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

FYZIKÁLNÍ SIMULACE VE 3D SCÉNĚ S VYUŽITÍM KNIHOVNY PHYSX

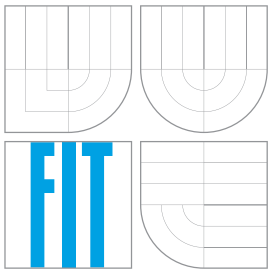
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TIBOR JAŠEK

BRNO 2013

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

FYZIKÁLNÍ SIMULACE VE 3D SCÉNĚ S VYUŽITÍM KNIHOVNY PHYSX

PHYSICAL SIMULATION IN 3D SCENE USING PHYSX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TIBOR JAŠEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN PEČIVA, Ph.D.

BRNO 2013

Abstrakt

Tahle bakalářská práce je zameřená na simulaci fyziky s využitím fyzikálního enginu PhysX. Program je vytvořen v XNA Frameworku. Aplikace zahrňuje síťovou hru pro dva hráče a zobrazení několika fyzikálních simulací.

Abstract

This bachelor thesis is focused on physic simulation utilized by physics engine PhysX. Program is made in XNA Framework. Application consists of network game for two players and presentation of few physical simulations.

Klíčová slova

XNA Framework, PhysX, PhysX.Net, Lidgren.Network, herní engine, detekce kolizí

Keywords

XNA Framework, PhysX, PhysX.Net, Lidgren.Network, game engine, collision detection

Citace

Tibor Jašek: Fyzikální simulace ve 3D scéně s využitím knihovny PhysX, bakalářská práce, Brno, FIT VUT v Brně, 2013

Fyzikální simulace ve 3D scéně s využitím knihovny PhysX

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Pečivu Ph.D.

.....
Tibor Jašek
15. května 2013

Poděkování

Rád by som poďakoval vedúcemu práce Ing. Janovi Pečivovi Ph.D. za jeho cenné rady a tipy.

Obsah

Kapitola 1

Úvod

Hranie hier je od nepamäti pre ľudstvo typické a hry sú súčasťou každej spoločnosti. Hry prešli vývojom cez klasické stolné ako go, liubo neskôr šach, tímové hry až po v súčastnosti veľmi populárne a obľúbené elektronické hry. A u každej elektronickej hry, aby sa stala úspešnou, je potrebné splniť niekoľko aspektov - hrateľnosť, grafická stránka, príbeh a čím ďalej viac žiadaná reálnosť interakcií objektov virtuálneho sveta.
V 90-tych rokoch hry ako Wolfenstein 3D, Doom a iné začali revolúciu, ktorá vytvarovala celkové smerovanie počítačového herného priemyslu s príchodom hardwarovej grafickej akcelerácie. Najskôr vo forme 3D prídavných kariet, ďalej s grafickými doskami, ktoré integrovali 2D a 3D funkcie do jedného čipu, neskôr nazvaného GPU [11]. Postupne grafické karty prešli rapídnym vývojom a spoločnosti ako nVIDIA a ATI v dnešnej dobe ponúkajú grafické karty s multivláknovým výpočtom a výpočetným výkonom rádovo v stovkách MHz. Za zlepšením dnešných hier môže stáť byť práve dokonalejšia herná fyzika, ktorá umožní hráčovi oveľa realistickejší herný zážitok a taktiež neporovnateľné herné možnosti (deštrukcia objektov, reálna interakcia na meniace sa počasie a pod.). Herná fyzika bola spočiatku otázkou výpočtov CPU, kedy sa GPU staralo len o výsledné vykreslovanie. Túto situáciu zmenila firma Ageia ktorá prišla na trh s technológiou PhysX - PPU (Physics Processing Unit) čipom schopným prevádzať herné fyzikálne výpočty oveľa rýchlejšie ako univerzálne výpočetné jednotky.
Táto práca je zameraná na možnosti fyzikálneho enginu PhysX a s jeho využitím implementuje jednoduchú sieťovú 3D hru PhysXTanks.

Kapitola 2

Možnosti fyzikálnych simulácií v hrách

Slovo simulácia je možné použiť vo veľa kontextoch, ako napríklad simulácia technológie využívaná pre jej optimalizáciu, bezpečnostné inžinierstvo, testovanie, trénovanie, vzdelávanie a video hry. Simulácia je taktiež používaná vedeckým modelovaním prirodzených systémov alebo ľudských systémov na nahliadnutie do ich fungovania [11]. Pre potreby tejto práce je relevantná počítačová simulácia ktorá je pokusom modelovať situácie z reálneho alebo hypotetického sveta s cieľom ich študovania. Vďaka možnostiam týchto systémov môžeme meniť rôzne aspekty sledovaného deja a tak získať predikcie fungovania daného systému [2]. Táto kapitola sa všeobecne zaoberá možnosťami fyzikálnych simulácií v počítačových hrách, ich postupným vývojom, existujúcimi programami na ich vykonávanie a taktiež ich limitáciách na dnešných CPU a GPU. Na simulácie môžeme všeobecne nahliadať z niekoľkých

2.1 Fyzikálny engine

Jedná sa o počítačový software, ktorý zabezpečuje simuláciu nejakých fyzikálnych systémov. V dnešnej dobe sa využíva tento software hlavne na simuláciu pevných objektov, kvapalín, tekutín, mäkkých častí a tiež látok (v zmysle plátno, tknanina) pre využitie vo filmoch počítačovej grafike a hrách. Bez adekvátnych fyzikálnych simulácií aj najkrajšia hra vyvoláva pocit statičnosti a vyzerá mŕtvo. U hier sa jedná o simuláciu fyziky v reálnom čase čo vyžaduje veľmi rýchle výpočty, zložité optimalizácie a rôzne prístupy.

Rôzne simulácie si vyžadujú rozdielny prístup preto výber vhodného enginu je dôležitým aspektom kvality výslednej aplikácie. Článok [4] sa zaoberá kvantitatívnym vyhodnotením zadarmo dostupných fyzikálnych enginov pre simulačné systémy a herný vývoj. Porovnávané sú presnosť a výpočetná efektivita integrovaných nastavení, vlastností materiálov, kopenie objektov na seba a systém detekcie kolízií [4].

Porovnávania sú vykonávané s využitím PAL [1] na siedmych testovaných enginoch - AGEIA PhysX, Bullet Physics Library, Dynamechs, JigLib, Meqon, Newton Phycs SDK, Open Dynamics Engine, OpenTissue Library, Tokamak a True Axis Physics SDK.

Existuje šesť esenciálnych faktorov, ktoré rozhodujú o celkovej výkonnosti fyzikálneho enginu [4]:

[1] PAL (Physics Abstraction Layer) - je abstraktná vrstva poskytujúca se jedinečných rozhraní k rôznym bežným vlastnostiam fyzikálnych enginov

- *Paradigma Simulátora* rozhoduje o tom, ktoré aspekty môžu byť správne simulované. To ovplyvňuje presnosť v rozhodovaní obmedzení.
- *Integrátor* rozhoduje o numerickej presnosti simulácie.
- *Reprezentácia objektov* prispieva k efektivite a presnosti kolízií v simulácií.
- *Detekcia kolízie a stanovenie kontaktu* tiež prispievajú k efektivite a presnosti kolízií v simulácií.
- *Vlastnosti materiálov* rozhodujú aký fyzikálny model simulácia dokáže aproximovať (napr. Coloumbove trenie).
- *Implementácia obmedzení* určuje aké obmedzenia sú podporované a ako presne ich môžeme simulovať.

Podľa výsledkov všetky analyzované fyzikálne enginy poskytujú prostriedky vhodné pre vývoj hier. Takmer každý fyzikálny engine dopadol najlepšie v rodielnom zo šiestich testov. Z open source[2] enginov dopadol najlepšie Bullet engine ktorý poskytol celkovo najlepšie výsledky, a predčil i niektoré komerčné enginy[4].

Jediný test ktorý nebol nesplnil žiadny z uvedených simulátorov bolo realistické hromadenie troch gúľ. Žiadny simulátor nezahrnul žiadny ruch na zvýšenie realičnosti simulácie[4].

2.2 Detekcia kolízií

Existuje veľa dynamických aplikácií, ktoré neberú v úvahu sily ktoré nastanú pri zrážke alebo dotyku telies. Avšak čím ďalej viac aplikácií sa zaoberá detekciou kolízií - od jednoduchých kolízií letiacej gule so zemou (napr. záleží len na tom, či sa guľa nachádza nad alebo pod povrchom) až po padanie teľa po schodoch (veľa priesečníkov tvarov, napr. kolízie ohybu s 3D objektom so zachovaním obmedzení v tele). SDK musí mať informácie o tvaroch tiel, ktoré sa môžu dotýkať a taktiež vlastnosťami rôznych povrchov - napr. drsnosť, pružnosť.

2.2.1 Obecná fáza detekcie kolízií

Existuje niekoľko spôsobov testovania kolízií medzi jednotlivými pármi objektov. Podstatné je zistiť ktoré páry zo všetkých možných párov v scéne sa môžu dotýkať. Testovanie kolízií všetkých objektov navzájom by bolo možné, ale časovo náročné a taktiež zbytočné (až n*n/2 potenciálnych párov v skupine n tvarov). Tento problém je vyriešený automatickým rozdeľovaním priestoru zabraného útvarmi, kedy sa testujú kolízie útvarov len voči blízkym útvarom. V PhysX SDK sú po vyhodnotení, že môže nastať kolízia páru útvarov vykonané 3 po sebe idúce kontroly, ktoré zisťujú, či sa používateľ "zaujíma"o tento pár. Len v prípade, že tieto tri kontroly prejdú sa vykoná časovo náročná kontrola kontaktu. Musia byť splnené nasledovné podmienky, aby sa vykonala detekcia kolízie:

[2]software s dostupným (otvoreným) zdrojovým kódom

```
(a->getActor()->isDynamic() || b->getActor()->isDynamic())
 && NxScene::getGroupCollisionFlag(a->getGroup(), b->
    getGroup())
 && (!(NxScene::getShapePairFlags(a,b) & NX_IGNORE_PAIR))
```

Fragment 2.1: Kontroly pred vykonaním detekcie kolízie

Prvá podmienka kontroluje, či sa nejadná o dva statické objekty, ktoré nemôžu spolu kolidovať, kvôli nemožnosti pohybu. Drohou podmienkou sa zisťuje, či sú daný činitelia v rovnakej koliznej skupine (defaultne sa všetci nachádzajú v skupine 0) a teda je ich vzájomná kolízia možná. Posledný test kontroluje či u daných útvarov nie je ignorovaná ich vzájomná kolízia.

