VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Moderní fyzikální enginy pro WebGL

Modern Physics Engines for WebGL

2017

Milan Šebela

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Milan Šebela**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Moderní fyzikální enginy pro WebGL
Modern Physics Engines for WebGL

Jazyk vypracování: čeština

Zásady pro vypracování:

V dnešní době s rostoucím výkonem počítačů rostou také nároky na kvalitu a rychlost grafických aplikací, a to nejen aplikací desktopových, ale s příchodem WebGL i aplikací běžících v moderních prohlížečích. Cílem této práce je nastudovat, popsat a demonstrovat možnosti dnešních fyzikálních enginů použitelných v internetových prohlížečích se zaměřením na technologii WebGL. Součástí práce bude srovnání vybraných fyzikálních enginů pro WebGL (dynamika pevných těles, dynamika měkkých těles, dynamika kapalin apod.). Následně u vybraných fyzikálních enginů vytvořit srovnávací ukázkové příklady, otestovat je a porovnat jejich výhody a nevýhody.

1. Nastudujte možností dnešních fyzikálních enginů použitelných u WebGL (Bullet, Cannon, JigLib, apod.).
2. Vytvořte ukázkové aplikace, na nichž budou demonstrovány jejich vlastnosti a budou umožňovat jejich testování.
3. Srovnejte možnosti vybraných enginů a popište výhody a nevýhody. U vybraných enginů srovnejte jejich možnosti pro webové a desktopové aplikace.
4. Praktické výsledky pečlivě teoreticky popište pro další využití.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Němec, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Rád bych na tomto místě poděkoval panu Ing. Martinu Němcovi Ph.D. za jeho čas a odbornou pomoc.

Abstrakt

S rozvojem hardwaru přichází další využití internetových prohlížečů přenesením interaktivní 3D grafiky. Technologie WebGL ukázala zatím největší potenciál pro vývoj 3D webových aplikací. Cílem této bakalářské práce bylo srovnat aktuální možnosti javascriptových fyzikálních enginů pro WebGL. Konkrétně byly integrovány opensource fyzikální frameworky. V rámci práce byla vytvořena experimentální aplikace, pomocí které jsme vyhodnotili u jednotlivých frameworků jejich úspěšnost, rychlost, kvalitu a sadu funkcí.

Klíčová slova: bakalářská práce, WebGL, fyzikální engine, Bullet, real-time rendering

Abstract

With the development of hardware comes another use of internet browsers transferring interactive 3D graphics. Technology called WebGL shows a lot of potentials for developing 3D web applications. The aim if this bachelor thesis is to compare actual possibilities of javascript physics engines for WebGL. Specifically we integrated opensource physics frameworks. Within thesis the experimental aplication was created through which we evaluated the various frameworks for their success, performance, quality and feature set.

Key Words: bachelor thesis, WebGL, physics engine, Bullet, real-time rendering

Obsah

Seznam použitých zkratek a symbolů **8**

Seznam obrázků **9**

Seznam tabulek **10**

1 Úvod **12**

2 WebGL **13**
2.1 Existující WebGL Frameworky 13

3 Fyzikální engine **15**
3.1 Existující fyzikální frameworky 15

4 Architektura experimentální aplikace **17**
4.1 Design testů 18

5 Výsledky experimentu **20**
5.1 Detekce kolize 20
5.2 Dynamika pevných těles 21
5.3 Omezení 25
5.4 Vozidlový systém 29
5.5 Dynamika měkkých těles 30
5.6 Simulace kapalin 31
5.7 Výkonnost 32

6 Závěr **34**

Literatura **35**

Přílohy **38**

A Přílohy **39**

B Ukázkový kód **40**

Seznam použitých zkratek a symbolů

WebGL	–	Web Graphics Library
OpenGL	–	Open Graphics Library
API	–	Application programming interface
GPU	–	Graphic processing unit
W3C	–	The World Wide Web Consortium
HTML	–	HyperText Markup Language
CSS	–	Cascading Style Sheets
SPH	–	Smooth Particle Hydraulics

Seznam obrázků

1 Architektura experimentální aplikace 17
2 Test komplexní scény 18
3 Test tkaniny 19
4 Primitivní tvary 21
5 Výšková mapa 22
6 Mesh těleso 23
7 Konvexní plášť 23
8 Bodové omezení 25
9 Pantové omezení 25
10 Posuvné omezení 26
11 Vzdálenostní omezení 26
12 Kuželové omezení 27
13 Pevné omezení 27
14 Generických 6 stupňů svobody 28
15 Základní vozidlo 29
16 Graf prvního měření 32
17 Test skládání - 819 koulí poskladaných do pyramidy 32
18 Graf druhého měření 33

Seznam tabulek

1 Podporované typy široké fáze 20
2 Podporované tvary kolizních těles 24
3 Podporované typy omezení 28
4 Podpora vozidlového systému 29

Seznam výpisů zdrojového kódu

1 Vytvoření krychle ve vybraných fyzikálních enginech 40

1 Úvod

Fyzikální enginy dnes hrají důležitou roli v počítačové grafice. Jejich praktické využití můžeme nalézt například v mechanice, fyzikálních simulacích a filmovém či herním průmyslu. Dosavadním trendem byl vývoj stand-alone aplikací ve spustitelném souboru, který obsahuje všechna potřebná data. S rozvojem internetu vznikají nové možnosti, jak mít takové aplikace velmi snadno přístupné.

Jedny z populárních technologií používaných k zobrazování interaktivní 3D grafiky na prohlížeči jsou Flash, Microsoft Silverlight a JAVA Applet, avšak jejich hlavním nedostatkem je, že přehrávač potřebuje stáhnout plugin dřív, než můžete spustit obsah aplikace[1]. V únoru 2011 byla představena renderovací technologie nazývaná WebGL[2]. Tato technologie je podobná OpenGL[3], ale může běžet na internetovém prohlížeči. Výhodou této technologie je, že je dnes ve většině majoritních internetových prohlížečích podporována nativně, takže vypadá velmi slibně pro vývoj 3D webových aplikací.

Průmyslové konsorcium Khronos vydalo 16. února 2016 finální verzi nízkoúrovňového API Vulkan 1.0[4], které by pravděpodobně mělo do budoucna nahradit OpenGL vyvíjené stejnou skupinou. Nízkoúrovňové API umožní vývojářům užší přístup k hardwaru a jeho ovladačům. Díky tomu je možné v softwaru snížit jeho nároky na běh, fungování lépe optimalizovat a dosáhnout tím vyššího výkonu typicky ve hrách a jiných náročných 3D aplikacích. Jednou z hlavních změn, které Vulkan oproti OpenGL přinesl, je podpora více vláken při úkolování GPU, což zásadně sníží latenci při vykreslování. Vulkan bude frameworkem pro všechny platformy kromě iOS, které zůstalo u Metal API[5]. Vulkan začal pracovat, se sadami funkcí obsahující hardwarové funkce, které je možné při implementaci API použít. Lišit se budou například možnosti desktopového a mobilního Vulkanu. Momentálně není umožněno aby Vulkan běžel i na webových prohlížečích. Největší překážkou je udělat ho dostatečně bezpečný a konzistentní. Jedním z důvodů proč Vulkan dosahuje větší rychlosti je, že neprovádí žádné ověřování a nechává správu paměti na aplikaci. Je nepřijatelné, aby náhodným webovým stránkám bylo umožněno provádět takové operace na vašem počítači. Pro tento účel by tedy musel být navrhnut "bezpečný Vulkan", takže nejvhodnější technologií pro zobrazování interaktivní 3D grafiky v prohlížeči zůstává stále WebGL[6].

Cílem práce je srovnání aktuálních možností dostupných fyzikálních enginů pro WebGL, jejich úspěšnost, kvalitu, použitelnost a sadu funkcí. Pro rychlý vývoj 3D aplikací je nezbytné použít renderovací framework. V rámci práce byl vybrán renderovací framework, který je vhodný pro testování fyzikálních enginů. K provedení srovnání byla vytvořena experimentální aplikace spustitelná ve webovém prohlížeči.

2 WebGL

WebGL 1.0 je multiplatformní grafické API, jehož běhovým prostředím je webový prohlížeč využívající HTML5[7]. Stabilní verze této technologie byla vydána po únoru 2011, takže tato technologie není úplně nová, ale momentálně není W3C standardem[8]. WebGL je založeno na OpenGL ES 2.0, které umožňuje běh na přidaných platformách, takže si jsou API velmi podobná. Momentálně většina hlavních internetových prohlížečů, jako jsou Mozilla Firefox 4, Opera 12, Google Chrome 18, Safari 5.1, Internet Explorer 11, Microsoft Edge a jejich novější verze podporují WebGL nativně[9]. V současnosti se stále hledají způsoby jak zefektivnit renderování, například nový algoritmus kombinující Raycasting a Octree pro vylepšení výkonu[10].

2.1 Existující WebGL Frameworky

Cílem WebGL frameworku je převzetí typických problémů při vytváření WebGL obsahu, čímž se usnadní vývoj bez budování od základu, aby se vývojář mohl soustředit pouze na vytváření scény a své zadání. Níže je výčet nejvýznamnějších WebGL frameworků[11].

Babylon.js

Babylon.JS[12] je údajně jeden z nejlepších 3D JavaScriptových enginů pro tvorbu profesionálních her. Babylon.js kromě jeho grafických funkcí využívá i zakomponovaný fyzikální framework Oimo.js, který v našem experimentu testujeme.

Three.js

Three.js[13] je další komplexní a výkonná 3D JavaScriptová knihovna pro dělání čehokoliv 3D od vytváření jednoduchých 3D animací po vytváření interaktivních 3D her. Three.js knihovna přináší kromě podpory WebGL rendereru i SVG, Canvas a CSS3D renderery. Three.js je k dispozici jako open source pod licencí MIT.

