Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Skeletální animace 3D modelů

Bakalářská práce

Autor: Josef, Zahradník
Studijní obor: Aplikovaná informatika (ai3-p)

Vedoucí práce: Ing. Bruno Ježek, Ph.D.

Hradec Králové duben 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 22.4.2024 Josef Zahradník

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Brunovi Ježkovi, Ph.D za metodické vedení práce a jeho podporu při průběhu vývoje.

Anotace

Práce se zabývá rozborem populárních 3D animačních technik používaných primárně pro tvorbu filmů a počítačových her s finálním důrazem na skeletální animaci. Skeletální animace je technika používaná pro pohyb postav, jejím použitím je zajištěna plynulost a realističnost výsledné animace. K zobrazení skeletální animace bude použita grafická knihovna OpenGL s její rozšiřovací knihovnou Assimp, která umožňuje čtení a konverzi animovaných objektů různých souborových formátů. Import a zobrazení objektu bude testováno na různých modelech lišících se komplexitou, formátem, ale i zpracováním. Účelem této práce je prozkoumání matematických algoritmů používaných v technikách skeletální animace a prozkoumání knihovny Assimp spolu s jejími strukturami. Finálním produktem této práce je funkční animovaný model.

Annotation

Title: Skeletal animation of 3D models

This thesis is delving into popular 3D animation techniques used primarily for creation of films and computer games with its final specialization on skeletal animation. Skeletal animation is technique used for the movement of characters, when used, skeletal animation ensures smoothness and authenticity of resulting animation. The OpenGL Assimp library will be used to display animated model. This library enables users to import models of various file types. The import and display of models will be tested on several models varying in complexity, file type and in the way they were created. The purpose of this thesis is to explore mathematical algorithms used in skeletal animation together with Assimp and its structures. Final outcome of this thesis is a functioning animated model.

Obsah

Seznam obrázků

1 Úvod

Animace je technika snažící se vytvořit iluzi pohybu. Animace původně začala ve 2D formě, kde její hlavní využití bylo ve filmovém průmyslu pro rozpohybování kreslených postav. V moderní době techniky animace pokročily a nyní kromě 2D ručně kreslených animací vznikají i 3D animace vytvořené pomocí počítačových algoritmů. Animace se, kromě technologického zpracování, posunula i v oborech, které se touto technikou zabývají. Nyní je možné se setkat s animací i v oborech jako je architektura, medicína či školství.

Skeletální animace je technika používaná primárně pro pohyb postav, jejím použitím je zajištěna plynulost a realističnost výsledné animace. V rámci řešení práce bude k zobrazení skeletální animace bude použita grafická knihovna OpenGL a knihovna Assimp umožňující načítání a konverzi různých souborových formátů pro uložení 3d scény obsahující animované objekty. Import a zobrazení objektu bude testováno na různých modelech lišících se komplexitou, formátem, ale i zpracováním. Účelem této práce je prozkoumání matematických principů skeletální animace a jejich implementačních algoritmů. Dále se tato práce bude zabývat prozkoumáním knihovny Assimp spolu s jejími strukturami. Finálním produktem této práce je funkční animovaný model.

2 Cíl a metodika práce

Cílem práce je prozkoumat jednotlivé fáze 3D animačních technik se zaměřením na implementaci skeletální animace. Zpracování a zobrazení animace bude provedeno pomocí OpenGL knihovny Assimp. Pro prezentaci výsledků budou použity externí, ale i mnou vytvořené modely různých souborových formátů. Pro implementaci animace bude použita technologie LWJGL s pomocí balíčku Transforms pro uchování a práci s matematickými strukturami

Teoretická část práce bude popisovat proces vzniku animovaného modelu, od koncepčního návrhu až po ladění finálního modelu. Následně budou popsány jednotlivé animační techniky a jejich využití. Poté se práce bude zabývat souborovými formáty pro uložení 3D scény a bude zmíněna scéna knihovny Assimp. Práce bude zakončena teoretickým rozborem a implementací skeletální animace.

3 Animace

„*Animace je metoda zobrazování na sebe navazujících snímků, modelů, či loutek za účelem vytvoření pohybu*“ [1]. Animaci můžeme mimo jiné rozdělit na 2D a 3D, přičemž každá z těchto animací má svoje využití a účel. V praxi se používá několik druhů animačních technik jako např. keyframe animace, motion capture nebo animace procedurální. U keyframe animace je výsledný pohyb získán interpolací mezi pózami animovaného modelu. K animaci se dále používá technologie zvaná motion capture nahrávající pohyby živého herce, které jsou následně použity pro základ animace v animačním softwaru. Neposledním příkladem je animace procedurální, zde se pohyb odvíjí od získaných dat vypočtených fyzikálními rovnicemi zahrnující faktory, jako je gravitace či vítr. Pro účely této práce budou probírány postupy pouze 3D animací s finálním zaměřením na animaci skeletální, která je druhem keyframe animačního postupu.

Animace se tradičně přisuzuje oborům, jako je filmový průmysl či tvorba počítačových her. Animace už dávno přesáhla i do odvětví jako je medicína, architektura či právo [2]. Animace se tedy postupem času transformovala z disciplíny, která byla primárně určena pro umělce, na disciplínu vyžadující znalost technických oborů jako např. matematika nebo počítačová grafika.

Na poli filmového průmyslu je možné se setkat se dvěma typy animovaných snímků: plně animované a filmy se speciálními efekty. Plně animované filmy používají 3D software pro vytvoření celého snímku [2]. Mezi tento typ filmů patří například Shrek, Frozen či Spiderman: Into the Spider-Verse. Filmy se speciálními efekty využívají reálných herců stejně jako běžné filmy. Od těchto filmů se ale liší tím, že všechny objekty scény jako jsou modely či prostředí jsou vytvořené počítačem. Za starší příklady těchto filmů je možno uvést Godzillu či Jurský Park. V současnosti jsou speciální efekty využity pro natáčení nespočtu filmů, přičemž nejznámějším příkladem je série superhrdinských filmů od studia Marvel.

Obrázek 1: Použití VFX ve filmu Avengers: Endgame od studia Marvel [3]

Tvorba animací pro počítačové hry se dělí na dva základní okruhy: Animace k podpoře hratelnosti (napří klad animace pohybu hráčem ovládané postavy) a animace tzv. cutscén neboli neinteraktivních filmů [2]. Tyto filmy jsou předrendrované segmenty, které slouží pro podporu příběhu a dodávají počítačovým hrám kinematický element. V některých hrách jsou cutscény vzácné nebo v nich nejsou vůbec, ale na druhou stranu je také spoustu herních žánrů, které jsou na cutscénách postavené.

Animátor počítačových her musí brát v úvahu mnoho faktorů, aby se jeho výtvor dal považovat za kvalitní, a tak stejně jako jakýkoli jiný animátor musí pracovat s rozpočtem animace, estetickou a funkční stránkou. Hlavním faktorem pro funkční animace je její průběh v reálném čase. Na rozdíl od filmů, videohry potřebují efektivně využívat výpočetní sílu koncového uživatele. Kvůli tomu musí vývojáři pokaždé zvážit, jak jejich umělecké rozhodnutí ovlivní optimalizaci a hardwarové nároky finálního produktu. U animace a animovaného modelu je potřeba brát zřetel na detail a komplexitu modelu. Modelování příslušného objektu je jednou z prvních fází 3D animace. [2]

3.1 Animační řetězec

Animování postavy není ani zdaleka jediný úkon, který je potřeba k finální animaci provést. Vývoj animace může zahrnovat mnoho týmů z různých odvětví. Kvůli komplexitě animačního vývoje byl proces animace rozdělen na několik fází, které jsou

obsaženy v animačním řetězci, kde jsou zahrnuty prvotní umělecké návrhy, technické zpracování, ale i ladění finálního produktu. [2]. Účelem animačního řetězce je zefektivnit vývoj animace např. produkční fáze nezačne, pokud předprodukce není dokončena [4]. Animační řetězec rozděluje vývoj animace do tří na sobě závislých kroků: předprodukce, produkce a postprodukce [5].

3.1.1 Předprodukce

V běžné praxi bývá předprodukce nejméně technickou částí celého procesu [4]. Tato fáze zahrnuje úkony jako je plánování, design či výzkum. Z animační stránky je zde nejdůležitější si uvědomit, čeho se animátoři snaží docílit, jaké emoce by animace měla vyjádřit a jaký příběh by měla říct [2]. Finálním produktem této fáze je umělecký směr a concept art [4].

Existuje mezikrok mezi předprodukcí a produkcí, který se v angličtině nazývá „pre-visualization“ neboli previzualizace (zkráceně pre-vis). Pre-vis slouží k ověření vytvořených konceptů. Animační tým kontroluje své výtvory, vyhodnocuje, jestli splňují všechny požadavky. Produkční část řetězce může být velice finančně nákladná, proto se v pre-vis animátoři snaží předejít zbytečnému předělávání designových konceptů [4]. Jeden ze scénářů, kterým je vhodné předejít, je např. předělání hotového 3D modelu. Kromě revize produktu vzniklého v předprodukci se prochází nástroje potřebné pro tvorbu animace. Mezi zmíněné nástroje je možné zařadit 3D softwary typu Unity 3D a Unreal Engine.

3.1.2 Produkce

V produkční fázi se teprve projeví znatelný pokrok. V ideálním případě je většina potenciálních problémů odchycena u kořenů již v předprodukci. Prvním krokem v produkční části je vytvoření tzv. 3D layoutu, sloužícího pro testování animace. Testování probíhá na proxy modelu, který sdílí tvar a velikost s modelem finálním [2]. Layout může probíhat ve všech fázích animačního řetězce, ale ideální je začít s testováním co nejdříve. Pokud je testování úspěšné, animátor získá informace o funkčnosti animace, tím pádem si ověří, jak dobře se concept art přenesl do 3D prostoru. Dále testování umožní simulovat práci s kamerou a tím je možné získat ideální úhel kamery pro danou animaci [2] [5].

Předchozí testy se prováděly na testovacím modelu, který však nebude použit pro finální produkt. V tuto chvíli nastává další krok produkční fáze animace: modeling. Existuje hned několik způsobů, jak vytvořit model objektu. Vše záleží na tom, k jakému účelu bude model využit. Animátor může vytvořit zbrusu nový model pomocí 3D animačního softwaru, jako je například Maya či Blender. Model může být generován procedurálně nebo je dokonce možné naskenovat reálný objekt a model vytvořit z něj [2] [5]. Dříve bylo zmíněno, že layout fáze se může využít při jakékoliv části animačního řetězce. Zde je možné použít hotový model pro testování animace a doladit případné detaily, jako je například úhel snímací kamery [2].

Pokud je modelování úspěšné, animátor má k dispozici model v základní barvě, která je ve většině 3D programů šedá [5]. 3D umělci mají za úkol přiblížit vizuální stránku modelu co nejvíce k prvotním návrhům v concept artu, toho lze dosáhnout pomocí nanesení materiálu nebo upravení nasvícení [2] [4].

Po modelování je model je hotový jen z estetické stránky. Vypadá sice tak, jak animátoři potřebují, ale nemá komponenty nezbytné pro animaci. Proto je v následujícím kroku vytvořen rig, který umožňuje objektu pohyb. Rig je kostra skládající se z kostí a kloubů, která zprostředkovává pohyb celého modelu. Rig by měl být co nejefektivnější, neboť všechny pohybové výpočty jsou prováděny na jeho komponentech [2]. Komplexita rigu se může lišit od základního vztahu předek-potomek až po rig s klouby či svalovým systémem [2]. S pohyblivým modelem je možné přistoupit k další fázi řetězce neboli animaci, kde se vytváří pohyb postavy. Animačních technik je řada a budou více rozvedeny v následující kapitole. Pro příklad je možné uvést keyframe animace či motion capture [2].

Animátor vytvořil pohyb postavy, jejího těla a jednotlivých končetin. Části postavy, které jsou přidány navíc, jako vlasy či oblečení, dokončuje tým starající se o vizuální efekty neboli VFX. Animace těchto částí postavy je příliš komplikovaná na ruční animaci, proto je potřeba najít jiný způsob [5]. Cílem je simulovat vlivy přírodních faktorů, jako vítr či gravitace, na již zmíněné předměty využitím matematicko-fyzikálních výpočtů [2]. VFX umělci se kromě animace částí postavy zabývají animací částicových systémů a simulací tekutin [5].

Nasvícení a rendrování jsou finální části produkční fáze animačního řetězce. Existuje několik druhů osvícení, přičemž jednotlivé druhy se liší počtem světelných bodů a jejich umístěním ve scéně. Jako příklady typů osvětlení je možné uvést směrové, bodové a plošné světlo [5].

S nasvícenou scénou přichází renderovací fáze. Rendrování je proces kde z 3D dat uložených v PC je vytvořen obraz [5]. Zde jsou rendrovány jednotlivé objekty scény. Jednotlivě zpracované objekty a jejich vlastnosti, jako jsou stíny, jsou v postprodukci seskupeny zase dohromady [2]. Tím, že jsou jednotlivé části scény rendrovány zvlášť, se nabízí animátorovi příležitost doladit detaily. Rendrování je možno rozdělit podle využití času na dvě metody. Rendrování v reálném čase kalkuluje a zobrazuje data za chodu aplikace, tento postup je využíván především v počítačových hrách, kde je potřeba počítat s uživatelskou interakcí. Druhý způsob je renderování neinteraktivních médií, jako jsou animované filmy nebo videa, která nevyžadují práci s reálným časem. Zde je možné dosáhnout lepší kvality obrazu s menšími hardwarovými nároky [5]. Tuto metodu je možno nalézt i v počítačových hrách ve formě předrenderovaných cutscén.