2.2.2 Diskrétna a spojitá detekcia kolízií

Pri rýchlo sa pohybujúcich objektoch môže nastať problém kedy sa dané objekty nezrazia aj napriek ich viditeľnému prieniku. Tento problém nastáva kvôli prechodu objektu iným v jednom časovom kroku, kedy prostredie nezaznamená kolíziu. Tento efekt nastáva napríklad pri pohybe guľky cez úzku dosku, kedy v jednom časovom kroku sa nachádza guľka na jednej strane dosky a v nasledujúcom už na druhej strane. Tento problém je možné riešiť niekoľkými spôsobmi - napr. pomocou testu diskrétneho prekrývania medzi dvoma statickými OBBs (object boundary boxes - priestorové ohraničenie objektu slúžiace na detekciu kolízií). Najefektívnejší test tohoto typu je pravdepodobne popísaný v [6], ktorý je založený na teoréme rozdeľovania osí. Ak predpokladáme, že prvý OBB je opísaný tromi osami e1, e2, e3, stredom Ta a polovičnými dĺžkami pozdĺž osí a1, a2, a3. Podobne je druhý objekt popísaný osami f1, f2, f3, stredom Tb a polovičnými dĺžkami pozdĺž osí b1, b2, b3. Rozdeľovací teorém osí hovorí, že dve statické OBBs sa prekrývajú len vtedy, ak všetkých 15 rozdeľovacích testov osí zlyhá. Rozdeľovací test je nasledovný: os a rozdeľuje OBBS len a len vtedy ak [10]:

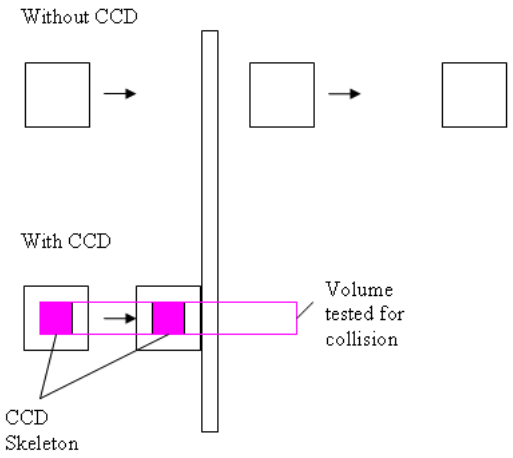
$$|a \cdot T_a T_b| > \sum_{i=1}^{3} a_i \, |a \cdot e_i| + \sum_{i=1}^{3} b_i \, |a \cdot f_i|$$

Päťnásť vhodných osí je odvodených z OBBs osí:

$$a \in \{e_i, f_j, e_i \times f_j, 1 \leq i \leq 3, 1 \leq j \leq 3\}$$

U spojitej detekcie kolízií je na rozdiel od exaktného diskrétneho OBB testu vykonaná konzervatívny test pozostávajúci z dvoch častí: 1. Vykonanie spojitej verzie testu 15 rozdeľovacích osí 2. Ak je zistené prekrývanie OBBs pre aktuálny časový interval, vykonáva sa "deliaci test", ktorý určuje, či má byť daný časový interval rozdelený [10].

Netestuje sa kolízia v diskrétnych bodoch, ale PhysX SDK tento problém rieši testovaním extrudovaného rozsahu, ktorý reprezentuje pohyb objektu po celý časový krok. Ak je zistená kolízia je možné vypočítať čas zrážky a pohyb objektu môže byť adekvátne obmedzený (vykonaný). Súčasná verzia SDK podporuje spojitú detekciu kolízií dynamických voči dynamickým a dynamických voči statickým objektom[?]. Vytváranie CCD skeletonov, ich aplikácia a limitácie sú bližšie popísané v dokumentácií danej verzie PhysX.



Obrázek 2.1: Spojitá kontrola kolízií

2.3 PhysX a využitie GPU

Marketing Nvidie tvrdí (v Cryostasis): "S adekvátne presnou simuláciou vody, deštrukcie cencúľa, a časticovými efektami, CPU sa ukazuje ako žalostne neadekvátna dodávať hrateľné snímkovanie. GPU ktoré nemá podporu PhysX sa vo výsledku stáva úzko-profilovým, a dodáva rovnakú úroveň výkonu bez ohľad na grafické možnosti hardwaru. GeForce GPUs s hardwarovou podporou fyziky ukazuje 2-4 násobné zvýšenie výkonu, dodávajúce skvelú rozšíriteľnosť naprieč GPU linkou."

Spomínaný 4 násobný rozdiel znie markantne, ale existuje niekoľko publikácií zaoberajúcich sa totuto problematikou (CPU verzus GPU, hardwarová akcelerácia). V publikácií [8] sa nachádza analýza systém výpočtu a spomínaný výkonostný rozdiel v simulácií fyziky medzi CPU a GPU. Reálne, ak by spoločnosť Nvidia chcela využiť CPU môže použiť zkomprinované SSE [3] s jednoduchou presnosťou pre PhysX. Každá inštrukcia by potom mohla vykonať 4 SIMD (viď. poznámka pod čiarou) operácie v jednom cykle, namiesto len jedinej skalárnej [8]. Týmto by sa rozdiel výkonnosti znížil zhruba na 2 násobok medzi CPU a GPU. Problém je teda v nezmenenej inštrukčnej sade (prevzatá od spol. Ageia) x87, ktorá je dnes už zastaralá a nepodporovaná väčšinou procesorov. Rozdiel výkonosti je tiež spôsobený implicitne jednovláknovým behov PhysX simulácie na CPU a mnohovláknovým behom na GPU. Je možné písať viac-vláknové aplikácie aj na CPU, ale viac-vláknové spracovanie nie je natívne podporované (prípadná optimalizácie je v rukách programátora).

Ďalším aspektom ktorý hovorí proti tejto technológií (v použitej verzii PhysX) je absencia hardwarovej akcelerácie pevných telies. V danej verzii sú hardwarovo akcelerované tekutiny, mäkké časti a látky (v zmysle plátno, tkanina). Tým pádom je podstatná časť objektov v hrách simulovaná len s využitím softwaru, čo môže ovplyvniť rýchlosť renderovania pri častej interakcii a vysokom počte takýchto objektov.

[3] Streaming SIMD Extensions - SIMD = Single instruction, multiple data. Jedná sa o streamové rozšírenie počítačov s viacerými procesnými elementami, ktoré vykonávajú rovnakú operáciu na viacerých dátových bodoch súčasne.

Kapitola 3

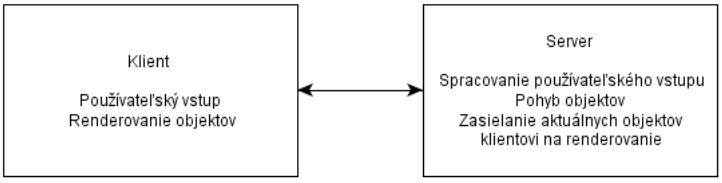
Použité technológie

Na dnešnom trhu je dostupné množstvo fyzikálnych enginov pracujúcich v reálnom čase ako Bullet, PAL, Box2D (open source), Havok, PhysX, ODE, Newton a iné. Podľa porovnávania herných enginov (PhysX, Havok, ODE, Netwon, Bullet) na konci roku 2009 PhysX dominuje počítačovému trhu, Havok dominuje na konzolovom trhu a za zmienku ešte stojí engine ODE v ktorom boli vytvorené najmä PC tituly [12]. PhysX tiež využíva výhodu veľkého výpocetného výkonu GPU pre komplexné fyzikálne výpočty. Kedže vlastním GPU GeForce 9300M GS a táto technológia je voľne použiteľná pre vývojárov zvolil som si ju ako východí engine pre moju bakalársku prácu. Využívam v nej PhysX SDK vo verzii 2.8.4.5.

PhysX je natívne písané v C++ ale je možné túto technológiu využiť spolu s XNA Frameworkom. Zvolenou technológiou na spojenie týchto dvoch technológií je projekt PhysX.Net. Na sieťovú komunikáciu je využitá knižnica lidgren.

Zasielanie dát v hrách sa odvíja od typu sieťového návrhu konkrétneho titulu. V našom prípade berieme do úvahy hry s klient-server prístupom. V týchto hrách je jeden autoratívny server, ktorý je zodpovedný za pribeh hlavnej hernej logiky. Na tento server je pripojený jeden, alebo viacero klientov. Títo klienti neboli pôvodne ničím viac ako prostriedkom ako spracovať používateľský vstup a preposlať ho serveru na spracovanie. Server vykoná vstupné príkazy, vykoná pohyb ostatných objektov a potom zašle klientovi zoznam objektov na renderovanie [3].

Tento prístup počíta s rôznymi sieťovými problémami, ktoré sú zmierňované rôznymi druhmi predikcie pohybu, kompenzáciou oneskoria príchodu paketov a podobne. Taktiež tento prístup je nevhodný pre hry v ktorých interagujú vysoké počty objektov a hry využívajúce pokročilú fyzikálnu simuláciu, kedy by objem dát prenášaný zo strany servera bol neúnostný. Z týchto dôvodov bol pre túto aplikáciu zvolený prístup klient/server v ktorom každý klient hýbe objektami na základe jeho lokálnych výpočtov a posiela sa len pozície protihráča a jeho projektilov.



