VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTERAKTIVNÍ OPENGL DEMO

INTERACTIVE OPENGL DEMO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PATRIK CHUKIR**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MILÉT,**
SUPERVISOR

BRNO 2017

Abstrakt

Práce se zabývá realnou implementaci jednoduchého dema v prostředí OpenGL.

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

OpenGL, 3D hra,

Keywords

OpenGL 3D game

Citace

CHUKIR, Patrik. *Interaktivní OpenGL demo*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Milét Tomáš.

Interaktivní OpenGL demo

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Patrik Chukir
15. května 2017

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tvorbou počítačových her se zabývá celé odvětví průmyslu, pracují na nich mnoha členné vývojářské týmy, i tak se vyvíjí měsíce i roky. Ale co vše se musí vlastně vytvořit, abychom se mohli projít ve virtuální 3D scéně? Tato práce popisuje implementaci jednoduché hry, dema, ovšem veškerá implementace bude směřována k případnému pozdějšímu rozšíření do plné hry. Vše bude řešeno buď vlastní implementací nebo *freeware/open-source* knihovnami. Práce se bude zabývat realizací základního osvětlení, načtení a vykreslení modelů, interakcí s uživatelem, dále vyřešením kolizí pomocí externí knihovny. K těmto cílům budou použity knihovny založené na *OpenGL*, jmenovitě *Assimp*[2] pro načítání, *Bullet physic engine*[1] pro zachytávání kolizí a fyzikální model a *irrKlang*[**?**] pro ozvučení a knihovny *glew[**?**], glfw[**?**], glm[**?**]* pro základní práci s grafikou.

Kapitola 2

Teorie

V této kapitole bude popsána netriviální teorie později použita v aplikaci. Zaměří se primálně na teorii mnou implementovanou, nikoliv na teorii obslouženou knihovnami třetích stran. Jako hlavní zdroj pro tuto kapitolu slouží kniha Moderní počítačová grafika[**?**].

2.1 Phogův osvětlovací model

Phongů osvětlovací model předpokládá, že většina světla nedopadá do kamery přímo, ale odrazem od objektů. Toto odražené světlo dělí na tři složky *Ambientní*, *Difuzní* a *spekulární*. Kdy Ambietní složka představuje světlo rovnoměrně rozptýlené v prostoru, tedy osvětluje všechny fragmenty stejně. Difuzní složka představuje světlo přímo od zdroje dopadající na fragment. Intenzita osvětlení difuzní složkou se odvíjí od úhlu dopadu, čím blíže k normále tím je fragment více osvětlen. Poslední složka, spekulární neboli zracodlová představuje světlo odrážející od fragmentu, tedy její viditelnost se mění na základě polohy kamery tzn. jak moc se světlo odráží směrem ke kameře. Phongův model v tomto případě předpokládá ideální odraz, tedy úhel dopadu se rovná úhlu odrazu. Intezitu celkového ovětlení fragmentu můžem obecně spočítat vzorcem:

$$I_V = I_A r_a + \sum_{k=1}^{M} I_{L_k}[r_s(\vec{v} \cdot \vec{r_k})^h + r_d(\vec{l_k} \cdot \vec{n})][3] \tag{2.1}$$

Vzorec 2.1 je obecný pro M zdrojů světla. Já v aplikaci použiji jako zdroj světla jen slunce, takže můžeme vzorec zjednodušit na:

$$I_V = I_A r_a + r_s(\vec{v} \cdot \vec{r_k})^h + r_d(\vec{l_k} \cdot \vec{n}) \tag{2.2}$$

V obou vzorcích I_A je intenzita *ambientní* složky a r_a její barva. V první vzorci se pak sčítají všechny *difuzní* a *spekulární* složky všech zdrojů světla. V případě této práce se suma redukuje jen na jeden vzorek.