Obrázek 2: Ukázka předrendrované cutscény ze hry Like A Dragon: Ishin! [6]

Obrázek 3: Ukázka cutscény v reálném čase ze hry Like A Dragon: Ishin! [7]

3.1.3 Postprodukce

Finálním stádiem animačního řetězce je postprodukce, která je určena ke korektuře chyb, finálnímu ladění a úpravě barev. Postprodukční fáze se dělí na tři kroky: kompozice, VFX a korektura barev [2] [5].

Z produkční fáze získal animátor spoustu jednotlivě rendrovaných objektů, v kompozici je potřeba tyto objekty seskupit podle původního modelu. Komplexita kompozice spočívá na množství vrstev, se kterými musí animátor pracovat. Nekombinuje se zde jen animovaný model, ale i všechny aspekty rendrované scény [2] [5].

2D visuální efekty (dále jako 2D VFX) se soustředí na vytvoření efektů jako jsou jiskry, prachové částice, kapky deště či třes kamery. Tyto efekty je možné vytvořit už v produkční fázi jako 3D objekty (kapky deště či jiskry), ale není to ideální [2]. Každý modelovaný objekt musí projít rendrovacím procesem a pak následnou kompozicí zvlášť. Této dlouhé implementaci se dá vyhnout, pokud jsou zmíněné efekty dodělány až v postprodukci. Animátor zde již nepracuje s 3D scénou ale pouze s 2D [2] [5].

Cílem korektury barev je ucelení jednotného vizuálního stylu. Při rendrování mohly být jednotlivé objekty nasvíceny jinak, či byly použité jiné barevné tóny. Zde animátor musí pracovat s odstíny jednotlivých barev a celkovým osvětlením scény. Všechny tyto kroky by měly být hotové už z produkční fáze, zde se jen ladí detaily.

3.2 Druhy animace

3.2.1 Keyframe animace

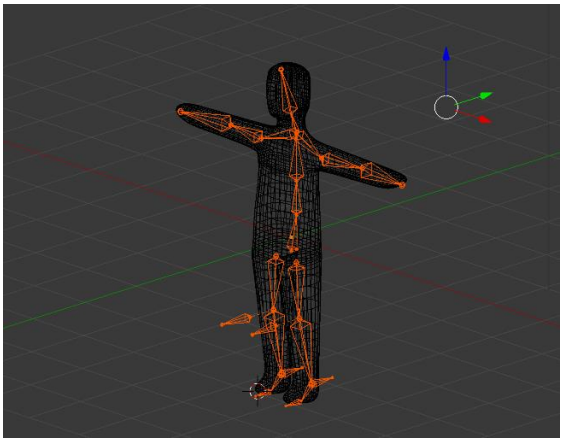
Animace pomocí klíčových snímků, nebo také póz, je základním animátorským přístupem. Celková animace se skládá z nespočtu snímků, avšak nejdůležitější z nich jsou tzv. pózy. Pózy jsou použity jako hlavní body pro pohyb postav, neboť kromě udávání počátku a konce animační sekvence nesou informace o souřadnicích objektu [2]. Pózy jsou ručně vytvořeny animátorem, ale jednotlivé přechody mezi nimi jsou generované. Tento přechod je způsoben interpolací mezi počáteční a konečnou fází [8] [2]. Obtížnost tohoto principu spočívá v dobrém nastavení klíčových snímků, pokud se snímky nenastaví správně, může dojít k několika problémům při interpolaci např. ke kolizi končetin.

Existují dva principy keyframe animace. Prvním způsobem je již zmíněná práce s pózami tzv. Pose-To-Pose [2] [8]. Před nástupem počítačů byl hlavní technikou princip zvaný straight ahead. Animátor zde musí vytvořit každý snímek animace ručně. Oprava dané animace je velmi obtížná, protože animace musí být dokončena celá, aby ji bylo možné posoudit. Pokud animace nesplňuje očekávání, jediný způsob opravy je začít znovu [2]. Straight ahead postup byl hlavně používán pro rukou kreslená média a jako samostatný postup není dostatečný pro oblast 3D animace [8].

Princip straight ahead ale není opuštěn úplně. Při animaci postav se používá kombinace obout postupů. Pose-To-Pose je použit k definování klíčových snímků, díky nim je nastaven základ animace, straight ahead je určen k dokončení finálních snímků. Tomuto postupu se říká hybrid, neboť je kombinací předchozích postupů [2].

3.2.1.1 Skeletální animace

Skeletální animace je druhem keyframe animace, která také provádí pohyb interpolací mezi jednotlivými animačními snímky. Model animovaný pomocí skeletální animace obsahuje kostru, která stejně jako u lidského těla, zprostředkovává pohyb celého modelu. Díky kostem může animátor simulovat realistický pohyb, kde se kosti vzájemně ovlivňují, to znamená, že když se pohne rameno, tak tímto pohybem bude ovlivněno i předloktí či ruka.



Obrázek 4: Kostra animovaného modelu [9]

3.2.2 Motion capture

Motion capture (mocap) je proces při kterém je snímán pohyb fyzických objektů, jež je následně překládán do virtuálních dat [10]. Lidští herci na sobě nosí snímací oblek, který slouží pro zachycení jejich pohybů. Tato data jsou následně

aplikována na 3D model postavy [2] [10]. Použití reálné postavy k animaci má několik nezpochybnitelných výhod. Data získána z mocap usnadňují animaci a zároveň snižují její náklady [2]. Animace je daleko přesnější a realističtější, protože jejím podkladem je člověk. Tím pádem je možno zachytit nuance lidského pohybu, které by jinak mohli být opomenuty. Při využití mocap je jednodušší splnit prvotně naplánovanou vizi pro animaci, protože režisér nebo zadavatel animace je v přímém kontaktu s hercem.

Animace pomocí technologie motion capture je používána v mnoha odvětvích lidské činnosti a to ne jen v té zábavní. Videoherní a filmový průmysl jsou nejznámějšími představiteli práce s mocap. Mezi nejznámější příklady filmů patří Star Wars (1999) nebo Avatar (2009) [10]. Na poli počítačových her je možné uvést např. novodobou sérii her God of War [11].

Obrázek 5: Motion capture pro hru GoW: Ragnarok [12]

3.2.3 Procedurální animace

Předchozí animační techniky byly založeny na předem definovaných pohybech. U keyframe animace to byly klíčové snímky a u motion capture to byl nahraný set pohybů lidského herce. Na rozdíl od předem zmíněných technik, procedurální animace využívá algoritmy k vytvoření celého pohybu [2] [13]. Procedurální animace je prováděna v reálném čase a její výsledky se odvíjí od externích, ale i interních faktorů. V počítačových hrách je možné jako příklady těchto faktorů uvést vstup z hráčovy strany či přírodní podmínky jako vítr manipulující s vlajkou atp. [13]. Výhodou tohoto přístupu je jeho robustnost pro komplexní sety pohybů. Například je možno uvést videoherní sérii Assassin´s Creed, kde je parkur pohybový systém procedurálně

generovaný, aby hráč mohl získat nejvhodnější animaci pro daný akrobatický kousek [14]. Procedurální animace nabízí několik druhů, které budou předvedeny v následující sekci.

3.2.3.1 Forward/Inverse Kinematics

Kinematika se zabývá studiem pohybu, konkrétně se zaměřuje na popis fungování hierarchické struktury skeletu a jeho vlivy na výsledný pohyb objektu [15]. Obecně jsou v animaci používány dva základní přístupy: přímá kinematika, anglicky forward kinematics neboli FK, a inverzní kinematika, anglicky inverse kinematice neboli IK [15]. Využití kinematiky je možné nalézt v mnoha oborech jako např. robotika nebo počítačová grafika [16].

Pohyb za pomocí přímé kinematiky je započat na začátku hierarchické struktury, to znamená, že koncová rotace cílené části objektu (např. dlaně) je odvozena z předchozích rotací rodičovský kloubů (např. rameno, či loket) [15]. Pro tento příklad animace začíná pohybem ramenního kloubu, přes kloub loketní, končívaje pohybem a rotací zápěstí. Tento postup může být velice časově náročný, protože se musí s rotacemi každého kloubu pracovat zvlášť [15]. Existence přímé kinematiky dala prostor pro vznik kinematiky inverzní (IK), která změnila pracovní postup vytváření animace. V IK nejsou potomci ovlivňováni rodiči, ale rodiče jsou ovlivňováni transformacemi potomků, což umožňuje snazší tvorbu animace a živější pohyb [15].

Inverse kinematice (IK) je princip, který podle kinematických rovnic určuje pohyb kloubů za účelem vytvoření pohybu [16]. Na rozdíl od kinematiky přímé se zde pohyb odvíjí od tzv. efektorů, které se nachází na spodu hierarchické struktury skeletu. Pokud tedy animátor chce pohnout chodidlem, tak pohyb celé nohy se bude odvíjet od pohybu chodidla. Efektor je při animaci přímo přemístěn na požadovanou pozici a rotace potomků se odvíjí od výsledné pozice efektoru. Jak inverzní, tak i přímá kinematika má své využití, přímá kinematika je například využívána pro animaci švihu končetin [15].

3.2.3.2 Ragdoll physics

Ragdoll physics je druh procedurální animace používané pro animaci 3D postav simulující pohyby končetin ovlivněných fyzikálními silami jako je proud větru, či gravitace. Ragdoll physics je používána zejména pro charaktery, nad kterémy hráč ztratí kontrolu, charakter například zemře nebo je omráčen [17].

Tento způsob animace rozebere model na jednotlivé části a následně na ně aplikuje fyzikální síly, které určují výsledný pohyb. Princip ragdoll physics je využíván nejen pro postavy, ale i pro neživé předměty jako je oblečení, vlasy nebo ohebné objekty. Technika manipulující neživými objekty podle fyzikálních vlivů se nazývá soft-body physics [17].

4 Souborové formáty pro uložení 3D scény

Pro export a používání modelů se používají různé formáty 3D souborů. Hlavním cílem těchto souborů je uchovávat informace o geometrii objektu, texturách, detailech scény a animaci [18].Ne všechny formáty umí ukládat všechny tyto složky. Informace mohou být do těchto souborů ukládány buď textovou nebo binární formou, záleží primárně na typu konkrétního souboru [18]. I přes ohromné množství různých typů, 3D soubory můžeme zařadit mezi jednu ze dvou kategorií: proprietární a neutrální.

Proprietární soubory jsou vytvořeny speciálně pro práci s určitým softwarem jako např. BLEND od Blenderu či DWG od AutoCADu. Tyto soubory jsou optimalizované pro práci v konkrétních systémech, což umožní danému sofwaru je zpracovávat rychleji a efektivněji [19]. Největší výhoda proprietárních souborů je zároveň jejich největší omezení, tyto soubory fungují jen s úzkou selekcí softwarů. Druhým typem jsou neutrální soubory, které fungují oproti jejich proprietárním protějškům na všech platformách.

4.1 Wavefront OBJ

OBJ je neutrální formát vytvořený společností Wavefront technologies určený pro uchovávání dat o geometrii tělesa jako jsou přímky, polygony, křivky a plochy. Kromě těchto atributů OBJ dokáže uchovat informace o barvě tělesa či jeho texturách [19]. Jednou z předností OBJ je jeho otevřenost, díky níž mohou vývojáři nahlédnout do souboru přímo, OBJ je totiž druh textového souboru [20]. Objekt exportovaný do OBJ má většinou menší velikost oproti svým konkurentům [21]. U některých formátů se může stát, že jejich interpretace nebude konzistentní s čímž OBJ nemá problém [19]. Hlavním nedostatkem OBJ je to, že nepodporuje animace, ani neuchovává informace scény jako je osvětlení či kamera [19].

4.2 Filmbox FBX

Filmbox (neboli FBX) je jeden z nejpoužívanějších proprietárních 3D souborových formátů současnosti. Tento formát je používán pro ukládání geometrie objektů, světel, materiálů a animací. FBX je podporován všemi moderními 3D grafickými softwary jako Blender, Max nebo Maya. Soubory tohoto typu jsou standartně reprezentovány binárně, ale je možnost jejich uložení i v ASCII podobě [22]

[23]. Díky jeho rozmanitým možnostem pro ukládání informací je tento formát oblíbený u filmových tvůrců či vývojářů počítačových her [21]. FBX je hojně využíván pro aplikace podporující virtuální a rozšířenou realitu. Narozdíl od neutrálního OBJ, který funguje s každým softwarem, FBX v některých aplikacích podporován není. I když aplikace FBX podporuje, může se stát, že objekt uložený v tomto souboru je interpretován chybně, neboť určité programy nedokáží s FBX správně manipulovat [24].

4.3 COLLADA

COLLADA stejně jako předchozí formáty podporuje geometrii a stejně jako FBX podporuje i uchovávání animací, ale jako jeden z mála formátů podporuje i kinematiku a fyziku [18]. COLLADA zapisuje informace ve formě XML značkovacího jazyka [18] a soubory využívající tuto technologii mají koncovky .dae [25]. Výhodou tohoto formátu je kompatibilita, takže uživatelé mají větší škálu aplikací, ve kterých s COLLADOU mohou pracovat. COLLADA patřila mezi špičku 3D formátů, ale poslední dobou se uživatelé obrací k novějším technologiím, jako je např. dříve zmíněné FBX [18]. COLLADA společně s OBJ spadá do neutrálních formátových souborů.