Turbulenz

Turbulenz[14] je jedním z nejstarších herních enginů datovaných do roku 2009 kdy HTML5 a WebGL byly stále ve vývoji. Turbulenz nebyl přístupný jako open source do roku 2013. Turbulenz přichází zabalen s hromadou doplňků jako 2D fyzika, 3D fyzika, zvuk, video a další služby jako žebříčky, multichat, platby and uživatelská data.

Famo.us

Famo.us[15] je dalším velkým hráčem na trhu HTML5 3D vývoje a je jedním z nejlepších open source JavaScriptových 3D frameworků. Nejlepší věcí na Famo.us je že je zabalen s 3D renderovacím enginem, který je plně integrovaný s 3D enginem fyzikálně řízených animací.

PlayCanvas

PlayCanvas[16] je open source podnikový JavaScriptový herní engine, který má hromadu

vývojařských nástrojů, které pomohou během okamžiku vytvořit 3D scény. PlayCanvas je postaven profesionální komunitou a původně nebyl open source. Také přichází s cloudovým editorem, který běží v prohlížeči. Za PlayCanvas stojí společnosti jako je Mozilla, Activision a ARM.

Goo Engine

Goo Engine[17] je silný soubor JavaScriptových API pro vytváření čehokoliv 3D v HTML5 a WebGL. K dispozici je on-line editor Goocreate, který běží na tomto enginu a je zabalen s mnoha funkcemi, jako je vizuální 3D editor, materiální editor, skriptování a snadná možnost publikování. Ačkoli editor má nějaké související náklady, tak Goo Engine je zcela zdarma.

CooperLicht

Ambiera je společnost která stojí za tímto enginem a používala jej pouze pro vlastní produkci, avšak nakonec jej zpřístupnila jako open source. CopperLicht[18] je jedním z nejlepších 3D enginů pro vytváření her pro prohlížeče. K dispozici má editor CopperCube, který podporuje všechny funkce potřebné k vytvoření 3D her a animací. Avšak editor má nějaké licenční poplatky a není k dispozici jako open source.

Voxel.js

Voxel.Js[19] je open source JavaScriptový 3D herní engine. Společenství má od jeho vydání rychlý růst vůbec. Voxel je skvělá volba, pokud jste příznivccem modulárního přístupu. Voxel-engine je jádrem modul pro vytváření 3D her, ostatní moduly mohou být zapojeny jak je požadováno. Je k dispozici více než 200 addonů.

Blend4Web

Blend4Web[20] byl vydán jako open source 3D framework ruskou společností s názvem Triumph v roce 2014. Je úzce integrován s nástrojem Blender a má nativní podporu pro uzel materiálů, částicový systém, fyzikální engine Bullet a další funkce. Blend4Web je pod aktivním vývojem, v blízké budoucnosti by mohl konkurovat knihovnám jako Babylon.js a Three.js.

Enchant.js

Enchant.js[21] je modulární, objektově orientovaný JavaScriptový framework pro vytváření aplikací a her v HTML5. Je k dispozici jako open source pod licencí MIT a tudíž zdarma k použití. 3D animace mohou být vytvořeny pomocí přídavných pluginů. Enchant.js je aktivně udržována a rozvíjena společností Akihabara Research Center.

Pro naší práci byl vybrán framework Three.js, protože je dodáván bez fyziky, a také kvůli jeho modularitě, kompatibilitě a navržením jako odlehčené 3D knihovny s velmi nízkou úrovní komplexity.

3 Fyzikální engine

Fyzikální engine je softwarový program, který se používá k simulací fyzikálních jevů. Jedním z prvních použití fyzikálního enginu bylo pro dělostřelectví k výpočtu trajektorie na základě hmotnosti a síly palby. Od té doby by také použit na pomoc při konstrukci vozidel, plavidel i letadel. Fyzikální engine je obvykle jednou z komponent herního enginu. Fyzikální engine, který je používán v herním enginu je běžně adaptován z komerčního fyzikálního enginu vyvinut třetí stranou. Příklady populárních komerčních fyzikálních enginů jsou NVIDIA PhysX[22] a Havok[23]. Fyzikální enginy používané v herních enginech mají za cíl poskytnout výpočetně jednodušší přibližnou simulaci na rozdíl od vysoce přesných vědeckých fyzikálních enginech[24].

Alternativou může být vyvinout fyzikální engine založený na existujícím fyzikálním frameworku. Rozlišujeme fyzikální framework a fyzikální engine. Fyzikální framework je knihovna poskytující nízkoúrovňové funkce, zatím co fyzikální engine poskytuje vyšší úroveň rozhraní pro uživatele. Fyzikální framework musí obsahovat dvě základní funkcionality. Detekci kolize a odezva kolize. Momentálně jsou zde dvě populární fyzikální teorie používané v real-time fyzikálních enginech. Newtonovská fyzika a pevné tělesa. Newtonovská fyzika je založena na Newtonových pohybových zákonech, zatím co pevné tělesa předpokládají, že objekty jsou pevné a nemůžou se deformovat. Fyzika pevných těles se stala populární, protože skvěle zjednodušovala potřebný výpočet a poskytovala akceptovatelné výsledky. Některé z pokročilích prvků fyziky jsou ragdoll physics, dynamika měkkých těles, nebo simulace kapalin[25].

3.1 Existující fyzikální frameworky

Teoreticky všechny frameworky napsané v javascriptu můžou být použity jako fyzikální engine. Pro tento výzkum byly vybrány frameworky, které jsou opensource, takže výsledky můžou být dále analyzovány zjišťováním struktury jejich kódu. Kritéria pro vybrání frameworků byla popularita napříč herními vývojaři a jejich schopnosti simulovat 3D fyziku. Vybrané frameworky jsou následující[26].

Ammo

Ammo.js[27] je JavaScriptový port fyzikální knihovny Bullet[28] napsané v C++ použitím programu Emuscripten[29]. Kód byl přeložen bez dalšího lidského zásahu, takže zdrojový kód by měl být funkcionálně identický s originálním Bullet fyzikálním enginem. Protože Emuscripten převádí knihovnu do JavaScriptu, jsou zde věci, které jsou náročnější na použití například ukazatelé. Naštěstí zde není moc míst, které by byli tímto ovlivněny, jinak je to velmi bohatá knihovna na doplňky. Bullet je jednou z nejkompletnějších fyzikálních knihoven dostupných v jakémkoliv programovacím jazyce. Bullet je použit v mnoha úspěšných komerčních produktech jako například hra Grand Theft Auto IV, nebo film Sherlock Holmes a je dostupný jako fyzikální engine v mnoha nástrojích pro tvorbu 3D grafiky jako například Blender nebo Cinema 4D. Licence knihovny je zlib, stejně jako Bullet.

Cannon

Cannon.js[30] je odlehčená JavaScriptová 3D fyzikální knihovna s poměrně širokou nabídkou funkcionality. Hlavní výhodou této knihovny je, že je psaná od základu v JavaScriptu, což snižuje její velikost na méně než 100 kB a je optimalizovaná pro chod na webovém prohlížeči. Licence knihovny je zlib.

JigLibJs

JiGLibJs[31] je port fyzikální knihovny JigLibFlash pro ActionScript 3.0, což je rovněž port další populární C++ fyzikální knihovny JigLib. Narozdíl od Ammo.js to není automatický port, ale je ručně předělaný do JavaScriptu s řadou optimalizací. Toto přizpůsobení dalo JibLibJS navíc výkon narozdíl od Ammo.js, ačkoliv není tak bohatá na doplňky. Licence knihovny je MIT.

Oimo

Oimo.js[32] je odlehčený 3D fyzikální engine pro JavaScript. Je to kompletní JavaScriptová konverze OimoPhysics pro ActionScript 3.0. Knihovna OimoPhysics je stále vyvíjena v ActionScript 3.0, přičemž je dále automaticky konvertována nejen do JavaScriptu, ale i do Javy. Licence knihovny je zlib.

Goblib Physics

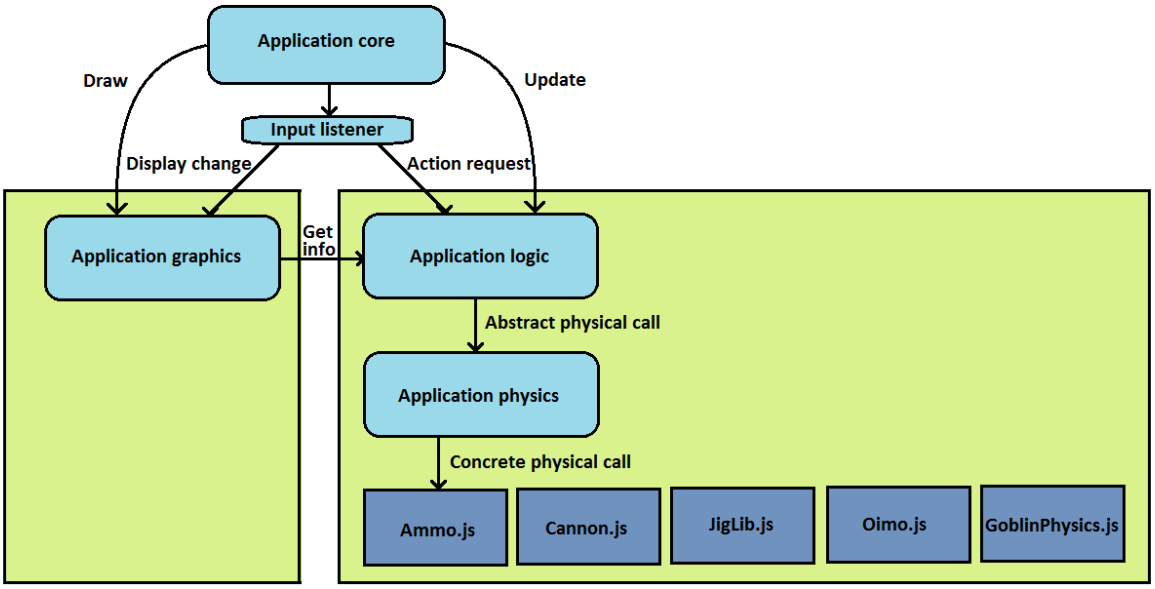
Goblin Physics[33] je další opensource fyzikální engine napsaný od základu v JavaScriptu. Cílem tohoto enginu je poskytnout rychlé a spolehlivé fyzikální simulace bez ohledu na platformu. Knihovna poskytuje pouze základní funkcionalitu, ale jinak dosahuje poměrně dobrých výsledků. Licence knihovny je zlib.