2.2 Shadow depth map //TODO

Shadow depth map, česky stínová paměť hloubky[3], je algoritmus na výpočet vlastních i vržených stínů hmotných těles. Vyznačuje se vysokou rychlostí výpočtu a nízkou kvalitou stínů, vyšší paměťovou náročností pro více zdrojů světla. Algoritmus spočívá ve vytvoření hloubkové mapy, podobně jako při řešení viditelnosti pomocí z-buffer algoritmu, kdy je

v mapě uložená vzdálenost nejbližšího bodu ke zdroji světla. Při vykreslování se porovná hloubka daného bodu s hodnotou v mapě, pokud je vyšší není ve stínu jinak je. V důsledků použití rastrové mapy se u toho algoritmu projevují všechny problémy spojené s mapováním textur, jako aliasing, nízké rozlišení mapy. Dále může v důsledku nepřesnosti rasterizace vzniknout jemná odchylka a potom hloubka bodu vyjde o něco větší než při zápisu do hloubkové mapy a začne vrhat stín sám na sebe. Algoritmus této metody se dá zapsat jako:

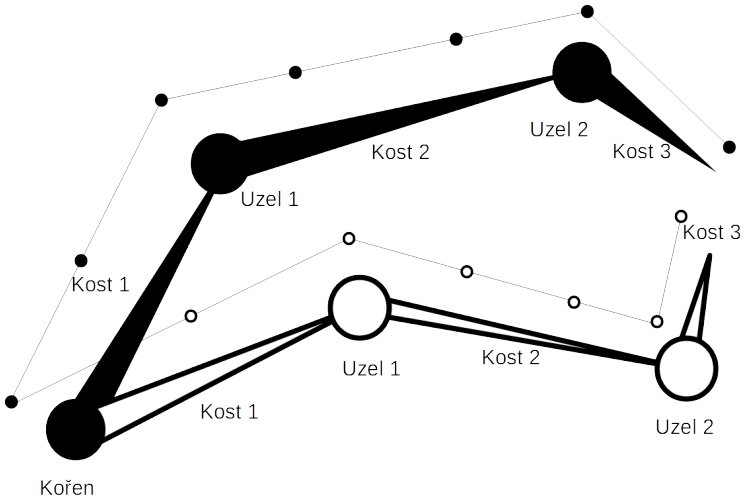
1 pro každý světelný zdroj:
   - vykresli scénu z pohledu zdroje;
   - pro každý pixel:
     - nejnižší z-souřadnici ulož do mapy, "místo barvy";

2 vykresli scénu normálně:
   - pro každý fragment:
     - vec3 sum = (0,0,0); výsledná osvětlenost
     - pro každou Shadow mapu M_i:
       - L-coord = souřadnice fragmentu v soustavě daného světla;
       - if(M_i.get(L-coord.xy)<z):
         - sum += 0.5 * světlo; //lezi ve stinu
       - else:
         - sum += světlo; // je osvětlen
     - sum /= počet světelných zdrojů;12

2.3 Skeleton animation

Skeleton animation je metoda inspirovaná stavbou lidského těla, tedy pohyb kosti způsobuje pohyb celé části těla a když pohnu například setehení kostí pohne se mi o stejnou rotaci i holení, vřetení kost i kosti kotníku a chodidla. Základní *Skeleton animation* využívá dvě vrstvi sít bodů jako kůži a stromovou strukturu kostí viz. obrázek 2.1. Pro jednoznačný popis a vykreslení animace potřebujeme znát:

- pro vrchol:
  - pozici, uv souřadnice, normálu
  - seznam kostí, kterými je ovlivněn
  - seznam vah, poměr ovlivnění, pro tyto kosti
- pro kost:
  - transformační matici pro přechod ze soustavy kosti do soustavy bodů
- pro uzel:
  - transformační matici pro přechod ze soustavy tohoto uzlu do soustavy rodičovského uzlu pro každý klíčový snímek animace