4.4 Graphics Library Transmission Format glTF

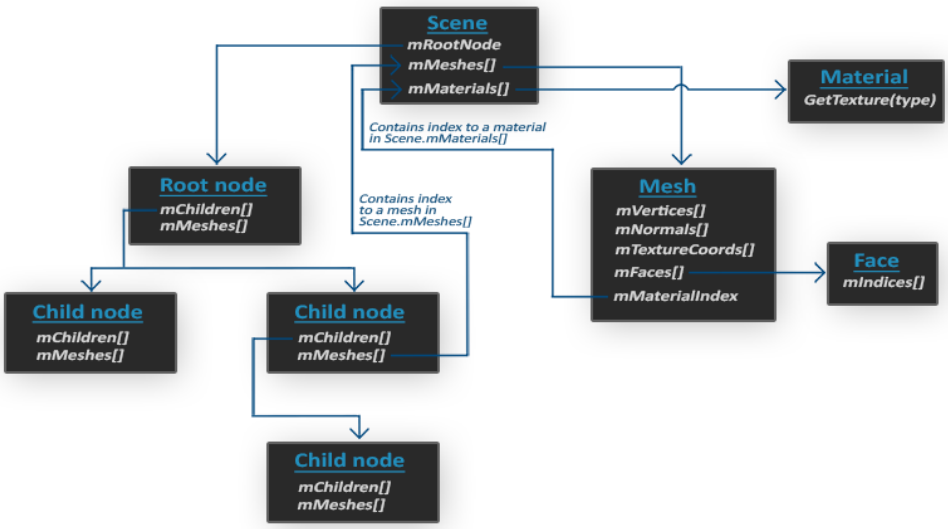
Graphics Library Transmission Format (glTF) je open-source souborový formát spravovaný společností Khronos Group, zahrnující ve svých souborech informace scény jako je osvětlení nebo kamera, ale také geometrické informace modelu a jeho animace [21]. Formát glTF používá pro své soubory dvě koncovky: .gltf a .glb. Rozdíl mezi těmito formáty je jejich reprezentace, .gltf je textový formát ve formě JSON, načež .glb je jeho binární forma. Tyto formáty se liší rychlostí a velikostí, .glb je menší formát díky své binární reprezentaci, ale zase se načítá výrazně pomaleji [26]. Khronos Group, autor formátů .gltf a .glb, stojí za vývojem grafického rozhraní OpenGL, zajišťující kompatibilitu mezi technologií OpenGL a zmíněnými formáty.

4.5 Knihovna Assimp

Assimp (neboli Asset-Importer-Lib) je OpenGL knihovna určená pro nahrávání scén z různých 3D formátů. Assimp podporuje nahrávání geometrie objektů, textur animací, a animačních dat jako jsou kosti, klíče a nody. Všechna data jsou získána ze struktury AIScene reprezentující importovanou scénu [27].

4.5.1 Scéna

Scéna je rozdělena do hierarchické struktury, přičemž v jejím kořeni je třída AIScene, zahrnující root node pro animaci, pole meshů a meteriálů.



Obrázek 6 – Zjednodušená struktura Assimp scény [28]

Každý mesh má přístup ke geometrickým datům modelu jako jsou vertexy, normály, souřadnice do textury, indexy materiálů a strany uchovávající pole indexů, které určují pořadí vertexů. Indexy materiálů slouží pro přístup k poli materiálů uchovaných ve třídě AIScene, každý materiál může uchovávat data jako je barva nebo mapy textur [28]. Animace jsou uchovávány v nodech, kde každá instance nodu má přístup ke svým potomkům. Nody uchovávají animační data jako jsou například transformace.

5 Skeletální animace

Skeletální animace postav i dalších objektů je silně inspirována anatomií lidského těla, z čehož vyplývá název skeletální (z latiny skelet = kostra). Stejně jako u živočichů i kostra (skelet) animovaného modelu je složena z kostí a kloubů [29]. Kosti jsou rozděleny do stromové hierarchie, kde je mezi jednotlivými kostmi uplatňován vztah předek – potomek, principem je vzájemná závislost dvou sousedních kostí kostry [30]. Základním pravidlem je: pokud se hne předek, hne se i potomek, respektive všichni potomci [31]. Kosti a klouby nejsou ale jediné komponenty, které jsou součástí 3D animovaného objektu.

5.1 Komponenty kostry

Skeletání 3D model je složen z kostí, kloubů a meshe představující kůži modelu. Ani jedna část není soběstačná jako samostatný prvek, k úplné animaci je potřeba jejich propojení. Povrch kůže je definován jako soubor bodů spojených do polygonů , které slouží k zobrazení požadovaného vzhledu objektu [31]. Klouby jsou určeny k ohybu kostí, které jsou s nimi spojeny [32]. Model kůže je rozdělen na segmenty a ty jsou přiřazeny k jednotlivým kostem. Kost jako taková má pouze informace o své lokaci a orientaci, pro mapování vztahu mezi kostí a kůží se používá metoda zvaná skinning.

5.1.1 Vertex

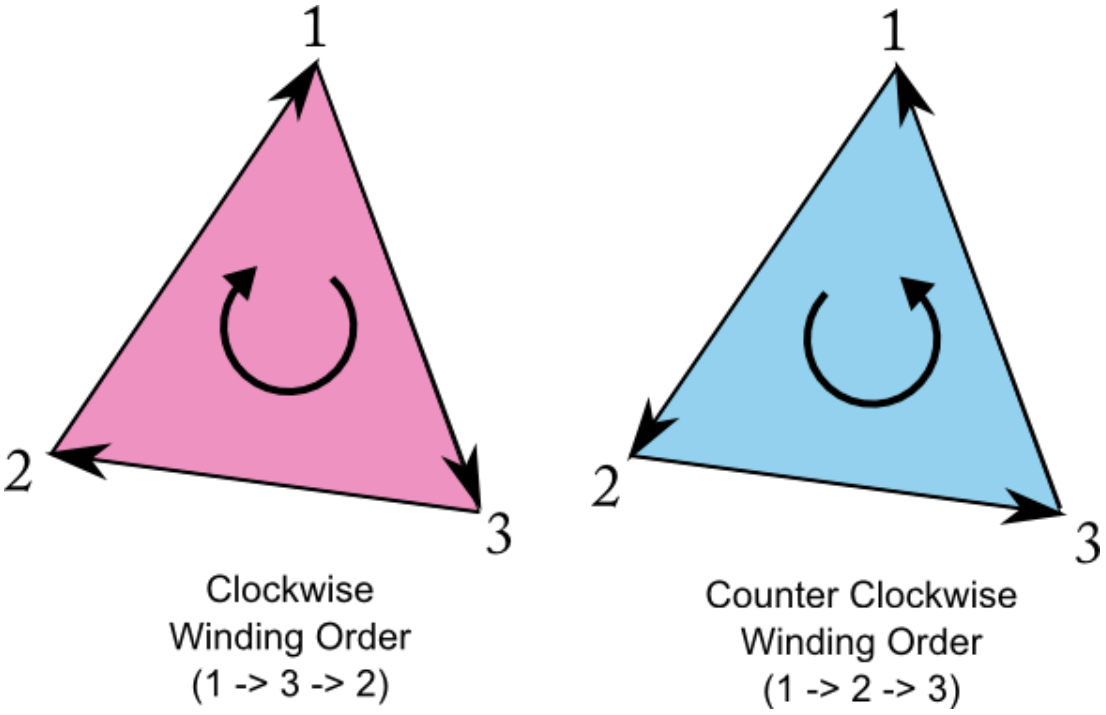
Základním kamenem pro tvorbu jak grafických primitiv, jako je např. bod, úsečka či polygon, tak i celých objektů, je geometrický bod neboli vertex [33]. Vertex uchovává informace nutné k jeho manipulaci a následnému vykreslení, mezi tyto vlastnosti patří např.: polohové souřadnice dané dimenze, hodnota barvy, normálový vektor nebo souřadnice do textury. Normála slouží při renderování modelu pro určení hodnoty osvětlení [34].

5.1.2 Mesh

Mesh je možno popsat jako soubor bodů a jejich propojení do polygonů. Mezi využívané polygony je možné zařadit např. trojúhelníky nebo čtyřúhelníky [35]. K vytvoření meshe jsou potřeba dvě datové struktury: seznam vrcholů označovaný jako vertex list či vertex buffer, a seznam polygonů, nazývaný jako polygon list nebo

index buffer. Vertex list je hlavní buffer informací, kde jsou uloženy všechny atributy zmíněné v podkapitole popisující vertexy. V seznamu polygon je uloženo propojení vrcholů důležité pro složení geometrických primitiv a jejich následnému vykreslení. Pořadí vrcholů je určeno index bufferem obsahujícím indexy do vertex listu.

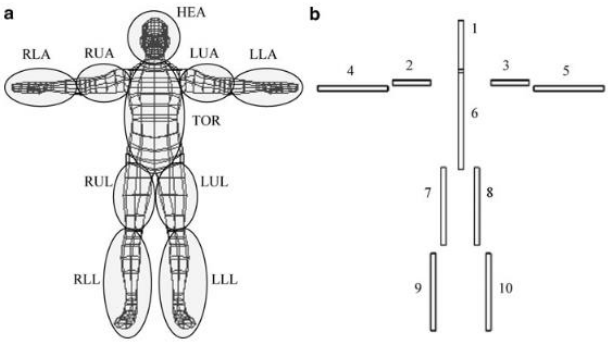
Pořadí vykreslovaných vertexů je důležité pro určení orientace vzniklých ploch, například trojúhelníků. Pokud existuje polygon s indexy bodů, tak jsou dvě možnosti, jak daný trojúhelník zapsat [34] [36] [37]. Posloupnost určuje orientaci normálového vektoru plochy, tedy její přední stranu. Standartním způsobem je zápis bodů proti směru hodinových ručiček (CCW), tj. podle pravidla pravé ruky. Pokud je taková plocha natočena k uživateli, je viditelná.



Obrázek 7: Zápis vertex po a proti směru hodinových ručiček. [38]

5.1.3 Kost

Animace všech vertexů a meshů zvlášť by sama o sobě byla velice výpočetně náročná, proto se hodí seskupovat vertexy do skupin a manipulovat následně s nimi. Stejně jako při pohybu lidského těla i 3D model obsahuje kosti, které určují jeho pohyb. Kosti značně zjednodušují proces animace, transformace je možno vypočítat pro každou kost a následně ji aplikovat na jí poddružené vertexy. Na *Obrázku 8* je uveden příklad umístění kostí v 3D modelu.



Obrázek 8: (a) Skupiny vertexů podle vlivu kostí (b) Rozvržení kostí [29]

Jeden vertex může být ovlivňován více kostmi a jejich celková váha se musí rovnat 1 (tedy 100 %). Váha je desetinné číslo pohybující se v intervalu <0;1>, které určuje míru vlivu dané kosti na konkrétní vertex.

Kosti nejsou fyzické struktury jako například vertexy, a tudíž nejsou součástí meshe. Kost je pouze pomocný nástroj pro tvorbu animací, takže při renderování objektu nebude vidět. Kost representuje oblast vlivu konkrétní transformace v rámci modelu [29].

5.2 Rigging

Rigging je proces, při kterém se objektu složeného z geometrických primitiv přiřadí ovládací systém umožňující animaci objektu. Tyto systémy se mohou lišit podle složitosti, jednoduchými vztahy předek-potomek počínaje a systémem simulujícím svalstvo konče. Rigging zajišťuje přiřazování kostí konkrétnímu modelu. Výsledek tohoto procesu je rig neboli kostra, která je schopná pohybu [2], [39].

5.3 Skinning

Výstupem rigging procesu je model s kostrou obsahující kosti, klouby a mesh. Pokud by teď byly aplikovány geometrické transformace na kosti a klouby, jediné, co by podléhalo animaci by byly právě kosti a klouby. Kosti by se sice pohybovali, jak bylo určeno, ale mesh a textura na něm by animována nebyla. Aby bylo možné animovat i mesh, je potřeba použít metodu zvanou skinning [32]. Skinning je proces přiřazení kostí jednotlivým meshům tak, aby každá kost měla vliv na část objektu [40].

5.3.1 Simple Skinning

Simple skinning prochází všechny vertexy meshe a mapuje je k jednotlivým kostem [41]. To znamená, že každý skinovaný vrchol je přiřazen právě k jedné kosti. Na rozdíl od následujících metod tato metoda nedodává animaci přirozený vzhled, neboť nebere v úvahu vliv okolních kostí [34].

5.3.2 Linear Blend Skinning

Linear blend skinnig (dale jen jako LBS) se považuje za standardní řešení pro přiřazování vertexů kostem a mapování meshe [41]. Tento algoritmus je rozšíření simple skinningu v tom, že vrchol je přiřazen k více kostem a s různou vahou. Tento faktor určuje vliv jednotlivých kostí na pohyb části meshe. Vzorec pro vypočítání výsledné pozice vypadá takto:

$$v_i^{'} = \sum_{j=1}^{m} w_{i,j} W_j B_j^{-1} v_i$$

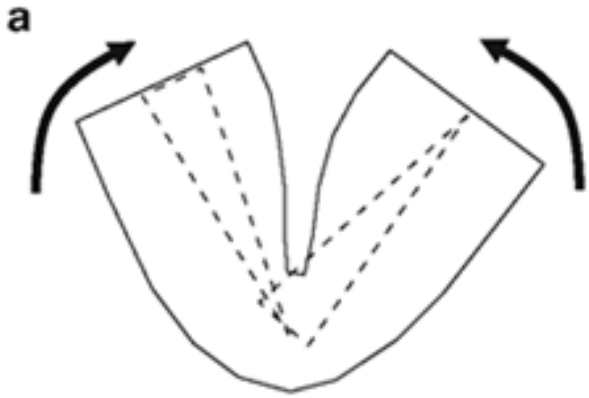
Rovnice 1: výsledná pozici vertexu [42]

Každý vertex je ovlivňován několika kostmi, které se podle váhy podílejí na výsledné poloze vrcholu a ta se projevuje na její následné změně tvaru. Proměnná v_i je výsledná pozice vertexu, W_j představuje transformační matici, která náleží konkrétní kosti ve své aktuální pozici. B_j^{-1} je pozice kosti ve své vázací póze (anglicky bind pose). Písmeno j označuje používanou kost [42].