Abychom si mohli udělat nějakou představu o designu API jednotlivých frameworků, můžeme si prohlédnout kód**??** nacházející se v příloze.

4 Architektura experimentální aplikace

Pro to aby experimenty bylo možné provést, byla vytvořena experimentální aplikace. Tato experimentální aplikace se skládá ze tří hlavních komponent. Jádro, renderovací engine a fyzikální engine. Jádro je odpovědné za správu paměti a působí jako hlavní kontrolor jakékoliv jiné komponenty. Renderovací engine je zodpovědný za zobrazující pohled na uživatele. V tomto případě, renderovací engine Three.js využívající technologii WebGL.

Hlavní důraz je kladen na fyzikální engine. Fyzikální engine je vytvořen pomocí fyzikálních frameworků, které budou testovány v této práci. S cílem usnadnit porovnávání, bude fyzikální engine poskytovat obecné rozhraní fyzikálních frameworků. Tak uživatel může jednoduše použít funkci rozhraní a zvolit framework, který bude použit namísto přímého volání funkcí, které jsou poskytované frameworky. Výhodou tohoto přístupu je, že uživatel nebude muset změnit kód, pokud bude chtít změnit fyzikální framework.



Obrázek 1: Architektura experimentální aplikace

Pro zpřehlednění testování, bylo přidáno uživatelské rozhraní, ve kterém můžete spouštět a přizpůsobovat testy, či vybrat si, na kterém fyzikálním frameworku daný test poběží. Uživatelské rozhraní je upraveno jednoduchým opensource CSS frameworkem W3.CSS[34].

V dnešním světě vícejádrových procesorů je velmi výhodné počítat fyzikální simulaci v jiném vlákně, než ve kterém provádíme funkce pro vykreslení. To je umožněno díky WebWorkers[36], které od příchodu HTML5 umožňují zpracovávání informací na novém vlákně se kterým komunikuje za pomocí asynchronních událostí. Před příchodem HTML5 byl běh JavaSriptu omezen pouze na jedno vlákno. V hlavním vlákně tak může běžet synchronizace a vykreslovací smyčka a ve vedlejším vlákně fyzikální simulace.

Experimentální aplikace je dostupná na webové stránce[35] a v příloze na CD.

4.1 Design testů

Každý z testů pokrývá jednu problematiku a dohromady testy pokrývají většinu funkcionality fyzikálních frameworků. Pokud vybraný fyzikální framework testovanou funkcionalitu nepodporuje, nebude možné daný test spustit. Tím docílíme srovnání úplnosti fyzikálních frameworků. Poskytnuto je i uživatelské prostředí, kde si jednotlivé testy bude uživatel moct libovolně přizpůsobit včetně rychlosti simulace, gravitace, či vzhledu materiálů. Je umožněno simulaci pozorovat přímo grafickou kamerou, nebo můžete být s fyzikálním světem v interakci a ovládat kouli, rozpohybovanou postavu, či auto. Aplikace bude obsahovat následující testy.

Testy pevných těles

Testy pevných těles budou zahrnovat test primitivních tvarů, kde bude vytvořeno několik instancí všech podporovaných primitivních objektů s kterými půjde manipulovat. Test výškové mapy matematicky vygeneruje výškovou mapu za pomocí algoritmu SimplexNoise a vytvoří z ní terén na kterém můžeme otestovat přesnost a výpočetní náročnost. Test komplexní scény bude využívat terén a načtené komplexní 3D modely, které budou pokrývat fyzikální tělesa složené z více primitivních tvarů.

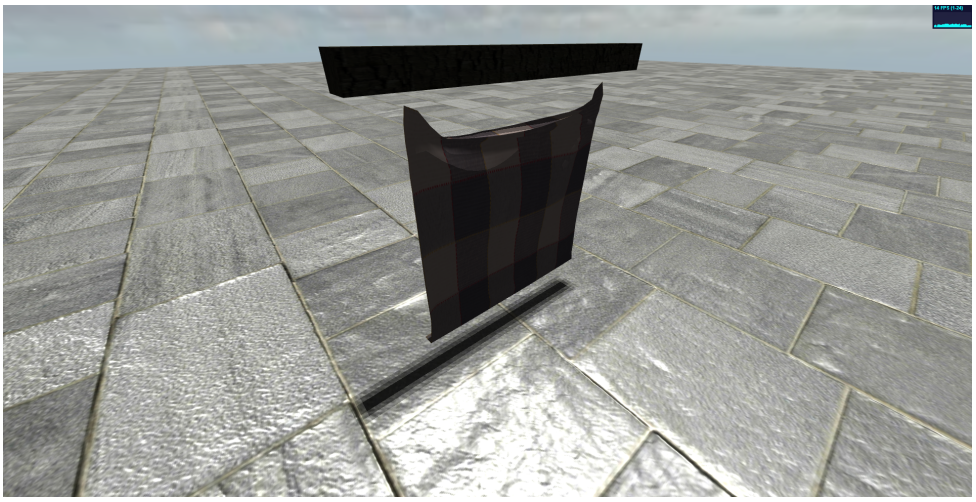
Obrázek 2: Test komplexní scény

Testy omezení

Testy spojů budou obsahovat test bodové vazby, pro který byla vybrána simulace mostu, jelikož každé prkno mostu je navzájem spojeno dvěma bodovými spoji se sousedním prknem. Test vzdálenostního spoje vyzkoušíme napodobením Newtonovy kolébky, což jsou kuličková kyvadla zavěšená na lanku, kde vzdálenost spoje reprezentuje délku lanka. Test pantové vazby vyzkoušíme vytvořením místnosti s různými typy dveří které půjdou otevřít.

Testy měkkých těles

Testy měkkých těles budou obsahovat test lana, který bude simulovat lano, na kterém bude zavěšena koule, kterou můžeme kurzorem rozhoupat. Test tkaniny, který bude simulovat zavěšenou látku. Test volumetrického měkkého tělesa bude simulovat obří polštář po kterém budete moci skákat a být s ním různými způsoby v interakci.



Obrázek 3: Test tkaniny

Test vozidlového systému

V tomto testu bude sestaveno základní auto sestavené z těla a čtyř kol reprezentující cylindr, které jsou spojené s tělem pantovou vazbou. Navíc bude sestavena základní dráha pro ježdění.

Testy kapalin

Testy kapalin budou obsahovat pouze test hladké hydrauliky částic, protože jiný algoritmus JavaScriptové fyzikální frameworky nepodporují. V tomto testu bude vytvořen bazén do kterého budou vhozeny částice pohybující se podle algoritmu hladké hydrauliky částic, která má simulovat pohyb vody.

Testy skládání

V těchto testech nám je umožněno skládat kvádry nebo koule na sebe různými způsoby a ověřit s jakou přesností fyzikální framework rozmisťuje objekty. Je umožněno z kvádrů poskládat zásobník, zeď, nebo pyramidu. Koule budou moct být poskládány pouze do pyramidy.

Test výkonnosti

Test výkonnosti spočívá ve spouštění skládacích testů s různým počtem těles. Výsledky jsou zaznamenány a následně porovnány v grafu.

5 Výsledky experimentu

Jednotlivé části fyzikálních frameworků jsou zde popsány a následně srovnány s výsledky testů i s ukázky zdrojového kódu.

5.1 Detekce kolize

Pojem detekce kolize označuje algoritmy pro zjištění kolize dvou objektů, neboli zjištění, zda existuje průsečík dvou daných těles. K řešení problémů detekce kolize vyžaduje rozsáhlé užití pojmů z lineární algebry a výpočetní geometrie. Pro urychlení výpočtu se obvykle používá dělení prostoru, což je proces, při němž je prostor rozdělen na více podprostorů, přičemž každý ze vzniklých podprostorů je obvykle dále rozdělen pro vytvoření hierarchie. Vyhledání geometrických entit v takto uspořádaném prostoru je výrazně rychlejší.

Detekce kolizí může být diskrétní (objekty jsou poskytovány v čase prostřednictvím předem stanoveném intervalu), nebo kontinuální (objekty jsou poskytovány pouze, pokud dojde ke kolizi). První z nich je rychlejší a snazší implementovat, ale vede k odhalování falešných kolizí, pohybují-li se objekty dostatečně rychle. Vybrané fyzikální frameworky a real-time systémy obecně musí použít diskrétní detekci kolizí s jinými způsoby, aby se zabránilo falešným detekcím kolizí. Detekce kolize je běžně vykonávána ve dvou fázích. Široká fáze a úzká fáze[37].