Obrázek 3.1: Obecná herná architektúra klient/server

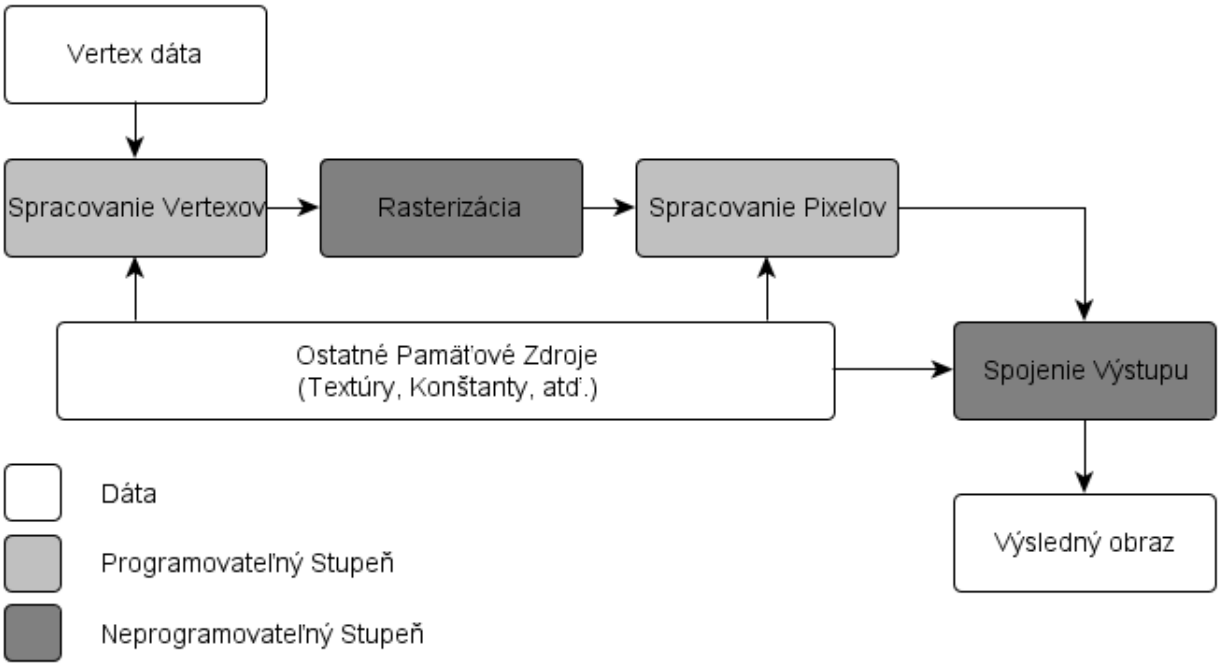
3.1 XNA Framework

Aplikácia je napísaná v jazyku C# s použitím XNA Game Studio 3.1 a .NET Frameworku 3.5. Síce sa jedná už o zastaralú technológiu, bola zvolená vzhľadom na použitú verziu PhysX.Net wrappera. Ako vývojové prostredie bolo použité Visual Studio 2008.
Telom aplikácie je trieda Game1 ktorá obsahuje päť predefinovaných metód, ktoré sú súčasťou každej XNA aplikácie.
V momente začiatku behu aplikácie sa práve raz volá *Game1* metóda (konštruktor). To znamená, že žiadne vnútorné hodiny neboli inicializované v momente keď bola táto metóda (konštruktor) zavolaná. To znamená, že pred vykonaním inicializácie nie je žiadny prístup k zdrojom (ako napríklad k *GraphicsDevice* triede), lebo ešte nebola inicializovaná [7].
Hneď po inicializácií *GameComponent* tried ktoré sa typicky vykonajú v metóde *Game1* sa zavolá metóda *Initialize*. V tejto metóde má programátor plný prístup k všetkým prostriedkom *Game* objektu [7]. V tejto metóde sa typicky inicializujú počiatočné hodnoty premenných a objektov vytvorených v hre (počiatočná poloha objektov, camera, farby priradené objektom a pod.).
Počas behu hry (aplikácie), sa XNA snaží zaručiť volanie metódy *Update* presne 60krát za sekundu (raz každých 0.0167 sekundy) [7]. V tejto metóde sú vykonávané všetky potrebné aktualizácie. Toto zahŕňa napr. detekciu kolízií, aktualizáciu polohy kamery, pohyb objektov, a iné ktoré budu rozoberané v ďalšej časti textu.
V *Draw* metóde sa nachádza kód, ktorý renderuje scénu na obrazovku. Defaultne je táto metóda volaná s rovnakou frekvenciou aká je nastavená na obnovovanie obrazovky[7]. V tejto metóde prebieha vykreslovanie všetkých 3D objektov, 2D indikátorov (`healthBar`, `reloadBar`) a tiež text vypisovaný na obrazovku.
V metóde *LoadContent* by malo byť vykonané všetko načítavanie grafických komponent hry. Táto metóda je zavolaná len raz na začiatku spustenia projektu. Toto načítavanie je vzhľadom na urýchlenie priebehu a umožneniu flexibility vykonané cez *content pipeline* [7]. V tejto metóde aplikácia načítava všetky potrebné zvuky, moddely, efekty, textúry a iné z externých súborov a taktiež sa tu vytvárajú niektoré objekty z knižnice PhysX.
Ak niektoré objekty použité v hre vyžadujú špeciálne odstraňovanie, alebo uvoľnovanie ideálnym miestom je metóda *UnloadContent*. Volá sa raz pred ukončením aplikácie [7].

3.1.1 Renderovacia pipeline XNA Frameworku

Kedže väčšina dnešných herných zariadení (PC, konzoly...) využíva 2D obrazovku je nutné transformovať 3D scénu na 2D obraz. Proces ktorý transformuje 3D scénu na obraz sa nazýva *renderovanie*.
Na obrázku 3.2 sa nachádza vysoko-úrovňový diagramu renderovacej pipeline používanej v XNA [5].

Objekt v 3D scéne je reprezentovaný 3D objektom (anglicky "mesh"), ktorý sa skladá z množiny bodov. Body reprezentujúce 3D objekt môžu mať rôzne odlišné atribúty, ako napríklad pozíciu, farbu, normálu a koordinát textúry [5].
Ako je zobrazené na obrázku 3.2 na začiatku vykreslovacieho procesu je množina bodov objektu poslaná do renderovacej pipeline, kde prechádza stavmi *spracovania množiny bodov*, *rasterizácie* a *spracovaním pixelov*. Na konci tohoto procesu je vygenerovaných veľa pixelov, pripravených na uloženie v konečnom obraze scény. Pretože veľa trojuholníkov jedného objektu sa môže uchádzať o rovnaký pixel na obrazovke posledná časť vykreslovacej pipe-



Obrázek 3.2: XNA renderovacia pipeline

line nazvaná *spojenie výstupu* rozhoduje, ktoré pixely sú najbližšie ku kamere. Časť *Spájač výstupu* ukladá tieto pixely vo výslednom obraze a rozhoduje ktoré pixely budú odstránené. Toto rozhodovanie je založené na vzdialenosti medzi kamerou a objektom, tým pádom sú zobrazené len najbližšie objekty, ale toto rozhodovanie môže byť ovplyvnené informáciou o priehladnosti objektov[5].
Od uvedenia DirectX 8.1 je možné naprogramovať niektoré časti renderovacej pipeline prostredníctvom vytvorenia malých programov nazývaných *shadery*. Typicky sú napísané v HSLS (High Level Shading Language - vysokoúrovňový jazyk na popis tienenia). Tieto programi umožňujú používateľovi ovplyvňovať každý stav spracovania v rámci jednotlivých programovateľných stupňov GPU a tiež určovať ktoré dáta sú vstupom a výstupom [5]. Používanie shaderov poskytuje viacero možností ako ovplyvniť výsledný vzhľad aplikácie. *Vertex shader* je možné využiť na spracovanie atribútov vrcholov ako napr. farby, normály a iných umožnujúcich zobrazovanie deformácie pevných objektov, pohybu častíc a pod. V *pixel shadery* má používateľ možnosť ovplyvniť farbu pixelu vzhľadom na svetelné podmienky, odrazy a taktiež meniť aspekty celej renderovanej scény ako napr. kontrast, sýtosť, nejasnosť. V tomto stupni je tiež možné zmeniť hĺbku pixelu, oproti implicitnej hodnote nastavovanej na základe vzdialenost od kamery.

3.2 Lidgren.Network

Lingren.Network je sieťová knižnica pre .NET Framework ktorá využíva jediný UDP socket na zabezpečenie jednoduchého aplikačno-programovacieho rozhrania na pripákanie klienta a serveru, čítanie a zasielanie správ [9]. V súčastnosti je táto knižnica 3. generácie a jej autorom je Michael Lidgren.

Základom komunikácie v tejto knižnici je zasielanie správ. Sú dva typy správ:

- Správy knižnice, ktoré slúžia na zasielanie správ oznamujúcich pripojenie účastníka, odpojenie z ustanoveného spojenia apod. alebo zasielanie diagnostických správ (varovania, chyby) v prípade ak nastane neočakávané správanie.
- Dátové správy, ktorými vzdialený (pripojený alebo nepripojený) účastník zasiela dáta.

3.2.1 Vytvorenie spojenia

Je možné využiť triedu NetPeer na ustanovanie p2p (účastník-účastník) siete, ale pre túto aplikáciu bol zvolený server-klient prístup, kedy herné inštancie vystupujú v roli klientov a pripájajú sa na spoločný server. Úlohou servera je po ustanovení spojenia s danými klientami preposielať správy od nich doručené iným klientom. Preto pre tento typ topológie sú využité špeciálne triedy NetServer a NetClient. Tieto triedy dedia od triedy NetPeer, ale rozšrujú ju o niektoré metódy a nastavenia.

```
NetPeerConfiguration config = new NetPeerConfiguration("xnaapp
    ");
config.Port = 14242;

NetServer server = new NetServer(config);
server.Start();
```

Fragment 3.1: Vytvorenie servera

Pre vytvorenie servera je nutné špecifikovať niekoľko údajov, aby sme zabezpečili správne pripojenie klientov k nemu, predišli nesprávnemu zasielaniu správ pri vytvorení niekoľkých inštancií aplikácií využívajúcich knižnicu lidgren a podobne. Najskôr je nutné vytvoriť konfiguráciu účastníka spojenia v ktorej zvolený reťazcom ("xmaapp") v konštruktore objektu slúži na rozlíšenie ustanovených spojení knižnicou lidgren. Rovnaký reťazec je nutné použiť aj pri vytváraní konfigurácie klienta. V ďalšom kroku sa nastavuje lokálny port na ktorom počúva server. Klienti daného serveru majú nastavené rovnaké číslo portu, aby bolo zabezpečené pripojenie na port na ktorom beží vytvorený server. V treťom kroku vytvoríme server podľa nastavenej konfigurácie a následne ho spustíme. Týmto krokom sa vytvorí nové sieťové vlákno, naviaže sa na socket a začne sledovať žiadosti o pripojenie.

3.2.2 Zasielanie a prijímanie správ

Je možné zasielať a prijímať rôzne typy správ, ale používateľ "ručne"ovplyvňuje len zasielanie, prijímanie a spracovávanie dátových typov správ a ostatné správy sú generované a spracovávané funkciami tried použitej knižnice. Zasielanie a prijímanie správ je vykonávané na strane klienta (PhysX aplikácia) v každom volaní funkcie Update(), aby aplikácia mala aktuálne dáta posielané z 2. klientskej aplikácie.
Zasielanie správ zo strany servera začína vytvorením správy metódou SendMessage(). Táto metóda vytvára novú správu alebo využíva recyklovanú správu preto nie je možné použiť na vytvorenie správy operátor new() [9].

```
NetOutgoingMessage sendMsg = server.CreateMessage();
sendMsg.Write("Hi");
sendMsg.Write(111);

server.SendMessage(sendMsg, recipient, NetDeliveryMethod.
   ReliableOrdered);
```

Fragment 3.2: Zaslanie správ - server

V kóde uvedenom vyššie zapisujeme reťazec (”Hi”) a číslo 111 (32 bitový integer) do správy. Následne je správa odoslaná použitím metódy SendMessage(), ktorej parametre sú nasledovné: správa, ktorá bude odoslaná, adresát a posledným parametrom je metóda zasielania ktorá je špecifikovaná nižšie.

Ukážka prijímanie správ na strane servera:

```
NetIncomingMessage msg;
while ((msg = server.ReadMessage()) != null)
{
    switch (msg.MessageType)
    {
        case NetIncomingMessageType.ErrorMessage:
            Console.WriteLine(msg.ReadString());
            break;
        case case NetIncomingMessageType.Data:
            Console.WriteLine("Data received");
            break;
        default:
            Console.WriteLine("Unhandled type: " + msg.
               MessageType);
            break;
    }
    server.Recycle(msg);
}
```

Fragment 3.3: Prijímanie správ - server

V kóde uvedenom vyššie sa najskôr vytvorí premenná typu NetIncomingMessage, ktorá reprezentuje typ prichádzajúcej správy. Následne prečítame správu a spracujemu ju. Toto prebiaha pokým sa na vstupe nachádzajú správy na výber. Po načítaní konkrétnej správy určí prepínač ktorá vetva programu sa vykoná. V tejto ukážke použitia rozlišujeme tri druhy správ - chybovú, dátovú a zvyšné. Pri chybovej správe sa metódou ReadString() extraktuje kópia chybového reťazca obsiahnutého v správe a vypíše do konzoly, u dátovej len uvedie, že boli prijaté dáta.