Obrázek 2.1: Skelatal animation

K výpočtu polohy vrcholu pro daný snímek se musí vypočítat kompletní transformační matice pro všechny kosti, kterými je vrchol ovlivněn. Každou z těchto matic se získá vynásobením všech matic uzlů od uzlu před danou kostí až ke kořeni násobeno ještě transformační matici kosti do soustavy bodů. Tyto matice po vynásobení příslušnými vahami a sečtení dají výslednou transformační matice pro daný vrchol. Stačí samozřejmě vypočítat pro každou kost matici jednou v daném snímku a pak ve vykreslování vrcholů k ní jenom přistoupit. Matice pro uzly se mění v závislosti na čase animace, ukládají se povětšinou jen matice pro klíčové snímky. Je-li čas animace mezi dvěma klíčovými snímky k určení matice pro každý uzel v čase t je třeba udělat vážený průměr matic z klíčového snímku před a po:

$$M_t = \frac{M_{pred} * (t - T_{po})}{M_{po} * (T_{pred} - t)} \tag{2.3}$$

Takže pro výpočet výsledné polohohy vrcholu bude vzorec bez optimaliazací(např. aby se každé kosti počítala jen jednou) bude vypadat takto:

$$M_t = \frac{M_{pred} * (t - T_{po})}{M_{po} * (T_{pred} - t)} \tag{2.4}$$

2.4 Perelinův šum //TODO

Kapitola 3

Návrh aplikace

První část této kapitoly bude pojednávat o obsahu a možnostech aplikace s důrazem na *frontend*, tedy nebude se zabývat vnitřní implementací těchto možností. Druhá část pak následně probere použité způsoby pro realizaci těchto vlastností a zdůvodní proč tyto a ne jiné, případně se krátce zmíní o alternativách, zde budou take rozebráno použití knihoven třetích stran jako *Assimp*[2] nebo *Bullet physic library*[1], dále jen *Bullet*. Na závěr uvede diagram tříd a jeho popis. Podrobnosti k zajímavým částem implemntace těchto tříd a vlastností aplikace budou popsány v následující kapitole č.4 Implementace.

3.1 Obsah aplikace

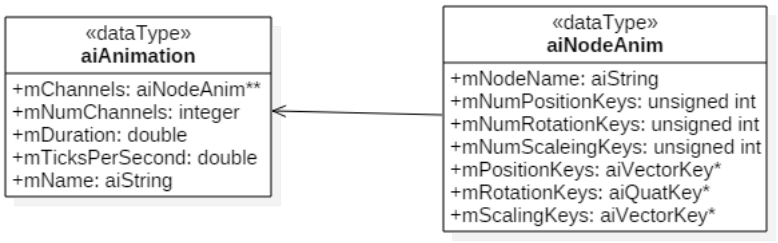
Cíle je vytvořit demo 3D hry, lukostřelecké střelnice. Demo bude tvořeno jen jednou scénou bez menu. Scéna bude obsahovat několik terčů a střeleckých pozic a umožňovat pohyb hráči v tomto prostoru, hráč bude vybaven lukem a několika šípy. Dále hráč vždy uvidí *head-up display*, dále jen *hud*, složený z počítadla bodů, ukazatele síly a směru větru, ukazatele zbylých šípů a zaměřovače.Pro šípy bude zajištěno fyzikální chování a ovlivnění silou a směrem větru. Scénu osvětlíme pouze sluncem/měsícem dle herního času.