Široká fáze je typicky výpočetně méně náročná operace, která má za úkol zjistit, které objekty mají velkou pravděpodobnost, že budou v kolizi. Metody řešení obsahují například Sweep and Prune[38] a dělení prostoru například pomocí dynamického AABB stromu[39]. Široká fáze může být řešena i algoritmem brutální síly, kde každé těleso porovnáváme s každým tělesem, takže tento algoritmus má kvadratickou náročnost a vyplatí se pouze pro malé scény.

Úzká fáze má za úkol zjistit, která část objektu A koliduje s objektem B. Je typicky výpočetně intenzivní a nemůže být provedena na každý pár objektu v čase, než bude vykreslen další snímek. Příklad technik úzkých fází je Hyperplane Separation Theorem[40], Pixel Perfect Collision Detection[41]. V následující tabulce1 můžeme porovnat jaké typy širokých fází podporují vybrané fyzikální frameworky.

Tabulka 1: Podporované typy široké fáze

Typ široké fáze	Ammo	Cannon	JigLib	Oimo	GoblinPhysics
BruteForce	X	✓	✓	✓	✓
Sweep and prune	✓	✓	X	✓	✓
Dynamic AABBTree	✓	✓	X	✓	X

Detekce kolize pro deformovatelné objekty může být důležitá pro dosáhnutí vizuálně realistického výsledku. To je náročné dosáhnout a to zejména v případě odhalení a vyřešení vlastních kolizí a vzájemných kolizí mezi dvěma a více deformovatelnými objekty, protože neexistuje žádný jednoznačný způsob, jak lokálně detekovat, zda uzel tkaniny, který pronikl, je na špatné straně,

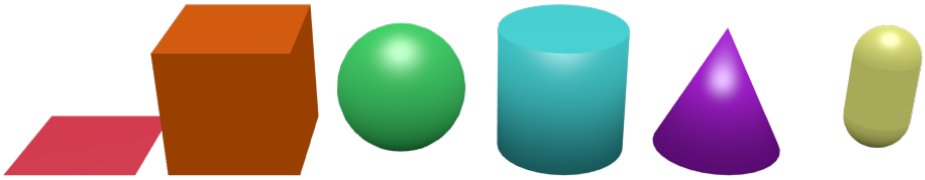
nebo ne. Řešení zahrnují historii pohybu tkaniny pro určení, zda došlo k události nebo vytvoření globální analýzy stavu tkaniny. Pixar představila způsob, který používá globální topologickou analýzu[42]. Obecně je ale příliš výpočetně náročný pro real-time látkový systém.

5.2 Dynamika pevných těles

Dynamika pevných těles je založena na tom, že pevné tělesa předpokládají, že objekty jsou pevné a nemůžou se deformovat. To znamená, že body tělesa budou od sebe stále vzdálené konstantní vzdáleností. Většina 3D objektů jsou reprezentovány dvěma separovanými objekty nebo tvary. Jeden z těchto objektů je komplexní a detailní tvar viditelný v grafické aplikaci. Pro účel zrychlení fyzikální engine používá k reprezentaci zjednodušený neviditelný objekt. Zjednodušený objekt používaný pro výpočet fyziky bývá nazýván kolizní geometrie[43]. Ta může být v několika tvarech.

Primitivní tvary

Primitivní tvary jsou obecně nejjednodušší geometrické objekty, které může systém zpracovat. Za primitivní tvary lze považovat například plocha, krychle, koule, cylindr, kužel, kapsle a částice.



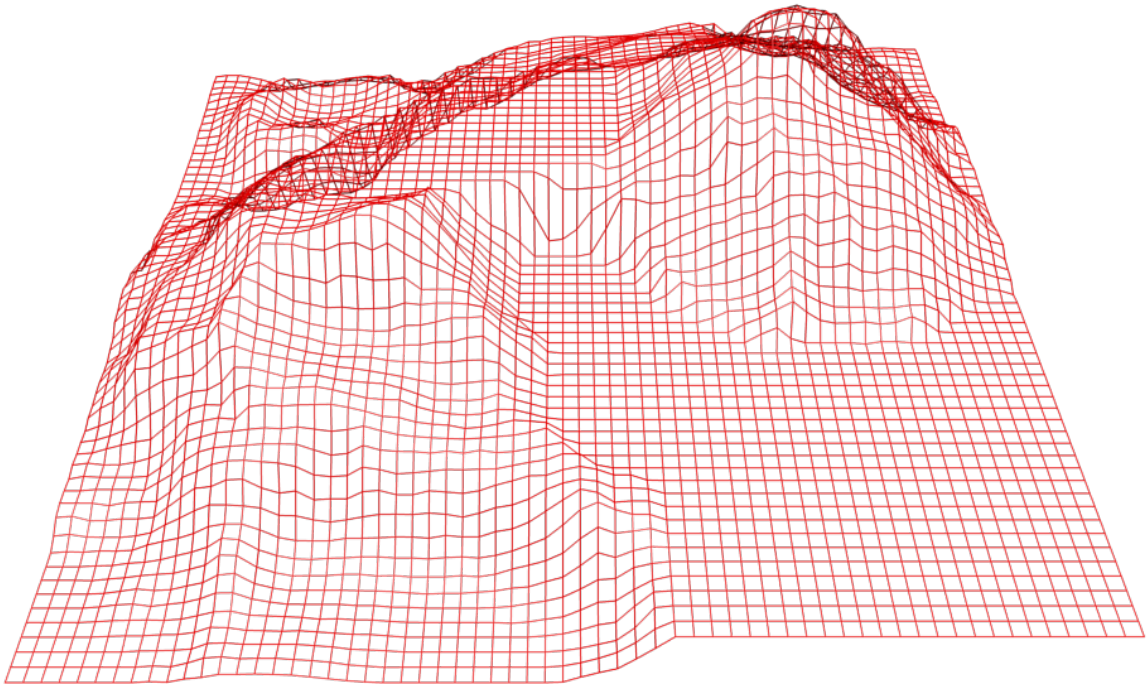
Obrázek 4: Primitivní tvary

Složené těleso(compound)

Sloučenina umožňuje spojit více tvarů a brát je jako jeden objekt s jednou váhou a jedním těžištěm. Sloučené objekty poté mezi sebou udržují stejnou vzdálenost a rotaci.

Výšková mapa

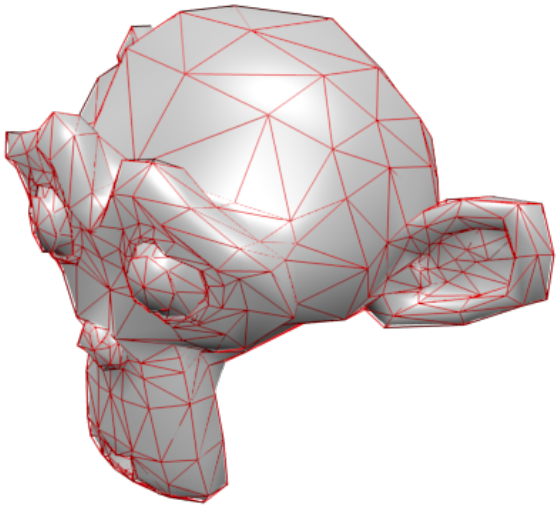
Výšková mapa se využívá pro simulaci terénu. Ammo.js, Cannon.js i JigLib.js podporují výškovou mapu. Goblin Physics.js nepodporuje přímo výškovou mapu, ale umožňuje dosáhnout podobného výsledku použitím funkce pro tvar tělesa. Takovéto řešení je sice přesné, ale není optimalizované pro paměť, jako tomu je u předešlých frameworků. Oimo.js taktéž přímo nepodporuje terén, avšak v oficiálních ukázkách dosahují podobného výsledku přidáním koule do každého vrcholu terénu. Koule se používá proto, že je to výpočetně nejjednodušší tvar. Tento způsob simulace terénu se však ukazuje jako poměrně nepřesný.



Obrázek 5: Výšková mapa

Mesh těleso

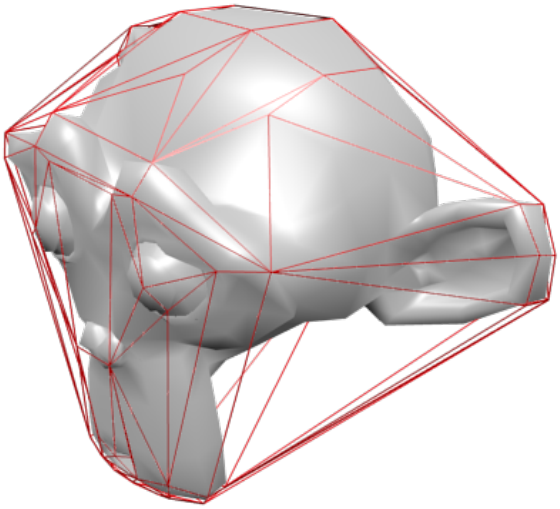
Mesh těleso je typ polygonálního tělesa skládajícího se ze sady trojúhelníků, které jsou spojeny jejich společnými hranami nebo rohy. Avšak jsou málo využívané kvůli jejich výpočetní náročnosti.



Obrázek 6: Mesh těleso

Konvexní plášť

Konvexní plášť je nejmenší konvexní množina vrcholů v prostoru, která obsahuje všechny vrcholy tělesa.



Obrázek 7: Konvexní plášť

V tabulce2 můžeme vidět srovnání podpory kolizních geometrií napříč vybranými fyzikálními frameworky.

