ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tkáč** Jméno: **Jakub** Osobní číslo: **421052**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Podpora pokročilejších rigovacích technik pro herní engine Unity3D

Název diplomové práce anglicky:

Advanced rigging techniques in Maya and Unity 3D

Pokyny pro vypracování:

Seznamte se s technikami používanými pro vytváření kostry, deformátorů a animačních kontrolerů pro animaci lidské postavy. Dále se seznamte s pokročilejšími technikami využívanými při tvorbě filmů, které zlepší výsledné deformace v programu Autodesk Maya. Tyto techniky jsou známé jako „twist joints“, „volume joints“, „interpolation joints“, „stretchy joints“ a další. Vytvořte nástroje, s jejichž pomocí bude možné exportovat model z programu Autodesk Maya do herního engine Unity3D spolu s těmito pokročilejšími technikami (podporujte alespoň dvě vybrané po dohodě s vedoucím práce). Dále vytvořte v programu Autodesk Maya knihovnu skriptů, která pomůže uživateli s tvorbou kostry a zjednoduší vytváření těchto pokročilejších technik. V programu Unity3D porovnejte vizuálně výsledky na postavách s pokročilejšími technikami a bez pokročilejších technik, porovnejte také výpočetní náročnost v zobrazování. Vytvořte tutoriál popisující použití vytvořených nástrojů s důrazem na kroky, které nejsou automatizovány.

Seznam doporučené literatury:

How to Cheat in Maya 2017: Tools and Techniques for Character Animation. Paul Naas, CRC Press (2018)
Art of Rigging Volume I. Kiaran Ritchie, Jake Callery, Karin Biri, CG Toolkit (2006)
Maya Python for Games and Film: A Complete Reference for Maya Python and the Maya Python API. Adam Mechtley, Ryan Trowbridge, Elsevier (2012)
Game Development with Unity. Menard, M., Course Technology (2011)

Jméno a pracoviště vedoucí(ho) diplomové práce:

MgA. Radek Smetana, katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.09.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **19.02.2021**

MgA. Radek Smetana podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Podpora pokročilejších rigovacích technik pro herní engine Unity3D

Bc. Jakub Tkáč

Vedoucí práce: Ing. MgA. Radek Smetana

6. ledna 2020

Poděkování

Rád bych poděkoval vedoucímu práce Ing. MgA. Radku Smetanovi za zasvěcení do tématu a čas strávený při vedení této práce. Dále děkuji Ing. Davidu Sedláčkovi, Ph.D. za konzultaci v rámci předmětu Softwarový a výzkumný projekt. Taktéž si zaslouží poděkování můj zaměstnavatel Beat Games s.r.o. za vstřícnost s časovým plánem při tvorbě této diplomové práce. V neposlední řadě děkuji rodině a kamarádům za podporu a trpělivost po celou dobu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 6. ledna 2020 .....

Odkaz na tuto práci

Tkáč, Jakub. *Podpora pokročilejších rigovacích technik pro herní engine Unity-3D.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2020.

Abstrakt

Tato práce se zabývá podporou pokročilejších rigovacích technik v herním engine Unity. Jedná se zejména o podporu technik, které jsou známé pod pojmy „twist joints“ a „interpolation joints“. V rámci této práce proto byly vytvořeny knihovny v programu Autodesk Maya a Unity, s jejichž pomocí bude možné tyto techniky vytvořit a poté přenést do Unity. Díky nim je možné dosáhnout v herním engine přesnějších deformací, a tím zajistit pro diváka lepší zážitek. Součástí práce je také porovnání výsledků na postavách bez pokročilejších technik a s pokročilejšími technikami.

Klíčová slova Rigování, pokročilejší rigovací techniky, Autodesk Maya, Unity

Abstract

The goal of this thesis was to create advanced rigging techniques for game engine Unity. It allows techniques known as „twist joints“ and „interpolation joints“ to be used in Unity. In order to achieve this goal, libraries in Autodesk Maya and Unity had to be made which allowed the creation of these techniques and their import into Unity. Thanks to that it is possible to achieve more precise deformations in the game engine which creates a better gaming experience for the player. The last part of the thesis was a comparative analysis of characters with and without advanced techniques used on them.

Keywords Rigging, advanced rigging techniques, Autodesk Maya, Unity

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V posledních letech se díky nárůstu výkonu počítačových systému máme možnost setkávat se stále více propracovanějšími virtuálními 3D modely postav, které mají využití v počítačových hrách či filmech. Pokud chceme tento model rozhýbat, je zapotřebí přidat do postavy kostru a definovat, jak se bude model deformovat v závislosti na pohybech této kostry.

Tento proces je známý jako rigování. Pro rigování postaviček je zapotřebí mít jak technický, tak umělecký vhled. Když se na to podíváme z abstraktního hlediska, jedná se prakticky o programování, při kterém se snažíme postupně skládat klouby, tvořit z nich kosti, přidávat k nim různá omezení a vkládat vazby mezi rodičem a potomkem. Některé z těchto opakujících se procesů se dají automatizovat. Vzniká proto od jednoduchých nástrojů přes modulární rigy, které automatizují určitou část (např. ruku), až po plně automatické rigovací nástroje nazývané jako autorig.

Tyto úkony dělají umělci zejména v některých z 3D grafických programů, kde následně dojde k vykreslení *(angl. rendering)*[1] scény. Případně se pro vykreslení použije externí program. Problém ale nastává ve chvíli, chceme-li přesunout takto připravený model do jiného programu. V našem případě model s rigem do herního engine, které mají pouze omezené možnosti. Například herní engine Unity dokáže ovládat pro humanoidní postavu[2] dle GUI až 25 hlavních kloubů a k tomu navíc prsty na rukou[3]. Další klouby sice můžeme v modelu mít, ale nebudou animovány. Zůstanou v původní pozici vzhledem k prvnímu rodičovskému kloubu, který je ovládán za pomoci Unity. Pro podporu dalších kloubů je proto třeba vymyslet způsob, jak je ovládat.

[1] Renderování je proces při kterém se z počítačového modelu vypočítává, jak bude daná scéna zobrazena [6].

[2] Humanoidní postava je taková postava, která se podobá stavbou těla člověku.

[3] Dokumentace mluví o 24 kloubech, kde kloub *upper chest* chybí [8].

Cíle

Tématem diplomové práce je vytvořit skripty, které dokážou přenést pokročilejší rigovací techniky *twist joints* a *interpolation joints* z programu Autodesk Maya do herního engine Unity. Díky tomu bude umožněno dosáhnout lepších deformací v herním engine, to například můžou využít herní vývojáři pro tvorbu filmových záběrů.

Struktura práce

V první části práce bude čtenáři představen lehký úvod do řetězce tvorby 3D modelů postav. Zde se seznámí se základními pojmy, kterými tento obor oplývá. Další kapitola bude věnována rigování, kde čtenáři budou přiblíženy techniky, se kterými se může setkat. Na konci této kapitoly budou představeny pokročilejší rigovací techniky. Ty umělci využívají pro tvorbu věrohodně se deformujících modelů. Ve třetí kapitole budou probrány existující řešení tohoto problému. Ve čtvrté kapitole se budeme zabývat jeho analýzou. Budou zde představeny požadavky na aplikaci a rozebrány další pojmy vyskytující se při řešení problému. V páté kapitole bude představen návrh aplikace, která vychází z požadavků představené v předchozí kapitole. Následuje šestá kapitola, kde budou popsány implementační detaily. Sedmá kapitola se věnuje testování. Poslední osmá kapitola bude obsahovat zhodnocení výsledků.

Kapitola 1

Úvod do řetězce tvorby 3D modelů postav a tvorby jejich rigu

V této kapitole bude popsán řetězec tvorby 3D modelů postav. Dozvíme se, v jaké části filmového řetězce se rig nachází a jak funguje tvorba modelu od návrhu po animaci. Podrobněji bude rig rozebrán v další kapitole číslo 2.

1.1 Základní popis rigu

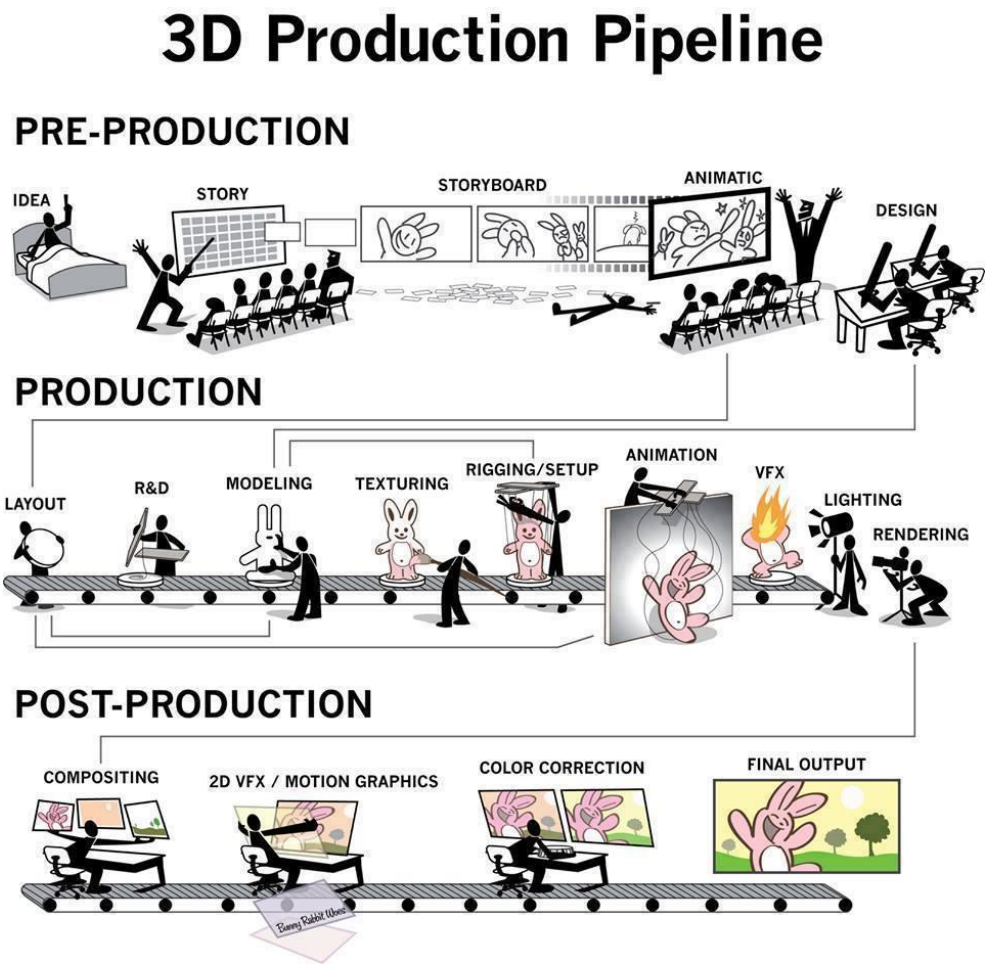
Rigování je proces, kde vytváříme pro zadaný model hierarchickou strukturu kostí z kloubů *(angl. joints)*. Tato struktura je poté využita pro animování modelu. Další nedílnou součástí je tzv. *skinning*. Zde dochází ke svázání *(angl. binding)* vrcholů z dodaného modelu k námi vytvořeným kostem. Tím zajistíme, jak se bude „kůže“ modelu pohybovat na základě změny pozice jednotlivých kloubů.

1.2 Umístění rigu v řetězci

Proces tvorby rigu se nachází v řetězci *(angl. pipeline)* mezi tvorbou modelu (vytvoření 3D sítě a namapování UV souřadnic[4]) a následnou animací. Ve velkých studiích mají pro každou část specialistu, který se nejlépe vyzná v dané problematice. Jeden člověk vytvoří koncept postavičky (nakreslí v grafickém programu nákres ve 2D) a předá ho modeláři. Ten návrh vymodeluje a vytvoří textury. Model dostane riger a on pro něj vytvoří kostru a udělá skinování *(angl. skinning)*. V případě tvorby rigu pro filmy definuje, jak se model bude

[4]Při mapování UV souřadnic dochází k projekci 2D obrázku na námi zvolený 3D model [1].

chovat podle požadavků kladených animátory a vytvoří kontrolery pro jednoduché ovládání. A nakonec se model s rigem předá animátorům, kteří na základě vytvořených kontrolerů jsou schopni s modelem jednoduše vytvářet požadované animace. Toto je základní řetězec tvorby 3D postavy. Ale může se mírně lišit dle určení použití, velikosti a zvyklostí studia. V malém studiu může člověk zastávat i několik pozic. Tento řetězec lze vidět na obrázku 1.1.



Obrázek 1.1: Popis produkčního řetězce u tvorby filmů, kde je patrné, že jeden z prvních kroků je vytvoření návrhu postavy. Dalším krokem je modelování postavy, nanesení textur a nakonec tvorba rigu. Ten se pak předá animátorům, aby vytvořili požadované animace. Poté následují ještě další kroky. Animace postavičky se již ale nemění. Převzato z [9]

Kapitola 2

Úvod do rigování

Tato kapitola se bude zabývat rigováním. Budou zde představeny některé pojmy, se kterými se můžeme v rigování setkat. Poté bude následovat popis pokročilejších technik, které dokážou vylepšit deformace. Dvě z těchto pokročilejších technik budou v rámci této práce rozebrány a přeneseny do herního engine Unity. I když pojmy probrané v této kapitole jsou obecné a principy platí ve většině grafických programů, tak práce je napsána zejména s přihlédnutím na program Autodesk Maya, viz kapitola 4.2, a následný export do Unity, viz kapitola 4.4.

2.1 Požadavky kladené na model

Aby výsledek vypadal uvěřitelně, je zapotřebí mít vhodný model, který splňuje několik požadavků. Pro model složený z polygonů[5] je nutné použít polygon o čtyřech vrcholech *(angl. quad)* namísto trojúhelníků *(angl. tris)*. V případě zjemnění modelu pomocí ploškového dělení *(angl. subdivision surface)* se dokážeme vyhnout anomáliím, které vznikají při použití trojúhelníků nebo víceúhelníků [3].

Dále je zapotřebí mít vhodné rozlišení sítě. Při nižším počtu polygonů bychom dosahovali v deformacích horších výsledků. Naopak při vysokém počtu nastává problém při skinování u kreslení vah.

Pokud dostaneme model s velkým počtem polygonů, je nutné pro hladší průběh animování vytvořit model s nižším počtem polygonů. Zpětné aplikování vah na model s vyšším počtem polygonů se dělá pomocí deformátoru `wrap`.

[5] Polygon je dvoudimenzionální útvar tvořený rovnými úsečkami.

2.2 Postup při rigování

Poté, co máme vhodný model, můžeme přejít k samotnému rigování. Je potřeba mít na paměti, že výsledný rig musí být „čistý“ a bez chyb. V případě odhalení chyby během animace je nutné ji v rigu opravit. A tím pádem některé návazné kroky v animaci musíme znovu předělat.

Také je vhodné si předem rozmyslet pojmenování jednotlivých skupin, kloubů a dalších částí rigu. Ať už z důvodu orientace v hierarchii, nebo hlavně z důvodu použití skriptu, kde jednotlivé skripty předpokládají určitou jmennou konvenci, díky které jsou schopny najít daný objekt v grafu scény.

2.2.1 Tvorba kostry

Vytváření kostry probíhá hierarchickým vkládáním kloubů. Kloub je v našem případě bod v prostoru, kterým lze rotovat. Při spojení dvou kloubů vzniká kost. Postupným spojováním s dalšími klouby vytvoříme celou kostru. Důležitá je v tomto případě lokální osa rotace *(angl. local rotation axis)*, viz kapitola 2.2.4.