3.1.1 Čeho všeho bude aplikace schopna

- Načítání a práce s modely a animacemi
- Fyzika letu a kolize objektů
- Osvětlení scény a stíny pro usnadnění odhadu vzdálenosti a realističnost
- Střídaní noci a dne
- Dostatečná náhrada slunce
- Vítr, plynulá změna jeho směru, poryvy, závětří

3.2 Načítání, ukládání modelů a jejich animací

Modelem se rozumí skupina vrcholů(angl. Mesh). Aby aplikace mohla model vykreslit musí být schopna jej načíst ze souboru a uložit si ho ve vhodné reprezentaci a vědět kterou texturu použít. K načtení bude použita knihovna *Assimp*[2], aby bylo možné načítat více



Obrázek 3.1: Část diagramu tříd knihovny *Assimp*[2]

různých formátů. Ukládat se budou jako pole vrcholů, kdy vrchol bude představovat struktura obsahující pozici v souřadném systému modelu, uv koordináty, normálu a indexy a váhy kostí na které bude navázán, více v sekci 2.3 o *skeletální animaci*. V tomto poli bude každý vrchol pouze jednou, neboť aplikace bude vykreslovat metodou **INDEX - zjisti odborný název**.

Animace jsou načítány společně s modelem ze souboru, *Assimp* je ukládá do dynamického seznamu structur *aiAnimation*. Každá z těchto struktur představuje jednu animaci modelu a vidět ji můžeme na obrázku3.1.

Vzhledem k tomu, že nad animací budou prováděny jen vyhledávací a čtecí operace je toto uspořádání dostačující. Pokud by ovšem vyhledávací a výpočetní funkce byly přidány do stejné třídy kde už jsou funkce pro práci s model, tak by se tato funkce stala velice přetíženou a těžko odladitelnou, vhodné bude zabalit tuto strukturu do vlastní třídy a obohatit ji o potřebné funkce a třídě model zpřístupnit jen výsledné transformační matice jednotlivých kostí.

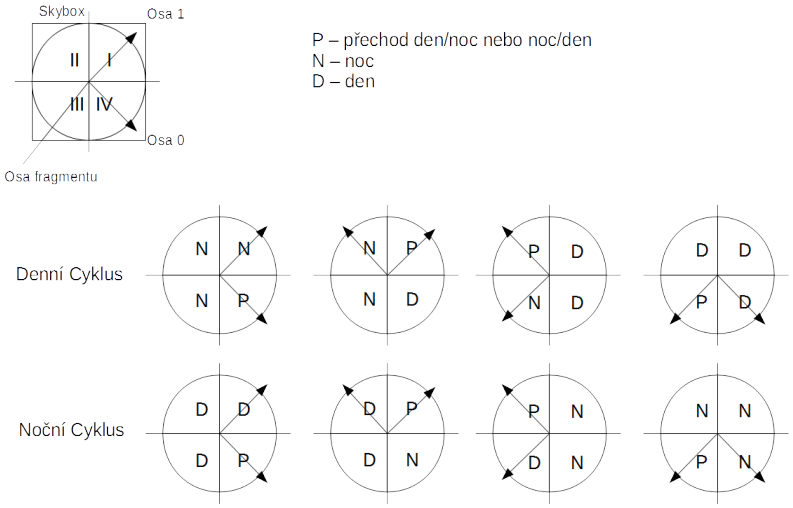
Modely se vyskytují ve spoustě různých formátech a variacích, rozhodl jsem se pracovat pouze se dvěma formáty:

- *Wavefront* od *Wavefront Technologies* s příponou *obj* pro modely bez animací
- *COLLADA* od *Sony Computer Entertainment* s příponou *dae* pro modely s animacema.

Ke čtení modelů je sice použit *Assimp::Importer*, který je schopen číst širokou škálu programů, ovšem aplikace je ozkoušena pouze na těchto dvou, při jiných formátech by potom mohlo dojít k problému při přiřazování do třídy *Model* viz diagram tříd4.1.