Čítanie dát zvyšuje vnútorný smerník správy, takže je možné čítať ďalšie dáta použitím Read*() metód [9].
Posledná (defaultná) možnosť do ktorej spadajú všetky ostatné typy správ (ladiace, varovné správy a iné) len vypíše uvedený reťazec a typ prijatej správy do konzoly.
Na koniec sa správa ”recykluje”, čo umožnuje znovupoužitie objektu knižnicou a tým pádom

sa vytvára menej odpadu.
Správy je možné zasielať piatimi rôznymi metódami:

- **Nespoľahlivá** - UDP sieťový protokol. Správy môžu byť stratené, doručené viackrát a v zlom poradí.
- **Nespoľahlivá usporiadaná** - Narozdiel od predchádzajúcej metódy dáta nemôžu byť doručené viackrát a nikdy nebudú doručené staršie dáta po už doručených.
- **Spoľahlivá neusporadaná** - Táto metóda zaručuje doručenie všetkých dát, ale nezaručuje správne poradie doručenia.
- **Spoľahlivá usporiadaná stratová** - Táto metóda doručovania je podobná ako Nespoľahlivá usporiadaná, ale u tejto metódy je zaručené, že budú doručené nejaké dáta.
- **Spoľahlivá usporiadaná bezstratová** - Táto metóda garantuje, že správy budú vždy doručené v poradí v akom boli odoslané.

3.3 PhysX

PhysX je šírený pod licenciou EULA[1] jedná sa teda o uzavretý kód, dostupný zdarma pre vývojárov a kupujúcich. Táto technológia má podporu na viacerých operačných systémoch a konzolách (napr. Windows 7, Mac OS X, Wii, PlayStation 3 a iných).

PhysX podporuje viacero obmedzení pohybu: pevné, generické (6D), distančné, rezolútne, sférické, valcové, bod na rovine, bod na priamke, pružiny, obmedenie typu "vozidlo"a remenicu. Ďalej podporuje geometriu typu: box, kapsula, konvexný 3D objekt (dynamický), zložený objekt, výškové pole (statické), guľu, statickú trojuholníkovú sieť reprezentujúcu 3D objekt (triangle mesh) a rovinu. Z materiálovej podpory sa jedná o statické trenie, dynamické trenia, anizotropné trenie a koeficient odskoku (angl. restitution) [4].

Jedná sa o najviac vybavený engine z porovnávaných enginov v článku [4]. Je možné použiť pevné, alebo premenlivé časové kroky. Podporuje niekoľko reprezentácií vozidiel a taktiež dynamickú trojuholníkovú sieť reprezentujúcu objekt. Podporované sú kvapaliny, ovládače postavy, látky, swept[2] geometrie a mäkké telá.

3.3.1 Operačný princíp

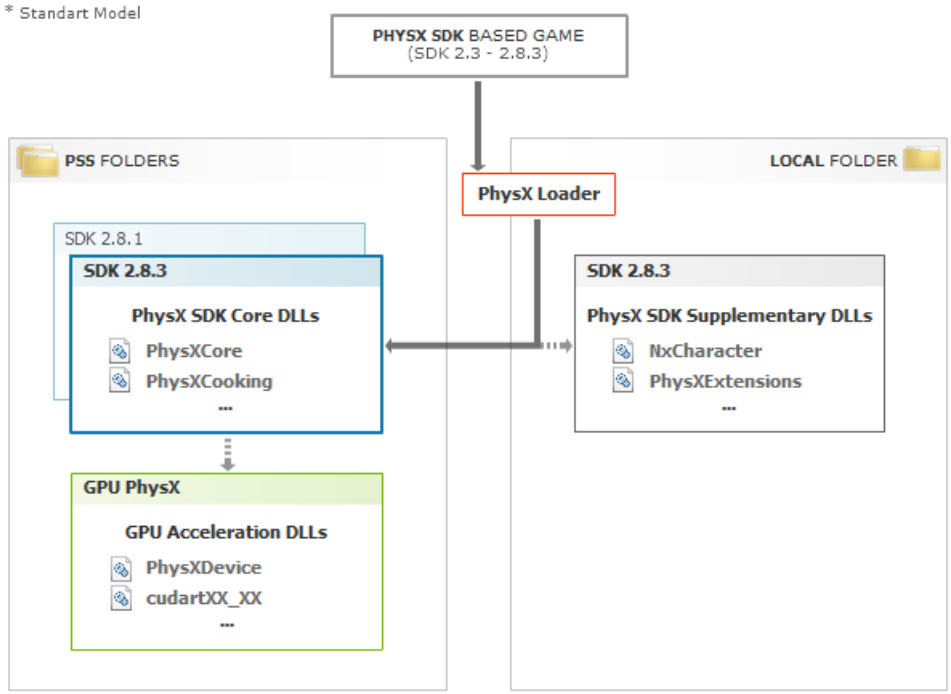
Štandardný model (validný pre hry a aplikácie používajúce PhysX SDK od verzie 2.3 po 2.8.3) Tento model odkazuje na situáciu, kedy súbory DLL(Dynamic-linked library) PhysX enginu sú uložené v PSS priečinkoch a teda inštalácia Systémového softwaru je nevyhnutná pre aplikácie, ktoré používajú tento model.

Zavádzací program PhysX, uložený v priečinku aplikácie, zavádza korespondujúce DLL súbory PhysX enginu z PSS instalačného priečinku. DLL súbory akcelerujúce GPU (PhysX a CUDA manažéri zariadenia) sa získaju z PSS distribúcie, ak sú potrebné. Doplnkové knižnice (napr. DLL ovládače postavy) v porovnaní môžu byť uložené v lokálnom priečinku.

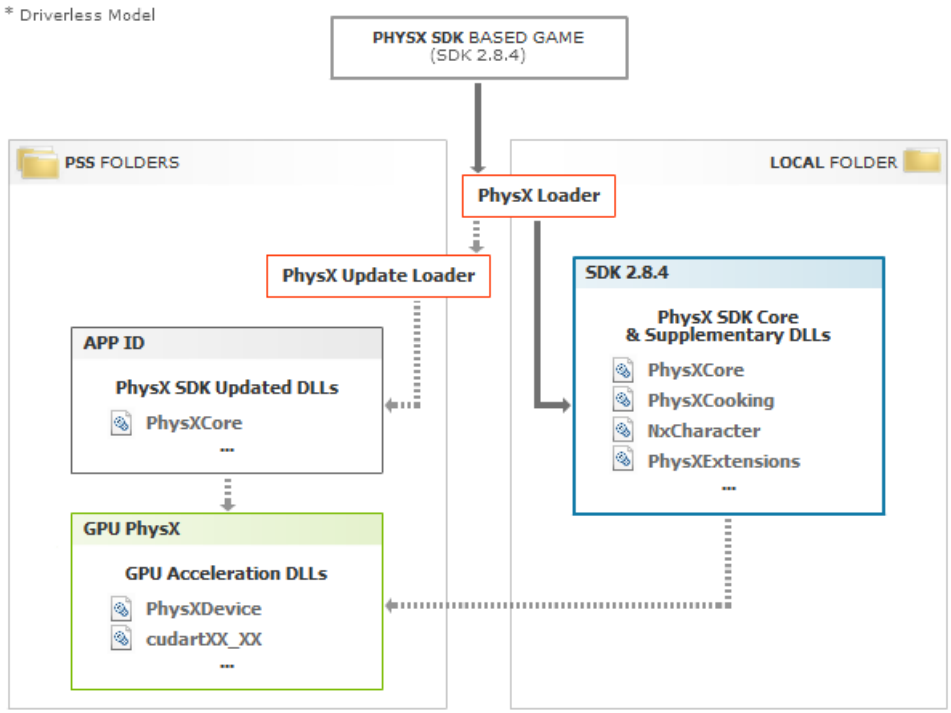
Bez-ovládačový model (validný pre hry a aplikácie používajúce PhysX SDK verzie 2.8.4 a 3.x) V tomto modele nie je inštalácia PSS nevyhnutná pre správne fungovanie aplikácie, pretože všetky potrebné súbory DLL sú uložné v lokálnom priečinku danej aplikácie.

[1] End-user license agreemet - jedná sa o kontrakt medzi osobou udeľujúcou licenciu a kupujúcim, ktorý vymedzuje jeho práva na používanie softwaru

[2] pomocou swept nástrojov sa vytvárajú geimetrie, ktoré pohybujú priesečníkom po špecifikovanej dráhe



Obrázek 3.3: Štandardný model



Obrázek 3.4: Bez-ovládačový model

Avšak ak aplikácia využíva GPU akceleráciu, stále môže vyžadovať aktualizácie na podporu nových GPU architektúr a pre zabezpečenie optimalizácie GPU výpočtov. V tomto prípade špeciálny PhysX Aktualizátor môže načítať nové DLL súbory, zabezpečené inštaláciou PSS, namiesto tých, ktoré sa nachádzajú v hernom priečinku.

PhysX systémový software (PhysX System Software - PSS)[13] Jedná sa o inštalačný balík, ktorý obsahuje firmware, software a PhysX SDK knižničné komponenty a je potrebný na inštaláciu na používateľských a developerských PC pre umožnenie korektného fungovania aplikácií založených na PhysX SDK.

PhysX systémový software bol predstavený spoločnosťou Ageia na zabezpečenie adekvátnej podpory ovládačov pre Ageia PhysX PPU(Physics Processor Unit) karty, ale čoskoro sa stal nočnou morou pre bežných používateľov, ktorí chceli hrať CPU PhysX hry, kvôli pokračujúcim chybám a inštalačným problémom.

3.3.2 Funkcionalita enginu

PhysX je navrhnuté špecificky pre hardwarovú akceleráciu na výkonných procesoroch so stovkami výpočetných jadier. Vďaka tohoto návrhu, NVIDIA GeForce GPUs poskytujú dramatické zvýšenie vo fyzikálnych výpoČtoch a dostávajú hry na nový level poskytujúci bohaté a zaujímavé herné prostredia s prvkami ako: Explózie, ktoré vytvárajú prach a vedľajšie úlomky Charaktery s komlexnou a spojenou geometriou, pre reálnejší pohyb a interakciu Veľkolepé zbrane so prepracovanými efektami Látky ktoré sa prirodzene hýbu a trhajú Hsustý dym a hmlu rozprestierajúce sa okolo objektov v pohybe

Nvidia PhysX akceleráciu podporujú všetky GeForce GPU 8. radu a vyššie s aspoň 256MB lokálnej pamäte na základnej doske a s najmenej 32 jadrami. Ak sa rozhodnete použiť podporovanú GPU ako kartu určenú na PhysX ostatné grafické karty v systéme musia tiež používať NVIDIA GPU.