Počet kloubů, kterých je potřeba pro vytvoření kostry, se liší rig od rigu. Určitě nevytváříme tolik kostí, kolik jich je v lidském těle. Snažíme se o co největší zjednodušení, které nám poskytne dostatečně dobré výsledky. V případě bipední postavy je dle [4] vhodné vytvořit tolik kloubů, aby bylo zajištěné pohodlné ovládání všech hlavních končetin, páteře, krku, hlavy, čelisti a rukou. Pokud připravujeme postavu pro herní engine, tak je vhodné si předem zjistit, kolik kloubů a které je potřebné vyplnit. Pro Unity v případě kostry humanoida je zapotřebí minimálně 15 kloubů [8].

2.2.2 Dopředná kinematika

Během rigování se můžeme setkat s dvěma přístupy ovládání kloubů. Jedním z nich je tzv. dopředná kinematika *(angl. forward kinematics (FK))*. Při ní určený kloub definuje otáčení všech jeho potomků v dané hierarchii [6]. Tím pádem je potřeba, např. pro položení ruky k desce stolu, rotovat manuálně několika klouby v řetězci, abychom dosáhli požadovaného výsledku.

2.2.3 Inverzní kinematika

Naproti tomu díky inverzní kinematice *(angl. inverse kinematics (IK))* je možné ovládat celý řetězec kloubů pomocí jednoho jediného bodu zvaného *IK handle*, který má stejnou pozici jako poslední bod v řetězci [4]. Animátor jen posune tento bod na požadované místo a o zbytek se starat nemusí. Jakmile se tento IK manipulátor pohne, tak *IK solver*[6] vyhodnotí, jak pootočit všemi

[6] *IK solver* si můžeme představit jako program, který spočte řešení řetězce kloubů, které je ovládáno pomocí inverzní kinematiky [13].

klouby, aby bylo požadavku animátora vyhověno [6]. V případě, že animátor posune tento bod dále než je součet délek všech kostí v řetězci (a první kloub v řetězci je fixní), tak všechny klouby budou v jedné linii směřovat k tomuto bodu, kterého se nedotknou.

V některých případech při tvorbě animací se hodí první přístup. V jiných druhý a pro některé (např. animace končetin) je vhodné dát animátorům oba dva přístupy (FK i IK).

2.2.4 Lokální osa rotace

Lokální osa rotace *(angl. local rotation axis (LRA))* nám určuje, jak se budou klouby lokálně otáčet v osách `x,y,z`. Dle [4] je dobrým zvykem, nikoliv pravidlem, mít osu `x` jako primární pro rotaci a osy `z,y` jako sekundární. Výjimka je např. v případě dat z Motion Capture[7], kde osa `y` je určena pro rotaci.

Je důležité, aby jednotlivé klouby směřovaly stejným směrem v rámci celého rigu. Zejména, pokud pro rigování používáme skripty, které dopředu počítají s určitým nastavení os rotace. Tato situace je ilustrovaná na obrázku 2.1.

2.2.5 Tvorba kontrolerů

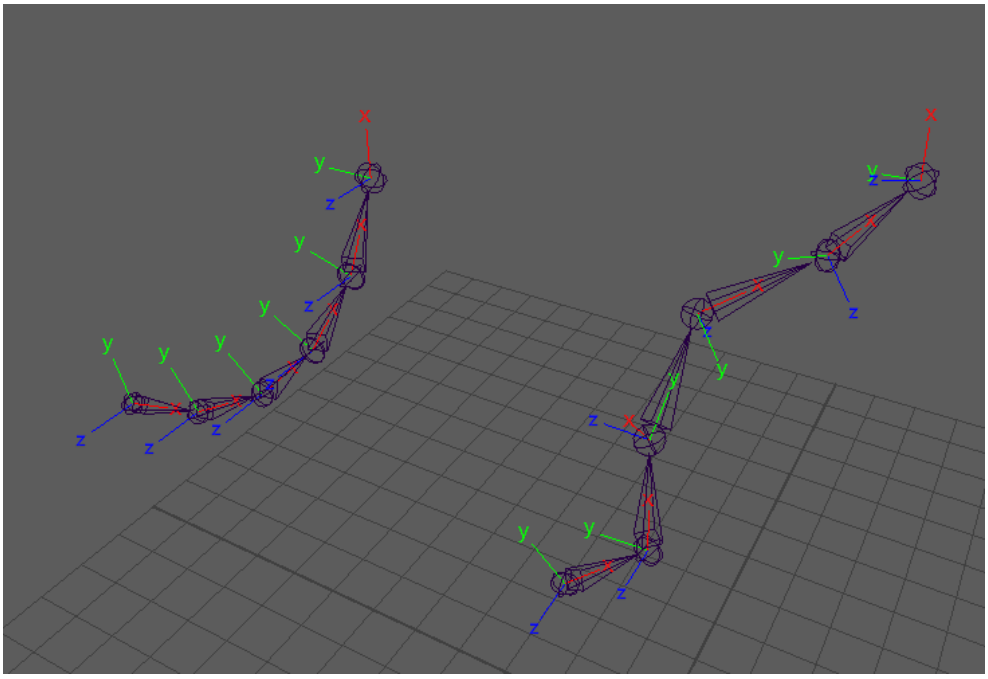
Tento krok je důležitý zejména pro tvorbu ručně animovaných postav. Jedná se o vytvoření kontrolerů, které slouží pro pohodlné ovládání postavy animátory. Využívají jednoduchých tvarů, jako jsou křivky či kruhy. Případně i složených tvarů, tak aby bylo zajištěné jednoznačné označení a animátoři se jednoduše v rigu vyznali. V závislosti na použití rigu a požadavcích animátorů tyto kontrolery ovládají základní části rigu (možnost ovládat končetiny, páteř, hlavu) až po podrobnější, umožňující animátorům pokročilejší možnosti (např. sevření celé pěsti pomocí jednoho ovládacího prvku, nebo ovládání obličeje). Hotový narigovaný model s kontrolery lze vidět na obrázku 2.2.

2.2.6 *Skinning*

Posledním krokem je tzv. *skinning*. Jeho úkolem je propojení hotové kostry s polygonální sítí, a také definování, jak se bude tato síť chovat v závislosti na změně poloh jednotlivých kostí.

Jednou z rozšířených technik je tzv. kreslení vah. Pro každý kloub v hierarchii „štětcem nanášíme“ na polygonální síť váhu, která definuje, jak moc změna pozice daného kloubu má vliv na pozici přidružené sítě.

[7] Motion Capture je proces, při kterém dochází k záznamu pohybu skutečných postav, kde je následně tento pohyb namapován na digitální postavu, a tím je zajištěna animace.



Obrázek 2.1: Lokální osy rotace. Na obrázku můžeme vidět dva řetězce složené z několika kloubů a jejich příslušné LRA. Oba dva řetězce máme iniciálně položené na ploše `xz` (vizuálně reprezentované mřížkou). Chceme rotovat každý kloub k ose `-y`. V případě levého řetězce můžeme vidět, jak se každý kloub otočil podle předpokladů v jednom směru. Bohužel v případě řetězce napravo dojde díky rozdílně nastaveným LRA pro nás k nežádoucímu chování

2.3 Pokročilejší techniky

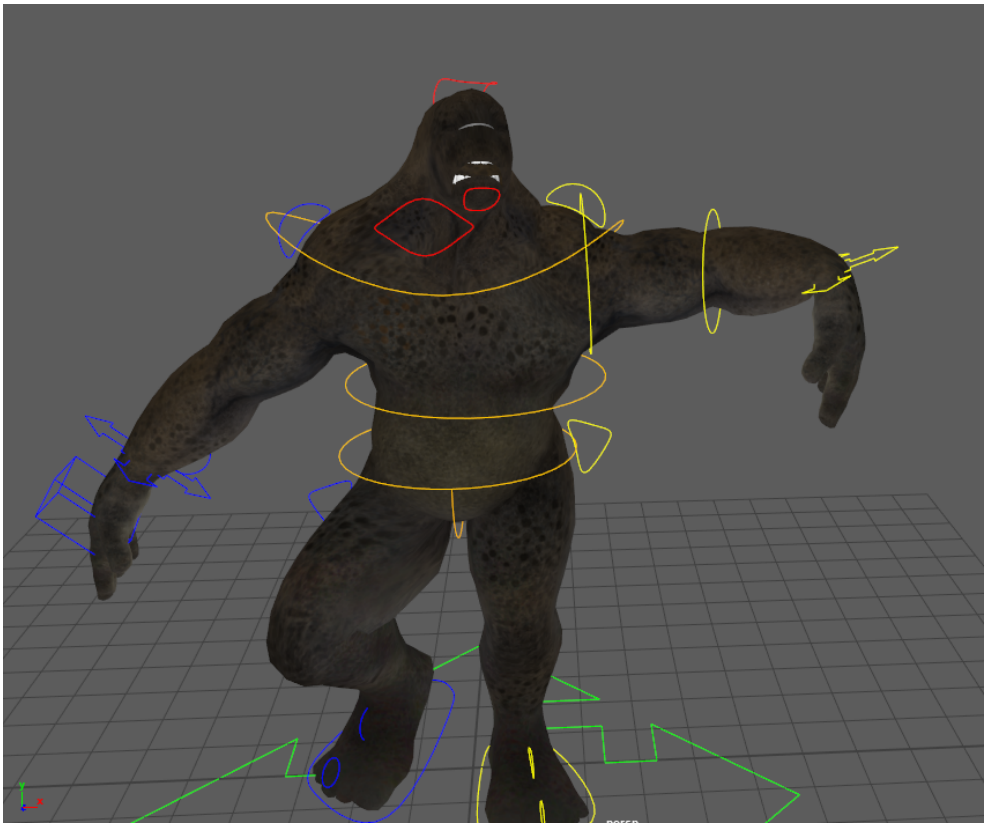
V této sekci bude zmíněno několik další technik, díky kterým můžeme vylepšit výsledný rig nebo rozšířit možnosti rigu. Dvě z nich *twist joints* a *interpolation joints* jsou v rámci této práce implementovány.

2.3.1 *Blendshapes*

Jedná se o deformátor, který se dá využít pro změnu geometrie jednoho objektu do jiného objektu [5]. Často se používá pro rigování tváře, kde se vytvoří několik výrazů, a poté můžeme mezi těmito výrazy interpolovat.

2.3.2 *Correctives*

Další metodou pro úpravu výsledné geometrie jsou tzv. *corrective blendshapes*. Vytvoříme novou, upravenou geometrii, kterou v nevhodně vypadajících pózách nahradíme za původní geometrii. Typické použití může být v případě rotace loketního kloubu, kdy bude na vnější straně ubývat objem, tak s po-



Obrázek 2.2: Model s kontrolery vytvořený na základě cvičení ze stránky Pluralsight [2]. Model byl poskytnutý v rámci cvičení

mocí této techniky můžeme původní geometrii nahradit novou, upravenou, kde k tomuto jevu docházet nebude. Můžeme do určité míry na některých kloubech tento problém vyřešit pomocí *interpolation joints*, viz kapitola 2.3.4.

2.3.3 *Twist joints*

S pomocí *twist joints* můžeme dosáhnout přesnějších výsledků rotace geometrie na místech jako je biceps nebo předloktí. V základním rigu mnohdy máme jeden kloub v oblasti lokte, druhý v oblasti zápěstí. Při rotaci zápěstí proto dochází k deformacím zejména v oblasti zápěstí. V místech směřujících k loktu deformace buď nejsou, nebo v horším případě jsou, ale při rotaci v dalších osách u zápěstí se hýbe i síť směřující k lokti. To můžeme vyřešit přidáním pomocných kloubů, které budou rotovat s částečným vlivem zápěstí v primární ose rotace. Tím dokážeme plynule rozložit deformace mezi dvěma klouby.

2.3.4 *Interpolation joints*

Naproti tomu *interpolation joints* dokáže zachovat ubývající objem, a tím pádem částečně supluje techniku *correctives*, která byla zmíněna v kapitole 2.3.2. Ta je náročnější o manuální tvorbu, a také výslednou velikost modelu. Používá se v místech jako je loket nebo rameno, kde se přidá navíc další kloub, který se bude otáčet o zlomek původního kloubu (který zdvojuje). Tím pádem při větších rotacích nebude tento úbytek tak patrný, protože přidružená polygonální síť nového kloubu se o tolik nezahne.

2.3.5 *Volume joints*

Tato technika nemá jednohlasně domluvené označení, proto každá firma ji pojmenovává jinak. Stejně jako *interpolation joints* slouží pro zachování objemu. Rozdíl je v tom, že zde nedochází ke zdvojování jednotlivých kloubů a jejich rotace, ale přidáním dalších kloubů ve tvaru hvězdice, která se propojí s již existujícími klouby. Můžeme si to představit jako vyztužení, které díky těmto kloubům zachovává objem. Používá se zejména u komplikovanějších nebo objemnějších modelů.

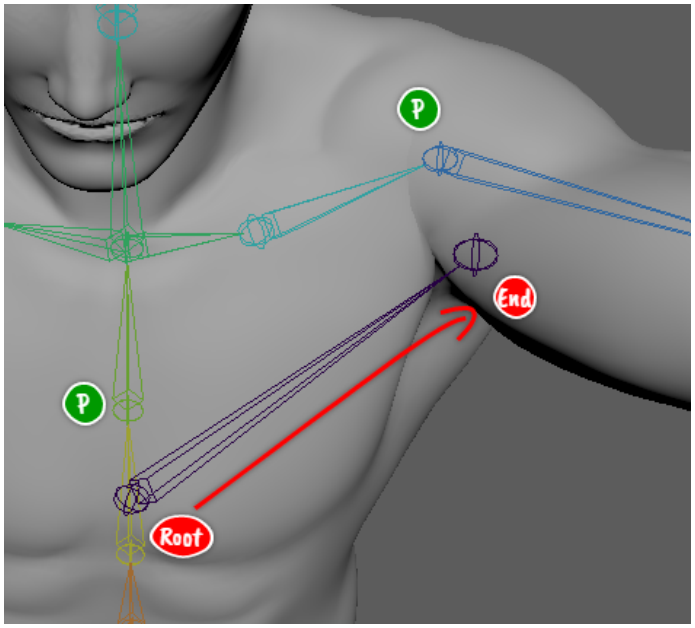
2.3.6 *Stretchy joints*

Další z pokročilejších technik, která se používá zejména pro simulaci výrazných svalů, jsou *stretchy joints*. Zde jsou vytvořeny dva nové klouby (na obrázku 2.3 označené jako `Root` a `End`). Každý z nich je svázaný pomocí *Parent constraint* k již existujícím kloubům (na obrázku označené pod písmenem `P`). Mezi těmito novými dvěma klouby je vazba rodič-dítě. Rodič v tomto případě míří pomocí omezení *Aim constraint* na potomka (druhý vytvořený kloub). U rodiče poté dochází ke škálování velikosti na základě vzdálenosti od jeho potomka. Čím blíže budou u sebe, tím rodič bude mít větší hodnotu v atributu *scale*.

2.4 Animace

Samotná tvorba rigu se v zásadě dělá za jedním účelem, a to je tvorba animací při které vdechneme život postavičkám. Jak Dariush Derakhshani zmiňuje ve své knížce *„Animace je změnou v čase [6]“*. To znamená, že při tvorbě animace vytvoříme několik statických obrázků, které jsou zvané jako rámce *(angl. frames)*, kde si napozicujeme, jakou postavička bude mít v daný rámec pózu. To uděláme pro každý jeden rámec, kde výsledkem přechodu mezi jednotlivými rámci vzniká animace. V případě ruční animace je také důležité, jaká je rychlost přehrávání vytvořené animace, neboli kolik přehrajeme snímků za jednu sekundu. Je zvykem ji značit jako *frames per second (fps)*.