Tabulka 2: Podporované tvary kolizních těles

Tvary	Ammo	Cannon	JigLib	Oimo	GoblinPhysics
Plocha	✓	✓	✓	X	✓
Krychle	✓	✓	✓	✓	✓
Koule	✓	✓	✓	✓	✓
Cylindr	✓	✓	X	✓	✓
Kužel	✓	✓	X	X	✓
Kapsle	✓	X	✓	X	X
Částice	✓	✓	X	X	X
Sloučenina	✓	✓	X	✓	✓
Výšková mapa	✓	✓	✓	X	X
Trojúhelníkové těleso	✓	✓	✓	X	✓
Konvexní plášť	✓	✓	X	X	X

Pěvné těleso definují kromě tvaru i vlastnosti jako jsou například váha, tření, lineární a úhlové tlumení, restituce, nebo typ tělesa. Fyzikální enginy rozlišují následující typy těles.

Statické těleso

Statické těleso se v průběhu fyzické simulace nepohybuje a slouží většinou jako blokující objekt, například podlaha.

Dynamické tělesoí

Dynamické těleso je plně simulovaný fyzikální objekt, jehož stav je v průběhu fyzikální simulace ovlivňován. Stavem se rozumí jeho pozice, rotace či jiné vlastnosti.

Kinematické těleso

Kinematické těleso může být animováno v průběhu fyzikální simulace, avšak není simulací nijak ovlivňováno. Dynamické objekty jsou při kolizi s kinematickými objekty odráženy.

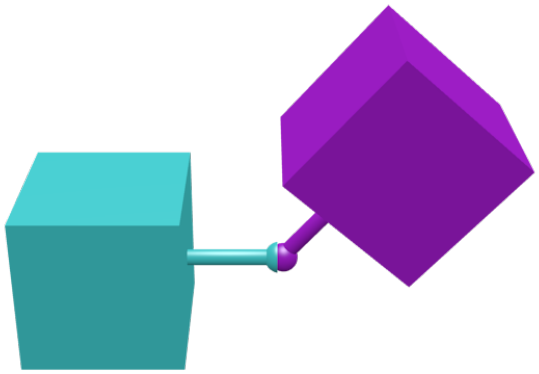
Výše zmíněné základní vlastnosti podporují všechny vybrané fyzikální frameworky, až na fyzikální framework Oimo.js, který nepodporuje lineární a úhlové tlumení a podobných výsledků musí dosahovat nastavením tření. Navíc u většiny z vybraných fyzikálních frameworků se objevují problémy s kinematickými tělesy, kvůli nedodělanosti, nebo nejsou vůbec implementovány. Například frameworku Cannon.js chybí naimplementované řešení kolize mezi statickým a kinematickým tělesem, což se ve výsledku jeví tím, že tělesa mezi sebou proletí. Korektně s kinematickými objekty zachází pouze framework Ammo.js.

5.3 Omezení

Omezení (constraint) ve fyzikálních enginech slouží k omezení lineární nebo úhlové rychlosti mezi objektem a světem nebo dvěma objekty[44].

Bodové omezení (Point constraint)

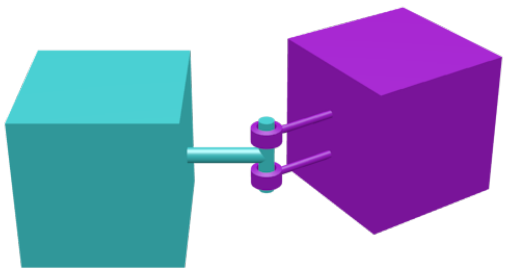
Bodové omezení omezuje objektu lineární pohyb na bod, ale umožňuje úhlový pohyb. Objekty jsou spojeny bodem, který volně umožňuje rotaci.



Obrázek 8: Bodové omezení

Pantové omezení (Hinge constraint)

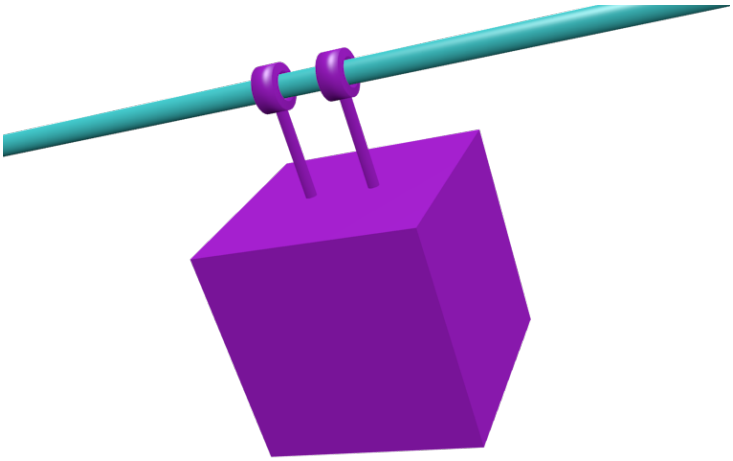
Pantové omezení připevní objekt na světovou pozici, nebo dva objekty mezi sebou jako pant, který umožní pouze úhlový pohyb na jedné ose.



Obrázek 9: Pantové omezení

Posuvné omezení (Slider constraint)

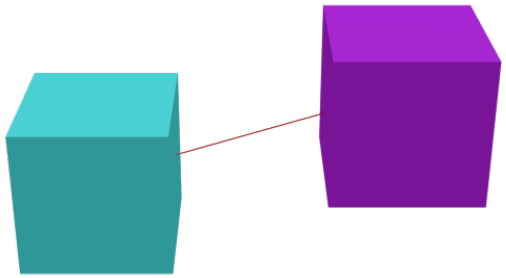
Posuvné omezení odstraní všechny úhlové pohyby a omezí lineární pohyb pouze na jednu osu.



Obrázek 10: Posuvné omezení

Vzdálenostní omezení (Distance constraint)

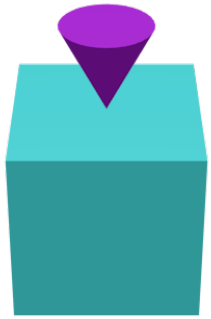
Je stejné jako bodové omezení s tím rozdílem, že body se od sebe mohou vzdálit na určitou maximální vzdálenost na rozdíl od bodového omezení, kde jsou od sebe body vzdáleny konstantní vzdáleností.



Obrázek 11: Vzdálenostní omezení

Kuželové omezení (Cone twist constraint)

Speciální bodové omezení které přidává kužel a limit rotační ose. Osa x většinou slouží jako rotační osa. Je to užitečné při vytváření ragdolls, speciálně pro končetiny jako horní část paže. Ragdolls jsou většinou použity pro animace mrtvých těl, které mají uvolněné končetiny



Obrázek 12: Kuželové omezení

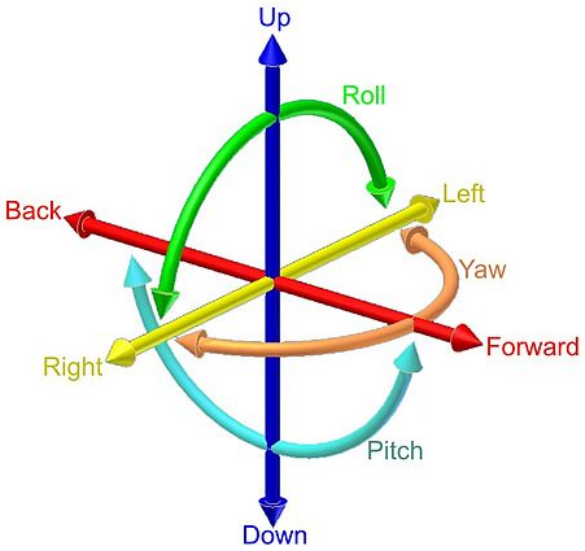
Pevné omezení (Fixed Constraint)

Omezení má všechny stupně svobody zamknuty. Objekty tak mají vůči sobě pevnou rotaci i polohu.

Obrázek 13: Pevné omezení

Generických 6 stupňů svobody (Generic 6 degrees of freedom)

Tohle generické omezení může emulovat různé standardní omezeni nastavením všech 6ti stupňů svobody. První tři stupně jsou lineární, a další tři reprezentují úhlový pohyb. Každá osa může být zamknuta, svobodná nebo limitovaná.



Obrázek 14: Generických 6 stupňů svobody

V tabulce3 můžeme vidět, které typy omezení podporují vybrané fyzikální frameworky.

Tabulka 3: Podporované typy omezení

Druh omezení	Ammo	Cannon	JigLib	Oimo	GoblinPhysics
Bodový	✓	✓	✓	X	✓
Pantový	✓	✓	X	✓	✓
Vzdálenostní	✓	✓	✓	✓	✓
Posuvný	✓	✓	X	✓	✓
Kuželový	✓	✓	X	X	✓
Pevný	✓	X	✓	X	X
6DOF	✓	X	X	X	X

5.4 Vozidlový systém

Ve vybraných fyzikálních frameworcích se můžeme setkat se dvěma typy vozidlového systému. Vozidlový systém využívající pevné tělesa a vozidlový systém využívající raycasting.

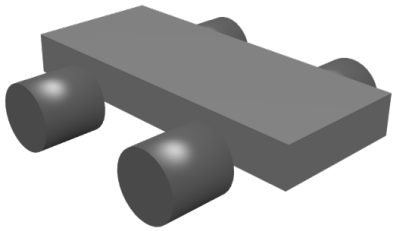
Základní vozidlo využívající pevné tělesa je ve fyzikálních enginech sestaveno z těla podvozku a čtyř kol reprezentující cylindr, či jiný tvar podobný kolu, které jsou spojené s tělem podvozku pantovou vazbou. Některé fyzikální frameworky poskytují naimplementovaný vozidlový systém zajišťující vytváření auta s pantovými vazbami a podpůrnými funkcemi pro roztáčení pantové vazby, nicméně pokud engine poskytuje pantovou vazbu jsme schopni si tento vozidlový systém naimplementovat sami. Tento způsob v principu funguje, ale má několik nevýhod. Potřebovaly bychom mnohem komplexnější omezení k simulaci suspenze. Posuvné chování kola se liší od standardních pevných těles, protože realné kola nejsou tuhé tělesa.