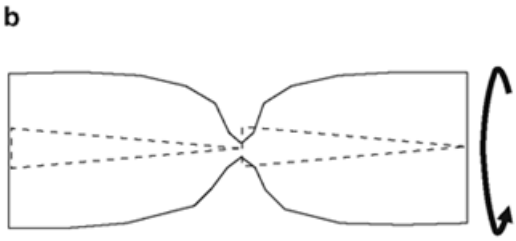
LBS ale není ideálním řešením pro skinning. Tento algoritmus je vhodný pro transformaci mezi polohami, které od sebe nejsou příliš rozdílné. Pokud by uživatel chtěl provádět interpolaci mezi významně odlišnými polohami, mohou nastat vizuální artefakty [42] [43].

Existují dva druhy vizuálních artefaktů. Prvním z nich je collapsing elbow effect. Tento efekt vzniká, když úhel kloubu, který svírají dvě kosti, je příliš malý. Vertexy jedné z těchto kostí se protnou s vertexy kosti předcházející, tímto animace ztrácí na věrohodnosti, neboť končetiny jsou nepřirozeně deformovány [44] [45]. Druhým efektem je candy-wrapper efekt. Jak název vypovídá, tento jev se projevuje na

končetinách zakroucením, stejně jako obal od bonbónu. Paže postihnutá tímto jevem se otočí o 180 stupňů, a proto připomíná zkroucený obal.



Obrázek 9: Collapsed elbow [45]



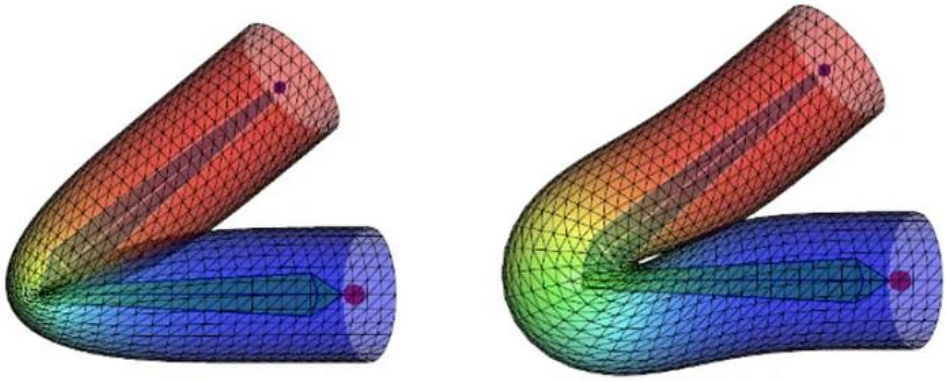
Obrázek 10: Candy-wrapper [45]

5.3.3 Dual quaternion skinning

Popsané problémy při skinningu postavy jsou způsobeny lineární podstatou zmíněných algoritmů. Je možné nahradit tyto lineární algoritmy metodou, které se říká Dual quaternion skinning (dále jen jako DQS). DQS funguje na stejném principu jako LBS, tedy také bere v úvahu váhu každé kosti při kalkulaci výsledné rotace. Na rozdíl od LBS tento algoritmus transformuje rotace do tzv. kvaternionů, které jsou uchovány a použity pro výpočet výsledné transformace, místo použití rotační matice jako takové [10].

LBS využívá pro výpočet animace mezi jednotlivými polohami lineární interpolaci. Výhodou lineární interpolace je snadná implementace a efektivita při jednoduchých transformacích, při složitějších transformacích ale dochází k deformaci. Při lineární interpolaci dvou bodů vzniká přímka a výsledný bod leží na ní. Při zvolení tohoto postupu může dojít např. ke kolapsu lokte, kde se dva meshe rotujících kostí

promítnou do sebe [41] [43]. Body transformované pomocí DQS neleží na přímce ale na kruhovém oblouku, tím pádem nedochází k nechtěnému smrštění meshů [43]. Ani DQS algoritmus není bezchybný, při ohybu mohou na kloubech vzniknout deformace ve formě boulí, jak je ukázáno na *Obrázku 11* a *Obrázku 12*.



Obrázek 11: Deformace loktu: smrštění meshů a boule [1]

Obrázek 12: Deformace meshe: boule na kloubu [1]

Zmíněné metody skinningu nejsou však jediné. Existuje celá řada metod jak lineárních, tak nelineárních. Jejich hlavním cílem je eliminovat vizuální artefakty. Jednou ze slibných metod je takzvaný spline skinning, který je stejně jako LBS lineární algoritmus. Spline skinning výrazně omezuje výskyt candy-wrapperu. Samozřejmě i na

poli nelineárních algoritmů je snaha eliminovat jakékoli artefakty, a proto se vývojáři věnují i rozšířením DQS [43].

5.4 Weighting

Weighting (nebo také weight painting) je proces, při kterém se mapují vlivy kostí na mesh a jeho vertexy [46]. Díky přidaným váhám může animátor kontrolovat deformace objektu způsobené pohybem skinnovaného modelu [47]. Díky kontrole získané pomocí přidělování vah, má animátor plnou kontrolu nad přechody modelu mezi jednotlivými pózami při průběhu animace. Pokud jsou váhy přiřazeny korektně, tak je možné předejít jevům jako jsou například nechtěné skoky nebo zkreslení [47]. Weight painting se nejvíce projeví při animování obličejů, kde je potřeba zachytit mnoho nepatrných pohybů. Animování obličejů je náročné, hlavně kvůli velkému počtu kostí podílejících se na jeho animaci [47].

6 Návrh řešení a implementace

6.1 Úvod

Hlavní nástrojem pro implementaci skeletální animace je C++ knihovna Assimp [48]. Assimp zvládá pracovat s mnoha formáty souborů a je navržen jako univerzální knihovna. Bohužel opravdová podpora knihovny Assimp jednotlivých formátů není stejná, např. při načítání proprietárních formátů jako je FBX může dojít k chybnému importu modelu. Velká část této práce se odvíjí od pochopení a následné implementaci metod vyžívající funkcionalitu této knihovny, protože uživatel se musí přizpůsobit strukturám Assimp knihovny a jejímu způsobu práce s daty.

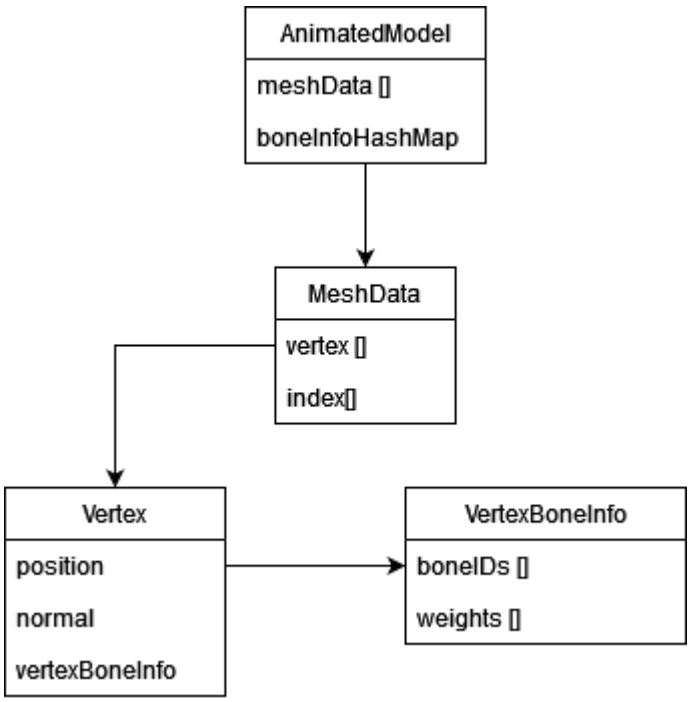
Celý projekt je strukturován do několika fundamentálních tříd, které se snaží izolovat jednotlivé úkony a tím zachovat princip objektového přístupu. Mezi hlavní třídy patří:

- *Renderer*: Obstarává vykreslení dat.
- *Animation*: Extrahuje animační data z Assimp
- *Animator*: Vede chod animace, aktualizuje jednotlivé animační snímky
- *MeshLoader*: Načitá objektová data z Assimp
- *AnimatedModel*: Uchovává a zpracuje objektová data

K tvorbě projektu byly kromě knihovny Assimp použity i třídy z balíčku transforms, které se starají o matematické struktury.

6.2 Načítání mesh struktur

Objekt se skládá ze skupiny meshů, každý z nich uchovává geometrická data jako jsou např. vertexy, normály nebo informace kostí. Pro použití dat z Assimpu je nutné vytvořit náležité struktury, jejich propojení je možné vidět na diagramu *Obrázku 13*.



Obrázek 13: Struktura tříd extrahující geometrická data (Zdroj: autor)

Všechny informace o modelu jsou obsaženy v AIScene. Použitím cyklu je možné ji projít, získat z ní jednotlivé meshe a ty následně uložit do vlastní struktury *MeshData.*

K vertexovým datům pozice a normály je možné přistoupit přímo pomocí získaného Assimp meshe. Pro extrahování indexů je nutné nejdříve přistoupit k jednotlivým stěnám a teprve z nich je možné získat indexy. Metody pro extrahování vertexů *processVertices* a indexů *processIndicees* jsou založeny na stejném principu. Do obou metod vchází prázdný list, který je pomocí cyklu procházejícího aktuální mesh plněn příslušnými hodnotami.

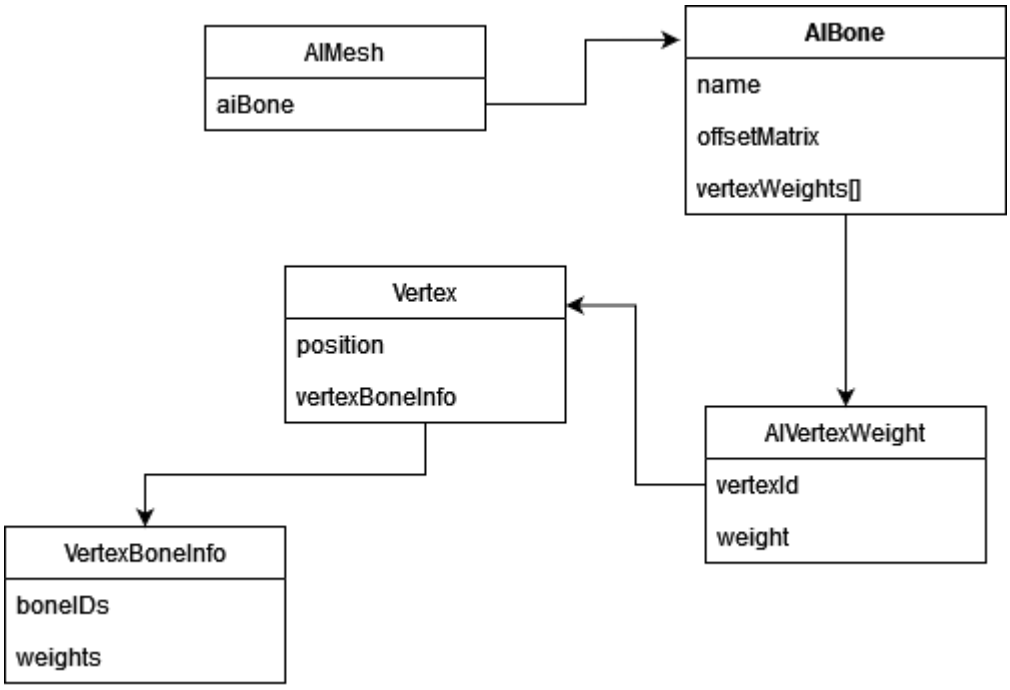
6.2.1 Vertex

Pro reprezentaci vertexů slouží třída *Vertex*, která uchovává informace o pozici, normále a strukturu *VertexBoneInfo.* Tato struktura ukládá názvy kostí ovlivňující daný vertex a jejich váhy pro skinning. Dále se ve třídě nachází konstanta *MAX_BONE_INFLUENCE,* která udává maximální počet kostí, jež mohou mít vliv na jeden vertex. Animace implementovaná v tomto projektu pracuje s nejvýše čtyřmi kostmi pro vertex.

6.2.2 Extrahování kostí

Cílem toho kroku je přiřadit každému vertexu kosti s jejich korespondujícími váhami. V *processVertices* se nachází metoda *setBoneInfoDefault*, která nastaví výchozí hodnoty pro pole *boneIDs* a *weights.* Pro jednotlivá pole jsou nastaveny výchozí hodnoty, které budou postupně nahrazeny hodnotami skutečnými. Výchozí hodnota kosti značí, že daná pozice v poli je prázdná neboli se na ní nenachází žádná kost. S těmito hodnotami se pracuje ve vertex shaderu.

Po načtení všech vertexů se volá metoda *getBoneInfoForVertices* získávající skutečné kosti a váhy. Získávání kostí je řešeno zde a získávání jejich vah se nachází v pomocné metodě *extractWeights.* Před ukázáním samotného kódu je potřeba pochopit provázání daných struktur v Assimp a jak jsou extrahovaná data ukládána.



Obrázek 14: Propojení tříd pro extrahování kostí (Zdroj: autor)

Na *Obrázku 14* je vidět provázání Assimpových tříd s vlastními třídami *Vertex* a *VertexBoneInfo* uchovávající vertexová data. Na rozdíl od pozic či normál, data kostí nemohou být přidána při extrahování vertexů. Propojit vertex s kostí a její váhou jde jenom pomocí atributu *vertexId.* V kódu se porovnává *vertexId* s pozicí vertexu v listu vertexů získaného z *processVertices.*

Jedinečnost kostí je v Assimp zaručena pouze jejich jménem, kosti nemají žádný jiný unikátní identifikátor. Používání jmen je vhodné pro propojení kostí s patřičnými animovanými nody, jména se ale nedají použít ve vertex shaderu, kde je potřeba ke

každé kosti přistoupit zvlášť. Je možné k daným kostem přistoupit pomocí unikátního id, o které se stará *boneCounter.* Počítadlo *boneCounter,* určující unikátní id každé kosti, je zvětšeno vždy po přidání nové kosti. Pokud daná kost už v HashMapě existuje, tak *boneId* je rovno id stávající kosti.