3.3.3 Základy API

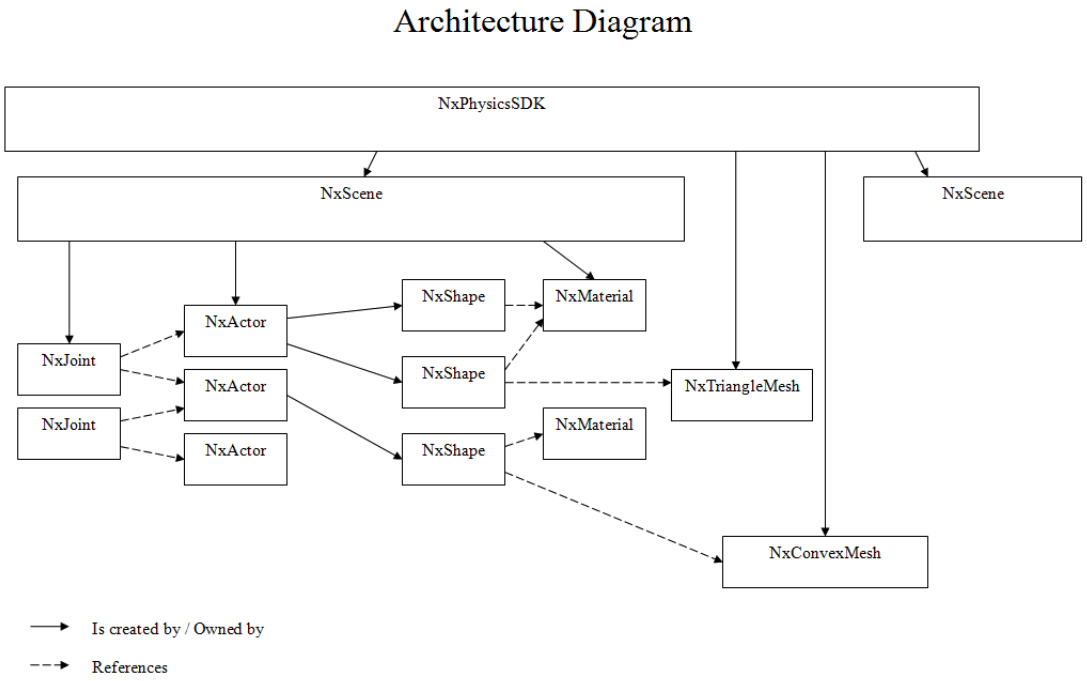
Architektúra Daný softwarový vývojový nástroj (SDK) má programovacie prostredie v ANSI C++. Vnútorne je SDK implementované ako hierarchia tried. Každá trieda, ktorá obsahuje funkcionalitu ktorá môže byť sprístupnená používateľom implementuje rozhranie. Toto rozhranie je abstraktnou bázovou triedou C++. Navyše sú tu exportované niektoré bezstavové užitočné funkcie.

Dátové typy Aby bol zabezpečený istý stupeň kompatibility, SDK využíva veľkostne špecifické dátové typy.

Triedy SDK NxVec3, NxMat33, NxMat34 a NxQuat reprezentujé 3-prvkový vektor, 3x3 maticu, 3x4 maticu a quaternion(kvaternion). Momentálne sú nakonfigurované, aby používali skalárne jednotky s pohyblivou desatinnou čiarkou a s jednoduchou presnosťou (32 bitov). A preto by používateľ mal využívať práve tieto typy pri práci s SDK [1].

Matematické triedy taktiež zabezpečujú veľký výber typových a formátových konverzných metód pre jednoduchú interakciu s vlastnými matematickými triedami. Kvôli spôsobu akým je vnútorný kód kompilovaný, by používateľ mal byť schopný pridať vlastné typovacie operátory do týchto tried bez nutnosti rekompilovať DLL súbory tohto SDK, alebo nutnosti vytvoriť ich ako závislosť svojich matematických tried.

Jednotky SDK nepotrebuje používať žiadne jednotky reálneho sveta. Avšak je dôležité definovať isté konvencie týkajúce sa jednotiek použitých pri návrhu aplikačnej stránky. SDK používa bezrozmerné jednotky na meranie troch základných veličín: hmoty, dĺžky a času. Tieto veličiny je možné definovať v ľubovoľných požadovaných jednotkách (napr. meter



Obrázek 3.5: Diagam architektúry PhysX SDK

ako základnú jednotku dĺžky atď.). Jednotky odvodených veličín sa získajú z jednotiek základných použitých veličín (napr. rýchlosť = vzdialenosť/čas -¿ meter/sekunda) [1].

PhysX SDK používa len čísla s pohyblivou desatinnou čiarkou a jednoduchou presnosťou a ako u každého numerického softwaru je dôležité udržať čísla v rozsahu relatívne veľkej presnosti. Táto presnosť závisí od konkrétnych potrieb týkajúcich sa presnosti simulácie.

3.3.4 Dynamika scény a činitelia

Simulácia sa vo PhysX odohráva v scénach (odvodených z treidy NxScene). Je možné vytvoriť niekoľko scén ktoré predstavujú kontainer pre simulovaných aktérov, spojenia a efektorov. Tieto scény je možné naplniť množstvom objektov a následne paralelne simulovať. Kedže je simulácia dvoch scén kompletne oddelená je nutné pre požadovanú interakciu jednotlivých objektov vytvorených v rozdielnych scénach pridať vonkajšie sily, ktoré vedú k istej komunikácií medzi scénami [1].

Scény sú v podstate priestorovo neobmedzené a umožnujú viacero praktických funkcionalít ako sú napr. konštantné gravitačné pole, ktoré ovplyvňuje všetky objekty, umožňovanie a znemožňovanie detekcie kolízií atď.

Väčsina simulácií využije len jednu scénu. Praktickým využitím viacerých scén je klient/ server implementácia hry pre viacerých hráčov. Teda jeden proces vykonáva simuláciu servera a lokálneho klienta, vytvorením dvoch scén. Klientská scéna simuluje len bezprostredné, vnímateľné okolie klientského avatara, zatiaľ čo server bude simulovať celý svet.

Činitelia (actors) sú základnou súčasťou simulácií. Môže sa jednať o statické objekty reprezentujúce budovy, zem, steny a pod. a tiež dynamické pevné objekty ako autá, rôzne stroje, ľudske telo atď. Jeden z dôležitých aspektov činiteľov je, že môžu mať pridelené rôzne tvary. Detekcia kolízií zabezpečuje, že tvar jedného sa neprekryje s tvatom druhého činiteľa [1].

Kapitola 4

Návrh a implementácia aplikácie

4.1 Pozadie

Aby hra pôsobila realistiky je vhodné vytvoriť pozadie, ktoré pokrýva všetky objekty v scéne a tak vytvára v hráčovi pocit, že sa nachádza v nekonečnom prostredí. Taktiež rozdielne pozadie umožnuje hráčovi sa lepšie orientovať v scéne. V našom prípade pozadie hry vystihuje celkovú krajinu obklopujúcu hráča zo strán a tiež oblohu. Tento efekt je vzhľadom na efektivitu vykreslovania, kedy by bolo nutné vykreslovať enormne veľké objekty vo veľkej vzidalenosti (nehovoriac o nízkej úrovni ich detailu) dosiahnutý vytvorením pevného obalu, ktorý zakryje celú scénu.

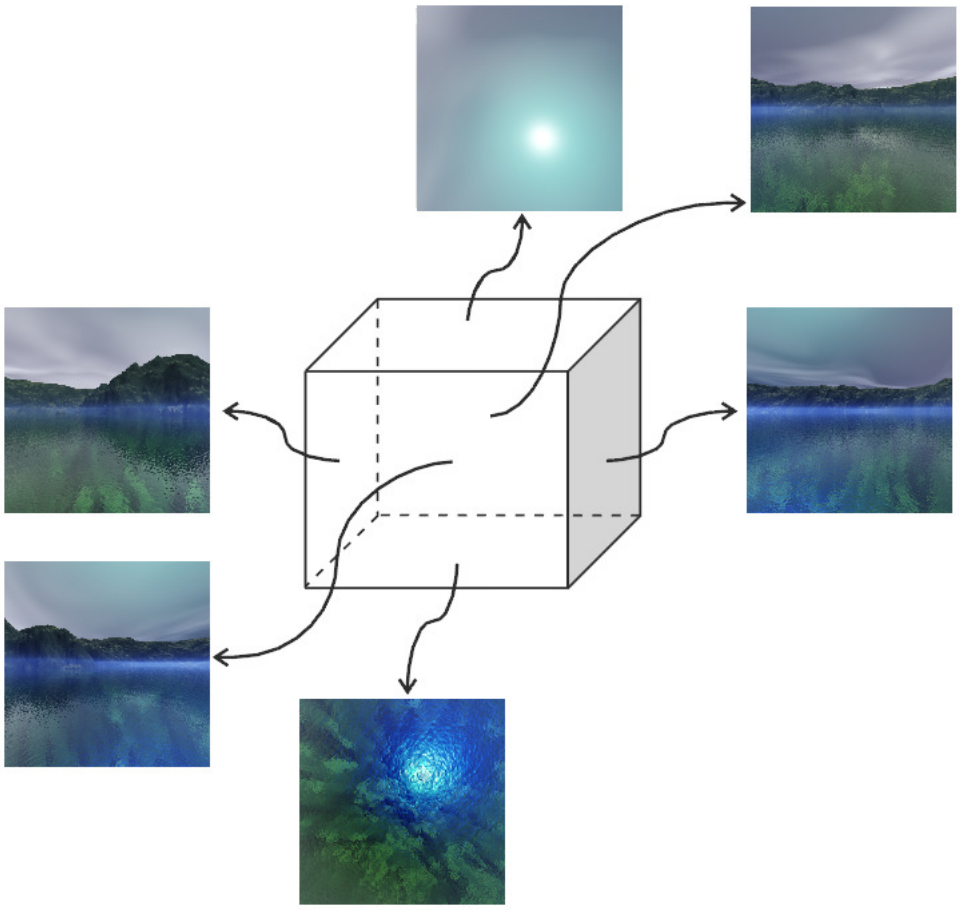
Za týmto účelom je použitý box (škatuľa, kocka) nazývaný *skybox*. Krajina vykreslovaná okolo hráča je uložená v textúrach, ktoré sú namapované na skybox. Na dosiahnutie efektu nekonečného horizontu sa stred boxu upne na kameru, alebo postavu a tak je hráč neustále v strede boxu, lebo pri jeho pohybe sa hýbe aj skybox.