3.3 Fyzika letu a kolize objektů //TODO

Fyzikou rozumím výsledek působení sil ať už mnou vytvořených nebo vzniklých při kolizích objektů, zachytávání kolizí a tvorbu sil plynoucích z nárazu. Tuto stránku aplikace jsem se rozhodl vyřešit použitím knihovny třetí strany z důvodů relativně vysoké fyzikální složitosti a náročnosti na implementaci dostatečně rychlého algoritmu pro řešení kolizí. Zvolil jsem si knihovnu *Bullet*, která poskytuje možnost řešit jak zachytávání kolizí, tak jednoduché zajištění fyzikálního chování pevných objektů i měkkých objektů Ovšem *bullet* provádí výpočty na základě vlastní fyzikální scény, je tedy potřeba zajisti provázání mezi fyzickou reprezentací modelu a jeho entitou určenou pro vykreslování. Vazba musí být obou směrná neboť



Obrázek 3.2: Graf střídání denní a noční oblohy

model se v každém vykreslovacím cyklu musí být schopen dotázat na polohu jeho fyzikálního ekvivalentu, a opačně, protože *Bullet* vrací kolize v jako pole dvojic *btColisionObject* a pro tyto objekty se musí zavolat příslušná funkce(výpočet bodů za zásah, zafixování polohy vůči zasaženému tělesu, ...).

3.4 Osvětlení a stíny //to finish

Scénu bude osvětlovat jen jedením až dvěma zdroji směrového světla představující slunce a měsíc za použití Phongova osvětlovacího modelu. Jeho poloha a intenzita se bude měnit dle herní denní doby. Stíny budou vypočteny a vykresleny metodou *shadow maping* jenž je popsána v kapitole 2.2. Protože aplikace bude mít jen jeden až dva směrové zdroje světla je tato metoda vhodná díky své rychlosti a tvoří se jen jedna - dvě mapy. Tato mapa se bude přepočítávat pro každý snímek může me ji tedy vždy namapovat tak, aby měla co nejmenší přesah přes plošné promítnutí tělesa vymezeného pohledem kamery a zdroje světla. Tento způsob namapování umožňuje pracovat s menší texturou, tím rychlejší výpočet a méně zabrané paměti.

3.5 Střídání noci a dne

Na *skybox* budou na mapovány celkem tři textury, pro mraky, denní a noční oblohu. Mraky budou vidět vždy, v ideálním případě se budou měnit pomocí *Perlinova* šumu. Střídání noci a dne je způsobenou změnou poměru míchání noční a denní textury pro jednotlivé fragmenty.

Poměr pro každý fragment je dán mezi úhlem jeho osou(polopřímka ze středu přes něj) a osami 1 a 0, jak je vidět na obrázku 3.2.Pokud se fragment nachází ve stejném kvadrantu jako osa 0 je poměr vypočítán na základě jeho úhlu, je-li úhel 45°je poměr je 0:1 v neprospěch aktuálního cyklu naopak pro -45°1:0 pro aktuální cyklus. Mezi těmito hodnotami je potom

nepřímá úměra. Ve zbylých třech kvadrantech je poměr vždy buď 1:0 nebo 0:1, odvíjí se toho, která čtvrtina denního/nočního cyklu je:

- kvadrant u osy 1:

  pro 1 - 3 část cyklu 0:1

  pro 4 část cyklu 1:0

- kvadrant naproti ose 0:

  pro 1 a 2 část cyklu 0:1

  pro 3 a 4 část cyklu 1:0

- kvadrant naproti ose 1:

  pro 1 část cyklu 0:1

  pro 2 - 4 část cyklu 1:0

Celý den má tedy 8 částí, 4 denní a 4 noční, trvá dvě otočení os o 360°. Aplikace při spuštění začíná v čase nula, což je úsvit nového dne.