AIBone má dva atributy, název kosti a její offsetová matice. Offsetová matice se používá k transformaci kosti ze souřadnic meshe do lokálních souřadnic kosti [49]. Pro uložení dat z AIBone je v projektu vytvořena struktura *BoneInfo* uchovávající id a zmíněnou matici.

Z diagramu zobrazujícího vztah mezi kostmi, váhami a vertexy je zřejmé. že propojení mezi vertexem a kostí je umožněno třídou AIVertexWeight reprezentující váhu, konkrétně jejím atributem *vertexId*. Metoda *extractWeights* zajišťuje právě toto propojení, nachází patřičné vertexy s kostmi a jejich váhami. Následně je použita metoda *setVertexBoneInfo* plňící dané vertexy získanými daty.

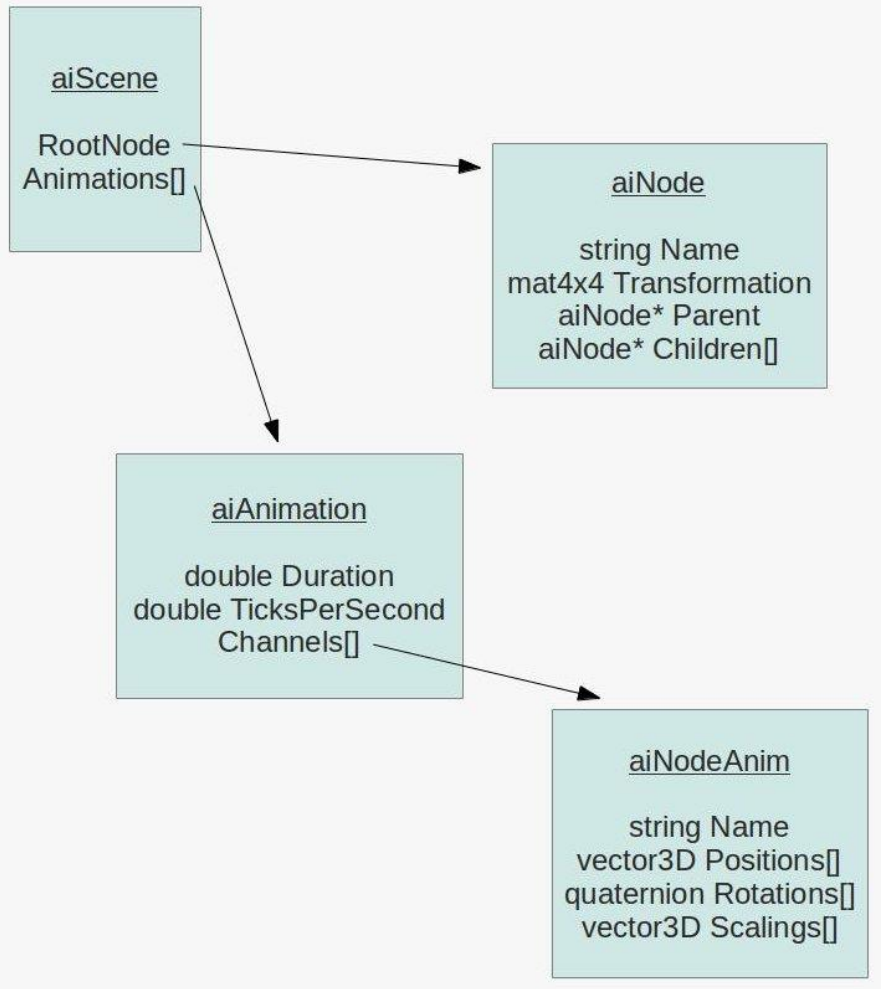
6.2.3 Model a mesh

Nahraný 3D model se může skládat z několika mesh struktur, proto je nutné reprezentovat každý mesh zvlášť a následně mít strukturu, která zahrnuje všechny tyto meshe. V projektu se tímto úkolem zabývají třídy *MeshData*, reprezentující jednotlivé meshe, a *AnimatedModel,* reprezentující celý model. *MeshData* má dva atributy ve formě listů uchovávající vertexy a indexy.

Animated model uchovává pole všech meshů objektu a HashMapu *boneInfoHashMap.* Hlavním účelem třídy *AnimatedModel* je uchování právě zmíněné *boneInfoHashMap,* která bude použita pro propojení kostí a animovaných nodů.

6.3 Zpracování animace

Všechna data, a to jak ta animační, tak ta geometrická se nachází v objektu AIScene, který je dále možné dělit. Doposud byl hlavním zdrojem dat objekt AIMesh, ze kterého se získávaly informace o mesh struktur a kostech. Animační data jsou uchována ve strukturách ukázaných na *Obrázku 15*.



Obrázek 15: Popis rozvržení animační dat [50]

AINode je první struktura navázána přímo na objekt AIScene, jejími atributy jsou jméno daného nodu, transformační matice 4x4 a odkazy na předka a potomky daného nodu. Transformační matice vyjadřuje transformaci kosti relativní k transformaci svého předka. Pro určení času a délky animace se používají atributy *TicksPerSecond* a *Duration.* Posledním atributem je pole animačních kanálů, skrz něj je možné se dostat ke struktuře AINodeAnim uchovávající animační snímky pro translaci, rotaci a změnu měřítka. Pro každou ze zmíněných struktur, kromě AIScene, je v práci vytvořena třída sloužící jako kontejner jejích hodnot.

Pro reprezentaci snímků jsou v balíčku *keys* tři třídy zastupující jednotlivé typy animačních snímků: *KeyPosition, KeyRotation* a *KeyScale.* Pro změnu měřítka a translaci je transformace reprezentována vektorem a pro rotaci je použit kvaternion. Získání snímků z knihovny Assimp je provedeno v třídě *Bone.* Tato třída nastavuje animační snímky a následovně provádí jejich interpolaci, která je využita při aktualizaci aplikace.

Podobně jako tomu bylo u získání vertexů, i zde je potřeba vytvořit cyklus, který tentokrát prochází snímky z Assimpu. Hodnoty snímků se ukládají do patřičné *Key*, kde je nutné uchovat transformaci a časové razítko. Procházení snímků a extrahování struktur obstarává patřičná *set* metoda.

6.3.1 Animace

Účelem třídy *Animation* je uchovat dříve zmíněná data z AIAnimation a vytvořit hierarchii nodů, která kopíruje strukturu získanou z knihovny Assimp. Při inicializaci třídy se provádí sestavení animačních nodů a získání časových údajů jako *Duration* a *TickPerSecond*. V konstruktoru se nachází dvě metody, a to *processNodes* spolu s *readMissingBones.* Tyto metody pracují s *Node*, která je reprezentací třídy Assimp AINode. Třída *Node* obsahuje již zmíněné atributy z AINode jako je transformační matice, jméno a seznam potomků.

K procházení nodů se nachází v *Animation* metoda *processBones,* která rekurzivně prochází potomky AINode, extrahuje její hodnoty a následně je ukládá do zmíněné Node struktury. Procházení stromu začíná u kořene a postupně směřuje ke všem potomkům. Metoda vrací hodnotu typu *Node* s odkazy na všechny své potomky.

Druhou metodou v konstruktoru třídy *Animation* je *readMissingBones,* která se snaží najít kosti nezachycené při načítání meshe. Pro interpolaci snímků, a tudíž aktualizaci animace se používají jen nody, které v modelu figurují zároveň jako kosti. Občas se ale stává, že se animační data objeví i na nodu, který původně kostí není. Pro tento případ je nutné najít takovéto nody a následně je přidat do HashMapy *boneInfoHashMap* animovaného modelu.

```
private void readMissingBones(AnimatedModel model){
    PointerBuffer aiChannels = aiAnimation.mChannels();
    int boneCount = model.getBoneInfoHashMap().size();

    for (int i = 0; i< aiAnimation.mNumChannels();i++){
        AINodeAnim aiNodeAnim = AINodeAnim.create(aiChannels.get(i));
        String boneName = aiNodeAnim.mNodeName().dataString();

        if (model.getBoneInfoHashMap().get(boneName) == null){
            BoneInfo boneInfo = new BoneInfo(boneCount,null);
            model.getBoneInfoHashMap().put(
               aiNodeAnim.mNodeName().dataString(),boneInfo);
            boneCount++;
        }
    }
}
```

Obrázek 16: Extrahování kostí modelu (Zdroj: autor)

Metoda přijímá referenci na animovaný model a následně prochází všechny animační kanály pro danou animaci. Pokud se najde animovaný node, který se nenachází v *boneInfoHashMap,* tak tam bude přidán s unikátním id.

6.3.2 Animátor

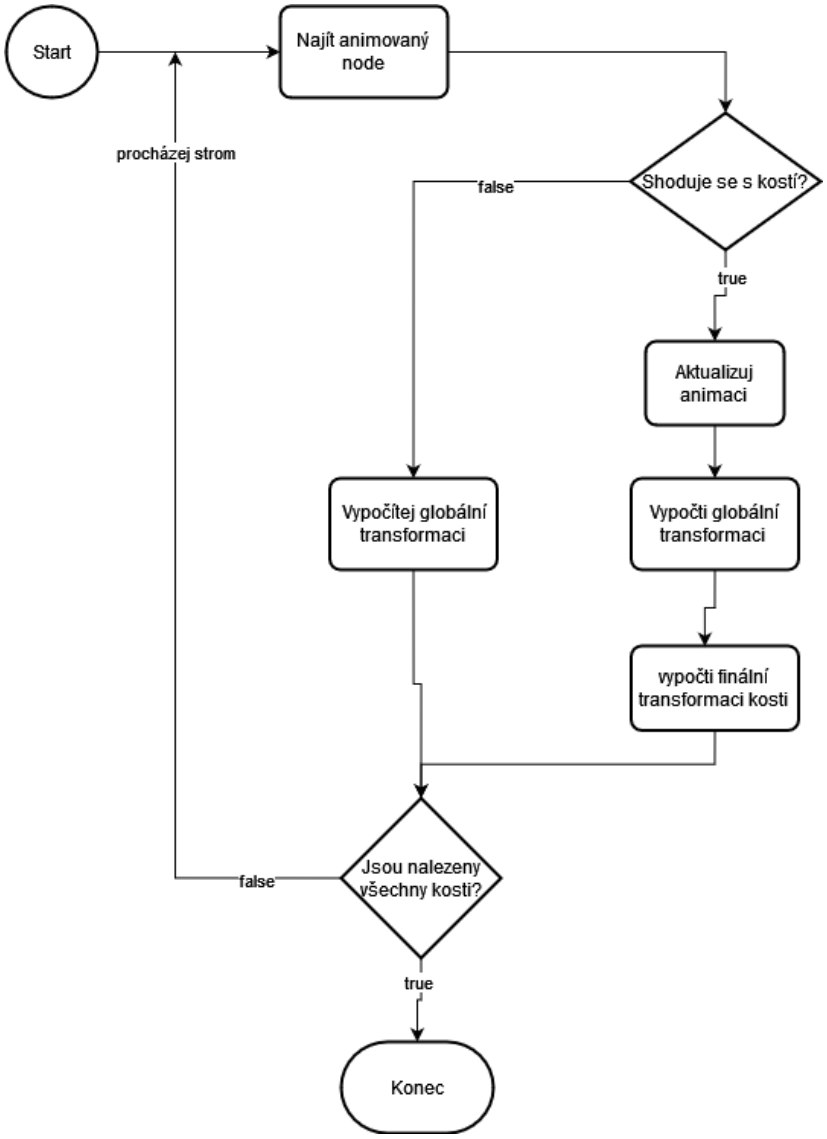
Animator je třída, jež řídí chod animace, stará se její o aktualizaci a dodává třídě *Renderer* výsledné transformační matice pro daný snímek. *Animator* aktualizuje animaci pomocí metody *updateAnimation* přijímající čas v sekundách, který je následně převeden na tzv. tiky vzniklé pronásobením atributů *timeInTicks* a *ticksPerSecond.*

```
public void updateAnimation(double timeInSeconds){
    if (animation != null){
        double ticksPerSecond = (animation.getTicksForSecond() != 0
        ? animation.getTicksForSecond() : 25.0d);
        double timeInTicks = timeInSeconds * ticksPerSecond;
        double animationTime = (timeInTicks % animation.getDuration());
        calculateTransformMatrices(animation.getRootNode(),
                        new Mat4Identity(),animationTime);
    }
}
```

Obrázek 17: Získání aktuálního času animace (Zdroj: autor)

Metoda má celkem tří parametry, a to kořenový node, matici reprezentující transformaci předka a čas animace. Cílem *calculateBoneTransforms* je získat finální transformace kostí, které budou využity pro animování postavy ve vertex shaderu.

Metoda musí projít hierarchii nodů, porovnat je s kostmi a vytvořit globální transformaci. Tato transformace je součinem předků konkrétní kosti, tedy na konkrétní kost musí být aplikovány transformace všech jejích předků. Teoretický rozbor metody je vidět na *Obrázku 18*.



Obrázek 18: Rozbor metody calculateTransforms (Zdroj: autor)

Každý procházený node má svojí transformační matici, která bude přepsána interpolovanou lokální transformací, pokud se jméno daného nodu shoduje s názvem kosti uloženým v *boneInfoHashMap*. Výpočet aktuální transformace dané kosti obstarává metoda *update* třídy *Bone.*

```
if (boneInfo != null){
    Bone bone = new Bone(aiNodeAnim);
    bone.update(animationTime);
    nodeTransform = bone.getLocalTransform();
}
```

Obrázek 19: Získání interpolované nodeTransform (Zdroj: autor)

Globální transformace je součin současné *nodeTransform* s transformací jejího předka.