Existují různé standardy, jak je zmíněno v [6]. Pro film se od roku 1927 využívá 24 *fps*, v Evropě je to formát PAL, který má 25 *fps*. Pro televizní



Obrázek 2.3: Technika *stretchy joints*. Šipka od `Root` směrem k `End` kloubu značí omezení *Aim constraint*. Klouby označené písmeny **P** jsou svázány s klouby `Root` a `End` za pomocí *Parent constraint*

vysílání se využívá formátu NTSC s 30 snímky za sekundu. V poslední době ale někteří režiséři zkouší natáčet filmy na 48 *fps*, mluví se i o 60 *fps*. Jedná se např. o snímek Petera Jacksona s názvem Hobit: Neočekávaná cesta [7].

Pro tvorbu animací je tedy dobré dopředu vědět, v jaké snímkovací frekvenci bude animace přehrávána. V případě přehrání animace s jinou snímkovací frekvencí má za následek, že výsledná animace bude přehrána rychleji, resp. pomaleji, než byla zamýšlena.

Kapitola 3

Existující řešení

V této kapitole budou rozebrány existující řešení, kterými lze docílit přenesení anebo vytvoření těchto speciálních technik z programu Autodesk Maya do herního engine Unity.

3.1 Problémy tvorby a přenosu

První důvod, proč řešíme přenos, je nemožnost vytvářet v herním engine rig a zejména pro něj udělat *skinning* (když se zaměříme na 3D postavy). Pro tyto úkony jsou určené zejména 3D grafické programy. Proto se uchylujeme k přenosu modelu s pokročilejšími rigovacími technikami z jiného programu. Zde je ale další problém, že nedokážeme do souboru (např. formátu `.fbx`) uložit všechny potřebné atributy, a ty pak v herním engine přečíst. Podrobněji tento problém bude rozebrán v kapitole číslo 4.3.

3.2 Řešení používané v jiných herních enginech

V mnohých proprietárních herních enginech můžeme předpokládat, že podobné technologie mají již implementované dle požadavků her, které tvoří. Zejména pokud se jedná o hry, jako je například Death Stranding[8], které se velmi zaměřují na filmové záběry. Druhým vodítkem nám můžou být firemní nabídky práce, jako je Character TD[9]. Jedním z příkladů je herní společnost Naughty Dog[10] a jejich hra The Last of Us, která běží na herním engine Havok[11]. Ta používá mnoho vylepšení svalů nebo přídavných kloubů pro zacho-

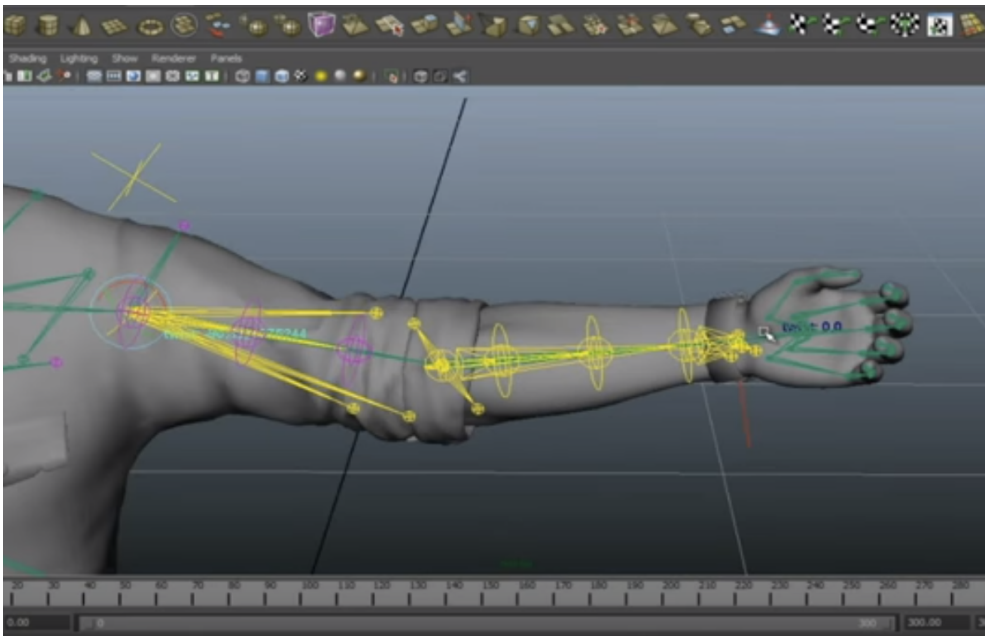
[8] Death Stranding je počítačová hra Hideo Kojimi, vytvořená firmou Kojima Productions, vydaná v roce 2019. Tato hra obsahuje velké množství filmových záběrů.

[9] Character TD *(technical director)* se zaměřuje zejména na rigování, ale mnohdy se v inzerátech dají dočíst informace o pokročilejších metodách a jejich exportování do herních enginů atp.

[10] `https://www.naughtydog.com/`

[11] `https://www.havok.com/`

vání přesnějších deformací. Ukázku z přednášky od 3D grafika Judd Simantova lze vidět na obrázku číslo 3.1.



Obrázek 3.1: Model postavy s pokročilými rigovacími technikami[12]ze hry The Last of Us. Můžeme zde vidět velké množství přídavných kloubů, které zaručují přesnější deformace. Na pravé straně vidíme *twist joints*, které mají navíc možnost manuálního ovládání pomocí kontrolerů [10]

3.3 Zapečení animace do modelu

Jednou z možností, jak se k problému postavit, je v programu Autodesk Maya za pomoci funkce *Bake Simulation*, která dle [15] převede animaci na klíčové snímky[13] a z mnoha animačních křivek udělá jednu. Tím pádem máme zajištěné, že dokážeme exportovat pokročilé rigovací techniky, animace pomocí inverzní kinematiky a mnoho dalšího. Nevýhodou je velikost souboru (případně rozkouskovaných souborů po jednotlivých animacích), ve kterém se nachází model, rig a celá animace. Druhou nevýhodou je omezená použitelnost. Dokážeme díky tomu sice převést pokročilé techniky do herních enginů, ale nemůžeme animaci pak nijak dynamicky upravovat, takže je tento přístup vhodný, když všechny animace vytváříme ručně a nadále se již nebudou mě-

[12]Na obrázku lze sice vidět pokročilejší rigovací techniky, ale stále může jít o přenos za pomocí tzv. zapečení animace do modelu viz 3.3.

[13]Klíčové snímky *(angl. keyframes)* při nichž je pro daný objekt zaznamenána pozice a další atributy v daném čase.

nit. Tuto metodu nemůžeme využít například pro *animation retargeting*, viz kapitola 4.4.3.

3.4 Existující řešení na přenos

V případě, že nechceme ukládat velké množství dat a nechceme mít animaci pevně svázanou s daným modelem, musíme využít jiného přístupu, které tyto techniky dokážou přenést. Na internetu existuje k dohledání jedno řešení, které ale není zcela veřejné pod názvem Unity Auto-Rigger [12]. Je zde odkrytá jenom část řetězce řešící Unity. Ze zveřejněných videí se dá usoudit, že nejspíše opravdu přenáší *twist joints* z programu Autodesk Maya do programu Unity. Z průzkumu zdrojových kódů se dá jen odhadovat, jak to celé funguje. Pravděpodobně při exportu v programu Autodesk Maya pro každý kloub ukládá velké množství dat. Ta potom aplikuje na klouby v Unity, které navíc vyhledává podle typu kloubu. Tato práce, jak bude představené v kapitole číslo 5, se naopak snaží ukládat co nejméně dat a díky získávání informací z grafu závislostí bude možná i další rozšiřitelnost.

3.5 Unity Animation Rigging

Novinkou od konce roku 2019 je balíček Animation Rigging[14], který se zabývá zejména úpravou a vylepšováním rigu za běhu hry. Jsou zde možnosti editace animací, přidávání omezení *(angl. constrains)*, IK, nebo i přidání *twist joints*. Je ale nutné model připravit v grafickém programu včetně skinování. Pro funkčnost je zapotřebí v Unity na daný model s kostrou přetáhnout několik skriptů, nastavit vazby rodič-dítě a vyplnit několik proměnných. Navíc musíme znovu nastavit pro každý kloub stejné chování, jako jsme již nastavovali v programu Autodesk Maya. Pro některé může být tato cesta zdlouhavá, a raději zvolí rychlou alternativu. Proto můžou uživatelé zvyklí vytvářet rigy v programu Autodesk Maya a chtějí je rychle přenést, využít knihoven vytvořených v rámci této práce.

Navíc v případě použití balíčku Unity Animation Rigging je zde problém s exportováním takto vylepšeného rigu ven z herního engine. S pomocí objektu zvaného *prefab*[15] sice můžeme v rámci Unity model přenášet, ale ven z herního engine ho už nedostaneme. Tento balíček je zatím stále jen v předběžné *preview* verzi a ne vše funguje tak, jak má.

Na druhou stranu přidáním tohoto balíčku ze strany Unity nám dává znamení, že má opravdu smysl zabývat se touto problematikou, protože je žádaná.

[14] Animation Rigging byl přidán do Unity dne 27. listopadu 2019 do verze 2019.3.0. [11]. Ale stále se balíček zatím nachází v předběžné *preview* verzi.

[15] *Prefab* obsahuje všechny komponenty a nastavení daného objektu. Díky tomu můžeme tvořit jednoduše nové instance.

Kapitola 4

Analýza

V této kapitole budou popsány požadavky na aplikaci. Na jejich základě bude probráno několik termínů, s jejichž pomocí nám bude umožněno tyto požadavky naplnit. V další kapitole bude na základě této analýzy vytvořen návrh aplikace, který se pak bude nacházet v kapitole číslo 6. Ta se zabývá implementací a je zde dopodrobna popsána.

Vzhledem k povaze zadání práce, která přímo předurčuje použité programy, bude lepší programy nestandardně rozebrat již v této kapitole. Je to z důvodu, že některé pojmy jsou vázány k dané aplikaci. V kapitole 5 věnované návrhu budou programy rozebrány již pouze z technického hlediska.

4.1 Analýza požadavků

Tato diplomová práce cílí na mírně zkušenější až zkušené rigery. Z tohoto důvodu je potřeba vytvořit knihovny, které budou použitelné pro obě skupiny uživatelů. Je nutné brát v potaz, že méně zdatní uživatelé se můžou v průběhu tvorby rigu ztratit. Proto je potřeba tuto skupinu nějak navigovat, aby dokázali provést jednotlivé kroky ve správném pořadí. V analýze požadavků proto bude popsáno, jaké jsou nároky na aplikaci. Poté vznikne návrh, který bude splňovat potřeby uživatelů. Tyto požadavky se dají rozdělit do dvou hlavních skupin. Funkční požadavky popisují funkcionalitu aplikace. Nefunkční požadavky kladou omezení na provedení.

4.1.1 Funkční požadavky

- Aplikace umožní vytvoření rigovacích technik *twist joints* a *interpolation joints* v programu Autodesk Maya.

- Při tvorbě těchto technik bude možné měnit parametry (například počet vytvořených kloubů).

- Bude možné tyto techniky z programu spolu s modelem exportovat do souborů.
- Poté v programu Unity soubory načíst a tím pádem pro model vytvořit lepší deformace.
- Bude umožněno takto vylepšené modely zakomponovat do výsledné hry.
- Aplikace by měla být snadno ovladatelná a přehledná.

4.1.2 Nefunkční požadavky

- Aplikace bude možné spustit minimálně na operačním systému Windows 10 a novější.
- Bude cílena na pokročilejší až velmi pokročilé rigery.
- Bude zaměřena zejména na co největší zjednodušení použitelnosti v programu Unity.
- Díky velkému množství ošetření chybových hlášek bude uživateli neustále napovídáno, aby se v průběhu tvorby neztratil.

4.2 Autodesk Maya

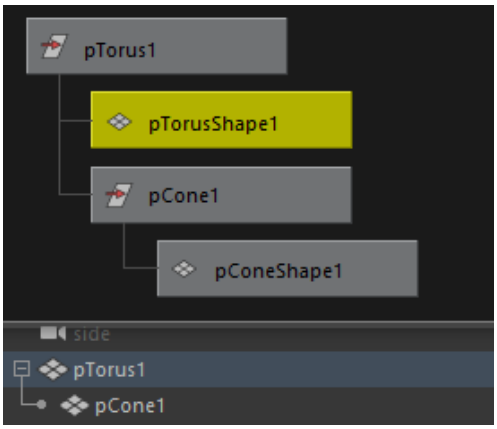
Pro tvorbu byl vybrán program Autodesk Maya[16]. Tento program má na poli profesionálního animačního software jednoznačně největší zastoupení. Další možné programy byly Autodesk 3ds Max[17], open-source program Blender[18] případně Houdini[19] od SideFX. Oproti ostatním programům má Autodesk Maya nejvíce rozvinuté animační nástroje a i v případě modelování a tvorbě filmových triků rozhodně nezaostává. Navíc díky saturovanému řetězci ve filmových studiích, který se v posledních 20ti letech ve filmovém průmyslu tvořil právě kolem programu Autodesk Maya, můžeme předpokládat, že tento program rozhodně pár let ještě na trhu bude hojně využíván.

4.2.1 Graf scény

Jedním ze základních prvků programu je tzv. graf scény *(angl. directed acyclic graph)*, který zaobaluje všechny objekty naší scény do hierarchické struktury. Jak zmiňuje [5], tak se zde nachází dva typy objektů: transformační uzly a tvarové *(angl. shape)* uzly daného objektu. *Shape* uzly jasně definují daný objekt např. geometrii objektu. Zatímco díky transformačnímu uzlu můžeme tento

[16] `https://www.autodesk.com/products/maya/overview`
[17] `https://www.autodesk.com/products/3ds-max/overview`
[18] `https://www.blender.org/`
[19] `https://www.sidefx.com/`

objekt různě natáčet, škálovat či posunovat. Na obrázku 4.1 je možné tuto strukturu vidět.



Obrázek 4.1: Graf scény, na němž lze vidět dva typy uzlů (transformační a tvarové). Ty jsou zasazeny do hierarchické struktury. Na dolní části obrázku se ještě nachází stejná situace v záložce *outliner*, se kterým uživatel přichází často do styku. Zde se zobrazují jen vazby mezi rodičem a dítětem

4.2.2 Graf závislostí

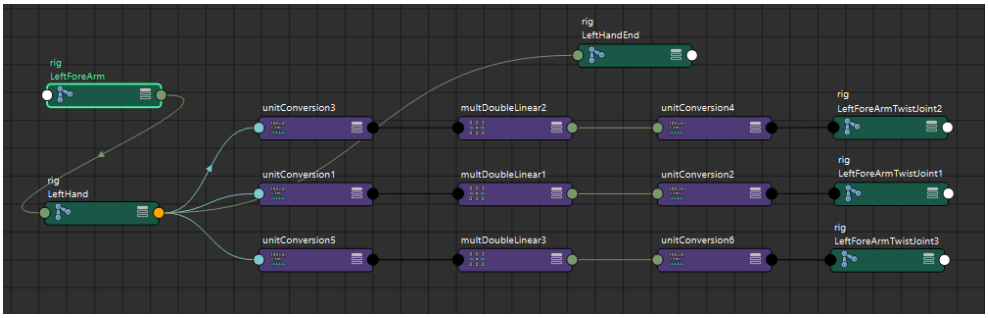
Druhým grafem je graf závislostí *(angl. dependency graph)*, který reprezentuje vazby mezi uzly a jejich propojení. *„Graf závislostí je zjednodušeně brán jako řada instrukcí, která popisuje, jak sestavit aktuální scénu úplně od začátku: vytvořte krychli A, přesuňte ji na jiné místo, vytvořte křivku B, promítněte křivku B na kouli A a vytvořte křivku na povrchu C atd.“* [16]. Například při vytváření *twist joints* můžeme poté v grafu závislostí schématicky vidět jednotlivé propojení mezi kanály, jak je zobrazeno na obrázku 4.2. Díky tomuto propojení zvanému *connection*, které má jasně daný směr, je možné řídit jednotlivé uzly.