Skybox je teda vytvorený ako box obsahujúci 6 plôch a každá z nich má inú textúru. Je nutné venovať špeciálnu starosť aby prechod textúr bol nerozoznateľný, lebo inak bodú viditeľné roky boxu a ilúzia vzdialeného horizontu sa rozplynie [5]. Pretože sa nachádzame vo vnútri boxu sú všetky plochy nasmerované dovnútra. Výhodou skyboxu je jeho jednoduché vytvorenie (má len 12 trojuholníkov) naopak nevýhodou je statická stránka boxu a tiež potreba vytvorenia plynulých prechodov medzi susediacimi textúrami.

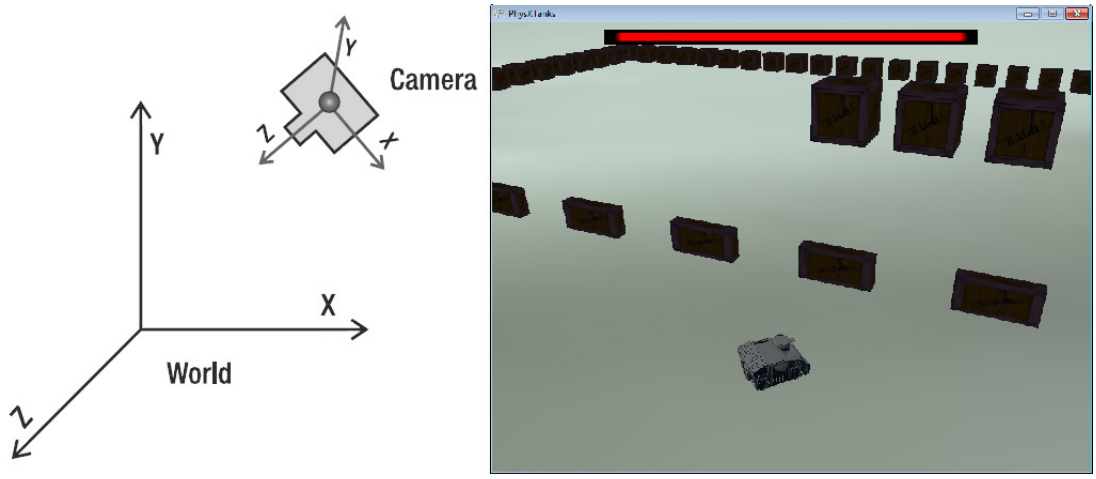
4.2 Kamera

Esenciálnou súčasťou scény je kamera. Podľa druhu herného žánru sa používajú rôzne druhy kamery v scéne - pohľad tretej osoby (kamera umiestnená za hráčom - 3rd person view), pohľad z prvej osoby kedy je kamera v podstate súčasťou herného charakteru a iné. Pre sieťovú hru je zvolená kamera vytvárajúca pohľad tretej osoby a pre simuláciu je určená voľne sa pohybujúca kamera.

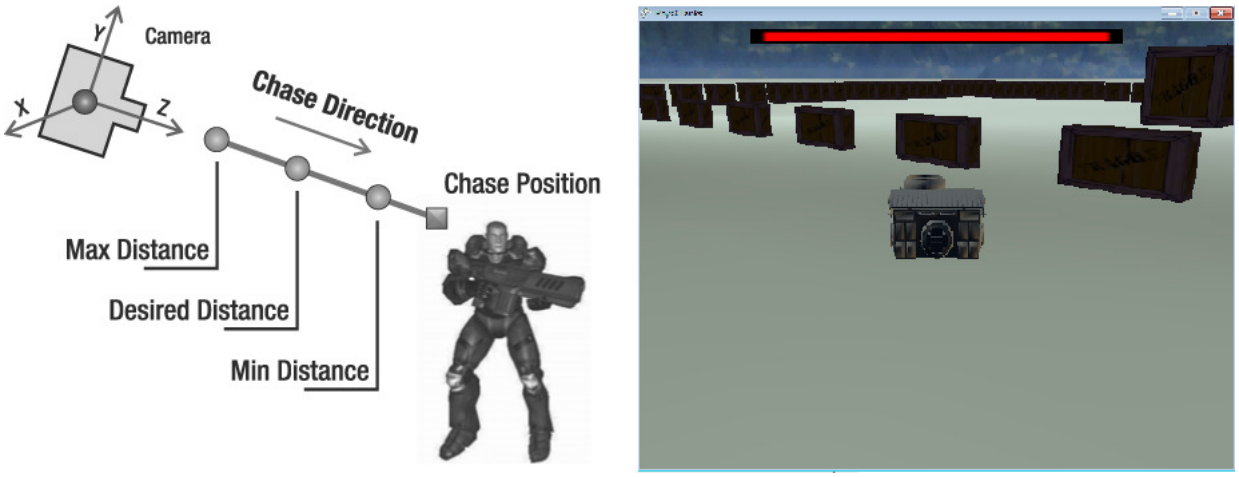
Kamera je vytvorená ako samostatná trieda v ktorej sa spracováva jej pohľad (camera view) a projekčné matice definujúce rozsah pohľadu. Jedná sa o jej *frustum* (zrezaný ihlan) a zobrazované sú len objekty ktoré sa nachádzajú v ňom [5]. V XNA sa tvorí pohľad kamery pomocou 3 vektorov určujúcich - polohu kamery v scéne, miesto na ktoré smeruje a vektoru, ktorý určuje jej natočenie (cameraUpVector). Následne sa vytvorí matica projekcie ktorá má 4 parametre: veľkosť poľa pohladu (0 až 180°), *aspect ratio*(pomer používaný na namapovanie 3D projekcie na koordináty obrazovky), vzdialenosť blízkej orezávacej plochy



Obrázek 4.1: Skybox - pozadie hry



Obrázek 4.2: Kamera - voľne umiestnená v priestore



Obrázek 4.3: Kamera - pohľad tretej osoby

a vzdialenosť vzdialenej orezávacej plochy. Hodnoty jednotlivých parametrov sú nastavené vzhľadom na prívetivosť vizuálnej stránky hry.

Kamera vytvárajúca pohľad tretej osoby je na začiatku vytvorená ako voľne sa pohybujúca kamera, ale v každom *Update* je jej poloha menaná vzhľadom na polohou a natočenie tanku. Kedže v jej smerovaní je zahrnutá aj aktuálna rotácia tanku problém nastáva v momente keď tank rotuje okolo vlastnej osi (prechádzajúcej pozdĺžne cez neho) vo vzduchu. Kamera vtedy nepríjemne rotuje, čo môže dezorientovať hráča, ale táto implementácia bola ponechaná kvôli možnosti "ustáliť"pozíciu tanku, ktorá je popísaná v časti ovládania aplikácie. Vzdialenosť kamery za tankom je pevne daná a nemenná (subjektívne najlepšie vyzerajúca možnosť).

4.3 Spracovanie vstupu

Spracovanie vstupu je implementované pomocou triedy *InputManager()* a rozhraní *KeyBoardListener* a *MouseListener*. Tieto rozhrania sú implementované triedami Game1 a Camera. V triede InputManager sú pri inicializácií vytvorené spomínané rozhrania a premenné zachycujúce počiatočný stav klávesnice a myši. V metóde Game1.Update() ktorá je pravidelne voľaná pred každým vykreslovaním (volaním metódy Draw()) sa volá metóda InputManager.Update(). V nej sa v každom priechode porovnáva predchádzajúci a súčasný stav klávesnice a myši. Na základe vyhodnotenia týchto stavov sa volajú nižšie uvedené metódy implementované v spomínaných triedach v ktorých sa vykonáva kód reagujúci na vstup.

Pri implementácií KeyboardListenera musí trieda implementovať tieto metódy:

- **KeyPressed** - klávesa stlačená v ľubovoľnom stave behu aplikácie
- **KeyPressedRunning** - klávesa stlačená v *Gamestate.Running*
- **KeyHoldedRunning** - klávesa držaná v *Gamestate.Running*
- **KeyPressedInMenu** - klávesa stlačená v *Gamestate.InMenu*

Pri implementácií MouseListenera musí trieda implementovať tieto metódy:

- **MouseMoved** - pohyb myši
- **MouseLeftButtonPressed** - stlačenie ľavého tlačidla myši
- **MouseRightButtonPressed** - stlačenie pravého tlačidla myši

4.4 Herný charakter a streľba

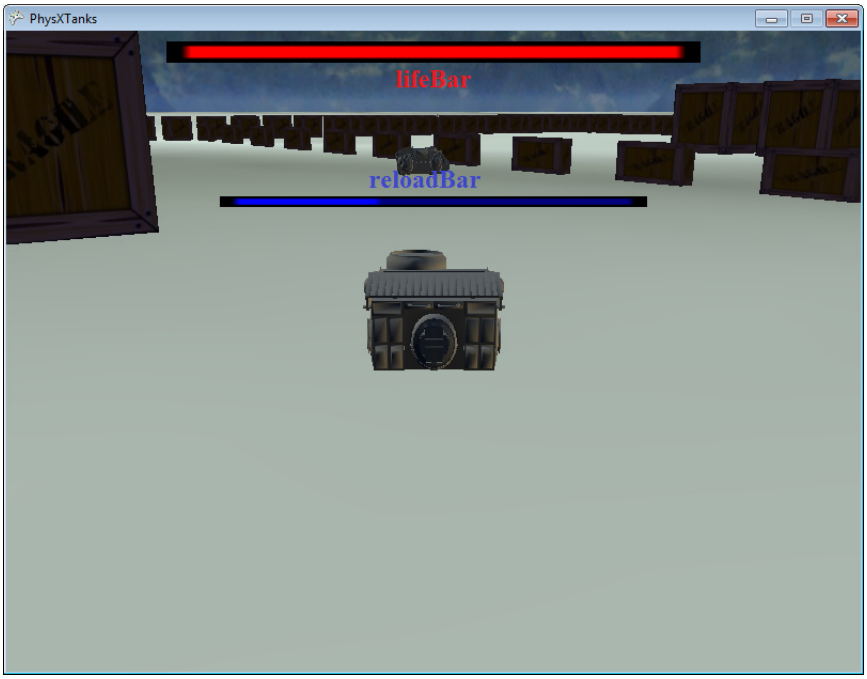
Tank ktorý sa vytvára v simulačnom móde a taktiež hernom móde je entita triedy *Tank*. V reprezentácií PhysX sa jedná o jednoduchý charakter (angl. actor) tvaru box na ktorý je namapovaný model tanku. Je vytváraný na špecifikovanej pozícií podľa charakteru spúšťanej aplikácie (viď kód aplikácie) so zadanou globálnou pozíciou a počiatočný stavom 100 *hitPoints* (životy).