Nejdůležitější částí této metody je správné skládání matic. Cílem skládání je vrátit souřadnice objektu z lokálních souřadnic zpět do souřadnic světa. V práci je tento úkon proveden následujícím způsobem.

```
if (boneInfo != null){
    Mat4 offset =  model.getBoneInfoHashMap()
                  .get(nodeName).getOffsetMatrix();
    Mat4 finalBoneTransform;
    if (offset != null && useOffset)
         finalBoneTransform = globalInverseTransform
         .mul(globalTransform).mul(offset);
    else
        finalBoneTransform = globalInverseTransform.mul(globalTransform);

    finalTransforms[boneInfo.getId()] = (finalBoneTransform);
}
```

Obrázek 20: Skládání transformačních matic (Zdroj: autor)

Nejdříve se získá offset matice z konkrétní kosti, ta může, ale také nemusí existovat, jelikož ne všechny kosti offsetovou matici mají. Finální transformace kosti je součin globální transformace spolu s offsetovou a globálně inversní maticí. Offsetová matice transformuje souřadnice do lokálních souřadnic relativních k příslušné kosti. Nejdřív se tedy aplikují transformace v lokálních souřadnicích a pak budou souřadnice transformací navráceny do souřadnic světa pomocí *globalInverseTransform.* Nakonec se finální transformace přidá do pole *finalTransforms* na pozici určenou podle id dané kosti.

V metodě *calculateTransformMatricies* se nachází dvě pomocné metody: *findAnimNode* a *update*. Metoda *findAnimNode* prochází animační kanály objektu AIAnimation a vrací proměnnou typu AINodeAnim, pokud se shoduje s jménem zaslaným v hlavičce metody.

6.3.3 Interpolace animačních snímků

Metoda *update* třídy *Bone* zprostředkovává interpolovanou transformační matici pro daný čas animace, její složení se skládá ze tří interpolačních metod pro dané transformace: translace, rotace a změna měřítka. Lokální transformace dané kosti se skládá ze součinu těchto maticí.

```
public void update(double animationTime){
    // interpolate position
    Mat4 translation = interpolatePosition(animationTime);
    …

    localTransform = translation.mul(rotation).mul(scale);
}
```

Obrázek 21: Finální interpolovaná matice složená z translace, rotace a změny měřítka. (Zdroj: autor)

Všechny interpolační metody jsou založeny na stejném principu, ale metoda pro interpolaci rotace se vymyká oproti zbytku použitím kvaternionů a nutnosti využití struktur a metod z Assimp pro korektní interpolaci. Metody přijímají čas animace, který je následně využit pro nacházení odpovídajících klíčů ze třídy *KeyPosition/Scale/Rotation.*

```
private Mat4 interpolatePosition(double animationTime) {
    int indexStart = getPositionIndex(animationTime);
    int indexEnd = indexStart + 1;
    KeyPosition startPos = positions.get(indexStart);
    KeyPosition endPos = positions.get(indexEnd);

    double factor = getFactor(startPos.getTimeStamp(),
                    endPos.getTimeStamp(),animationTime);

    Vec3D start = startPos.getPosition();
    Vec3D end = endPos.getPosition();
    Vec3D delta = end.sub(start);
    Vec3D interpolatedPosition = start.add(delta.mul(factor));

    return new Mat4(interpolatedPosition.getX(),
                    interpolatedPosition.getY(),
                    interpolatedPosition.getZ());
}
```

Obrázek 22: Metoda interpolace pozice (Zdroj: autor)

Metoda začíná tím, že najde index počátečního a konečného klíče a tento index je použit k získání správných *KeyPosition.* Hodnota proměnné *factor* se nachází v intervalu <0;1> určující, který snímek má na výslednou pózu větší vliv. Následně jsou získány počáteční a konečné vektory pozice. Lineární interpolace použita v této práci je řešena získáním rozdílu mezi konečnou a počáteční pozicí a poté přičtením počáteční pozice k součinu proměnných *factor* a *delta.* Metoda vrací translační matici založenou na interpolované pozici.

Pro získání finální rotace není použita standartní lineární interpolace, tak jako tomu je u translace a změny měřítka, ale je použita interpolace sférická, zprostředkovaná Assimp metodou *aiQuaternionInterpolate.* Pro Assimp metodu je nutné pracovat s Assimp strukturami, proto jsou zde používány struktury AIQuaterion. Výsledkem této metody je transponovaná rotační matice. Matice musí být transponovaná, protože Assimp pracuje s maticemi jinak než balíček transforms. Matice získáné z Assimp a matice získaná z *toRotationMatrix* mají prohozené řádky a sloupce.

6.4 Zobrazení dat

Jádro animace spočívá ve vertex shaderu, který se stará o provádění jednotlivých transformací. V projektu jsou tyto shadery dva, jeden zobrazuje anivaný

model a druhý animovaný skelet objektu. Vertex shader pro zobrazení modelu je vidět na *Obrázku 23*.;

```
for (int i = 0; i < MAX_WEIGHTS; i++) {

    if (int(in_boneIDs[i]) != -1) {

        vec4 localPosition =
            boneTransforms[int(in_boneIDs[i])] * vec4(in_position, 1.0f);

        position.xyz += localPosition.xyz * in_weights[i];
    }
}
```

Obrázek 23: Aplikování jednotlivých kostí na daný vertex (Zdroj: autor)

Cyklus nacházející se ve vertex shaderu zajišťuje aplikování transformací na daný vertex. Na každý vertex může mít vliv až čtveřice kostí naráz. Při aplikování všech transformací je nutné přičítat jednotlivé lokální pozice k pozici finální. Finální pozice je tedy součtem pozic získaných po aplikaci všech čtyřech kostí. Podmínka je použita k ověření, jestli se na dané pozici pole nachází opravdová nebo jen výchozí, tedy žádná, kost. Váha zde určuje, jakou intenzitu má konkrétní transformace. Na výslednou pozici jsou následně aplikována pohledová a projekční matice.

Fragment shader je zde použit pro výpočet finální barvy a implementaci difúzního osvětlení. Difúzní osvětlení je získáno skalárním součinem vektoru normály a pohledu, přičemž oba vektory musí být v normalizované formě.

6.4.1 Renderer

Třída, která všechny předchozí pojí dohromady je třída *Renderer,* která se stará nejen o předávání dat do vertex bufferu, ale také vykresluje scénu a aktualizuje animaci. Nejdůležitější jsou zde metody *init* a *display* obstáravající o chod celé aplikace.

Metoda *init* se volá jen při spuštění aplikace, jejím cílem je načíst model, naplnit příslušné buffery a inicializovat shadery. Aplikace dokáže zobrazit nahranou animaci a skelet animovaného modelu. Pro každý z těchto módů je vytvořena metoda vytvářející buffery a vlastní shader. Do vertex shaderu se kromě atributů posílají i uniformní proměnné jako pohledový vektor, projekční a pohledová matice, a nakonec jednotlivé transformace.

Metoda *loadModel* nahrává model a inicializuje proměnné animace a animátora. Nahrávané modely jsou uložené v *resources* složce a je k nim přistupováno relativní

cestou pomocí systémové třídy File. Metoda *createBuffers* vytváří buffery pro animovaný model a určuje vertexové atributy. Na každý vertex spadá celkově sedmnáct floatů, kde tři jsou pro pozici, tři pro barvu, čtyři pro id kostí, čtyři pro jejich váhy, a nakonec tři floaty pro normály.

Vykreslování modelu, aktualizování animace a plnění bufferu je prováděno v metodě *display*, která se volá s každým snímkem. Prvním úkonem pro aktualizaci animace je zde vypočítání času animace vyjádřeného v sekundách. Právě proměnná *timeInSeconds* je používána metodou pro aktualizaci animace.

```
long currentTime = System.currentTimeMillis();
long delta = currentTime - lastTime;
if (anim) {
    startUpTime += delta * speed;
}
double timeInSeconds = ((double)startUpTime / 1000.0);
lastTime = currentTime;
```

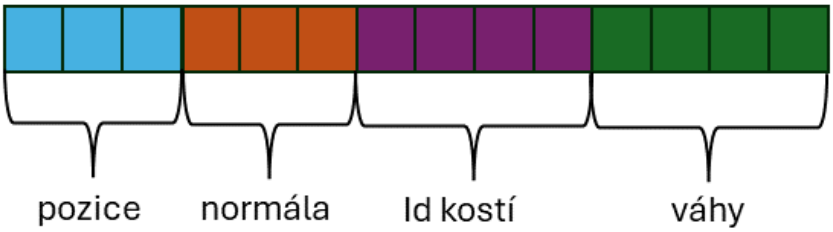
Obrázek 24: Získání času v sekundách pro aktualizaci animace. (Zdroj: autor)

Po aktualizaci animace jsou získány ze třídy *Animator* finální transformace, které jsou transformovány z původního pole matic na FloatBuffer, který je poslán do vertex shaderu. Metoda *display* je zakončena metodou *drawModel*, která vykresluje jednotlivé meshe objektu. Pro vykreslování byla zvolena topologie trojúhelníků *GL_TRIANGLES.*

6.4.2 Vertexové atributy

Získávání pole floatů pro vertexové hodnoty je řešeno třídou *MeshData* a to konkrétně její veřejnou metodou *getVertexAttributes.* Metoda *getVertexAttributes* zpracovává vertexy získané z načítaní meshů a extrahuje z nich jednotlivé informace. Metoda vrací pole floatů určené pro vertex shader.

Samotný vertex je pole floatů a jeho základní varianta se skládá pouze z pozice zabírající tři místa. Pomocí vertexových atributů je ale možné vertex obohatit o další vlastnosti jakými může být barva, normála atp. Složení vertexu pro animaci je zobrazeno na *Obrázku 25*.



Obrázek 25: Vertexové Atributy (Zdroj: autor)

Na *Obrázku 25* je možné vidět pole reprezentující jeden vertex a následně barevně oddělené atributy. Je zde také ukázáno, kolik pozic má každý atribut přiděleno. V kódu je toto přiřazování řešeno metodou třídy *MeshData getVertexAttributes,* která prochází vertexy z meshe, extrahuje zmíněné atributy a následně jimi plní pole floatů, které je posláno do vertex shaderu.

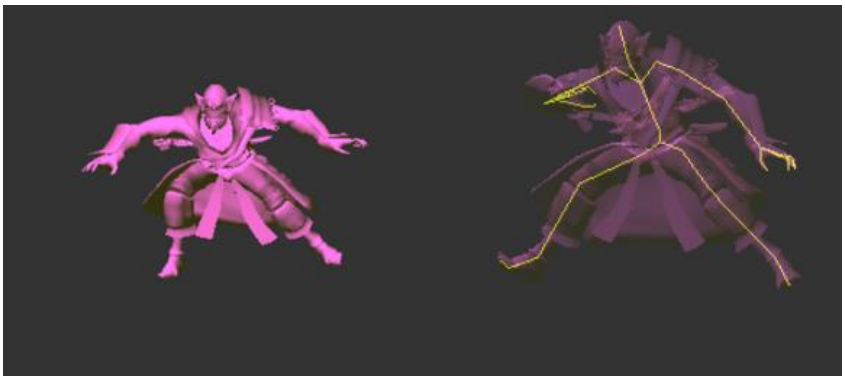
6.5 Zobrazení kostry

Aplikace nabízí možnosti zobrazit kostru daného objektu pomocí úseček reprezentujících polohy kostí. Úsečky se skládají z počátečního bodu, do kterého je uložen index kosti předka a konečného bodu, kde je uložen potomek. Indexy kostí jsou získávány stejným způsobem, jako tomu bylo u počítání finálních transformací. Opět se prochází hierarchie nodů akorát teď se místo počítání transformací tvoří spojnice. Metoda zajišťující vytváření úseček se nazývá *calculateSkelet* a nachází se ve třídě *Animator.*

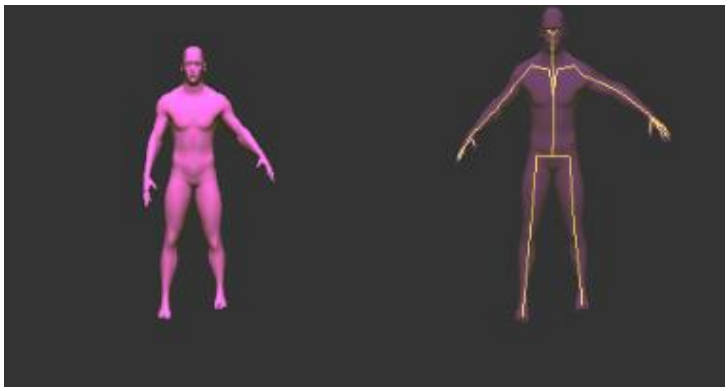
Uchováním přímek jsou v projektu pověřeny dvě třídy: *Line* a *Skeleton.* Skeleton zde má podobnou funkci jako *MeshData*, zpracovává vertexy, které následně předává v *Rendereru* jako pole floatů. Vertex je složen pouze ze čtyř floatů, kde první tři zastupují pozici a finální představuje id kosti.