Utility nodes

V grafu závislostí můžeme využít tzv. *utility nodes*. Ty nám umožňují provádět další operace, jako je například převod 2D textury do bump mapy, nebo podpora matematických operátorů jako je násobení, dělení, sčítání a odečítání. Jsou jednou ze skupin patřící do *shading nodes*.

4.2.3 Využití skriptování

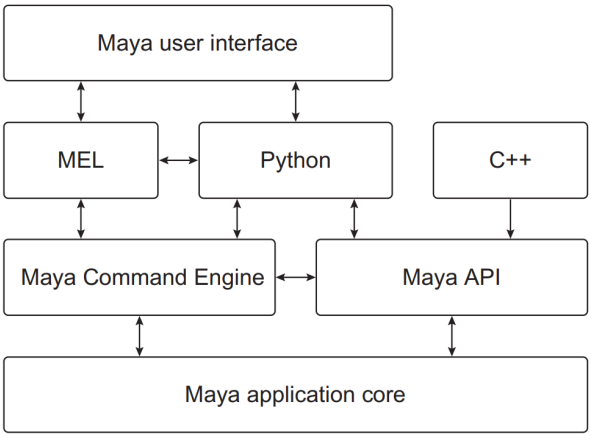
I když nám Maya poskytuje bohaté nástroje pro tvorbu rigu, tak se neobejdeme bez drobného skriptování pro jednodušší úkony až po tvorbu komplex-



Obrázek 4.2: Graf závislostí zobrazující propojení uzlů *twist joints* v kostře s využitím *tility nodes*. Pro zobrazení byl využit vestavěný *Node Editor*

ních nástrojů, které dokážou s pomocí uživatelského rozhraní sestavit automaticky rig[20].

Maya umožňuje psaní skriptů v jazyku MEL a Python. Pro programování nových modulů lze využít i programovací jazyk C++. Jak lze z obrázku 4.3 vidět, tak vývojáři vytvořili 4 různá programová rozhraní pro interakci, pro kterou využívají 3 různé programovací jazyky [17].



Obrázek 4.3: Komunikace v programu Maya na úrovni kódu. Převzato z [17]

MEL

Jazyk MEL *(Maya Embedded Language)* je zabudovaný skriptovací jazyk programu Autodesk Maya. Jak si lze povšimnout na obrázku, tak tento jazyk

[20]Do této skupiny patří tzv. modulární rigy. Ty automatizují rigování jednotlivých částí těla, např. ruky, až po sofistikovanější značené jako autorigy, které mají širokou paletu nástrojů pro sestavení celého rigu.

tvoří grafické rozhraní programu a dokáže komunikovat s Maya Commands Engine[21]. Bohužel jazyk nepodporuje objektově orientované programování.

Python

Tento jazyk se v průmyslu v poslední době těší stále větší a větší oblibě. Oproti jazyku MEL nabízí navíc datové struktury, lepší práce s řetězci a soubory, rozsáhlé knihovny třetích stran, možnost napojení na grafickou knihovnu PyQt[22] a podporuje objektově orientované programování.

S pomocí tohoto jazyka můžeme volat stejné MEL příkazy pomocí knihovny `maya.cmds`. Další možností je využití nástavby vytvořené třetí stranou zvanou PyMEL (knihovna `pymel.core`), která naopak využívá objektový přístup. Při programování může být tento přístup přívětivější, protože místo referencování objektu na základě textového řetězce jako v případě `maya.cmds` máme odkaz na objekt. Dokonce můžeme využít možnosti volání příkazů z Maya API, a to pomocí obalových tříd z knihoven `maya.OpenMaya` a novější verze `maya.api.OpenMaya`.

C++

Poslední možností je využití jazyka C++. Má rychlejší běh spouštěného kódu než dříve jmenované skriptovací jazyky. Navíc některé metody Maya API nejsou z jazyka Python přístupné [18]. Nevýhoda je ta, že při úpravě kódu je zapotřebí tento kód přeložit a znovu nahrát do programu Autodesk Maya. A navíc už jen příprava prostředí pro možnost programování je o mnoho náročnější než v případě spouštění skriptů v Pythonu nebo MELu.

Animation expression

Nejedná se sice o programovací jazyk jako takový, ale s jeho pomocí je možné také do určité míry ovládat chování objektů ve scéně. *„Expressions jsou instrukce, pomocí kterých Maya ovládá atributy v průběhu času. Jsou to vlastnosti objektu, například měřítko v* **x***, posun v* **y** *atd.“* [19]. Také najdou využití pro propojení různých objektů ve scéně, které se pak dokážou různorodě chovat na základě příchozích příkazů. Jednoduché využití může být v případě tvorby natahovací se páteře, kde nastavíme jednotlivým kloubům uvnitř páteře, jak se budou natahovat na základě změny pozice kontrolerů.

[21] Maya Command Engine obsahuje seznam příkazů, které jsou možné zavolat pomocí jazyka MEL nebo Python. Příkladem může být příkaz `polyCube`. Po jeho zavolání se vytvoří krychle.

[22] `https://pypi.org/project/PyQt5/`

4.2.4 Uživatelské rozhraní

Vytváříme-li pokročilejší skripty, zejména pokud jsou uživatelem konfigurovatelné, je žádoucí pro jednoduché ovládání vytvořit uživatelské rozhraní.

Maya pro tvorbu uživatelského rozhraní používá tzv. Maya ELF *(Extension Layer Framework)* příkazy, které vytvářejí a modifikují grafické elementy.

Alternativou je možnost využití frameworku Qt. Tento framework má pro Python několik verzí API, mezi nimi je to PyQt4, PyQt5, PySide, PySide2. V zásadě nabízejí podobné vlastnosti a kód lze překonvertovat pomocí nástroje `Qt.py`[23]. Z těchto možností je nejrozumnější volbou PySide2, protože navíc sama Maya toto API podporuje. Pro podporu jiných API by bylo nutné samostatně sestavit tyto API, a pak pomocí nástroje `SIP` přidat do programu Autodesk Maya. Další nespornou výhodou je možnost využití pro tvorbu grafického rozhraní v programu QT Designer[24], namísto kódování „naslepo“, jako v případě Maya ELF.

4.3 Problém exportu

Při exportování modelu se setkáváme s dvěma zásadními problémy:

1. export včetně pokročilejších technik z programu Autodesk Maya,
2. správné načtení a interpretace v programu Unity.

V prvním případě nám Maya bude odmítat uložit pokročilejší rigovací techniky, když je budeme ukládat jinam než do nativních formátů `.ma` a `.mb`. Ať už se jedná o tvorbu pomocí *utility nodes* nebo *animation expression*, výsledek bude stejný a bude se nám snažit tyto techniky „zapéct“ do modelu, jak bylo popsáno v kapitole 3.3. Je to z toho důvodu, že každý 3D grafický software má vlastní škálu nástrojů a atributů. K nim přísluší rozdílný koncept jejich ukládání. Proto pro zachování kompatibility formátu napříč programy lze do něho uložit pouze základní informace o modelu. Unity sice dle [20] nabízí možnost načíst `.ma` nebo `.mb` soubory (v případě, že máme na počítači nainstalovaný Autodesk Maya), ale nebudou tam tyto techniky zaneseny.

Druhému problému se nijak nevyhneme. Je potřeba v koncovém engine vymyslet, jak správně přečíst příchozí data, a na základě nich poté ovládat klouby ve hře. Možností by bylo uložit tato data do souboru spolu s modelem (pokud by zde byla možnost). Případně je uložit vedle a ta pak spolu s modelem přečíst a nastavit vše potřebné.

[23] `https://pypi.org/project/QtPy/`

[24] `https://doc.qt.io/qt-5/qtdesigner-manual.html`

4.4 Unity

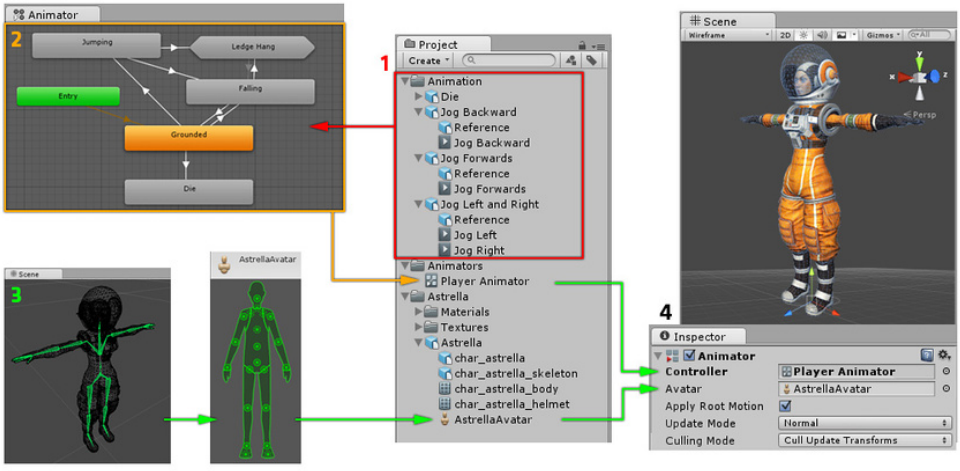
V rámci zadání práce byl vybrán engine Unity. Jedná se o jeden z nejrozšířenějších mezi nezávislými vývojáři. Ti si jej cení pro jeho jednoduchost, širokou škálu assetů[25], rychlost při prototypování a tvorby her. Vznikly v něm úspěšné hry jako je například Beat Saber, Hearthstone nebo Firewatch.

Další z dostupných enginů na trhu je Unreal Engine 4. Ten se spíše hodí na rozsáhlejší scény plné vegetace a pro realistickou vizualizaci. Je vhodnější spíše na větší projekty. Navíc vývojáři z Epic Games zde již přidali možnost, jak za pomoci nástroje *twist corrective* [21] vylepšit deformace, které se dají využít na krku, nebo právě na ruce.

4.4.1 Unity Mecanim

Pro animace je zde zakomponovaný vlastní animační systém. Přezdívá se Mecanim. Dle oficiální dokumentace [22] se mezi nejdůležitější prvky řadí:

- ovládání a jednoduché nastavení animací Unity objektů,
- možnost importovat animační klipy,
- *humanoid animation retargeting* - možnost přenosu animací mezi jednotlivými modely, viz kapitola 4.4.3,
- animování jen určitých částí modelu a maskování animací.



Obrázek 4.4: Unity animační systém [22]

[25]Herní asset je jakýkoliv znovupoužitelný objekt, ze kterých se skládá výsledná hra. Může jím být nějaký model, hudba nebo animace.

Pro přidání animace na námi zvolený model je potřeba udělat několik jednoduchých kroků. Ty jsou znázorněny na obrázku 4.4 a budou zde schématicky vysvětleny. V prvním z nich je potřeba disponovat příslušnou animací. Tu můžeme vytvořit v nějakém z 3D grafickém programu, který nám umožňuje tvorbu animací. Anebo využijeme webovou stránku `mixamo.com`[26], která dokáže po nahrání našeho modelu postavičku narigovat, aplikovat *skinning*, a poté si stáhnout předpřipravené animace z databáze. Případně můžeme stáhnout pouze animace. Ty dokážeme jednoduše namapovat na náš model. O této tématice podrobněji pojednává kapitola 4.4.3. Tyto animace, jak je vidno na přiloženém obrázku, potom vložíme do komponenty *Animator Controller*, na obrázku označen jako *Animator*. Zde můžeme přidat jednotlivé animační klipy, nastavovat přechody a s pomocí drobného programování nastavit, kdy dojde k přechodu mezi přehráváním jednotlivých animací. Chová se jako stavový automat. Pod bodem 3 na obrázku se skrývá vytvoření tzv. Avatara, který je nutný v případě humanoidní postavy. Ten se poté v bodě 4 spolu s *Animator Controller* předá komponentě daného modelu, jehož bude Unity animovat.

4.4.2 Typ rigu

Zásadním nastavením rigu u modelu je jeho typ. Unity nám dává celkem 3 možnosti: Humanoid, *Generic* a *Legacy*. Každá z nich má různé vlastnosti a využívá se v rozdílných případech.

Humanoid

Jak již název napovídá, tak se tento typ rigu využívá pro humanoidní postavy. Pro správnou funkci je nutné mít připravený rig s minimálně 15 hlavními klouby, které jsou na obrázku 4.5a znázorněné pod plným zeleným kolečkem. Tento rig můžeme rozšířit až na 25 kloubů. Po přidání prstů na ruce se dostaneme až k 55 kloubům, které bude animační systém schopen ovládat. Zásadní funkcionalitou je pak *animation retargeting*.

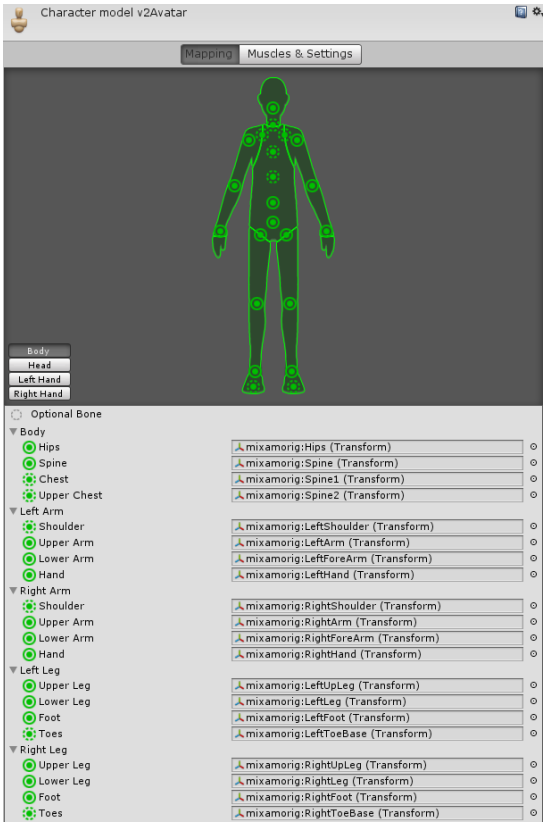
Generic

Naproti tomu zde žádné omezení na rig není. Lze ho využít libovolně. Pro postavičky, čtyřnohé charaktery nebo i pro animace střelby z pušky. Nevýhoda je ta, že nemůžeme využít *animation retargeting* a každá animace musí být vytvořena v animačním programu nebo s pomocí zabudovaného animačního editoru.

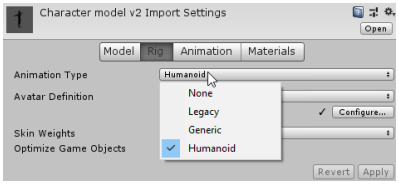
Legacy

Tento systém je zde pouze kvůli zpětné kompatibilitě. Byl využíván dříve, než byl přidán stávající Unity Mecanim.

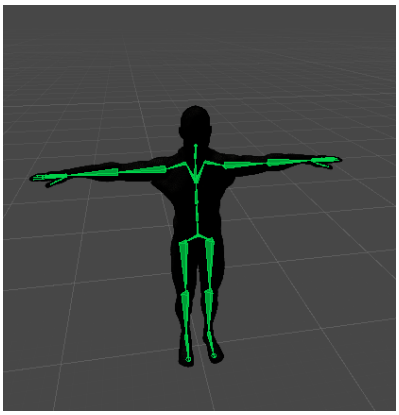
[26] `https://www.mixamo.com/`



(a) Ukázka humanoidního avatara s možností nastavení jednotlivých kloubů



(b) Nabídka pro výběr typu rigu



(c) Obrázek modelu včetně rozestavení kloubů, které budou ovládány pomocí systému Mecanim

Obrázek 4.5: Nastavení rigu u humanoidní postavičky v programu Unity

4.4.3 *Animation retargeting*

Tento mocný nástroj nám umožňuje vzít animace, které byly vytvořeny na jiný model, a přehrát je na našem libovolném modelu. Lze to ale provést jen u humanoidního rigu. Umožňuje nám to podobně vygenerovaná kostra (alespoň co se počtu a umístění základních kloubů týče). Tím pádem můžeme v grafickém programu vytvořit model postavičky a tento model nahrát do Unity. Pak dle libosti si stáhnout z nějaké databáze příslušné animace, které potom ve hře aplikujeme. Což nám dokáže výrazně urychlit tvorbu výsledné hry.