Obrázek 4.4: Streľba tanku

Streľba tanku je implementovaná s prihliadnutím na charakter aplikácie, kedy je žiaduce aby letiaci projektil bol reprezentovaný pevným objektom, ktorý môže kolidovať s inými objektami. Jedná sa o objekt typu guľa, ktorá sa vytvára v strede tanku a v momente jej vytvorenia je na ňu aplikovaná sila smerujúca zo stredu cez prednú časť tanku. Veľkosť sily je statická, je smer závisí od natočenie tanku. Na streľu nepôsobí gravitácia (jednoduchšia predikcia pre hráča) a kedže je vytváraná vo vnútri tanku je na určitý čas "nehmotná"(nekoliduje sa žiadnymi objektami) aby mohla byť vystrelená predpokladaným spôsobom bez okamžitej kolízie s telom tanku. Po vystrelení je každá steľa pridaná do zoznamu strieľ *bulletsList* a je vytvorená premenná určujúca čas, kedy je kolízia stely s ostatnými objektami znovu aktívna. V simulačnej časti je počet súčasne vytvorených strieľ obmedzený na 10 a v hernej časti len na 1 (kvôli nutnosti prenášať údaje o jej polohe v každom volaní metódy Update()). Pri vytvorení viacerých strieľ sa odstráni najskôr vytvorená a nahradí práve vytvorenou.

4.5 Zvuky a ukazatele stavu

V hernej časti aplikácie sú využité zvuky a ukazatele stavu životov a stavu nabíjania (*lifeBar* a *reloadBar*). Použité sú tri jednoduché zvukové efekty reprezentujúce výstreľ, nabíjanie, explóziu a pri spustení hry sa spustí hudba dotvárajúca herné pozadie.
Ukazatele stavu sa vykresľujú nad charakterom hráča a reprezentujú aktuálny stav (progres). Ukazateľ životov sa nachádza v hornej časti obrazovky a reprezentuje aktuálnu hodnotu životov tanku. Jeho hodnoty sú v rozptyle 0-100 pričom hodnota nula reprezentuje zničenie tanku. Ukazateľ stavu nabíjania je vykreslovaný vždy po vystrelení projektilu a reprezentuje aktuálny progres nabíjania vďaka ktorému je hráč schopný odhadnúť čas, možnej opätovnej streľby.



Obrázek 4.5: Ukazatele stavu životov a nabíjania

4.6 Detekcia kolízií

Detekcia kolízií je pre fyzikálnu aplikáciu priam nevyhnutná, inak by všetky objekty jednoducho prepadli cez rovivu (podlahu). Implementácia detekcie kolízií (vrátane spojitej detekcie kolízií) je realizovaná vrámci výpočtov PhysX enginu. Problém nastal kvôli použitému wrapperu, ktorý vo využitej verzii neobsahuje udalosť *SimulationEventCallback.OnContact* ktorá sa vyvolá v prípade kontaktu dynamického charakteru. Preto je implementovaná detekcia kolízií v rámci možností XNA. Táto detekcia je implementovaná v metóde *CollisionWithEnemy* a v súčastnej verzii aplikácie sa testuje kolízia medzi zoznamom projektilov a zoznamom "nepriateľských"tankov. Do zoznamu nepriateľských tankov je možné pridať aj tank hráča, ale táto možnosť nie je využitá, kvôli hrateľnosti (hráč nemôže projektilmi zabiť sám seba). Detekcia je kontrolovaná priechodom cez jednotlivé časti tvarov a ich častí konkrétnych objektov a porovnávania prekrývania sa sfér týchto objektov. Po zistení kolízie s nepriateľským projektilom sú tanku ubrané životy a ak sa tank dostane na hodnotu 0 životov je zničený.

Kapitola 5

Meranie záťaže CPU a GPU

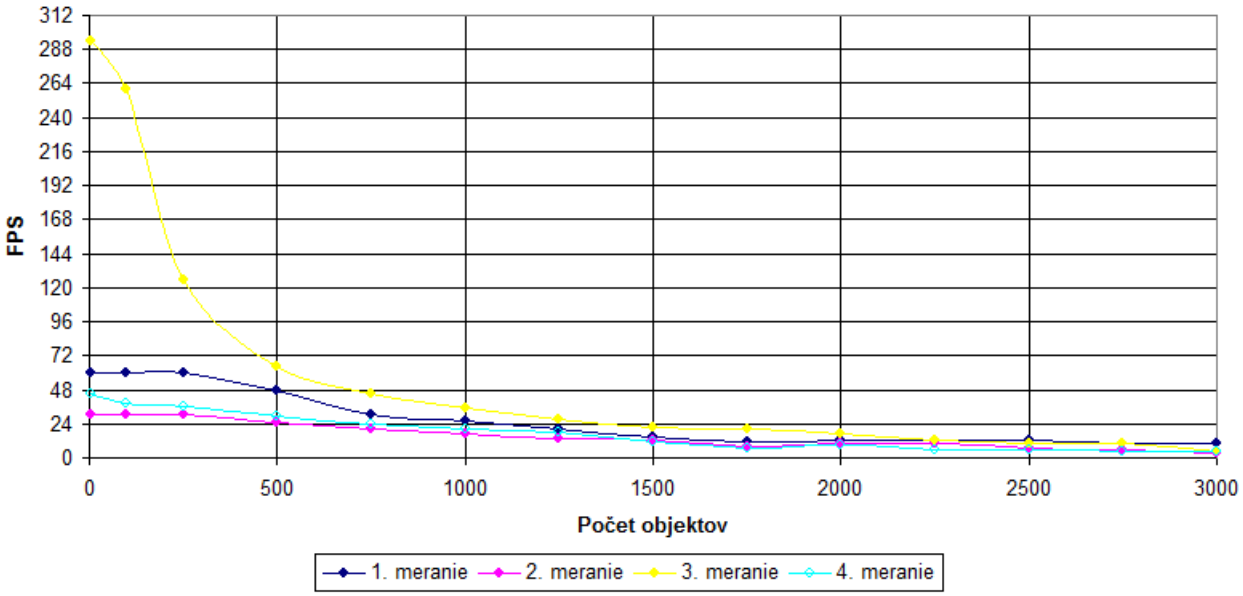
Parametre testovaného systému:
Operačný systém: Windows 7 Professional, 32-bit (Service Pack 1)
Grafická karta: GeForce 9300M GS
DirectX verzia 11.0
CUDA Jadrá: 8
Celková dostupná grafická pamäť: 1533 MB
Dedikovaná video pamäť: 256 MB GDDR3
Tieňovacia (angl. shader) systémová pamäť: 1277 MB

3D Nastavenia
NVCUDA.DLL 8.17.13.0623 NVIDIA CUDA 5.0.1 driver
PhysX 09.12.0604 NVIDIA PhysX

5.1 Záťažový test vykresľovania

Prvý test je zameraný na porovnanie výkonu vykresľovania veľkého počtu objektov. Jedná sa o pevné objekty, u simulácie ktorých sa nevyužíva hardwarová akcelerácia. Vykonané boli 4 testy, v každom z nich je vytvorených 5, 100, 250 a ďalej s prírastkom 250 až po 3000 objektov. Meră sa počet vykreslovaných snímkov za sekundu vzhľadom na počet vytvorených objektov. V prvých dvoch testoch je zapnutá vertikálna synchronizácia (obmedovanie frekvencie vykreslovania vzhľadom na nastavenú frekvenciu monitora) v ďalších dvoch testoch je naopak vypnutá. Ďalej v 1. a 3. teste je zamerané okno aplikácie (angl. focused) a v 2. a 4. nie je. Hodnoty sú namerané v stave, v ktorom sú vygenerované telesá v kľude, kvôli väčšej stabilite vykresľovania.

Na grafe 5.1 je možné vidieť, že 2. a 4. merania majú nižšie hodnoty vykeslovania snímkov za sekundu, čo je výhodné pri prepnutí sa do iného okna, kedy daná aplikácia menej zaťažuje GPU.



Obrázek 5.1: Počet snímkov vykreslovaných za sekundu pri generovaní objektov

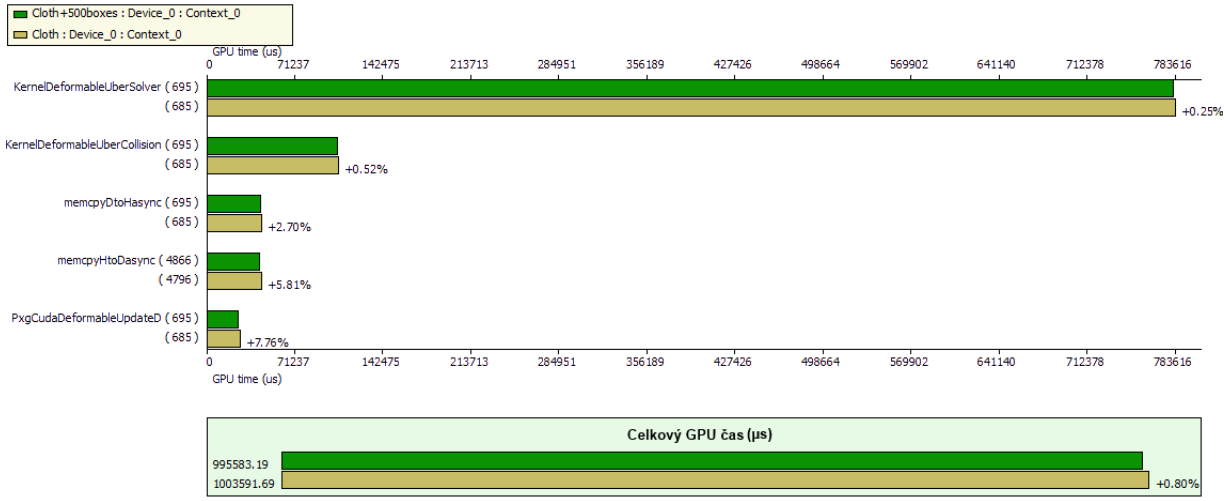
5.2 Testy pri využití hardwarovej akcelerácie

Tieto testy porovnávajú aplikácie v ktorých je implementovaná látka (cloth) a teda je aplikovaná hardwarová akcelerácia po ich spustení. Na zistenie konkrétnych dát je použitý nástroj Compute Visual Profiler on spoločnosti Nvidia, ktorý je schopný spustiť CUDA alebo OpenCL program a profilovať rôzne aspekty bežiaceho programu. Pri získavaní údajov sa vytvorí sekcia do ktorej sú uložené spriemerované dáta z 5 behov aplikácie. V tomto meraní je sledovaných 5 metód GPU, jedná sa buď o "memcpy*"(kopírovanie pamäte), alebo o názov GPU jadra. Sledované boli tieto metódy:

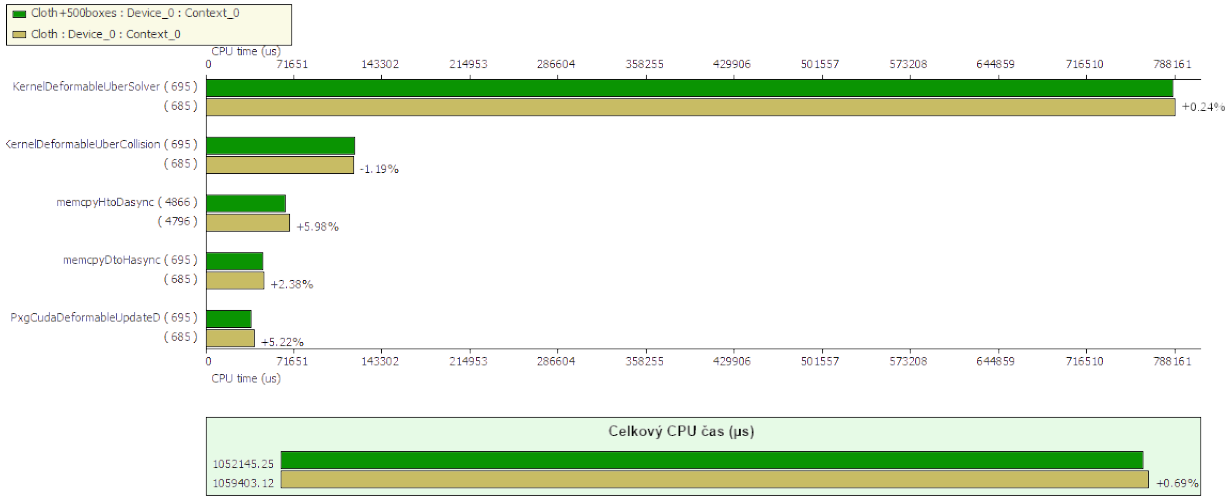
- `KernelDeformableUberSolver` - jadro riešiaca reakcie deformovateľných častí (objektov)
- `KernelDeformableUberCollison` - jadro riešiace kolízie deformovateľných častí (objektov)
- `PxgCudaDeformableUpdateD` - Cuda jadro aktualizujúce deformovateľné časti
- `memcpyDtoHasync` - asynchrónne kopírovanie pamäte typu zariadenie - hostiteľ
- `memcpyHtoDasync` - asynchrónne kopírovanie pamäte typu hostiteľ - zariadenie

V jednotlivých grafoch je uvedené porovnanie dvojice sekcií meraní[1] a sledovaným aspektom je celkový čas vykonávania uvedeych metód so zameraním na GPU, alebo CPU. Čas je uvedený v mikrosekundách a v zátvorkách za každou metódou je uvedený počet volaní danej metódy.

[1] 3Cloth + 4balls - 3 látky interagujúce so 4 guľami, Cloth - jedna látka na ktorú pôsobí jedna guľa, Cloth+500boxes - jedna látka interagujúca s jednou guľou v prostredí s postupným vytváraním objektov typu box (max. 500 kusov)



Obrázek 5.2: Porovnanie GPU záťaže v 1. meraní



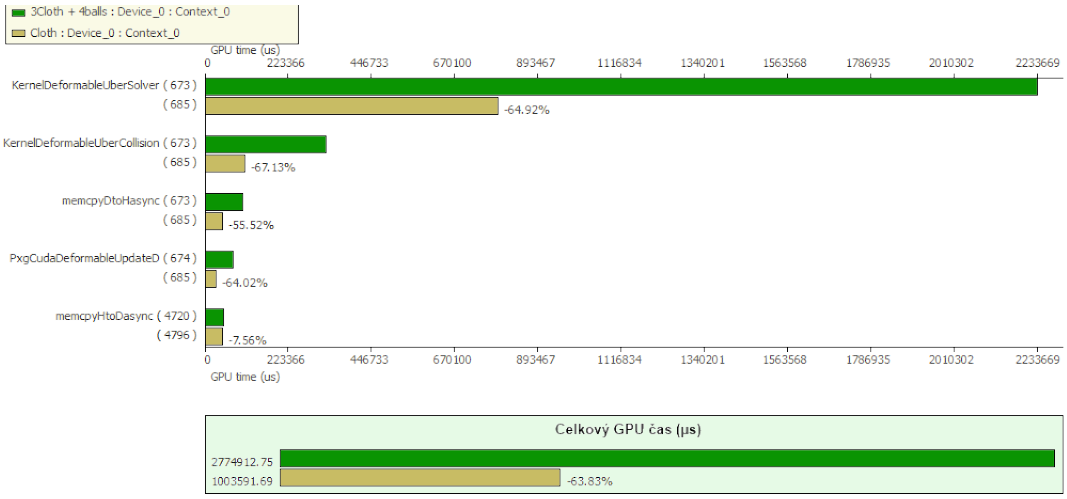
Obrázek 5.3: Porovnanie CPU záťaže v 1. meraní

V 1. meraní je porovnaný beh aplikácie s vytvoreným jedným objektom typu látka a s guľou ktorá na ňu padá a aplikácie s rovnakými objektomami s rozšírením o postupné vytváranie pevných objektov typu box (krabica).

Na grafoch 5.2 a 5.3 je možné vidieť je malé rozdiely trvania vykonania operácií jednotlivých metód, s celkovým rozdielom len 0,8% u GPU, resp. 0,69% u CPU.

V 2. meraní sú v jednej aplikácií vytvorené tri látky ktoré interagujú so štyrmi guľami (pre zvýšenie výpočetnej záťaže) a druhá aplikácia pozostáva s jedného objektu typu látka na ktorú pôsobí jedna guľa.

V grafe 5.4 je možné vidieť veľký rozdiel času potrebného na výpočet v GPU pri podobnom počte volaní jednotlivých metód. Týmto testom je teda potvrdená hardwarová akcelerácia látok v systéme PhysX.



Obrázek 5.4: Porovnanie GPU záťaže v 2. meraní

Kapitola 6

Záver

Cieľom tejto bakalárskej práce bolo vytvoriť fyzikálnu simuláciu v 3D scéne s využitím PhysX enginu. Práca implementuje sieťovú hru PhysXTanks s využitím frameworku XNA, wrapperu na spojenie týchto dvoch technológií PhysX.Net a sieťovej knižnice Lidgren.Network. Herná aplikácia bola testovaná lokálne na OS Windows 7 s grafickou kartou GeForce 9300M GS a bolo vykonaných aj niekoľko testov simulácií zameraných na zaťaženie systému a zistenie toho, aká časť simulácie je akcelerovaná hardwarom (GPU). Hra je pre dvoch hráčov, ktorí sa snažia zničiť nepriateľský tank skôr, ako sa podarí protihráčovi zničiť ich. Hra má zvukový doprovod symbolizujúci streľbu, nabíjanie, zásah protihráča a hráč tiež vidí svoj aktuálny stav (životy) a priebeh nabíjania.

Ďalší vývoj môže spočívať v rozšírení hry pre viacerých hráčov, pridaním nových modelov, kvapalín, "mäkkých objektov"(angl. soft body) a tiež vytvorením realistickej reprezentácie tanku. Použitý je taktiež len základný osvetľovací model a pomerne jednoduché textúry, takže je možnosť aplikovať vylepšienia aj u týchto stránok grafickej aplikácie.

Napriek tomu, že Havok stále konkuruje medzi AAA titulami PhysX je mu už zdatnou konkurenciou ktorú využíva čím ďalej hráčov a vývojárov. V dnešnej dobe je používaný viac ako 150 hrami a využívaná ho viac ako 10 000 vývojárov.

Zdrojové kódy, binárne súbory, Readme a dokumentácia sú dostúpné na stránke: http://merlin.fit.vutbr.cz/wiki/index.php/Graphics_Projects_2013#Physical_Simulation_in_3D_Scene_UsingPhysX

Literatura

[1] NVIDIA PhysX SDK 2.8. 2008, compiled HTML Help file.

[2] Banks, J.; Carson, J.; Nelson, B.; aj.: *Discrete-Event System Simulation, 3rd Edition.* Prentice Hall, 2001, ISBN 0-13-088702-1.

[3] Bernier, Y. W.: Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization, srpen 2009, [Online], [cit. 2013-05-10].

[4] Boeing, A.; Bräunl, T.: Evaluation of real-time physics simulation systems. In *GRAPHITE '07 Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-912-8, s. 281–288, conference.

[5] Evangelista, B. P.; Lobao, A. S.; Grootjans, R.; aj.: *Beginning XNA 3.0 Game Programming: From Novice to Professional (Expert's Voice in XNA)*. Apress, 2009, ISBN 978-1430218173, 1 edition.

[6] Gottschalka, S.; Lin, M.; Manocha, D.: OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In *SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM, 1996, ISBN ISBN:0-89791-746-4, s. 171–180, conference.

[7] Grootjans, R.: *XNA 3.0 Game Programming Recipes: A Problem-Solution Approach (Expert's Voice in XNA)*. Apress, 2009, ISBN 978-1430218555, 1 edition.

[8] Kanter, D.: PhysX87: Software Deficiency. http://www.realworldtech.com/physx87/5/, červenec 2010, [Online], [cit. 2013-05-10].

[9] Lidgren, M.: Basics - The basics of using the library. http://code.google.com/p/lidgren-network-gen3/wiki/Basics, květen 2011, [Online], [cit. 2013-05-13].

[10] Redon, S.; Kheddar; A.; aj.: *Fast Continuous Collision Detection between Rigid Bodies*. květen 2003, str. 279–287, computer Graphics Forum.

[11] Smith, R. D.: Simulation Article. In *Encyclopedia of Computer Science 4th Edition, červen 2000*, 1998, str. 6.

[12] Zogrim: Popular Physics Engines comparison: PhysX, Havok and ODE. http://physxinfo.com/articles/?page_id=154, červenec 2009, [Online], [cit. 2013-05-13].

[13] Zogrim: PhysX System Software. http://physxinfo.com/wiki/PhysX_System_Software, prosinec 2012, [Online], [cit. 2013-05-13].

Příloha A

Obsah CD

Zdrojové súbory
Manuál k aplikácií (Readme.txt)
Plagát
Spustiteľný program
Zdrojové texty dokumentácie
Dokumentácia

Příloha B

Manual

Po spustení sieťovej hry, alebo simulácie je možné použiť tieto klávesy na ovládanie aplikácie:

- Esc - ukončenie aplikácie
- P - prerušenie hry (pauza)

Kamera

- W - pohyb kamery vpred
- S - pohyb kamery vzad
- A - pohyb kamery doľava
- D - pohyb kamery doprava

Pohyb tanku a streľba

- Up - pohyb tanku vpred
- Down - pohyb tanku vzad
- Left - otáčanie tanku smerom doľava
- Right - otáčanie tanku smerom doprava
- T - ustálenie polohy tanku
- Space - streľba z tanku

Pôsobenie silou na objekty

- J - pôsobenie silou v smere osi x
- L - pôsobenie silou proti smeru osi x (-x)
- U - pôsobenie silou v smere osi y
- M - pôsobenie silou proti smeru osi y (-y)

- I - pôsobenie silou v smere osi z
- K - pôsobenie silou proti smeru osi z (-z)
- N - vytváranie boxov (pre potreby simulácie)
- R - zvolenie iného objektu
- - vypnutie/zapnutie vertikálnej synchronizácie
- Y - vypnutie/zapnutie zvukov

Pohyb kamerou je možný len v simulácií a taktiež zvolenie iného objektu a pôsobenie silou na objekty je možné len v simulácií, nie sieťovej hre.