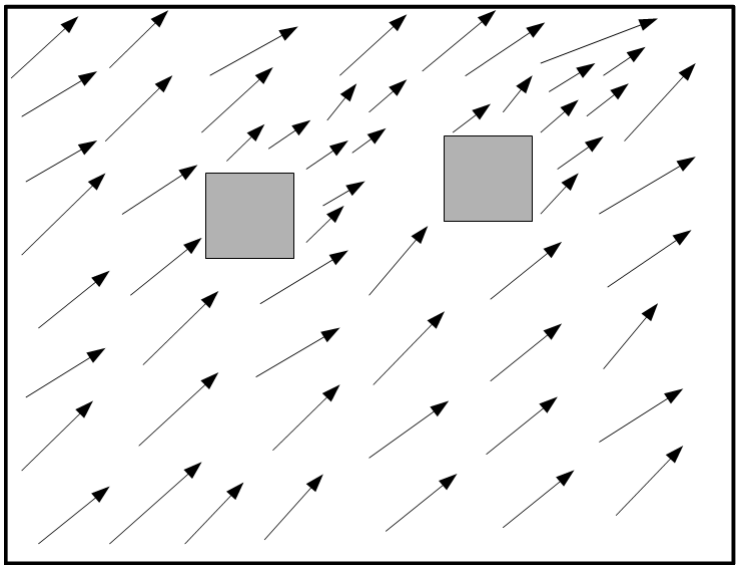
3.6 Slunce a měsíc

Slunce a měsíc se realizují pomocí plochých modelů rotujících kole středu *skyboxu*. Doba oběhu musí trvat celý den, tedy dvě otočení os *skyboxu*. V rámci třídy pro Slunce a měsíc se bude řešit i změna směru světla příslušného objektu. Pro správný pohy se musí zajisti dvě věci správné natočení plochy objektu, aby nebyl vidět jako elipsa či jako pouhá čára, uchování vzdálenosti od středu *skybox*, potažmo avatára. Dosáhne se toho pomocí vektoru $\vec{a}$ mezi středem a objektem a rotací $\vec{a}$ i objektu kolem stejné osy(vyjádřená stejným předpisem, počítána vždy v relativní soustavě objektu nebo globální soustavě v případě $\vec{a}$). Poloha objektu se potom vždy nastaví podle vektoru $\vec{a}$ posunutého do aktuálního středu *skyboxu*.

3.7 Vítr

Vítr je na vykreslování relativně jednoduchý jev, není totiž vidět, ale na druhou stranu má spoustu viditelných projevů. Zatím se bude práce zabývat jen působením větru na šíp za letu. Vítr bude mít globální sílu a směr, které se budou v čase měnit, změna bude vypočtena po každé herní vteřině pomocí perlinova šumu. Tak to by vznikl vítr, který by všude byl stejný, jenže vítr se mění, dá se zastavit a vznikají závětří, tady pofukuje víc, tady méně, tamhle trochu jiným směr a tak, aby se dalo dosáhnout tohoto chování aspoň do určité míry realističnosti musí se učit závětrná míst, větrné stíny, a pro každý bod vypočíst drobnou odchylku směru a síly jak je znázorněno v obrázku 3.3.

Na obrázku 3.3 nám směr a velikost větru znázorňují šipky. Za oběma čtverci jsou vidět větrné stíny s výrazně nižší intezitou větru. K výpočtu větrných stínů se dá použít téměř jakákoliv osvětlovací metoda, proto že vítr se v podstatě ničím neliší od světla, stále jsou to paprsky/proudy, jen jejich projev je jiný místo osvětlení místa dopadu jej posunou. Já jsem se rozhodl použít metodu *shadow maping* a znovu použít již nutně implementované funkce pro výpočet stínů osvětlení. Tímto postupem lze získat větrné stíny, ale jinak se síla a směr větru nijak nebude měnit. Proto výslednou mapu ze *shadow mapingu* zašumíme, například již zmíněným *perlinovým šumem*.Jsou dvě možnosti buď použít 2D perlinův šum na celou



Obrázek 3.3: Větrná mapa výsledné aplikace

mapu nebo 1D pouze na aktuálně čtený bod. Vzhledem k tomu, že vítr bude ovlivňovat jen několik těles a pro každé těleso se bude vyčítat jen jeden bod z mapy,vysvětlím dále, bude rychlejší na výpočet použít pouze 1D šum při čtení.