4.4.4 A póza a T póza

Umělci při tvorbě modelu si musí rozmyslet, v jaké póze postavičku chtějí ztvárnit. Pokud se nerozhodnou zpodobnit ji do umělecké pózy, ale předpokládají i návazné kroky (rigování a tvorba animací), je nutné tomu přizpůsobit

model. Ve světě rigování se rozlišují dvě takové pózy, jsou jimi A póza a T póza.

- **A póza**

  Ta se vyznačuje mírně roztaženýma nohama a pokrčenými rameny pod úhlem 45°, připomínají proto písmeno A. Jak se mluví v [5], tak zásadní změna deformací se nachází v oblasti krku a ramen. V případě uložení rukou v tomto úhlu budou tyto části mezi dvěma extrémy, kterými jsou plně roztažené ruce jako v případě T pózy a uložené podél těla. Tím pádem máme v případě rigování realistických postav dobrou představu, jak se budou ruce deformovat v oblasti ramen.

- **T póza**

  Naproti tomu T póza má rovně uložené nohy a ruce pod úhlem 90° od těla. Při tvorbě IK, kdy je nutné mít končetiny v jedné rovině, nám dokáže tato konfigurace ulehčit práci. Další skutečností je, že se stále používá v herních enginech a Unity tuto konfiguraci přímo vyžaduje, aby mohla vytvořit avatara a správně provést *animation retargeting*. Pro Unity proto použijeme tento typ pózy, kde je doporučené navíc mít palec na ruce pod úhlem 45° ve směru pohledu postavy.

4.5 Gimbal Lock

Problém, který trápí animátory již dlouhá léta, se jmenuje Gimbal Lock. Stává se to v situacích, když se jedna osa dostane paralelně s druhou osou rotace [24]. Při rotaci jak jedné, tak druhé osy bude rotace stejná, a tím pádem ztrácíme možnost rotovat kolem jedné z os. Někdy označováno jako ztráta stupně volnosti.

4.5.1 *Rotate order*

Tento krok je často přehlížen. Může přitom vyřešit mnohé problémy, kdy jedním z nich je i gimbal lock [5]. Jedná se o nastavení, v jakém pořadí budou na objekty (klouby) aplikovány jednotlivé rotace za sebou. V programu Autodesk Maya si můžeme povšimnout základního nastavení jako `xyz`. Rotace jsou ale aplikovány odzadu, to znamená, že první rotuje objekt kolem osy `z` (a tím ovlivní osu `y` a `x`), poté kolem `y` (a ovlivní i osu `x`) a nakonec kolem `x`. Tohoto můžeme využít a nastavit osy rotace tak, aby nejčastěji používaná osa pro daný kloub byla aplikována jako první a nejméně používaná uprostřed.

Úplně stejná situace je i v Unity, kde s tímto také musíme počítat nejen v případě vyhnutí se gimbal locku, ale také v případě čtení hodnot rotací u Eulerových úhlů, které jsou tímto ovlivněné. U globálních Eulerových úhlů se nejprve rotuje kolem `z` osy, poté kolem osy `x` a nakonec kolem osy `y`. V lokálních souřadnicích je to naopak rotace kolem `y`, pak `x` a nakonec `z`.

4.5.2 Možnosti reprezentace rotací

Existuje několik způsobů, jak reprezentovat a nakládat s rotacemi. Každá má své výhody i nevýhody.

- **Eulerovy/Tait-Bryan úhly**

  Eulerovy úhly mají pro uživatele příjemnou reprezentaci, hodnoty představují úhel ve stupních kolem jednotlivých os. Většina komunity je ale přezdívá nesprávně jako Eulerovy úhly. Lepší označení je jako Tait-Bryan úhly, protože nabývají některé z 6ti možných posloupností rotací `xyz, yzx, zxy, xzy, zyx, yxz`. V případě „Pravých Eulerových úhlů“ by se vždy jedna osa vyskytovala v rotaci dvakrát `zxz, xyx, yzy, zyz, xzx, yxy`. Mezi jejich výhody je možnost reprezentovat úhel i větší než 180°, takže při interpolaci animačních křivek můžeme zadat rotaci 540° kolem jedné osy a program tuto rotaci vykoná[27]. Bohužel v případě interpolování zde může vzniknout gimbal lock.

- **Quaterniony**

  Naproti tomu quaterniony se dokážou vyhnout gimbal locku. Ale nedokážou reprezentovat větší než 180° rotace. V případě rotace 540°, kde bychom za použití Eulera udělali 1,5násobek rotace kolem jedné osy, tak zde vykoná jen 180°. Navíc při překročení 180° bere vždy ten menší úhel. Proto místo rotace 260° v jednom směru udělá rotaci 100°, ale v opačném směru. Pro uživatele jsou navíc hůře čitelné (jsou reprezentovány 4řmi `x,y,z,w` složkami s hodnotami od -1 do $+1$).

  Při skládání rotací quaternionů v Unity musíme dodržet správné pořadí skládání rotací. Pro quaterniony je:

  ```
  QvyslednaRotace = QrotaceZ * QrotaceY * QrotaceX;
  ```

Mezi další způsoby se řadí reprezentace pomocí rotační matice nebo osou–úhlem *(angl. axis-angle)*.

4.6 *Skinning*

Jak již na začátku bylo řečeno, tak tento krok zajišťuje propojení vytvořené kostry s polygonální sítí, aby bylo možné deformovat kůži postavičky na základě změny kostry. Pro jejich tvorbu se využívá zejména tzv. *smooth skinning*, na který se v této kapitole zaměříme. Existuje ještě druhá skupina. Jmenuje se *rigid skinning*, ale z důvodu, že každý vrchol polygonální sítě může být ovládán pouze jednou kostí, nám neposkytuje nejlepší výsledky[28].

[27] V programu Maya se animační křivky počítají tímto způsobem, pokud není nastaveno jinak.

[28] Tento efekt můžeme docílit i pomocí *smooth skinning*, kde nastavíme počet kloubů ovlivňující každý vrchol polygonální síťě na jeden.

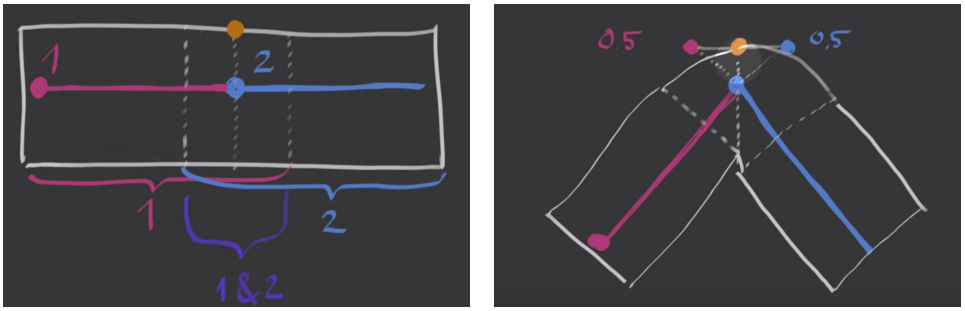
V programu Autodesk Maya jsou poskytnuté dvě různé metody s možností použití obou dvou zároveň, a tím nakombinovat jejich silné stránky. V Unity je dostupný bohužel jen *linear blending skinning* [27] s možností maximálně 4 kloubů ovlivňující daný bod.

- ***Classic linear blending skinning***

  V programu Maya se nachází pod pojmem *classic linear*. Není výpočetně náročná a poskytuje dostatečně uspokojující výsledky, ale v některých případech selhává, kdy dochází k úbytku objemu. Může jít o větší rotace kloubů a nebo v případě většího kroucení v kloubu, jak lze vidět na obrázku 4.7c.

  Uvažujme situaci jako na obrázku 4.6. Máme dva klouby (označené jako 1 a 2) a polygonální síť ve tvaru válce. Ta je ovlivňována na levé straně pomocí prvního kloubu a na druhé pomocí druhého. Na středu vzniká situace, kdy je síť ovládána pomocí obou kloubů. Chceme zjistit novou pozici bodu, který je označen oranžovou barvou. Na pravé straně obrázku je znázorněná situace po ohybu. Pro výpočet nové pozice posuneme tento bod do prozatímních pozic, na kterých by byl v případě ovlivnění čistě jen pomocí jednoho nebo druhého kloubu. Výsledná pozice bodu se nachází na spojnici s posunem, závislým na váhách, kterými jednotlivé klouby přispívají.

  Z tohoto důvodu v případě mnohem extrémnějších rotací bude docházet k úbytku objemu. V případě 180° rotace vznikne defekt, jak je vidět na obrázku 4.7c, protože bod na spojnici se protne přesně do středu objektu. Úbytek objemu se dá do jisté míry řešit pomocí *interpolation joints*.



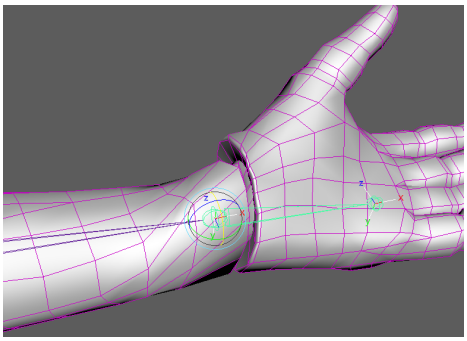
(a) Před ohybem (b) Po ohybu

Obrázek 4.6: Ohyb v metodě *linear blend skinning*. Obrázek převzatý z: [26]

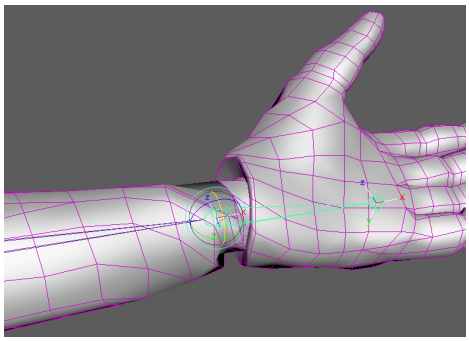
- ***Dual quaternion***

  Hlavní výhodou *dual quaternion* metody je ta, že se snaží zachovávat objem [25]. Je ale výpočetně náročnější. Používá se proto hlavně v místech jako jsou ramena nebo loket, abychom předešli úbytku objemu.

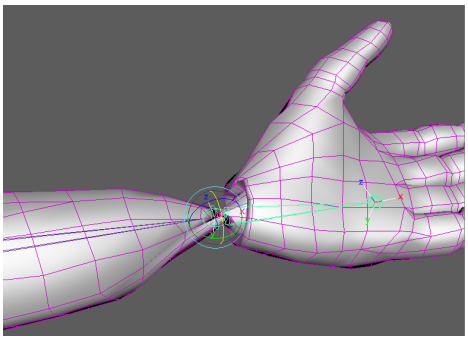
Na přiložených obrázcích jsou patrné rozdíly. Z obrázků 4.7a, 4.7b a 4.7c můžeme vidět, jak v případě použití metody *dual quaternion* nedochází k úbytku objemu, a tím vzniku nepřirozené situace. A mezi obrázky 4.7a a 4.7b sledujeme „přeskočení“ v případě překročení úhlu nad 180°. Je to z důvodu, že `skinCluster`[29] nedokáže reprezentovat rotaci větší než 180°.



(a) *Dual quaternion*, rotace v ose `x` 180.0°



(b) *Dual quaternion*, rotace v ose `x` 180.1°



(c) *Classic linear*, rotace v ose `x` 180.0°

Obrázek 4.7: Porovnání metod skinování

[29]SkinCluster v programu Maya je uzel, který se používá jako deformátor ke svázání geometrie ke kloubům.

Kapitola 5

Návrh

V této kapitole budou popsány použité technologie, které byly využity při implementaci. Dále budou představeny minimální požadavky nutné, jak pro tvorbu a přenesení rigu s těmito technikami, tak pro koncové uživatele, kteří si budou chtít aplikaci spustit na svém počítači. Na základě požadavků z analýzy byl sestaven postup přenosu pokročilejších technik. Konec kapitoly se věnuje návrhu uživatelského rozhraní a zjednodušenému modelu tříd pro knihovny v obou programech.

5.1 Použité technologie

5.1.1 Autodesk Maya

Programovacím jazykem byl vestavěný Python ve verzi `2.7.11`. K němu bylo dopomáháno nástavbou PyMEL ve verzi `1.0.10`. Pro tvorbu uživatelského rozhraní postačila knihovna Maya ELF. Při implementaci bylo využito standardních vestavěných Python knihoven včetně modulu pro exportování do `.JSON` formátu. Pro vývoj byla použita verze programu Autodesk Maya 2018.

I když podpora jazyka Python ve verzi 2.7. končí k datu 1.1.2020, tak Autodesk zatím nevydal žádné prohlášení, že se bude přecházet na verzi `3.x`. Až Autodesk přestane verzi `2.7.` podporovat, tak by mělo stačit překonvertovat kód pomocí nástroje `2to3`[30]. Program by momentálně měl fungovat ve verzích Maya, kde je dostupný Python ve verzi `2.7.`

5.1.2 Unity

Pro Unity z důvodů využívání moderních konstrukcí jazyka je zapotřebí minimální verzi jazyka C# `7.1`. Ta je podporovaná v Unity od verze 2018.3. Jsou zde také využity knihovny třetích stran SimpleJSON a JsonDotNet, které po-

[30] `2to3` je oficiální Python program pro konverzi Python kódu verze 2 do verze 3.

žadují .NET Framework ve verzi `4.4` a vyšší. Pro vývoj byla využita verze Unity 2019.1.1f1.

5.2 Minimální požadavky pro běh vytvořené hry

Pro možnost spuštění vytvořené hry je potřeba dodržet minimální systémové a hardwarové požadavky, kterými jsou podle oficiální stránky [23]. Operační systém: Windows 7 s nainstalovaným Service Pack 1 a vyšší, macOS ve verzi 10.12 a vyšší, Ubuntu 16.04[31] a vyšší. Grafická karta, která podporuje DirectX 10 (shadery verze 4.0).

5.3 Řetězec pro přenos pokročilejších rigovacích technik

Tvorba a přenos technik z programu Autodesk Maya do Unity je přímočará. Na vývojovém diagramu 5.1 je znázorněno, jak jednotlivé kroky při přenosu na sebe navazují. Těmito kroky jsou:

1. Řetězec začíná v programu Maya, kdy si uživatel načte hotový model, pro který by rád vytvořil rig a přidal některou z pokročilejších technik.

2. Poté buď načte předpřipravenou kostru, nebo si vytvoří vlastní. Ta ale musí být kompatibilní s Unity animačním systémem, jak je popsáno v kapitole 4.4.2.

3. Pak uživatel musí vlastnoručně klouby této kostry napozicovat do modelu. Na místech, která budou obsahovat pokročilejší techniky, nesmí zapomenout správně nastavit lokální osy rotace. Jak má postupovat, je uvedené v uživatelské příručce C.2.2.1.

