objekty a tudíž těsně u těchto předmětů změnil typ na kovovou, odstranil z cesty objekty a znovu pokračoval skleněnou kuličkou. Při změně kuličky a objevení se nové v mírné výšce nad zemí se dále na určitou dobu znemožní provedení další změny kuličky. To kvůli případům spadnutí kuličky mimo level a následující ztrátě života, aby se nemohl hráč rychlou změnou kuliček vyšplhat do výšky, odkud by mohl pokračovat dále bez toho, že přijde o život.

Dále tento modul zajišťuje pohyb kuličky a to prostřednictvím jiného objektu, jak bylo popsáno v návrhu (Kapitola 3.5). V prvních fázích implementace bylo použito přímé pohybování kuličky prostřednictvím funkce `setMove()`. Tato funkce je využita i v konečné podobě hry. Slouží k zastavení kuličky při spuštění nové hry, ztrátě života nebo změny typu kuličky.

4.4 Testování a výsledky

Výsledná hra byla testována na 3 různých počítačích, pod operačním systémem Windows 7 (PC1, PC2) a Windows XP (PC3). V průběhu vývoje byla hra testována i pod Linuxem (Ubuntu - webový prohlížeč Firefox). Rozdíl oproti verzi pod Windows byl pouze v použití jiného druhu písma textu, jinak se obě verze nelišily. Výsledky testování jsou shrnuty v Tabulce 4-2.

Z nejpoužívanějších webových prohlížečů WebGL nepodporuje Internet Explorer, Safari pouze pod operačním systémem Mac OS X. Opera nezobrazuje korektně všechny vlastnosti. Zbývají tedy prohlížeče Firefox a Chrome, pro které byla hra vyvíjena a testována. Hodnoty v tabulce byly měřeny konkrétně ve verzi Firefox 12 a Chrome 18. Měření probíhalo pro 3 údaje: *avg* – průměrná hodnota snímků za vteřinu (fps), *max* – maximální dosažená hodnota fps během měření a *efekt* – průměrná hodnota fps při zapnutých efektech post processingu. Důvod snížení výkonu u zapnutých efektů je vykreslování scény do textury ve vyšším rozlišení, než je velikost původní scény. Nutnost většího rozlišení je popsána v kapitole 3.3.2 v části rybí pohled. Dalším faktorem ovlivňujícím rychlost hry je fyzika a množství objektů, pro které musí být počítána (viz kapitola 3.4.4).

Pro měření počtu fps byla použita knihovna Stats.js. Z výsledků dále vyplývá, že ze současných verzí testovaných prohlížečů je, co se týče optimalizace WebGL, na tom nejlépe Chrome,

který na všech počítačích vykazoval nejvyšší hodnoty. U PC3 se nepodařilo, kvůli problémům s ovladači grafické karty, hru spustit v prohlížeči Chrome.

	PC1			**PC2**			**PC3**		
Procesor	Intel Core 2 Duo - 2,4 GHz			Intel Core i3 - 2,4 GHz			Pentium Dual-Core - 2,0 GHz		
OP. Paměť	4 GB			4 GB			2 GB		
Grafická karta	Radeon HD 3650 mobile			Radeon HD 6370 mobile			Radeon HD 3410 mobile		
	avg	**max**	**efekt**	**avg**	**max**	**efekt**	**avg**	**max**	**efekt**
firefox	48	60	28	64	88	48	15	62	8
chrome	95	128	45	122	153	54	-	-	-

Tabulka 4-2: Přehled různých počítačů a jejich výkon ve hře Inball pro prohlížeče Firefox 12 a Chrome 18. Avg – průměrná hodnota snímků za vteřinu (fps), max – nejvyšší hodnota fps během testování a efekt – průměrná hodnota fps při zapnutém post processingu.