Vozidlový systém využívající raycasting řeší detekci kolize pro kola vysíláním paprsku pro každé kolo a anizotropním modelem tření. Použitím průsečíku bodu můžeme vypočítat délku suspenze a poté je síla suspenze aplikovaná na tělo podvozku, což udržuje podvozek nad zemí. Ve skutečnosti se tedy vozidlo vznáší nad paprsky[45].

V následující tabulce4 můžeme vidět které z vybraných fyzikální frameworků podporují vozidlový systém.

Tabulka 4: Podpora vozidlového systému

Druh vozidlového systému	Ammo	Cannon	JigLib	Oimo	GoblinPhysics
Vozidlový systém využívající pevné tělesa	X	✓	✓	X	X
Vozidlový systém využívající raycasting	✓	✓	X	X	X



Obrázek 15: Základní vozidlo

5.5 Dynamika měkkých těles

Dynamika měkkých těles je pole počítačové grafiky, které se zaměřuje na vizuálně realistickou fyzikální simulaci pohybu a vlastností poddajných objektů a měkkých těles. Na rozdíl od simulaci pevných těles se může tvar měkkých těles měnit, což znamená, že vzdálenost dvou bodů na objektu není stanovena. Zatímco relativní vzdálenosti bodů nejsou pevné, očekává se, že udržuje tvar do určité míry na rozdíl od tekutiny. Rozsah dynamiky měkkých těles je poměrně široký a zahrnuje simulaci měkkých organických materiálů, jako jsou svaly, tuky, vlasy a vegetace, nebo další deformovatelné materiály, jako jsou oblečení a tkaniny. Obecně platí, že tyto metody poskytují pouze vizuálně věrohodné emulace, než přesné technické simulace. Pro simulaci měkkých těles existuje několik různých modelů[46]. Mezi nejznámější patří model Pružina-Hmota. V tomto přístupu, je těleso modelováno jako soubor uzlů propojených pružinami v ideálním stavu beztíže řídící se jedním z Hookova zákonů. Uzly mohou být derivovány ze hran jednodimenzionálního, dvoudimenzionálního i třídimenzionálního polygonálního objektu. K dosažení požadovaných výsledků je možné sílu pružin modifikovat nebo přidat přídavné pružiny. Aplikováním druhého Newtonova zákona do hmotných bodů, včetně sil, působících pružinami a jakýchkoli vnějších sil v důsledku kontaktu, gravitace, odporu vzduchu, větru, a tak dále nám poskytne soustavu diferenciálních rovnic pro pohyb uzlů, který je řešen pomocí standardních numerických systémů pro řešení[47]. Mezi další známé modely patři Finite element simulation, což je fyzikálně více přesná metoda, která používá často používanou finite element method k vyřešení parciální diferenciální rovnice, které upravují dynamiku elastického materiálu. Tělo je modelováno jako třídimenzionální elastické kontinuum rozděleného do velkého množství pevných celků, které do sebe zapadají. Napětí v každém prvku je řešeno použitím modelu materiálu. Dané lokální napětí může být vypočítáno pomocí zobecněné formy Hookova zákona. Pohybová rovnice z uzlů prvku se dosáhne tím, že se integruje pole napětí nad každým prvkem a vztahující se uzly zrychlí prostřednictvím druhého Newtonova zákona[48].

Mezi další požívané modely můžeme dále zařadit například Energy minimization methods[49] a Shape matching[50]. Deformace měkkého tělesa může být také zpracována tradičními pevnými tělesy, které mohou modelovat pohyb měkkého tělesa pomocí sítě více těles propojených omezeními.

Vzhledem k tomu, že se jedná o pokročilí prvek fyzikálního enginu, tak podporu měkkých těles nabízí pouze fyzikální framework Ammo.js, který využívá model pružina-hmota s pozicově založenými řešiteli a nabízí pomocné funkce ulehčující vytvoření měkkého tělesa. Nicméně v některých fyzikálních enginech jsme schopni dosáhnout věrohodné simulace měkkého tělesa za pomocí částic, která jsou se sousedními částicemi navzájem propojeny vzdálenostním omezením, kde každá částice reprezentuje jeden vertex grafického tělesa tkaniny. Nastavením lineárního tlumení částic a tuhosti omezení pak můžeme simulovat měkkost tkaniny, tento přístup však nefunguje moc dobře u volumetrických těles.

5.6 Simulace kapalin

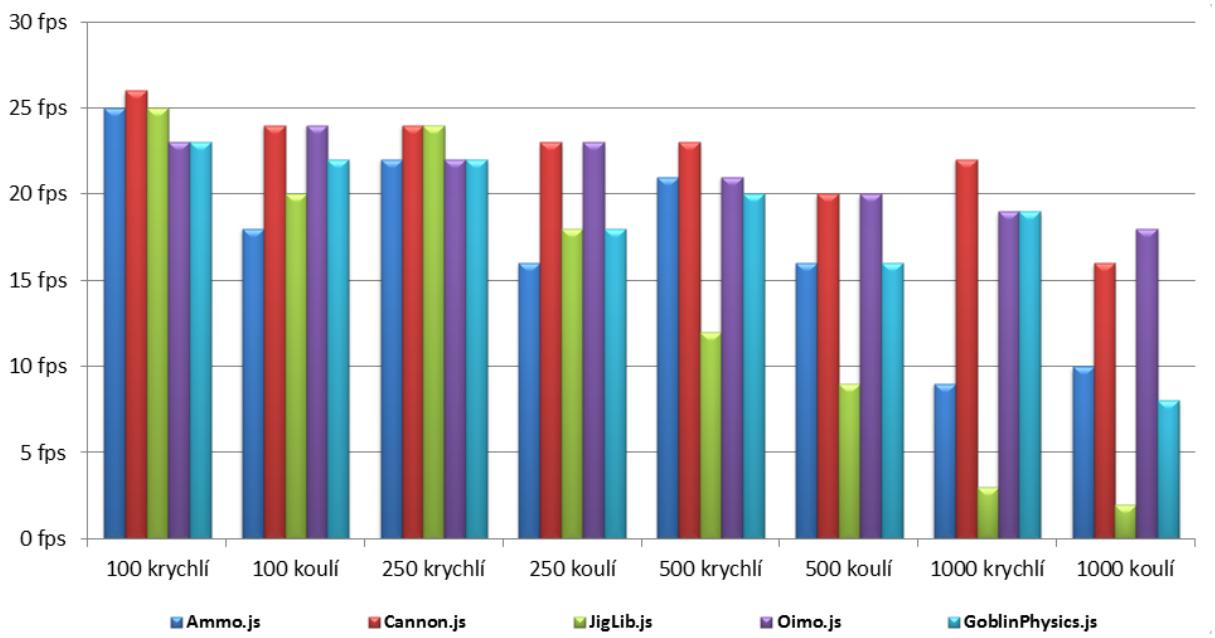
Simulace kapalin je oblast v počítačové grafice, která se zaměřuje na generování realistické animace kapalin, jako je voda a kouř. Simulátor tekutin emuluje pohyb popsané tekutiny s využitím Eulerových rovnic nebo Navier-Stokesových rovnic, které popisují fyziku tekutin, nebo jakoukoli jejich zjednodušenou verzi. Simulace kapalin se liší od výpočetní dynamiky kapalin v tom, že simulace kapalin je použita primárně pro vizuální efekty, zatím co výpočetní dynamika kapalin se používá k studování chování kapalin vědeckým způsobem. Nicméně některé z technik simulace kapalin mají původ ve výpočetní dynamice kapalin. Klíčový rozdíl mezi jednotlivými technikami je, že výsledky, jsou pouze pravděpodobné. To znamená, že lidský pozorovatel schopen určit, zda daná animace je fyzicky správná a výsledky jsou dostačující, zatímco ve fyzice, strojírenství, nebo matematice jsou zavedeny přísnější chybové metriky. Existuje několik konkurenčních technik pro tekuté simulace s různými kompromisy[51]. Nejběžnější jsou metody založené na Eulerově mřížce, metody založené na částicích, nebo nejnovější Lattize-Boltzmann metody[52].

V našem případě se budeme zabývat pouze částicově založenou výpočetní metodou Vyhlazená hydrodynamika částic(SPH), která má využití v hodně oblastech výzkumu včetně astrofyziky, balistiky, vulkanologie a oceánografie. Tento přístup poskytuje určité výhody oproti tradičním mřížkově založeným metodám. SPH zaručuje zachování hmoty bez dalších výpočtů, protože samotné částice představují hmotu. SPH vypočítává tlak z vážených příspěvků sousedních částic, namísto řešení soustavy lineárních rovnic. Na rozdíl od technik na bázi mřížky, která musí sledovat hranice tekutin, SPH vytváří volný povrch pro dvoufázové interakce tekutiny[53]. Pro tyhle důvody je možné simulovat pohyb kapaliny použitím SPH v realném čase. Nicméně, mřížkové i částicově založené techniky stále vyžadují použití polygonizačních technik, jako je marching cubes[54].

Javascriptové fyzikální enginy nejsou zatím příliš rozvinuté, pouze fyzikální framework Cannon.js nabízí real-time experimentální SPH systém. SPH podporuje i fyzikální engine Bullet, nicméně kvůli konverzi z jazyka C++ je u Ammo.js obtížné tuto výpočetní techniku zprovoznit.