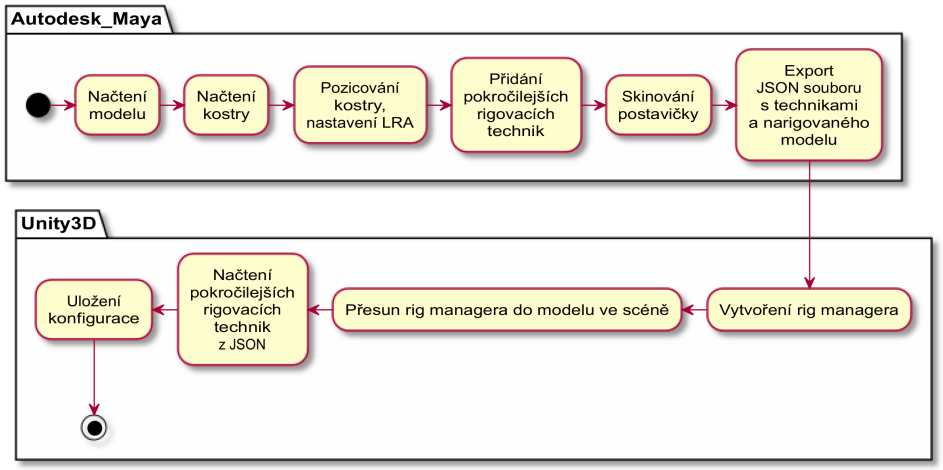
4. V dalším kroku následuje samotné přidávání pokročilejších technik.

5. Nakonec se provádí skinování postavičky. Pro nejlepší výsledky je doporučeno využít metodu kreslení vah, minimálně na nově přidané pokročilejší techniky.

6. Až jsme s výsledkem spokojeni, tak exportujeme model a spolu s ním speciální soubor ve formátu `.JSON`, který obsahuje informace o aplikovaných pokročilejších technikách. Tento soubor je klíčový pro přenos do Unity.

[31] Na internetu se dají najít i návody, jak rozjet Unity i na jiných distribucích linuxu než na Ubuntu, respektive distribuci Debian. Je proto možné bez záruky předpokládat, že to bude fungovat i na dalších.

7. V herním engine jsou jednotlivé kroky už velmi rychlé. Uživatel vytvoří speciální objekt, který zajišťuje sestavení rigu.
8. Tento objekt přesune ve scéně do modelu (ten, který byl v programu Maya exportován). Nesmí zapomenout pro tento model nastavit typ rigu na `humanoid`.
9. A poté pro tento model pomocí tohoto objektu načte `.JSON` soubor, který obsahuje popis, jak ovládat tyto nové klouby.
10. Nakonec Unity uloží tato data, aby po opakovaném zapnutí engine nebo i vytvořené hry se nemuselo nic dalšího nastavovat.



Obrázek 5.1: Vývojový diagram přenosu pokročilejších rigovacích technik. Obrázek vytvořený pomocí programu PlantUML[32]

5.4 Uživatelské rozhraní

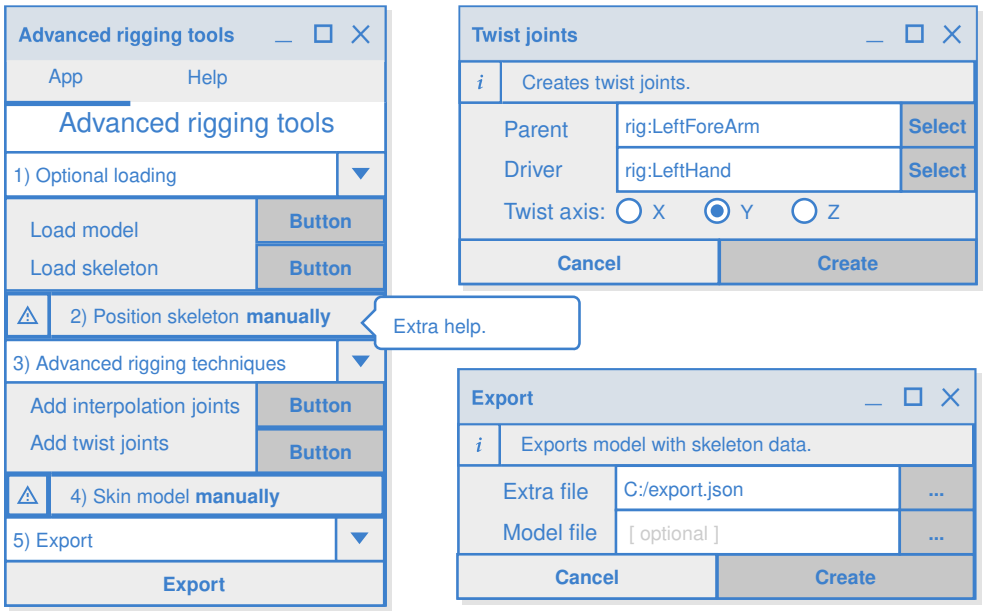
Z důvodu, že tuto aplikaci budou využívat dvě skupiny uživatelů (zkušenější a méně zkušení), je potřeba navrhnout takové uživatelské rozhraní, které bude pochopitelné pro obě skupiny (méně zkušená skupina se neztratí). Proto v obou programech si můžeme povšimnout různých informačních textů. Ty méně zkušeným uživatelům radí a pomáhají neztratit se.

5.4.1 Autodesk Maya

Bylo vybíráno mezi dvěma návrhy. V prvním návrhu bylo zamýšleno udělat nástroj kompaktních rozměrů a zakomponovat jen nejdůležitější tlačítka. Takže

[32] `https://plantuml.com/`

by aplikace obsahovala pouze možnost načtení kostry, přidání pokročilejších technik a exportování. Tento návrh by jistě ocenili pokročilejší rigeři, kteří mají většinou při tvorbě otevřeno mnoho nástrojů. Méně zkušení by mohli mít problémy a nevěděli by, v jaké fázi rigování mají tyto nástroje použít. Z tohoto důvodu byl vybrán druhý návrh. Ten zde bude rozebrán.



Obrázek 5.2: Návrh uživatelského rozhraní pro program Autodesk Maya. Vytvořeno v programu Pencil[33]

Hlavní okno aplikace (na obrázku 5.2 vlevo) shromažďuje všechny potřebné ovládací prvky, které uživatel bude potřebovat. Je rozděleno do 5ti částí, kdy při tvorbě modelu je zapotřebí dodržet správné pořadí návazných kroků. Uživatel proto postupuje při tvorbě odshora dolů. Jsou jimi:

1. Nepovinné načtení modelu a kostry.

2. Poté uživatel musí sám napozicovat kostru do modelu a nastavit LRA.

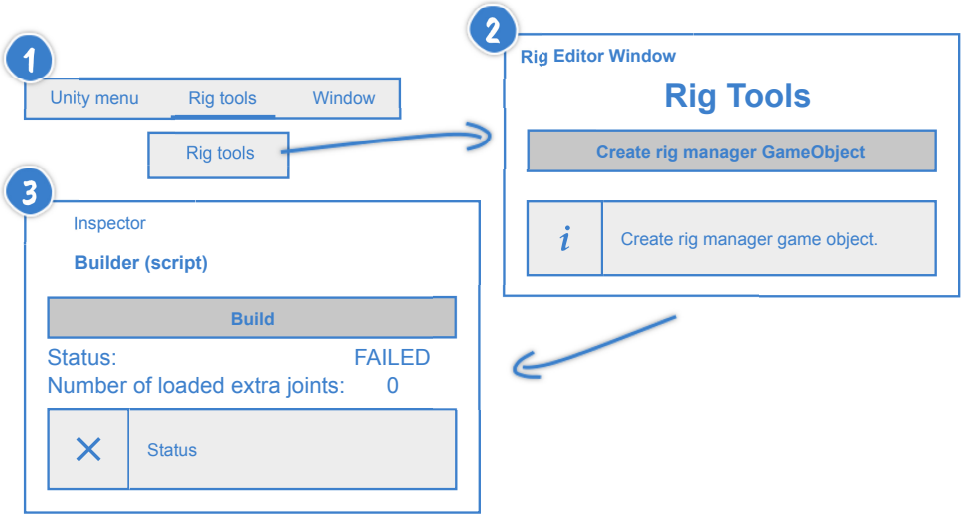
3. Přidání pokročilejších technik. Po kliknutí na tlačítka vyskočí další okno (na obrázku 5.2 vpravo nahoře), které se zabývá přidáváním dané pokročilejší rigovací techniky. Pro zadání kloubu je nutné označit příslušný kloub ve scéně, a poté kliknout na tlačítko **Select**. Vytvoření technik bude provedeno po stisku tlačítka **Create**.

4. Manuální skinování.

[33] `https://pencil.evolus.vn/`

5. A posledním krokem je export. Uživatel si vybere místo uložení souboru s pokročilými technikami a volitelně místo uložení narigovaného a naskinovaného modelu.

Kroky číslo 2 a 4 jsou zapotřebí udělat manuálně. Je proto tento krok zvýrazněn ikonkou. Kroky 1, 3 a 5 mají samorozbalovací nabídku pro ušetření místa. Aplikace je navíc rozdělena do dvou záložek, kde v jedné se nachází samotná aplikace a v druhé dodatečné informace k postupu. Navíc i po najetí myší na některá tlačítka jsou uživateli představeny dodatečné informace. Pro zajištění správného provedení jednotlivých kroků bude v případě chyby uživateli poskytnuta zpětná vazba, aby ji mohl napravit.



Obrázek 5.3: Návrh uživatelského rozhraní pro program Unity. Vytvořeno v programu Pencil

5.4.2 Unity

U Unity je situace mnohem jednodušší, jak ukazuje obrázek 5.3. V menu programu se nachází tlačítko (bod 1). Po jeho stisknutí se otevře okno. Má jediný úkol - vytvořit objekt ve scéně, který aplikuje pokročilejší techniky na model (bod 2). Ten po najetí na objekt v záložce *inspector* bude nabízet načtení `.JSON` souboru kostry daného modelu (bod 3). V tomto bodě bude uživateli nabídnuta zpětná vazba pomocí stavového textu a informačních zpráv. V případě nějaké chyby bude upozorněn a přečte si podrobnější informace o chybě v konzoli.

5.5 Doménový model

Pro zjednodušení a čitelnost obrázku je místo třídního modelu použit doménový model. Na něm se nachází vztahy mezi třídami a jen nejdůležitější metody.

5.5.1 Autodesk Maya

Doménový model se nachází na obrázku 5.4. Skripty jsou rozděleny do několika složek dle určení:

- **maya_rigging** - zde je implementace pokročilejších technik,
- **maya_windows** - má na starosti tvorbu hlavního okna,
- **rig_utils** - pomocné skripty, pro exportování, ukládání atp.,
- poslední skripty se nacházejí v kořenovém adresáři. Ty se zabývají přípravou a spuštění hlavního okna.

Vstupem do aplikace je soubor `import_all.py`, ve kterém dojde k naimportování všech potřebných knihoven a jejich znovu načtení v programu Maya. V souboru `art_rig_app.py` pak dojde ke spuštění hlavního okna v programu Maya.

- **maya_win**

  Tato třída vytváří hlavní okno programu a zaobaluje celou aplikaci. Z ní jsou pak volána další okna zodpovědná za tvorbu *twist joints* a *interpolation joints*, případně exportu. Každá z nich se volá metodou, po jehož zavolání vznikne další samostatné okno. Jednotlivá okna jsou nezávislá, takže uživatel může mít otevřené všechny typy oken najednou.

- **twist**

  Tato třída je zodpovědná za tvorbu *twist joints*. Je vytvořena po zavolání z hlavního menu. Uživateli je ihned zobrazeno okno metodou *_twist_window()*, které mu zprostředkovává tvorbu kloubů. Nachází se zde dvě třídní proměnné `parent_twist_joint` a `driver_of_twist_joint`. Ty jsou použité jen v případě tvorby nových kloubů z důvodu větší čitelnosti kódu. Důležitá metoda je *_create_twist_joints(joint_amount_nmb)*, která se volá po obdržení požadavku od uživatele na tvorbu nových kloubů. Metoda zastřešuje celou sekvenci, která je potřebná pro vytvoření kloubů, volá proto postupně další metody. Jmenovitě je to *_create_curve_between_2_joints()*, která nejprve vytváří křivku, na kterou následně v dalším kroku v metodě *_create_jnts_along_curve()* vytváříme nové klouby. Zde je správně nastavíme a vytváříme pro ně nadřazené transformační uzly, jak je popsáno v kapitole

6.1. Výstupem z této metody je list nově vytvořených *twist joints*, které vstupují do poslední metody *__setup__by__matrix__nodes()*, kde jsou propojeny kanály pomocí *connections* a je jim vytvořeno ovládání. Po tomto posledním kroku se okno znovu inicializuje a je připraveno pro další použití.

- **interpolation**

  Zde je podobná situace jako v předchozím případě. Třída je zodpovědná za tvorbu *interpolation joints*. Po zavolání z hlavního menu vznikne opět nové okno, kde si uživatel nastavuje parametry. Třídní proměnné jsou opět využité pro ulehčení práce a větší čistotu a jednoduchost kódu. Po zadání požadavku tvorby nových kloubů se jde do metody *__create__interpolation__joints(percent__of__move)*, která zastřešuje celou tvorbu. Zde jsou tyto klouby vytvořeny, správně nastaveny a nakonec v další metodě *__setup__by__matrix__nodes()* je jim obdobně nastaveno chování.

- **exporter**

  Třetí, poslední okno se zabývá exportováním. Znovu je okno vyvoláno z hlavního menu. V něm uživatel nastaví cesty pro jednotlivé soubory. Pak se spustí export pomocí metody *__export__extra__file()*. V ní nejprve dojde ke smazání nepoužitých *connetions*. Zbylé si uložíme do listu, to má na starosti metoda *__list__all__nodes()*. Překonvertujeme na `.JSON` formát pomocí *__prepare__json__data()* a ta nakonec uložíme pomocí *__write__to__JSON__file()*. Poté ještě v případě exportování modelu exportujeme i daný model za pomoci *__export__model()*.

- **save__load**

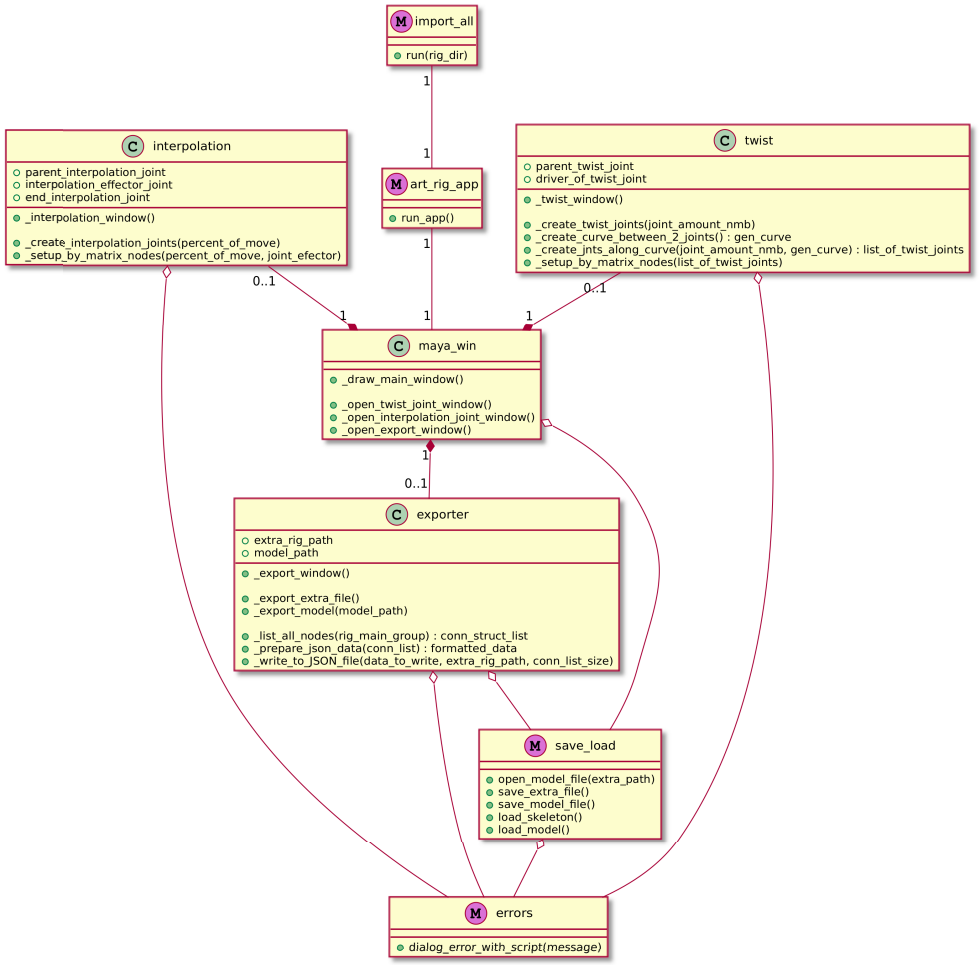
  Zde se nachází skupina metod, které mají za úkol otevírat načítací okna a připravovat správné cesty, kde mají tato okna otevřít. Metoda *__open__model__file(extra__path)* má za úkol načítat modely ve formátu `.fbx`, `.ma` nebo `.mb`. Tyto metody jsou volány ze tříd `maya_win` a `exporter`.