5 Závěr

Cílem této práce bylo navrhnout a vytvořit 3D počítačovou hru Inball použitím technologie WebGL. Jedná se o arkádovou hru založenou na hrách typu Ballance a Neverball. Hra pro spuštění nepotřebuje žádnou instalaci, pouze webový prohlížeč a grafickou kartu podporující tuto technologii. Aplikace byla vyvíjena pro prohlížeče Firefox a Chrome. Testována byla pro verze Firefoxu 9 až 12 a Chrome 16 – 18. Jedná se o multiplatformní aplikaci testovanou na operačních systémech Windows a Linux.

V průběhu vývoje hry jsem se seznámil s řadou frameworků WebGL, ze kterých jsem si posléze vybral k použití Three.js. Podstatnou část hry tvoří její fyzikální model. Zde jsem postupně vyzkoušel řadu postupů a metod od řešení detekce kolizí pomocí nástrojů obsažených v knihovně Three.js, vlastní řešení pohybu herní kuličky až po řadu knihoven implementující všechny potřebné prvky. Při vývoji jsem se také seznámil s psaním vlastních shaderů v jazyce GLSL. Ve hře je tak implementováno několik efektů např. komiksový vzhled a rozmazání obrazu.

Počítačových her ve WebGL zatím neexistuje příliš mnoho, vznikají především jednostranně zaměřené aplikace prezentující konkrétní vlastnost či funkcionalitu. Existující hry jsou většinou jednodušších typů, jak po stránce herní logiky, tak vzhledu na rozdíl od klasických počítačových her vytvořených pomocí DirectX nebo OpenGL. Důvodem jsou u WebGL větší nároky na výkon počítače především grafickou kartu než u DirectX a OpenGL. Zlepšení lze očekávat v budoucích verzích webových prohlížečů a jejich optimalizaci. Důkazem je Chrome a vydání verze 18, kde oproti starším verzím je znatelné zvýšení výkonu u WebGL aplikací.

Další vývoj hry může spočívat v přidání zvukového doprovodu. Pro snazší tvorbu levelů by bylo také vhodné vytvořit editor pro přímé vkládání a editaci jednotlivých objektů. Grafická stránka hry lze dále vylepšit přidáním propracovanějších modelů vytvořených v nástrojích typu 3ds Max nebo Blender. Další možností vývoje může být vývoj vlastní fyzikální knihovny zaměřené na vyšší výkon hry.

Literatura

[1] A. Munshi, D. Ginsburg, D. Shreiner. *OpenGL ES 2.0. Programming Guide*. Addison-Wesley Publishing Company. 2008. ISBN 978-0-321-50279-7.

[2] J. Neider, T. Davis, M. Woo. *OpenGL Programming Guide. The Official Guide to Learning OpenGL*. Addison-Wesley Publishing Company. Release 1. 1994. ISBN 0-201-63274-8.

[3] R. Škultéty. *JavaScript. Programujeme internetové aplikace*. 2. vydání, Computer Press, Brno 2004. ISBN 80-251-0144-4.

[4] M. C. Lin. *Efficient Collision Detection for Animation and Robotics*. University of California. Berkeley, 1993. Dostupné z www: <ftp://ftp.cs.unc.edu/pub/users/manocha/PAPERS/COLLISION/thesis.pdf>

[5] R. Szeliski. *Computer Vision. Algorithms and Applications*. Texts in Computer Science. Springer-Verlag, London 2011. ISBN 978-1-84882-934-3.

[6] P. Kršek. *Základy počítačové grafiky*. Studijní opora. VUT Brno.

[7] L. Caballero. *An Introducion to WebGL* [online] October 2011. Dostupné z www: <http://dev.opera.com/articles/view/an-introduction-to-webgl/>.

[8] T. Giles. *Learning WebGL. The Lessons* [online]. Dostupné z www: <http://learningwebgl.com/blog/?page_id=1217>.

[9] J. Groff. *An Intro to Modern OpenGL*. Chapter1: The Graphics Pipeline [online]. April 2010. Dostupné z www: <http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-1:-The-Graphics-Pipeline.html>.

[10] N. G. Belmonte. *WebGL Fundamentals* [online]. November 2011. Dostupné z www: <http://senchalabs.github.com/philogl/>.