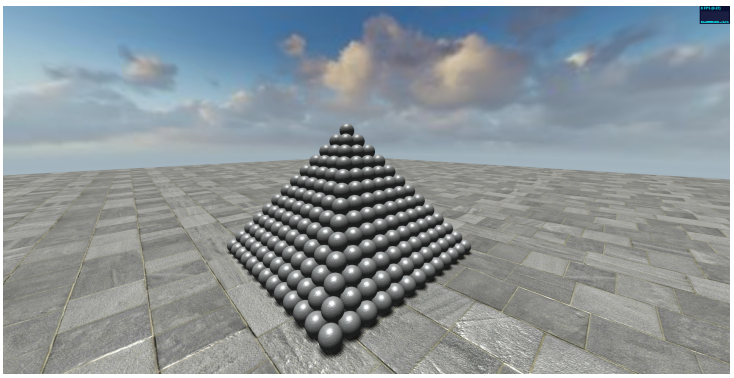
5.7 Výkonnost

Výkonnost byla měřena zaznamenáním fps a spouštěním scén s různým počtem různě poskládaných objektů. Výsledky byly testovány na notebooku DELL s procesorem Intel(R) Core(TM) i5-6300U CPU @ 2.40 GHz, integrovanou grafická karta Intel(R) HD Graphics 520 a 16 GB paměti RAM. První výsledky se měřili pro 100, 250, 500 a 1000 padajících krychlí nebo koulí. V grafu můžeme vidět výsledky zátěžových testů.

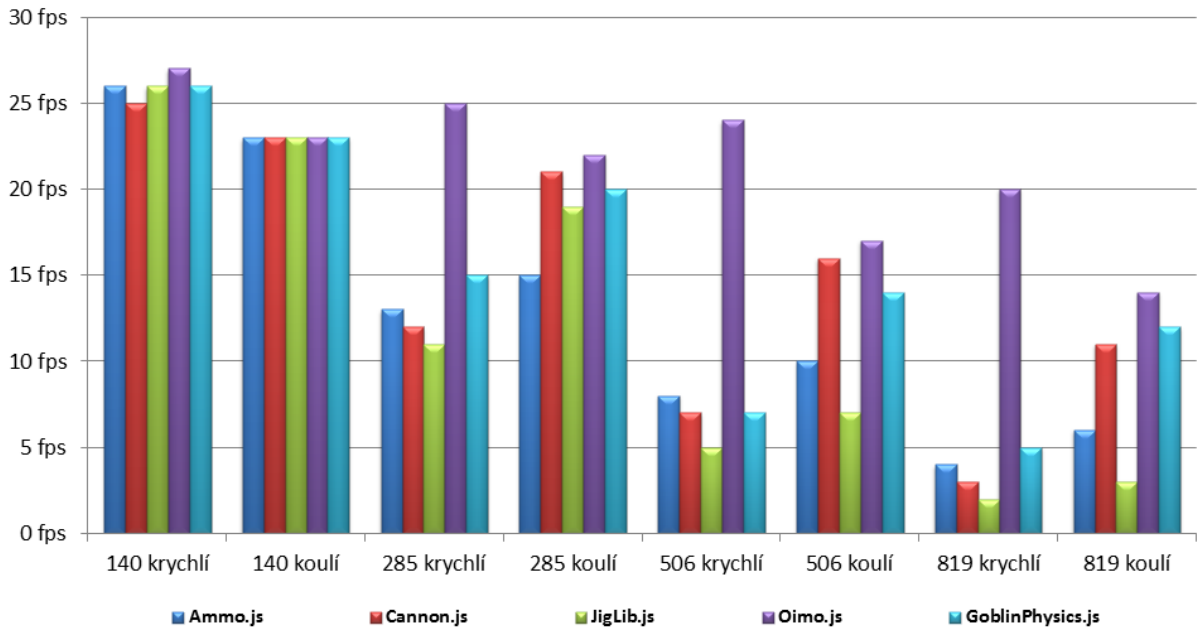


Obrázek 16: Graf prvního měření

Druhé výsledky se měřili pro 140, 285, 506 a 819 na sebe naskládaných krychlí nebo koulí do tvaru pyramidy. Počet objektů v druhém případě byl zaokrouhlen tak, aby odpovídal počtu objektů potřebných k sestavení pyramidy. Každé následující číslo odpovídá přičtení dalších dvou pater pyramidy.



Obrázek 17: Test skládání - 819 koulí poskladaných do pyramidy



Obrázek 18: Graf druhého měření

6 Závěr

Cílem práce bylo srovnání aktuálních možností dostupných fyzikálních enginů pro WebGL. V úvodu práce jsem se zaměřil na zobrazování interaktivní 3D grafiky na webu. Následně jsem popsal fyzikální enginy a použité technologie. Jednotlivé části enginu, včetně použitých algoritmů, jsou popsány v části výsledky experimentu, kde na konci každé části je srovnání vybraných fyzikálních enginů v dané oblasti. Hlavním praktickým výstupem je vytvoření experimentální aplikace, kde můžeme demonstrovat většinu funkcionality vybraných fyzikálních frameworků.

Výsledky ukázaly, že frameworky, které jsou napsány přímo v JavaScriptu jsou více optimalizovány pro běh na webu. Ammo.js je funkcionálně nejbohatší framework, avšak mohou u něj nastat problémy s výkonem. JigLib.js je ovlivněna svým designem API a nedostatkem funkčnosti. Fyzikální frameworky jako Cannon.js, Oimo.js a GoblinPhysics.js psané přímo v JavaScriptu mají spousty výhod jako nižší velikost a poměrně dobrý výkon. Hlavní výhodu těchto enginů však vidím v příznivějším designu API pro vývoj než konvertované frameworky, které vyžadují specifické zacházení, avšak ani jeden z nich zatím není připraven na to, být široce využíván. I když velmi složité a dynamické scény v internetovém prohlížeči ještě nemusí být dosažitelné, tak nabízí mnoho dalších zajímavých možností. Míchání snadno přístupných technologií v dnešním prohlížečích vytváří velmi zajímavé prostředí pro nové aplikace.

Literatura

[1] *Why Browser Plug-Ins Are Going Away and What's Replacing Themb* [online] [cit. 2017-02-25]. Dostupné z: https://www.howtogeek.com/179213/why-browser-plug-ins-are-going-away-and-whats-replacing-them/

[2] *WebGL - OpenGL ES 2.0 for the Web* [online] [cit. 2017-02-25]. Dostupné z: http://www.khronos.org/webgl/

[3] *OpenGL 4.5* [online] [cit. 2017-02-25]. Dostupné z: http://www.khronos.org/opengl/

[4] *Vulkan 1.0* [online] [cit. 2017-02-25]. Dostupné z: http://www.khronos.org/vulkan/

[5] *MetalAPI* [online] [cit. 2017-02-25]. Dostupné z: https://developer.apple.com/metal/

[6] *Will Vulkan have any effect on WebGL?* [online] [cit. 2017-02-25]. Dostupné z: https://www.quora.com/Will-Vulkan-have-any-effect-on-WebGL

[7] *A vocabulary and associated APIs for HTML and XHTML* [online] [cit. 2017-02-25]. Dostupné z: https://www.w3.org/TR/html5/

[8] *W3C* [online] [cit. 2017-02-25]. Dostupné z: https://www.w3.org/

[9] *WebGL stats* [online] [cit. 2017-02-25]. Dostupné z: http://webglstats.com/

[10] Subratty Mohammad A. I.,Tonta Yannick 1. T.,Raj Kishen Moloo, *Increasing Efficiency in Visualizing Large Sets of 3D models on the Web* (26 August 2015), 1st IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2015(SVS College of EngineeringCoimbatore, India) [cit. 2017-02-25]

[11] *Top 10 HTML5, JavaScript 3D Game Engines and Frameworks* [online] [cit. 2017-02-25]. Dostupné z: http://noeticforce.com/best-3d-javascript-game-engines-frameworks-webgl-html5

[12] *Babylon* [online] [cit. 2017-02-25]. Dostupné z: http://www.babylonjs.com/

[13] *Three.js* [online] [cit. 2017-02-25]. Dostupné z: https://threejs.org/

[14] *Turbulenz* [online] [cit. 2017-02-25]. Dostupné z: http://biz.turbulenz.com/developers

[15] *Famo.us* [online] [cit. 2017-02-25]. Dostupné z: http://famous.org/learn/hello-famous.html

[16] *Playcanvas* [online] [cit. 2017-02-25]. Dostupné z: https://playcanvas.com/

[17] *Goo engine* [online] [cit. 2017-02-25]. Dostupné z: https://goocreate.com/

[18] *CopperLicht* [online] [cit. 2017-02-25]. Dostupné z: http://www.ambiera.com/copperlicht/

[19] *Voxel.js* [online] [cit. 2017-02-25]. Dostupné z: http://voxeljs.com/

[20] *Blend4Web* [online] [cit. 2017-02-25]. Dostupné z:https://www.blend4web.com/

[21] *Enchant.js* [online] [cit. 2017-02-25]. Dostupné z: http://enchantjs.com/

[22] *PhysX - GeForce.* [online] [cit. 2017-02-25]. Dostupné z: http://www.geforce.com/hardware/technology/physx

[23] *Havok physics* [online] [cit. 2017-02-25]. Dostupné z: http://www.havok.com/

[24] *Physics engine* [online] [cit. 2017-02-25]. Dostupné z: http://www.computerhope.com/jargon/p/physics-engine.htm

[25] *Physics engines for dummies* [online] [cit. 2017-02-25]. Dostupné z: http://www.wildbunny.co.uk/blog/2011/04/06/physics-engines-for-dummies/

[26] *Physics libraries* [online] [cit. 2017-02-25]. Dostupné z: https://github.com/bebraw/jswiki/wiki/Physics-libraries

[27] *Ammo* [online] [cit. 2017-02-25]. Dostupné z: https://github.com/kripken/ammo.js/