V práci je ještě soubor `rig_errors.py`, který se zabývá výpisem chybových hlášek do konzole a tvorbou vyskakovacího okna. Ve třídách se nachází ještě několik validačních metod. Ty mají na úkol kontrolovat uživatele a v případě nevhodného vstupu uživatele na tuto skutečnost upozornit.

5.5.2 Unity

Pro Unity se doménový model nachází na obrázku 5.5. Rozdělení skriptů je dle jejich určení následovné:

- **Rig** - skripty zabývající se sestavováním a ovládáním rigu,



Obrázek 5.4: Doménový model v programu Autodesk Maya

  - **JointData** - pomocné skripty pro ovládání a tvorbu kloubů,

- **UI** - skripty pro uživatelské rozhraní,
- **Utils** - pomocné skripty pro načítání a vyhledávání.

Byly napsány ještě skripty pouze pro testování a tvorbu testovací aplikace. Nemají vliv pro samotnou funkci knihovny, proto zde nejsou rozebrány. Tyto skripty se nachází ve složce **ScriptsForTesting** a jsou dostupné na přiloženém médiu v Unity projektu.

- **RigEditorWindow**

  Samostatná třída, která vytváří v Unity menu novou položku na místě *Advanced Rig -> Rig Tools*. Ta po stisku tlačítka **Create Rig Manager gameObject** vytvoří objekt s názvem `Rig Manager` a uloží ho do scény.

- **BuilderEditor**

  Má na starosti úpravu *inspector* okna na objektu `Rig Manager`. Jedná se prakticky o vstupní bod, ze kterého se vše ostatní volá. Po nastavení pokročilejších kloubů se zde volá funkce *SaveExtraJointsData(builder)*. Ta má za cíl serializaci nově vytvořených dat.

- **Builder**

  Tato třída nese hlavní sestavovací metodu. Klíčovou metodou je *Build(path)*, která zaštiťuje celé sestavování. Nejprve se načte `.JSON` soubor pomocí třídy `Loader`. V dalším kroku jsou tato data v třídě `DependencyGraph` přečtena a jsou předpřipravována spojení, která budou nastávat mezi dvěma klouby a jejich kanály. Tato data jsou uložena do listu ve třídě `DependencyGraph`, na kterou máme v této třídě odkaz. Proto můžeme v dalším kroku jít do metody *ApplyToModel(dependencyGraph)*, kde projdeme všechna uložená spojení a pro jednotlivé speciální klouby vytvoříme komponentu `JointTransformModifier`. Ta má za cíl řízení daného kloubu na základě chování řídících kloubů (jeho odkazy jsou předány parametrem). Pak, jak již bylo popsáno v předchozí třídě, dojde k uložení dat. Naopak při spouštění hry je nutné zase tato data načítat. Stará se o to vestavěná Unity metoda *Start()*, která volá metodu *LoadFromJSON(rigManager)* ve třídě `JointDataPersistent` pro načtení všech serializovaných dat, aby bylo možné ve hře klouby ovládat.

- **DependencyGraph**

  Tato třída se jmenuje `DependencyGraph`, i když v momentální implementaci spíše jen přeformátovává příchozí `.JSON` soubor do struktur dále použitelných v Unity. V budoucnosti se zde, v případě rozšíření, bude nacházet metoda, která načte větší část grafu a přetransformuje do použitelných struktur v Unity.

  V této třídě se nachází metoda *ConstructGraph(loader)*. Ta je volána z `Builder` třídy. Má za účel z příchozí neformátované `.JSON` struktury ji projít a vytvořit list spojení, které se dají aplikovat mezi jednotlivými klouby v Unity. Prochází proto v této metodě všechna načtená spojení a přetransformovává je do speciálního listu, který je tvořen za pomoci struktur `Connection`. Jedna struktura obsahuje informace propojení mezi kanály na dvou objektech a hodnotu, s jakým násobkem bude hodnota tohoto spojení násobena. Při tvorbě těchto kanálů je využito abstraktní třídy `ChannelObject` a jejich odvozených tříd `RotateX`, `RotateY`, `RotateZ`.

- **JointTransformModifier**

  Tato komponenta se přidává na klouby, které budou vytvářet speciální techniky. Je tvořena v `Builder` třídě a jeho metodě *ApplyToModel(de-*

pendencyGraph), kde pro každé příchozí spojení vytváříme toto propojení v Unity. Všechna data na daném kloubu jsou ukládána do třídy `JointSerializedValues`. Ta se nachází v souboru `JointTransformSerializer.cs`.

- `JointTransformSerializer.cs`

  Z důvodu, že Unity nedokáže jednoduše serializovat pokročilejší datové typy, bylo nutné udělat zřetězení jednotlivých struktur, či tříd, aby serializace byla možná. Proto jsou jednotlivé prvky roztahány po několika třídách, které jsou využity při serializaci. První z nich je `JointSerizalizedValues`. Je vytvořena pro každý jednotlivý kloub a obsahuje list propojených kanálů. Jeden propojený kanál je definovaný pomocí `OneSerializedConnection`. Ta je tvořena dvojící `ChannelObject` objektů představující jednotlivé kanály. Do druhého z nich je pak ještě přidána hodnota, která představuje, s jakým poměrem se bude příchozí hodnota škálovat na novou hodnotu.

- **ChannelObject**

  Abstraktní třída představuje popis chování transformačního kanálu. Z této třídy se dále dědí na `RotateX`, `RotateY` a `RotateZ`. Díky polymorfismu je proto zajištěné, že v průběhu programu můžeme nakládat s každým `ChannelObject` objektem nezávisle na jeho typu. A v případě přidání dalšího kanálu je možné jednoduše přidat další třídu. Ta bude dědit z této abstraktní třídy. Nevýhodou je ale složitější serializace dat.

- **JointData**

  Zde se nachází další data, která jsou spojena s tvorbou kloubů. Je zde výpis možných podporovaných kanálů nebo klíčů, které se dají číst z `.JSON` souboru. Při validaci dat jsou proto příchozí data kontrolována s těmito hodnotami. Nakonec se zde nachází ještě struktura `Node`, která v sobě udržuje název kloubu a jeho transformační kanál, který pak bude ovládaný programem.

- **JointDataPersistent**

  Zde se nachází dvě metody *SaveIntoJSON()* a *LoadFromJSON()*. První z nich zajišťuje serializaci dat pro ovládání všech kloubů, které zprostředkovávají pokročilejší metody. Je volána ihned po sestavení kloubů v třídě `BuilderEditor`. Druhá metoda naopak zajišťuje načtení těchto hodnot, je volána v metodě *Start()* v třídě `Builder`. Tím pádem je zajištěné načtení všech nastaveních při startu aplikace.

- **BuildStatus**

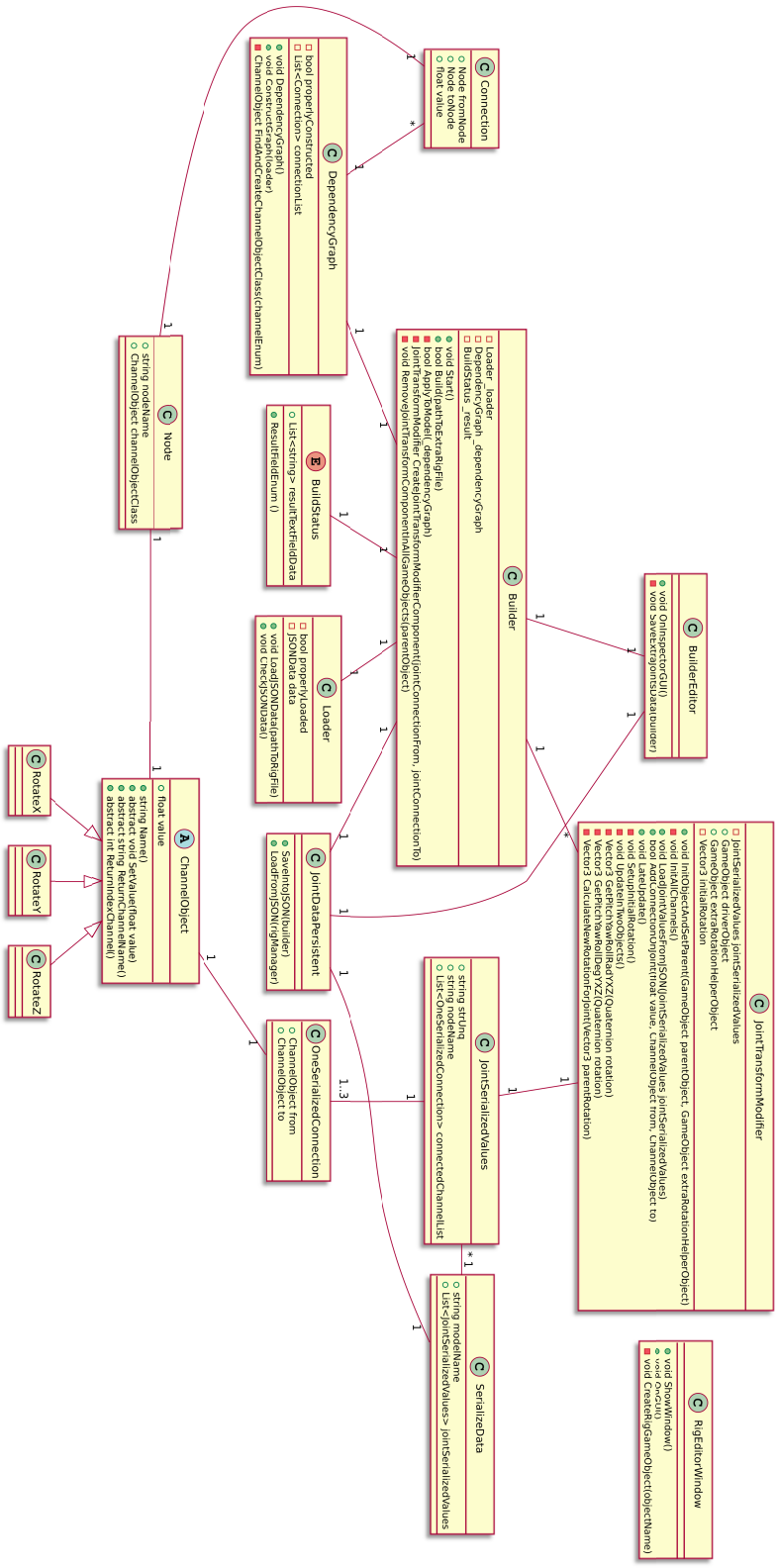
  V tomto výčtu hodnot se jen nachází možné stavy, kterých může `Rig Manager` objekt nabývat. Bylo to nutné provést z důvodu, kdybychom

měli více `Rig Manager` objektů ve scéně, tak aby si každý držel svůj status při přecházení mezi nimi.

- **Loader**

  Má za úkol načíst `.JSON` data z programu Maya do interní struktury. Tato data pak budou překonvertována v třídě `DependencyGraph`.

Mezi další třídy můžeme zařadit `AssetTestHelper`. Stará se o kontrolu a vyhazování chybových hlášek do konzole. Třídu `StringHelper` sloužící pro parsování a kontrolu řetězců, a nakonec třídu `GameObjectHelper` obsahující další pomocné skripty pro vyhledání objektů ve scéně.



Obrázek 5.5: Doménový model v programu Unity

Kapitola 6

Implementace

V této kapitole bude popsána implementace obou knihoven. Vychází se ze znalostí popsaných v předchozích kapitolách.

Diagram popisující řetězec je představený v kapitole pojednávající o návrhu 5.3. V této kapitole budou rozebrány důležité části z implementačního hlediska, které jsou využity při jednotlivých krocích přenosu.

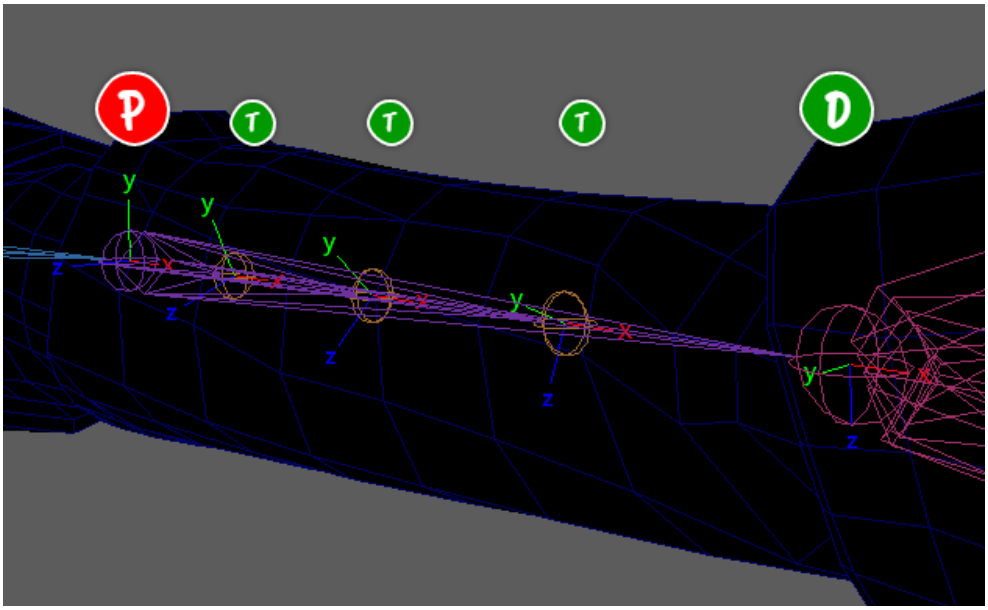
6.1 Implementace pokročilejších technik

Tvorba pokročilejších technik je v obou případech dělána za pomoci skriptů v jazyku Python, která volá příkazy z Maya Command Engine. Takže uživatel je odstíněn od složité manuální tvorby a jen si vybere pozici, kde tuto techniku chce vytvořit a provede dodatečné nastavení. O vše ostatní se program postará sám.

- ***Twist joints***

  Tato technika má za cíl propagovat deformaci. Ta je způsobena rotací jednoho kloubu v hlavní ose rotace po délce kosti směrem k jeho rodiči. Využívá se zde proto vytvoření dalších kloubů, které jsou připojené ke společnému rodiči. Jejich rotace je řízena původním následujícím kloubem v řetězci. Budou rotovat s částečným vlivem řídícího kloubu. Na obrázku 6.1 můžeme vidět situaci při tvorbě 3 *twist joints* na předloktí. Řídícím kloubem je zápěstní kloub, který v základní formě by měl deformovat síť jen v okolí kloubu a nepropagovat žádnou deformaci ve směru k loketnímu kloubu.