Zbývá vyřešit jak se bude vítr projevovat, lze k tomu přistoupit dvěma způsoby jednodušším a méně reálným nebo složitým a reálným. Jednoduší znamená, vítr se projeví jako pouhá směrová síla, bez ohledu na dopadovou plochu na tělese. Složitý znamená, že vypočítat dopadovou plochu z ní odvodit výslednou sílu větru. K zjištění dopadové plochy bychom museli ovšem každý fragment tělesa otestovat kde leží v mapě, jestli je ve stínu nebo ne. Pokud se tento test prováděl na CPU, došlo by k výraznému zpomalení aplikace, na GPU by to nebyl problém a v ničem by se to nelišilo od vykreslování stínů, ovšem standardní knihovny pro práci s GPU neumožňují přesun dat z GPU zpět do CPU. Pokud by šlo o simulátor plachetnice tak by se určitě muselo jít složitější cestou, ovšem tím že v rámci práce tvořím lukostřeleckou střelnici a větrem budou primárně ovlivněny šípy, které mají malou plochu a v podstatě stejnou ze všech stran, bude dostačující jednoduchá cesta. Z toho i vyplívá proč se bude vždy číst jen jeden bod mapy pro těleso a stačí tím pádem 1D šum.

Kapitola 4

Implementace // TODO

4.1 Diagram Tříd// TODO popis

Na obrázku 4.1 je zjednodušený diagram tříd výsledné aplikace. Aplikace je složena ze tří hlavních částí:

1. Třída Scene, která zapouzdřuje celou aplikaci.
2. Třída BulletWorld, přes kterou je řešena veškerá fyzika.
3. Třída Model, která je výchozí třídou pro většinu prvků ve scéně.

Dále pak aplikace má mnoho dalších tříd, jako třídy pro HUD nebo pro práci s animacemi či zvukem, mezi zajímavější pak patří třídy *Skybox* a *SunMoon*, které mají na starosti celé střídání noci a dne a pohyb zdrojů světla.