[11] L. Caballero. *Porting 3D graphics to the web – WebGL intro part 2* [online]. November 2011. Dostupné z www: <http://dev.opera.com/articles/view/porting-3d-graphics-to-the-web-webgl-intro-part-2/>.

[12] S. Downes. *Fun and Games With DHTML. Stephen's web: Stephen Downes* [online]. August 1999. Dostupné z www: <http://www.downes.ca/post/276>.

[13] G. Jonathan. *The History of Flash*. Adobe Systems Inc. 2001 [online]. Dostupné z www: <http://www.adobe.com/macromedia/events/john_gay/page02.html>.

[14] L. Lacko. *Silverlight 4*. Brožury s vývojářskou tématikou [online]. Dostupné z www: <http://msdn.microsoft.com/cs-cz/dd727769>

[15] P. Minařík. *Detekce kolizí v DirectX (collision detection)*. Série článků: Programování pro DirectX [online]. 2004/03. Dostupné z www: <http://programovani.net-mag.cz/?action=art&num=459>.

[16] P. Lewis. *Shaders. Part 2* [online]. Dostupné z www: <http://aerotwist.com/tutorials/an-introduction-to-shaders-part-2/>

[17] R. Koonce. *GPU Gems 3. Chapter 19. Deferred Shading in Tabula Rasa*. [online] 2007. Dostupné z www: <http://http.developer.nvidia.com/GPUGems3/gpugems3_ch19.html>

[18] R. B. Whitaker. *RB Whitaker's Wiki. A Game Development Launchpad. Creating a Toon Shader* [online]. Dostupné z www: <http://rbwhitaker.wikidot.com/toon-shader>

[19] E. W. Weisstein. *Normal Distribution. MathWorld – A Wolfram Web Resource* [online]. Dostupné z www: <http://mathworld.wolfram.com/NormalDistribution.html>

[20] R. Larson. *Gaussian Blur Filter Shader. Game Rendering* [online]. October 2008. Dostupné z www: <http://www.gamerendering.com/2008/10/11/gaussian-blur-filter-shader/>

Seznam příloh

Příloha A: Manuál
Příloha B: Obsah přiloženého DVD
Příloha C: Plakát

Příloha A

Manuál hry Inball

Spuštění hry
Aplikace se spouští souborem index.html v hlavním adresáři hry. Pokud je použit nepodporovaný prohlížeč, je zakázána technologie WebGL nebo jsou problémy s grafickou kartou či jejími ovladači, zobrazí se varovné upozornění. V tomto případě menu hry bude fungovat, ale samotná hra se nespustí.

Herní režimy
Classic – cílem je projít co nejvíce herních kol a nasbírat co možná nejvyšší skóre. Hráč si může zvolit level, ve kterém hru začne. *Survival* – hráč si nemůže zvolit počáteční level, začíná od prvního a musí jej dokončit v přiděleném časovém limitu. Po dokončení kola se zbývající čas převádí na skóre a zároveň se přičítá k časovému limitu v dalším levelu. Cílem hry je projít všechny levely v co nejkratším čase. *Best Time* – cílem je dohrát zvolený level v nejkratším možném čase.

Speciální typy polí
Bonusový život – přidání +1 k herním životům. Pole sestává ze čtvercové základny a malé průhledné kuličky kmitající nad základnou. Bonusová pole – přidávají +500 ke skóre a zapínají daný grafický efekt podle barvy pole: *červená* – Toon Shader v1, *zelená* – Toon Shader v2, *modrá* – invertované barvy, *žlutá* – odstíny šedé a *azurově modrá* – rozmazání obrazu.

Ovládání hry
Hra se ovládá klávesnicí a myší. Myš slouží k rotaci kamery a to po stisknutí levého tlačítka myši a pohybu ve zvoleném směru. Otáčení kamery je omezeno na 180 stupňů. Po puštění tlačítka myši se kamera postupně vrátí do výchozí pozice za kuličku. Pohled kamery je dále možné přiblížit nebo oddálit klávesami *page up* a *page down*. Pro zapnutí a vypnutí pauzy slouží klávesa *p*. Pro přechod mezi hlavní nabídkou a hrou slouží klávesa *escape*. Herní kulička se ovládá kurzorovými šipkami. Pro změnu typu kuličky slouží klávesy *q*, *w* a *e*. Další klávesy slouží pro změnu efektů zobrazení. První varianta efektů jsou změny materiálů objektů tvořící nepohyblivé části levelu – klávesy *a*, *s* a *d*. Druhá sada efektů mění vzhled celé scény – klávesy s číslem 1 až 7.