7 Shrnutí výsledků

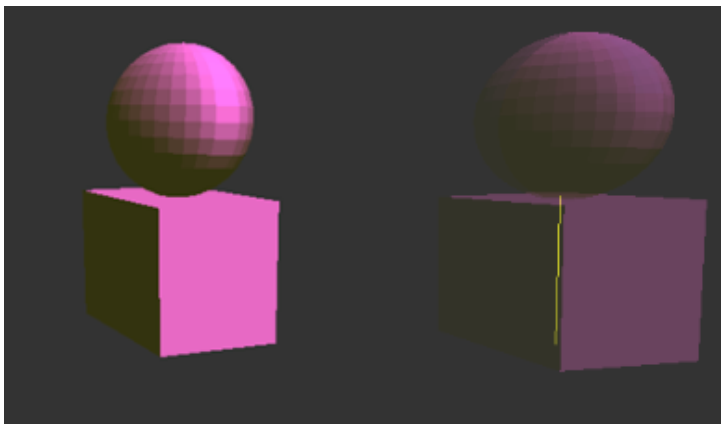
Animace byla testována na několika modelech různých datových formátů. Některé objekty byly získané z internetových zdrojů a jiné byly vytvořeny pomocí softwaru Blender. Hlavní modelem pro testování animace byl model *dancing_vampire.dae*, a jeho glb varianta *vampire.glb.* Modely ve formátu dae a glb se většinou načítaly bez problémů viz. Obrázky 13, 14, 15. Na *Obrázcích 26, 27 ,28* jsou ukázky animovaných modelů a jejich koster.



Obrázek 26: Animace objektu, zobrazení kostry objektu vamprie.glb (Zdroj: autor)

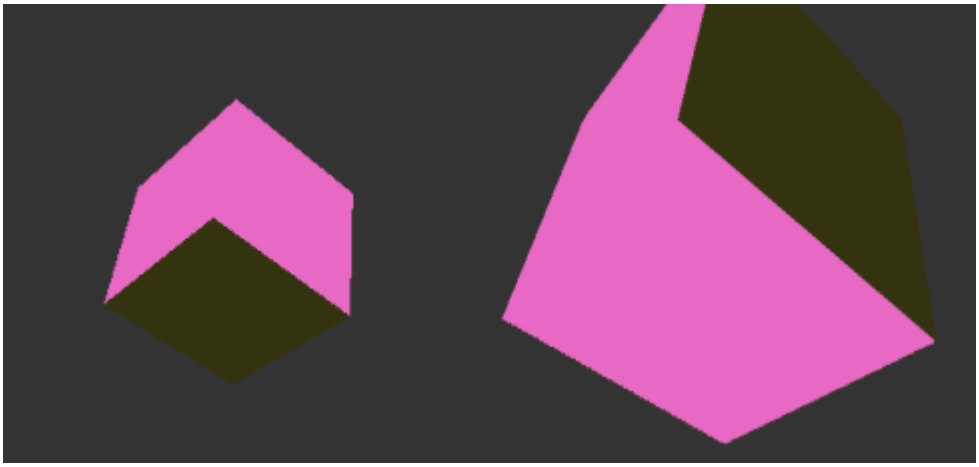


Obrázek 27: Animace objektu, zobrazení kostry objektu test.dae (Zdroj: autor)



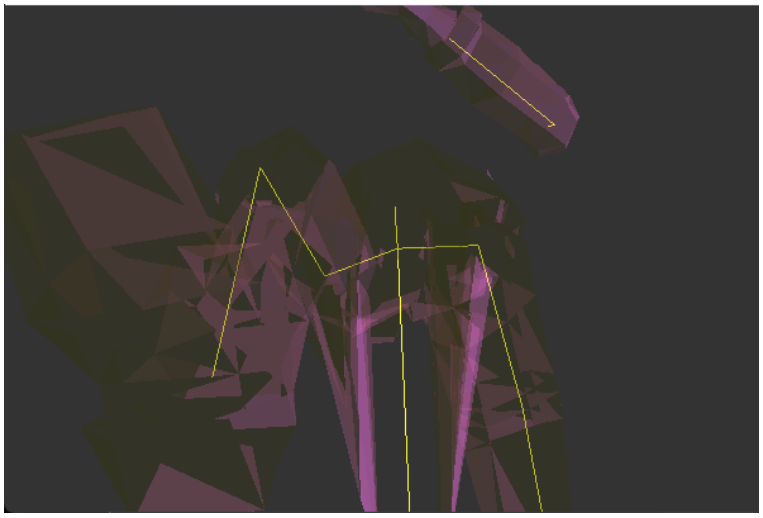
Obrázek 28: Animace objektu, zobrazení kostry objektu cubeSphere.glb (Zdroj: autor)

Problémy nastaly při načítání .fbx, soubory typu .fbx jsou importovány s větším měřítkem oproti .dae a .glb. Modely se bez explicitního zmenšení ani nevešly do scény. Modely *cubeSphere* a *cubeRotation* jsou vytvořeny mnou v modelovacím softwaru Blender. Objekty byly stvořeny pouze pro testování transformací s malým počtem kostí. Je možné, že při exportu modelů došlo k chybnému uchování normál. Cílem těchto modelů je ukázání animací na vlastních modelech různých formátů.



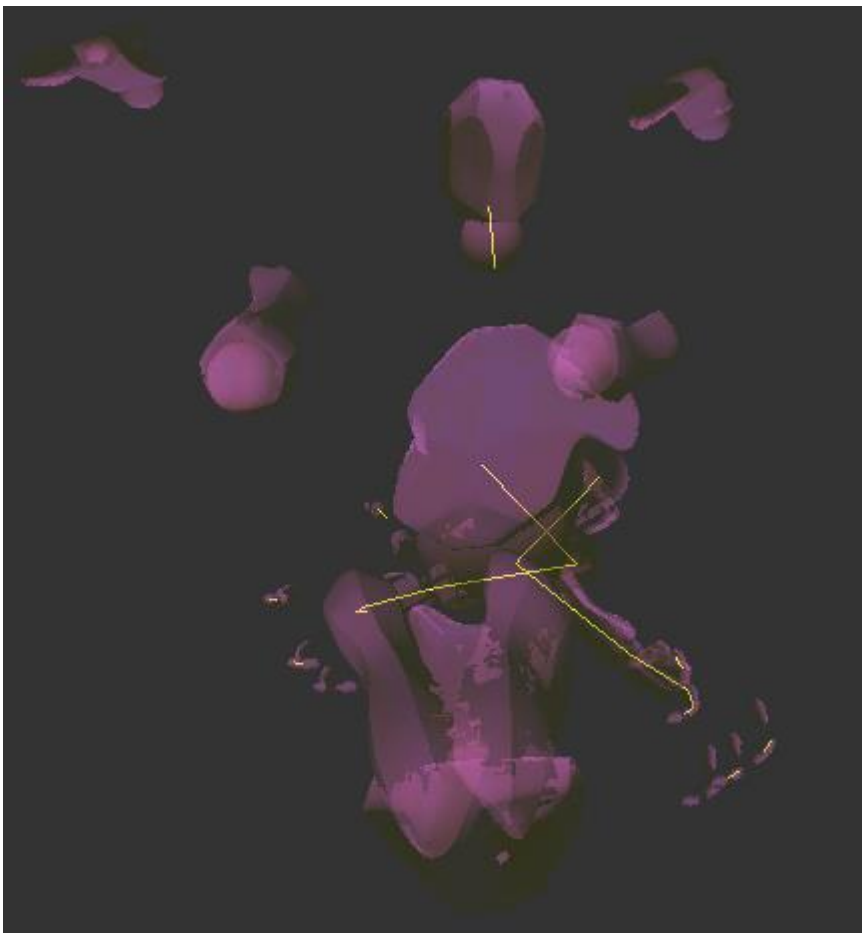
Obrázek 29: Objekt cubeRotation v .glb a .fbx formátu (Zdroj: autor)

Nalevo *Obrázku 29* je možné vidět objekt *cubeRotation.glb,* který byl přiblížen pro lepší viditelnost. Ten samý objekt, akorát že exportován do formátu .fbx je vidět napravo. Objekt byl oddálen co možná nejvíce aby se do scény vešel celý, a i přes to je sotva vidět. Při dalším oddalování objekt mizel ze scény. Další přiklad tohoto jevu je model *Character Running.fbx,* který se nevešel do scény ani s oddálením.



Obrázek 30: Model typu .fbx zobrazující se mimo scénu (Zdroj: autor)

Některé modely byly invertované, deformované nebo se jejich animace nenahrála vůbec. Soubory s animovanou scénou lze získat z mnoha zdrojů, ale jedním z nejspolehlivějších je stránka mixamo.com spravovaná společností Adobe. Animace jsou zde zdarma a v dobré kvalitě s velkým výběrem. Díky těmto vlastnostem je mixamo.com považován jako kvalitní zdroj pro animaci a modelů postav. Je tedy nepříjemné, že kompatibilita mezi implementovanou animací pomocí Assimp a mixamo.com nebyla jedna z nejlepších.



Obrázek 31: Chybné nahrání modelu a jeho kostí (Zdroj: autor)

Na *Obrázku 31* je možné vidět nesprávné nahrání kostí a celkovou deformaci postavy, kde celá spodní část je invertovaná, přičemž paže jsou zkřivené.

8 Závěry a doporučení

Teoretická část charakterizuje proces animace scény uspořádaný do animačního řetězce, který pomocí svých fází nabízí efektivní způsob tvorby animované scény. Důraz je kladen na vysvětlení populárních animačních technik, jejich výhody, nevýhody a následné využití v praxi. Detailně je představena skeletální animace jako jedna z tzv. keyframe animačních technik a její fáze rigging, skinning a weight painting. Součástí teoretické části je i představení nejpoužívanější 3D souborových formátů používaných pro uložení animované scény. Pro načítání dat v jednotlivých typech formátů byla vybrána softwarová knihovna Assimp, proto je popsána struktura reprezentace animované 3D scény používaná v této knihovně.

Praktická část je zaměřena na implementaci metody skeletální animace. Pro načtení animačního modelu byla použita knihovna Assimp, která zprostředkovala přístup k jednotlivým strukturám animované scény. Data animované scény byla uložena do patřičných modelových tříd a zpracována pro potřeby zobrazení. Vlastní zobrazení animovaného modelu je prováděno grafickou kartou s využitím programovatelných shaderů. Klíčovou úlohu zde tvoří vertex shader, který vypočítává výslednou pozici vrcholů povrchu geometrického modelu aplikováním transformačních matic reprezentujících kosti modelu. Model je skinnovaný lineární technikou linear blend skinning, která umožňuje aplikovat vliv kostí na jednotlivé vertexy pomocí vah. Pro přechod mezi animačními snímky byla použita lineární interpolace obstarávající výpočet posunutí a změny měřítka. Interpolace rotací byla řešena sférickou lineární interpolací kvaternionů. Výsledná scéna je složena z animovaného geometrického objektu, znázorněného částečně průhledným povrchem, a animované kostry znázorňující pohyb kostí. Povrchový model je osvětlen difúzním osvětlením vypočteném na základě normál načtených ze vstupního souboru. Pohyb zobrazeného animovaného modelu je plynulý a rozfázování pohybu odpovídá nastavenému časování.

Při testování řady vstupních souborů v různých formátech se ukázalo, že Assimp knihovna je velmi citlivá na různé typy modelů, kde jeden model může být nahrán úspěšně a druhý nikoliv. Otázkou je, zda zobrazené chyby vznikly nekompatibilitou knihovny se strukturou testovaných modelů, nebo způsobem implementace načítání a animace modelů. Pro načítání .fbx modelů by bylo do budoucna vhodné zvolit jinou knihovnu, popřípadě se snažit o napsání vlastního loaderu určeného pro daný formát. Při testování nebyly nalezeny problémy s formáty

.dae a .glb a to jak u modelů získaných z internetu, tak i u těch vytvořených v rámci práce. Na druhou stranu, i u špatně nahraných objektů probíhala interpolace snímků korektně. Pohyb animovaného objektu odpovídal původní animaci.

Do budoucna by bylo vhodné optimalizovat animační proces. Některé cykly používané v aplikaci při každém překreslení scény by mohly být předpočítány. Jako příklad je možné uvést procházení hierarchie nodů, kde při každém vykreslení je nutné projít jednotlivé nody a vypočítat finální matice. Efektivnější způsob by mohl být držení referencí na předka jednotlivé nody, aby se zamezilo zbytečným cyklům. Assimp knihovnu by bylo možné prozkoumat více, a to např. získáním materiálů objektu nebo extrahováním světla importované scény.

9 Seznam použité literatury

[1] „What is Animation — Definition, History and Types of Animation". Viděno: 29. září 2023. [Online]. Dostupné z: https://www.studiobinder.com/blog/what-is-animation-definition/

[2] A. Beane, *3D Animation Essentials*, 1., roč. 2012. Sybex. Viděno: 29. září 2023. [Online]. Dostupné z: https://www.pdfiles.net/pdf/view/3D-Animation-Essentials

[3] J. S. Peters, „Marvel Reportedly Pays VFX Artists 20 Percent Less Than Other Studios", CBR. Viděno: 19. duben 2024. [Online]. Dostupné z: https://www.cbr.com/marvel-studios-poor-vfx-artists-pay/

[4] A. Gouvatsos, „3D storyboarding for modern animation.", doctoral, Bournemouth University, 2018. Viděno: 7. říjen 2023. [Online]. Dostupné z: http://eprints.bournemouth.ac.uk/31265/

[5] A. Naghdi a P. Adib, „3D Animation Pipeline: A Start-to-Finish Guide (2022 update + video)", Dream Farm Studios. Viděno: 7. říjen 2023. [Online]. Dostupné z: https://dreamfarmstudios.com/blog/3d-animation-pipeline/

[6] „Like a Dragon: Ishin! Review", PCMAG. Viděno: 9. duben 2024. [Online]. Dostupné z: https://www.pcmag.com/reviews/like-a-dragon-ishin

[7] TheMartinV, „Like A Dragon: Ishin! - Sakura Index Review", Sakura Index. Viděno: 9. duben 2024. [Online]. Dostupné z: https://www.sakuraindex.jp/games/like-a-dragon-ishin-sakura-index-review/

[8] T. P. Thesen, „Reviewing and Updating the 12 Principles of Animation", *Animation*, roč. 15, č. 3, s. 276–296, lis. 2020, doi: 10.1177/1746847720969919.