4.1.1 Třída Scene

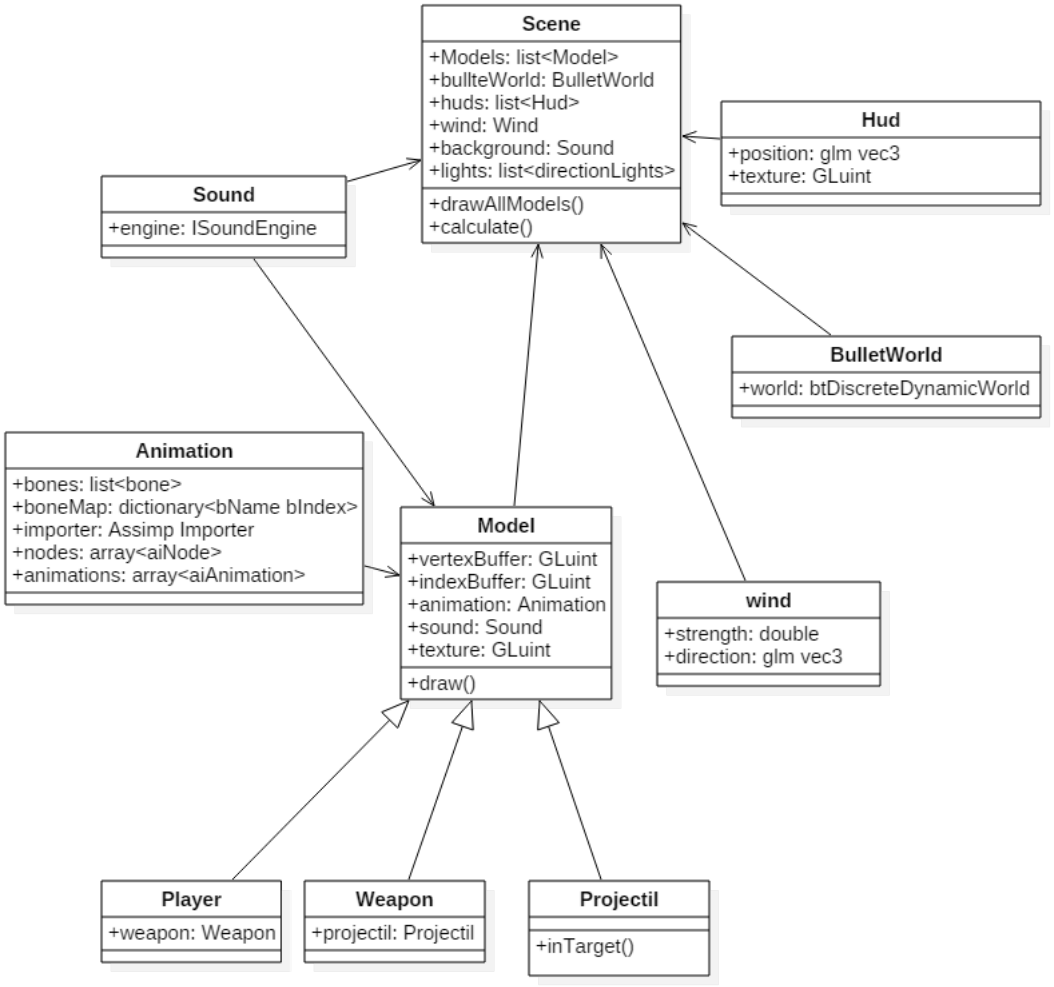
Třída Scene je ústřední třídou, z ní se přistupuje do třídy BulletWorld pro detekci kolizí a změny poloh modelů na základě působení sil, zde se také řeší jednotlivé kroky vykreslování jako vykreslení do *wind mapy, shadow mapy* a výsledné.

Třída tím pádem obsahuje převážně funkce pro nastavení objektů, jejich vkládání případné nastavení propojení mezi objekty, které spolu nějak komunikují(např.:zbraň a projektil nebo objekt a jeho fyzikální představitel). Většina chování objektů je v jejich vlastní režii, třída Scene volá jen funkce pro jejich nastavení při vzniku a během cyklu už jen vykreslovací funkce.

Tato třída ještě obsahuje slovníky načtených modelů a textur. Slovník pro modely má tvar <cestaKsouboru,ImportModel>. Při načítání souboru se zkontroluje zdali již není ve slovníku, je-li použije se příslušný záznam, není-li tak se soubor načte a zpracuje do třídy ImportModel a vrátí se opět záznam ze slovníku. U textur to probíhá podobně jen místo třídy ImportModel se použije přímo adresa textury na GPU.

4.2 Zajímavé nebo problémové části implementace

- problémy při implementaci větru
  - nekonečnost stínu



Obrázek 4.1: Zjednodušený diagram tříd výsledné implementace

  - simulace přirozených poryvů větru
- skeleton animation
- zajištění jednoho načtení modelu

Kapitola 5

Shodnocení náročnosti a efektivnosti implementace// TODO

- paměťová náročnost ku počtu modelů
- jak vylepši tuto věc
- framerate Obsah...

Kapitola 6

Závěr// TODO

Literatura

[1] Coumans, E.: *Bullet physic library.* [Online; navštíveno 26.04.2017].
URL http://bulletphysics.org/wordpress/

[2] Gessler, A.; Schulze, T.; Kulling, K.; aj.: *Open Asset Import library.* [Online; navštíveno 26.04.2017].
URL http://assimp.org/index.html

[3] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika.* Computer press, 2004, ISBN 80-251-0454-0.

Přílohy

Příloha A

Obrázky

A.1 Obrázky2