[28] *Bullet* [online] [cit. 2017-02-25]. Dostupné z: http://bulletphysics.org/

[29] *Emuscripten* [online] [cit. 2017-02-25]. Dostupné z: https://github.com/kripken/emscripten/wiki

[30] *Cannon* [online] [cit. 2017-02-25]. Dostupné z: http://www.cannonjs.org/

[31] *JigLib* [online] [cit. 2017-02-25]. Dostupné z: https://github.com/bartdeboer/JigLibJS2/

[32] *Oimo* [online] [cit. 2017-02-25]. Dostupné z: https://github.com/lo-th/Oimo.js/

[33] *GoblinPhysics* [online] [cit. 2017-02-25]. Dostupné z: http://www.goblinphysics.com/

[34] *W3.CSS* [online] [cit. 2017-02-25]. Dostupné z: https://www.w3schools.com/w3css/

[35] *Modern Physics Engines* [online] [cit. 2017-02-25]. Dostupné z: http://homel.vsb.cz/ seb0064/web/index.html

[36] *Web Workers* [online] [cit. 2017-02-25]. Dostupné z: https://w3c.github.io/workers/

[37] *Broad And Narrow Phase Collision Detection Systems* [online] [cit. 2017-02-25]. Dostupné z: https://ianqvist.blogspot.cz/2010/07/broad-and-narrow-phase-collision.html

[38] *Broad Phase Collision Detection Using Spatial Partitioning* [online] [cit. 2017-02-25]. Dostupné z: http://buildnewgames.com/broad-phase-collision-detection/

[39] *Dynamic AABB Tree* [online] [cit. 2017-02-25]. Dostupné z: http://www.randygaul.net/2013/08/06/dynamic-aabb-tree/

[40] *Collision Detection Using the Separating Axis Theorem* [online] [cit. 2017-02-25]. Dostupné z: https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem–gamedev-169

[41] *Pixel perfect collision detection in DirectX* [online] [cit. 2017-02-25]. Dostupné z: https://www.gamedev.net/resources/_/technical/directx-and-xna/pixel-perfect-collision-detection-in-directx-r2939

[42] David Baraff, Andrew Witkin, Michael Kass, *Untangling Cloth* ACM SIGGRAPH 2003, Annual Conference Proceedings [online] [cit. 2017-02-25]. Dostupné z: http://graphics.pixar.com/library/UntanglingCloth/paper.pdf

[43] *Rigid-body dynamics* [online] [cit. 2017-02-25]. Dostupné z: http://encyclopedia2.thefreedictionary.com/rigid-body+dynamics

[44] *Video Game Physics Tutorial - Part III: Constrained Rigid Body Simulation* [online] [cit. 2017-02-25]. Dostupné z: https://www.toptal.com/game/video-game-physics-part-iii-constrained-rigid-body-simulation

[45] *Vehicle physics* [online] [cit. 2017-02-25]. Dostupné z: http://digitalrune.github.io/DigitalRune-Documentation/html/143af493-329d-408f-975d-e63625646f2f.htm

[46] *Multi-Core Simulation of Soft-Body Characters using Cloth* [online] [cit. 2017-02-25]. Dostupné z: https://software.intel.com/en-us/articles/multi-core-simulation-of-soft-body-characters-using-cloth

[47] *Spring Physics* [online] [cit. 2017-02-25]. Dostupné z: http://new.gafferongames.com/post/spring_physics/

[48] *A Smoothed Finite Element-Based Elasticity Model for Soft Bodies* [online] [cit. 2017-02-25]. Dostupné z: https://www.hindawi.com/journals/mpe/2017/1467356/

[49] *Introduction to molecular dynamics* [online] [cit. 2017-02-25]. Dostupné z: http://www4.ncsu.edu/ franzen/public_html/CH795N/lecture/IV/IV.html

[50] *Particle-based Rigid Bodies using Shape Matching* [online] [cit. 2017-02-25]. Dostupné z: http://labs.byhook.com/2010/06/29/particle-based-rigid-bodies-using-shape-matching/comment-page-1/

[51] *The science of fluid sims* [online] [cit. 2017-02-25]. Dostupné z: https://www.fxguide.com/featured/the-science-of-fluid-sims/

[52] *Lattice Boltzmann* [online] [cit. 2017-02-25]. Dostupné z: http://physics.ujep.cz/ mlisa-l/nm_2/dkramolis/Menu_CZ.html

[53] *Smoothed Particle Hydrodynamics Fluid Simulation* [online] [cit. 2017-02-25]. Dostupné z: http://rlguy.com/sphfluidsim/

[54] *Marching Cubes* [online] [cit. 2017-02-25]. Dostupné z: http://iris.uhk.cz/grafika/mcb/mc0.htm

A Přílohy

Součástí bakalářské práce je příloha obsahující zdrojové kódy v javascriptu spolu s podklady. Struktura složek je následující:

/web/content/
: Obsahuje jednotlivé části html obsahu nahrané do těla indexu.

/web/css
: Obsahuje kaskádové styly pro html.

/web/images/
: Obsahuje textury pro 3D scénu.

/web/js
: Obsahuje vlastní javascriptové kódy.

/web/models/
: Obsahuje vlastní a upravené volně dostupné 3D modely.

/web/shaders/
: Obsahuje vlastní a upravené volně dostupné vertex a fragment shadery.

/web/vendor
: Obsahuje použité javascriptové knihovny třetí strany.

/web/index.html
: Index je jedinou stránku na kterou se lze přesměrovat a běží v něm celá aplikace.

/SEB0064_ModernPhysicsEngines.pdf
: Index je jedinou stránku na kterou se lze přesměrovat a běží v něm celá aplikace.

B Ukázkový kód

Ukázkový kód pro vytvoření fyzikálního světa a přidání krychle ve všech vybraných fyzikálních enginech

```
/* Ammo.js */
var collisionConfig = new Ammo.btDefaultCollisionConfiguration();
var dispatcher = new Ammo.btCollisionDispatcher(collisionConfiguration);
var solver = new Ammo.btSequentialImpulseConstraintSolver();
var  broadphase = new Ammo.btDbvtBroadphase();
var world = new Ammo.btDiscreteDynamicsWorld(
   dispatcher,
   broadphase,
   solver,
   collisionConfig
);
var pairCallback = new Ammo.btGhostPairCallback();
world.getPairCache().setInternalGhostPairCallback(pairCallback);
world.setGravity(new Ammo.btVector3(0, -9.8, 0));

var mass = 10, sx = 1, sy = 1, sz = 1, x = 0, y = 0, z = 0;
var localInertia = new Ammo.btVector3(0, 0, 0);
var transform = new Ammo.btTransform;
transform.setIdentity();
transform.setOrigin(new Ammo.btVector3(x, y, z));
var shape = new Ammo.btBoxShape(new Ammo.btVector3(sx, sy, sz));
shape.calculateLocalInertia(mass, localInertia);
var motionState = new Ammo.btDefaultMotionState(transform);
var rbInfo = new Ammo.btRigidBodyConstructionInfo(
   mass,
   motionState,
   shape,
   localInertia
);
var boxBody = new Ammo.btRigidBody(rbInfo);
world.addRigidBody(boxBody);
```

```
/* Cannon.js */
var world = new CANNON.World();
world.quatNormalizeSkip = 0;
world.quatNormalizeFast = false;
world.broadphase = new CANNON.NaiveBroadphase();
world.solver = new CANNON.SplitSolver(solver);
world.solver.iterations = 7;
world.solver.tolerance = 0.1;
world.gravity.set(0, -9.8, 0);

var mass = 10, sx = 1, sy = 1, sz = 1, x = 0, y = 0, z = 0;
var boxShape = new CANNON.Box(new CANNON.Vec3(sx, sy, sz));
var boxBody = new CANNON.Body({
   mass : mass
});
boxBody.addShape(boxShape);
boxBody.position.set(x, y, z);
world.addBody(boxBody);


/* JigLib.js */
var world = JigLib.PhysicsSystem.getInstance();
world.setCollisionSystem(true);
world.setSolverType("ACCUMULATED");

var mass = 10, sx = 1, sy = 1, sz = 1, x = 0, y = 0, z = 0;
var boxBody = new JigLib.JBox(null, sx*2, sz*2, sy*2);
boxBody.set_mass(mass);
boxBody.moveTo(new JigLib.Vector3D(x, y, z, 0));
if(mass == 0) boxBody.set_movable(false);
world.addBody(boxBody);
```

```
/* Oimo.js */
var world = new OIMO.World({
   timestep: 1/60,
   iterations: 8,
   broadphase: 2, // 1 brute force, 2 sweep and prune, 3 volume tree
   worldscale: 1,
   random: false,
   info: false,
   gravity: [0,-9.8,0]
});

var mass = 10, sx = 1, sy = 1, sz = 1, x = 0, y = 0, z = 0;
var move = true;
if(mass == 0) move = false;
var boxBody = app.Container.SceneManager.World.world.add({
   type: 'box',
   size: [sx*2, sy*2, sz*2],
   pos: [x, y, z],
   move: move
});


/* GoblinPhysics.js */
var world = new Goblin.World(
   new Goblin.BasicBroadphase(),
   new Goblin.NarrowPhase(),
   new Goblin.IterativeSolver()
);
world.solver.warmstarting_factor = 0.4;
world.solver.sor_weight = 1;

var mass = 10, sx = 1, sy = 1, sz = 1, x = 0, y = 0, z = 0;
var boxBody = new Goblin.RigidBody(new Goblin.BoxShape(sx, sy, sz), mass);
boxBody.position.set(x, y, z);
world.addRigidBody(boxBody);
```

Výpis 1: Vytvoření krychle ve vybraných fyzikálních enginech