[9] „Character Rigging in Blender". Viděno: 10. duben 2024. [Online]. Dostupné z: https://michaelcollins.xyz/3d-modeling-rendering-animation-fa19--oer/assignments/rigging.html

[10] M. C. Wibowo, S. Nugroho, a A. Wibowo, „The Use of Motion Capture Technology in 3D Animation", *International Journal of Computing and Digital Systems*, roč. 14, č. 1, s. 1–13, zář. 2023, doi: 10.12785/ijcds/XXXXXX.

[11] *God of War | How to Fight Like Kratos | PS4*, (2018). Viděno: 4. leden 2024. [Online Video]. Dostupné z: https://www.youtube.com/watch?v=yojcX5NRMuY

[12] *Behind the Scenes - God of War PS5 | Mocap Footage*, (2021). Viděno: 9. duben 2024. [Online Video]. Dostupné z: https://www.youtube.com/watch?v=HVXoOK4R8M0

[13] D. Ipacs, „Procedural Animation in Video Games: A Guide", Bluebird. Viděno: 8. leden 2024. [Online]. Dostupné z: https://bluebirdinternational.com/procedural-animation/

[14] „Best Books for Animation - Game Animation Techniques - 99bookscart.com". Viděno: 8. leden 2024. [Online]. Dostupné z: https://www.99bookscart.com/blog/view/64a8d7c4c495715168231c6f/animation-techniques-for-games

[15] Z. Bhatti, A. Shah, F. Shahidi, a M. Karbasi, „Forward and Inverse Kinematics Seamless Matching Using Jacobian". arXiv, 7. leden 2014. doi: 10.48550/arXiv.1401.1488.

[16] A. Aristidou, J. Lasenby, Y. Chrysanthou, a A. Shamir, „Inverse Kinematics Techniques in Computer Graphics: A Survey", *Computer Graphics Forum*, roč. 37, č. 6, s. 35–58, 2018, doi: 10.1111/cgf.13310.

[17] D. Ipacs, „Ragdoll Physics In Video Games: What Is It, Exactly?", Bluebird International. Viděno: 27. únor 2024. [Online]. Dostupné z: https://bluebirdinternational.com/ragdoll-physics/

[18] „The 10 Most Popular 3D File Formats", All3DP. Viděno: 28. únor 2024. [Online]. Dostupné z: https://all3dp.com/2/most-common-3d-file-formats-model/

[19] „The OBJ File Format – Simply Explained", All3DP. Viděno: 28. únor 2024. [Online]. Dostupné z: https://all3dp.com/2/obj-file-format-simply-explained/

[20] A. L. Possemiers a I. Lee, „Fast OBJ file importing and parsing in CUDA", *Comp. Visual Media*, roč. 1, č. 3, s. 229–238, zář. 2015, doi: 10.1007/s41095-015-0021-5.

[21] „Různé typy formátů 3D souborů – Adobe". Viděno: 28. únor 2024. [Online]. Dostupné z: https://www.adobe.com/cz/products/substance3d/discover/3d-files-formats.html

[22] M. Lazor, D. B. Gajić, D. Dragan, a A. Duta, „Automation of the avatar animation process in FBX file format", *FME Transactions*, roč. 47, č. 2, s. 398–403, 2019, doi: 10.5937/fmet1902398L.

[23] B. Foundation, „FBX binary file format specification", Developer Blog. Viděno: 28. únor 2024. [Online]. Dostupné z: https://code.blender.org/2013/08/fbx-binary-file-format-specification/

[24] „Everything You Need to Know About FBX Files". Viděno: 28. únor 2024. [Online]. Dostupné z: https://vection-technologies.com/blog/Everything-You-Need-to-Know-About-FBX-Files-A-Comprehensive-Guide/

[25] „Co jsou soubory COLLADA a jak je používat | Adobe". Viděno: 28. únor 2024. [Online]. Dostupné z: https://www.adobe.com/cz/creativecloud/file-types/image/vector/collada-file.html

[26] „GLB and GLTF files: Purpose, Difference, and Area of Application in 3D Modeling Services", Media. Viděno: 10. duben 2024. [Online]. Dostupné z: https://ikarus3d.com/media/3d-blog/glb-and-gltf-files-purpose-difference-and-area-of-application-in-3d-modeling-services/

[27] „The API-Documentation — Asset-Importer-Lib March 2022 v5.2.3 documentation". Viděno: 10. duben 2024. [Online]. Dostupné z: https://assimp-docs.readthedocs.io/en/latest/API/API-Documentation.html?highlight=AIScene#_CPPv47aiScene

[28] „LearnOpenGL - Assimp". Viděno: 10. duben 2024. [Online]. Dostupné z: https://learnopengl.com/Model-Loading/Assimp

[29] R. Mukundan, *Advanced Methods in Computer Graphics*. London: Springer, 2012. doi: 10.1007/978-1-4471-2340-8.

[30] A. S. Kushwah, „LearnOpenGL - Skeletal Animation". Viděno: 24. srpen 2023. [Online]. Dostupné z: https://learnopengl.com/Guest-Articles/2020/Skeletal-Animation

[31] J. Petty, „What is 3D Rigging For Animation & Character Design?", Concept Art Empire. Viděno: 24. srpen 2023. [Online]. Dostupné z: https://conceptartempire.com/what-is-rigging/

[32] I. Mas Ortega, „TroMotion: skeletal animation library", Bachelor thesis, Universitat Politècnica de Catalunya, 2019. Viděno: 21. srpen 2023. [Online]. Dostupné z: https://upcommons.upc.edu/handle/2117/174228

[33] M. Segal a K. Akeley, *The OpenGL R Graphics System*, roč. 2022. The Khronos Group, 2022. [Online]. Dostupné z: https://registry.khronos.org/OpenGL/specs/gl/glspec46.core.pdf

[34] K. Lehn, M. Gotzes, a F. Klawonn, *Introduction to Computer Graphics: Using OpenGL and Java*. in Undergraduate Topics in Computer Science. Cham: Springer International Publishing, 2023. doi: 10.1007/978-3-031-28135-8.

[35] R. Mukundan, *3D mesh processing and character animation: with examples using OpenGL, OpenMesh and Assimp*. Cham, Switzerland: Springer, 2022.

[36] B. J. Evans, J. Clark, a D. Flanagan, „Java in a Nutshell, 8th Edition", Red Hat Developer. Viděno: 24. srpen 2023. [Online]. Dostupné z: https://developers.redhat.com/e-books/java-nutshell-guide

[37] D. J. Eck, *Introduction to Computer Graphic*, roč. 2023. Hobart and William Smith Colleges. [Online]. Dostupné z: http://math.hws.edu/graphicsbook

[38] C. Michel, „Understanding front faces - winding order and normals | cmichel", Understanding front faces - winding order and normals | cmichel. Viděno: 30. srpen 2023. [Online]. Dostupné z: https://cmichel.io/understanding-front-faces-winding-order-and-normals

[39] „What Is Rigging in Animation? Skeletal Animation Explained | Adobe". Viděno: 24. srpen 2023. [Online]. Dostupné z: https://www.adobe.com/uk/creativecloud/animation/discover/rigging.html

[40] „Understanding Skinning: A Vital Step for Any Rigging Project". Viděno: 4. březen 2024. [Online]. Dostupné z: https://www.pluralsight.com/blog/film-games/understanding-skinning-vital-step-rigging-project

[41] J. Wang, „Comparison between linear blend skinning and dual quaternion skinning", Medium. Viděno: 19. srpen 2023. [Online]. Dostupné z: https://medium.com/@junyingw/comparison-between-linear-blend-skinning-and-dual-quaternion-skinning-369e727fdcf6

[42] J. Pan, L. Chen, Y. Yang, a H. Qin, „Automatic skinning and weight retargeting of articulated characters using extended position-based dynamics", *Vis Comput*, roč. 34, č. 10, s. 1285–1297, říj. 2018, doi: 10.1007/s00371-017-1413-6.

[43] N. Aburumman a M. Fratarcangeli, „Skin Deformation Methods for Interactive Character Animation", prezentováno v Communications in Computer and Information Science, čvc. 2017, s. 153. doi: 10.1007/978-3-319-64870-5_8.

[44] R. Mukundan, *Advanced methods in computer graphics: with examples in OpenGL*. London: Springer, 2012. [Online]. Dostupné z: http://link.springer.com/10.1007/978-1-4471-2340-8

[45] „Skeletal Animation (Advanced Methods in Computer Graphics) Part 3". Viděno: 29. srpen 2023. [Online]. Dostupné z: http://what-when-how.com/advanced-methods-in-computer-graphics/skeletal-animation-advanced-methods-in-computer-graphics-part-3/

[46] DeepMotion, „Character Animation 101: Weighting Your Rig", Medium. Viděno: 5. březen 2024. [Online]. Dostupné z: https://deepmotion.medium.com/character-animation-101-weighting-your-rig-d63118b86edc

[47] U. A. Studio, „Importance of Weight Painting in 3D Animation". Viděno: 5. březen 2024. [Online]. Dostupné z: https://www.ultimateanimationstudio.com/2023/05/importance-of-weight-painting-in-3d.html

[48] „The Asset-Importer Library Home". Viděno: 31. březen 2024. [Online]. Dostupné z: https://assimp.org/

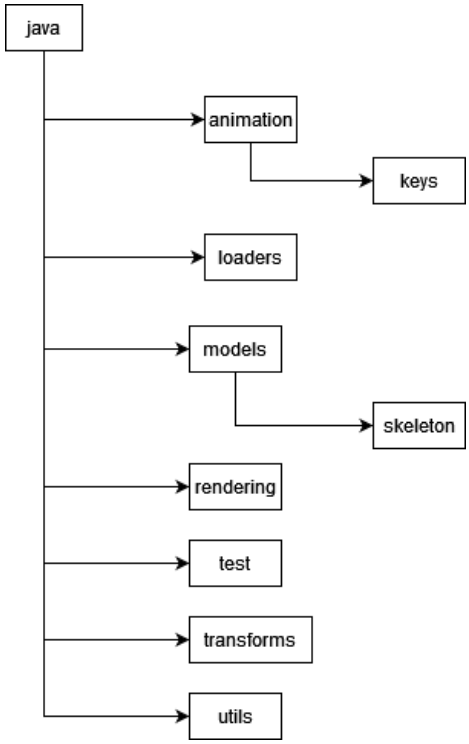
[49] „Working with Asset-ImporterLib — Asset-Importer-Lib March 2022 v5.2.3 documentation". Viděno: 1. duben 2024. [Online]. Dostupné z: https://assimp-docs.readthedocs.io/en/latest/usage/use_the_lib.html?highlight=offset%20matrix#bones

[50] „Tutorial 38 - Skeletal Animation With Assimp". Viděno: 5. březen 2024. [Online]. Dostupné z: https://ogldev.org/www/tutorial38/tutorial38.html

10 Přílohy

10.1 Verzování a adresář

Projekt používá verzovací systém GitLab na adrese: https://gitlab.com/zahradcore/skeletalanimation



Obrázek 32: Struktura adresářů třídy (Zdroj: autor)

Shadery a modely se nachází v rootu projektu, shadery se nachází ve složce *shaders* a modely ve složce *res.*

10.2 Extrerní repozitáře

Práce byla inspirována několika implementacemi, zde budou odkazy na jejich repozitáře. V projektu byly použity modely nacházející se v daných repozitářích.

- https://github.com/TheThinMatrix/OpenGL-Animation
  - Model: Character Running.fbx
- https://github.com/emeiri/ogldev/tree/master/tutorial24_youtube
- https://github.com/RagnarrIvarssen/Assimp-Tutorial-LWJGL-3
  - Model: test.dae
- https://github.com/JoeyDeVries/LearnOpenGL
  - Model: dancing_vampire.dae

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2022/2023

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Josef Zahradník**
Osobní číslo:	**I2000444**
Adresa:	V Koutech 1752/7, Hradec Králové – Pražské Předměstí, 50002 Hradec Králové 2, Česká republika

Téma práce:	Skeletální animace 3D modelů
Téma práce anglicky:	Skeletal animation of 3D models
Jazyk práce:	Čeština

Vedoucí práce:	Ing. Bruno Ježek, Ph.D. Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

Cílem práce je prozkoumat jednotlivé fáze 3D animačních technik se zaměřením na skeletální animaci. Navrhnout a implementovat zobrazení 3D scény obsahující animaci zvoleného modelu.
Postup prací:
Prozkoumat principy a metody používané pro animaci ve 3D scéně.
Vytvořit přehled technik se zaměřením na skeletální animaci.
Popsat formáty pro ukládání modelů včetně animačních dat
Navrhnout a implementovat řešení pro zobrazení 3D animované scény s využitím vhodné grafické knihovny.
Provést testování a srovnání s používanými nástroji.
Zhodnotit dosažené výsledky.

Seznam doporučené literatury:

MUKUNDAN, Ramakrishnan. 3D Mesh Processing and Character Animation: With Examples Using OpenGL, OpenMesh and Assimp [online]. 1. Švýcarsko: Springer Cham, 2022, 201 s. [cit. 2023-10-15]. ISBN 978-3-030-81354-3. Dostupné z: doi:https://doi.org/10.1007/978-3-030-81354-3
BEANE, Andy. 3D Animation Essentials. 1. Kanada: Sybex, 2012, 352 s. ISBN 1118147480.
LEHN, Karsten, Merijam GOTZES a Frank KLAWONN. Introduction to Computer Graphics [online]. Using OpenGL and Java. Švýcarsko: Springer Cham, 2023, 467 s. [cit. 2023-10-15]. 3. ISBN 978-3-031-28135-8. Dostupné z: https://link.springer.com/book/10.1007/978-3-031-28135-8
ECK, David J. Introduction to Computer Graphics [online]. New York: Hobart and William Smith Colleges, 2023, 527 s. [cit. 2023-10-15]. Dostupné z: https://math.hws.edu/graphicsbook/