Přehled nastavení ovládání:
Hráč:
- pohyb kuličky – kurzorové šipky
- změna typu kuličky:
    - q – dřevěná (výchozí typ)
    - w – kovová
    - e – skleněná

Kamera:
- přiblížení/oddálení kamery – page up, page down
- otáčení kamery: stisk levého tlačítka myši + pohyb

Obecné:
- pauza – p
- přepínání mezi hlavní nabídkou a hrou – escape

Efekty pro změnu materiálu:

- falešné světlo – s
- efekt 1 – a
- efekt 2 – d

Post Processing:

- Toon Shader v1 – 1
- Toon Shader v2 – 2
- černo/bílé zobrazení – 3
- invertování barev – 4
- odstíny šedé – 5
- rozmazání obrazu – 6
- rybí pohled - 7

Konec hry

Předčasné ukončení hry - dojde k němu po vyčerpání všech herních životů nebo při vypršení časového limitu v módu survival. Řádné ukončení hry – po dokončení cíle daného módu hry (viz Herní režimy).

Podpora WebGL u nejběžnějších prohlížečů a jejich nastavení

I přes podporu WebGL většinou prohlížečů je u některých v základním nastavení tato technologie zakázána. Zde si ukážeme přehled několika nejběžnějších prohlížečů a jejich nastavení. Dále v případě problémů je vhodné ujistit se, že je nainstalována nejnovější verze prohlížeče a zkontrolovat ovladače grafické karty. U operačního systému Linux může dojít k problémům i u podporovaných webových prohlížečů a to z důvodu špatných nebo neoficiálních verzí ovladačů grafické karty.

- **Firefox**: je vyžadována minimálně verze 4 nebo novější. Pro nastavení je nutné do adresního řádku zadat: `about:config`, do filtru zadat webgl a zkontrolovat nastavení hodnot.
- **Chrome**: minimálně verze 9 nebo vyšší. Je nutné zapnout WebGL v nastavení podobně jako u firefoxu, ale příkazem `about:flags`, zde najít odpovídající položku a aktivovat ji.
- **Safari**: verze 5.1 nebo novější. Pouze na operačním systému Mac OS X od verze 10.6. Ověření zapnutí: v nastavení, pod záložkou *Advanced* zaškrtnout položku *Show develop menu in bar*. Poté v nabídce *Develop* zaškrtnout *Enable WebGL*.
- **Opera**: od verze 12 alpha. Nastavení příkazem `opera:config`, záložka *user Prefs/* a zkontrolovat položky *enable Hardware Acceleration* a *enable webgl*
- **Internet Explorer**: WebGL není podporováno

Pozn.: hra Inball je vyvíjena a testována pro prohlížeče Firefox a Chrome. Pro ostatní prohlížeče není zaručena plná funkcionalita aplikace!

Příloha B

Obsah přiloženého DVD:

/doc – zdrojové soubory tohoto textu ve formátech docx , pdf a plakát prezentující vytvořenou hru Inball

/video – demonstrační videa ukazující běh aplikace

/screenshots – obrázky ze hry

/src – zdrojové kódy a všechny potřebné soubory nutné pro spuštění hry Inball:

- **index.html** – spouštěcí soubor
- **/css** – vzhled menu a ovládacích prvků pomocí kaskádových stylů
- **/js** – převzaté javascriptové knihovny nutné pro běh aplikace
- **/js2** – vlastní zdrojové kódy hry v javascriptu
- **/pic** – obrázky použité v aplikaci

Příloha C

Plakát prezentující vytvořenou hru Inball