  Při tvorbě je zde vytvořena křivka mezi řídícím a rodičovským kloubem. Poté na základě počtu kloubů, které si uživatel navolil na obrazovce, jsou tyto klouby uniformně vytvořeny na křivce. Tvořeny jsou v prostoru rodiče, proto jejich směr při nastavení nulových rotací a *joint orient* bude směřovat na řídící kloub. Nakonec je zde ještě vytvořen prázdný transformační uzel, ten je vložen mezi nově vytvořený kloub a rodiče. V této



Obrázek 6.1: Ukázka techniky *twist joints*. Rodičem všech kloubů je kloub označený písmenem **P**, řídícím kloubem je kloub **D** *(driver)*, a *twist joints* jsou označené pod písmenem **T**. Na obrázku lze vidět, jak jsou tyto klouby rotované jen s určitým poměrem vůči řídícímu kloubu

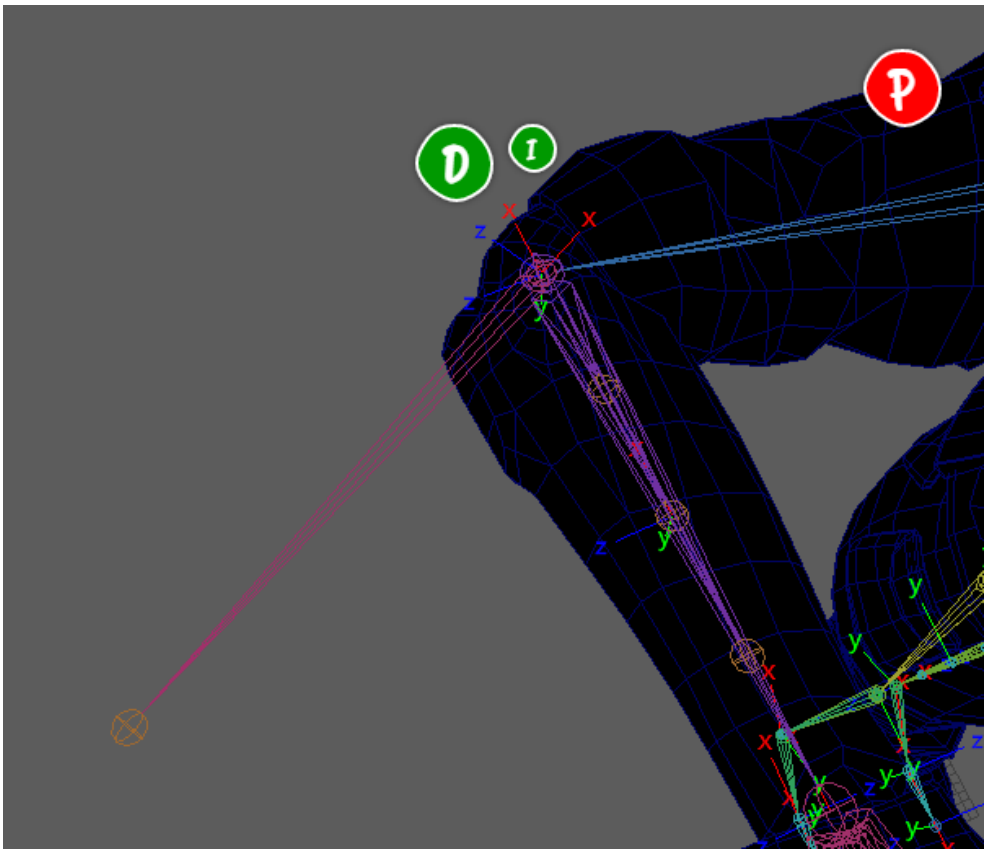
transformaci se bude nacházet informace o posunu *(angl. translate)*. Zde bude také uložen na pozici rotace offset z daného řídícího kloubu, z jeho jednoho řídícího kanálu (a opačná hodnota bude uložena v *joint orient* daného *twist joint*). Díky tomuto triku můžeme tyto hodnoty poté v Unity načíst a vypočítat offset, který zajistí, že iniciální pozice bude stejná, jako v případě tvorby v programu Autodesk Maya. Je to z důvodu, že Unity spojí u kloubu dohromady hodnoty z rotace a z *joint orient* do jedné položky. Druhý důvod je kvůli rotacím v programu Unity, kdy v případě umístění všech tří rotací na jednom objektu docházelo ke vzájemnému ovlivňování, jak bude podrobněji popsáno v kapitole 6.5.2.

- ***Interpolation joints***

  V tomto případě se snažíme kompenzovat úbytek objemu, který by jinak při větších rotacích nastával. Nově vytvořený kloub v tomto místě se proto bude rotovat jen s určitým poměrem do všech ostatních směrů. V případě naskinování této části nebude díky pomalejší rotaci úbytek tak znatelný, protože méně rotovaná část bude přispívat k zachování objemu.

  Při tvorbě, jak již bylo řečeno, vzniká kloub na stejném místě jako původní, jehož chování má upravovat, viz obrázek 6.2. Tento kloub je taktéž připojen vazbou rodič-dítě k předchozímu kloubu v řetězci. Znovu je zde

využito vytvoření pomocného transformačního uzlu, který se v hierarchii nachází mezi nově vytvořeným kloubem a původním rodičem. V tomto případě je tady také uložena hodnota posunu a do atributu rotace jsou uloženy hodnoty z *joint orient* řídícího kloubu. Hodnota rotace a *joint orient* v *interpolation joint* zůstává prázdná.



Obrázek 6.2: Ukázka techniky *interpolation joints*. Kloub, který zprostředkovává tuto techniku, je označen jako **I**. Ten je řízen pomocí kloubu **D**. Tyto klouby jsou pak spojeny vazbou rodič-dítě k předchozímu kloubu v řetězci nacházející se ve směru k písmeni **P**. Lze si povšimnout na koncovém uzlu, že při rotaci řídícího kloubu byl *interpolation joint* rotován jen o určité procento (v tomto případě 30 %)

6.1.1 *Utility nodes*

Pro řízení technik by se dalo použít *animation expression* a při exportu z kloubů získat informace o chování. Sice by byla jistota správného exportování a načtení, ale v Unity bychom museli rozlišovat každý typ kloubu a na něho aplikovat data. Vedoucím práce proto byla doporučena druhá varianta, a to za pomoci *utility nodes* z grafu závislostí. V něm při tvorbě technik vznikají spojení

(angl. connections). Jsou to spojení mezi jednotlivými klouby, které představují vazbu rodič-dítě, ale také zde budou vytvořené speciální uzly zvané jako *utility nodes*. Ty zajišťují matematické operace. Najdeme zde propojení jednotlivých kanálů mezi řídícím kloubem, přes matematický uzel, který bude násobit koeficientem tuto hodnotu až do cílového kanálu ovlivňovaného kloubu. To bylo možné vidět na obrázku věnovaném grafu závislostí 4.2. Implementace propojování kanálů lze najít v metodách `_setup_by_matrix_nodes`. Ty jsou k dohledání v souborech `twist.py` a `interpolation.py`.

Nevýhodou je, že při procházení grafu se exportují všechna spojení mezi klouby, která obsahují uzel pro násobení `multDoubleLinear`. Takže v případě použití tohoto uzlu mimo nějakou z pokročilejších technik můžeme buď způsobit nedefinované chování v Unity, ale pravděpodobněji budeme v Unity upozorněni na chybu. Pro práci je toto chování dostatečné, ale v budoucnosti by se dalo udělat rozšíření, které by procházelo celý graf, podporovalo několik dalších uzlů s možností vícenásobného skládání uzlů. A pak v Unity celý tento graf sestavit. Tím pádem by aplikace dokázala přenášet ještě pokročilejší a složitější chování.

6.2 Maya export

Při exportování má uživatel na výběr. Buď uloží všechny klouby, ze kterých chce získat popis pokročilejších technik do skupiny pojmenované jako `rig_group`. Anebo si vybere rodičovský kloub nebo skupinu ve scéně, od které dál v hierarchii bude program prohledávat graf závislostí. V něm bude procházet všechny vazby mezi klouby a hledat *utility nodes* a uzel `multDoubleLinear`. A pro každé toto spojení si uloží názvy kloubů a kanálů, mezi kterými se spojení nachází, a také násobek, se kterým bude řídící hodnota kanálu násobena pro zisk výsledné hodnoty. Tato data uloží do `.JSON` souboru, jeho ukázka je ve výpisu 6.1.

```
{
        "program": "art_autorig_extra_data",
        "version": "1.0.0",
        "scene_name": "gonzales_model.ma",
        "connections": [
          {
                "fromNode": "LeftHand.rotateX",
                "toShadeN": "multDoubleLinear2.input1",
                "value": 0.5,
                "fromShadeN": "multDoubleLinear2.output",
                "toNode": "LeftForeArmTwistJoint2.rotateX"
          },
          {
                "fromNode": "LeftHand.rotateX",
                "toShadeN": "multDoubleLinear1.input1",
                "value": 0.25,
```

```
17                  "fromShadeN": "multDoubleLinear1.output",
18                  "toNode": "LeftForeArmTwistJoint1.rotateX"
19              },
20              {
21                  "fromNode": "LeftHand.rotateX",
22                  "toShadeN": "multDoubleLinear3.input1",
23                  "value": 0.75,
24                  "fromShadeN": "multDoubleLinear3.output",
25                  "toNode": "LeftForeArmTwistJoint3.rotateX"
26              }
27          ]
28  }
```

Listing 6.1: Ukázka exportovaného souboru

Tyto závislosti poté v Unity budeme načítat. Na základě jména kloubů, názvů kanálů a hodnoty jsou pak v Unity vazby vytvořeny. V současné chvíli je podporován pouze matematický uzel `multDoubleLinear`. Mezi kanály jsou to rotace. V momentální implementaci nejsou klíče `toShadeN` a `fromShadeN` využity.

6.3 Unity Import

Aplikace technik probíhá v editoru po vytvoření speciálního objektu z menu, který se jmenuje `Rig Manager`. Tento objekt má za zásluhu sestavení a aplikaci pokročilejších technik pro daný model, ve kterém se objekt nachází. O celé sestavení se stará jeho komponenta `Builder`.

Nejprve se načte soubor s popisem pokročilejších technik ve formátu `.JSON` a data se uloží do interní struktury. Pro načtení je proto využito knihovny SimpleJSON. Ta je jednoduchá a pro tyto účely postačuje. Dalším krokem je pak procházení této struktury, která se koná v třídě `DependencyGraph`. Pro každé načtené spojení se vytvoří speciální struktura. Ta obsahuje jména kloubů, kanály, které propojuje, a hodnotu. Jednotlivé kanály jsou reprezentovány jako samostatné třídy. Mají společnou abstraktní třídu `ChannelObject`. Tím je zajištěné, že v aplikaci se dále nenachází žádné složitější větvení. Navíc přidání dalšího ovládacího kanálu znamená jen přidat další třídu, která bude dědit z `ChannelObject`. Tato data ukládáme do listu, který se jmenuje `_connectionList`.

Je navíc možné prohazovat jednotlivé propojené kanály. To znamená z řídícího kloubu pro rotaci v ose `x` ovládat kloub v ose `z`. Tato funkcionalita je naimplementována, ale prozatím se nevyužívá. V případě požadavku na ovládání jednoho cílového kanálu z dvou vstupů bude uvažován jen poslední kanál a uživatel bude na tuto skutečnost upozorněn.

Posledním krokem je metoda `ApplyToModel(DependencyGraph)` v třídě `Builder`. Ta se stará o vytvoření komponenty `JointTransformModifier`, která je přidána na kloub zprostředkovávající pokročilejší techniku. Pro tyto

klouby je provedeno základní nastavení, jako je správné propojení s řídícím kloubem a nadřazenou skupinou, nastavení iniciální pozice či uložení všech spojení mezi kanály, kterých se tento kloub účastní. Tyto kanály jsou uložené do proměnné typu `JointSerializedValues`. Ta obsahuje list propojených kanálů.

6.4 Uložení technik

O úspěšném sestavení technik budeme informováni v našem objektu `Rig Manager` a v konzoli. Během toho proběhne poslední krok, a tím je serializace dat[34]. Je nutné uložit právě načtené informace o ovládání těchto kloubů ze dvou důvodů. První z nich je ten, že při zapnutí hry se nad všemi *MonoBehaviour* objekty volají metody *Awake()* a *Start()*. A původně nastavená data, pokud nemají atribut `[Serializable]`, budou smazána. Navíc Unity dokáže serializovat jen některé datové typy. Dle [28] jimi jsou primitivní datové typy jako je `int, float`, struktury s `[Serializable]` atributem, některé základní vestavěné datové Unity typy jako je `Vector3` a některé neabstraktní třídy. Problémem je, že v aplikaci se nachází polymorfní třídy pro odlišení kanálů, které Unity serializovat nedokáže[35]. Druhým důvodem je pak výsledné nasazení hry, kdy je nutné zajistit správně načtení pokročilejších technik na vytvořené hře.

Bylo proto použito řešení serializace za pomocí knihovny JsonDotNet, která dokáže serializovat polymorfní třídy. Pro uložení dat je využita složka *Resources*. Z té se poté dají načíst data při tvorbě hry v editoru, anebo také po přeložení hry a distribuci na koncové zařízení uživatelů. Byla také testována varianta za pomoci *PlayerPref*, kde se povedla serializace i bez nutnosti knihoven třetích stran. Ale fungovala pouze na počítači, na kterém byla vytvořena. Z toho důvodu není tato volba vhodná.

6.5 Ovládání kloubů

Tato podkapitola je rozdělena na dvě části. Na nastavení iniciální rotace kloubu a na sekci pojednávající o výpočtu rotace kloubu za běhu hry.

6.5.1 Nastavení iniciální rotace

Pro nastavení iniciální rotace kloubu zajišťující pokročilejší techniky načteme před prvním výpočtem uložené hodnoty rotace z daného kloubu a jeho nadřazeného objektu (v programu Maya to byl transformační uzel). Z nich bude

[34]Serializací se rozumí převedení dat do takového formátu, který se pak dá uložit např. na disk a v budoucnosti znovu načíst.

[35]Ve verzi Unity 2019.3, která vyšla po naprogramování této části, by mělo být možné serializovat i polymorfní třídy. Tato verze ale nebyla testována.

vypočten offset rotace. Ten zajistí správné přenesení technik, i když v řídících kloubech budeme mít uložené nenulové hodnoty *joint orient*. Schématická ukázka (liší se pro přehlednost kratšími názvy) se nachází ve výpisu 6.2.

```
private void SetupInitialRotation() {

        initialRotation = GetPitchYawRollDegYXZ(extraGroup);
        initialRotJoint = GetPitchYawRollDegYXZ(this);

        scaledInitRot =
            CalculateRotationForJoint(initialRotation);

        initialRotation = initialRotation + initialRotJoint
            - scaledInitRot;
        initialRotationWasSetup = true;
}
```

Listing 6.2: Schématická ukázka iniciálního nastavení rotací

Představme si situaci, že máme jeden *interpolation joint*. V programu Maya má řídící kloub nastavenou hodnotu *joint orient* v ose `x` na hodnotu 30°, hodnota rotace je nastavena iniciálně správně 0°. Tento kloub rotuje s poměrem 50 % rotace řídícího kloubu. Úkolem je přenést správně tuto iniciální rotaci pro tento kloub do Unity.

Uložíme proto do skupiny na atribut rotace hodnotu *joint orient* řídícího kloubu 30° a samotný kloub necháme prázdný. V Unity zjistíme hodnotu rotace této skupiny a uložíme do proměnné `initialRotation`, z ní pak spočítáme přeškálovanou rotaci na základě poměru, který byl definovaný jako 50 %. Do proměnné `scaledInitRot` proto uložíme hodnotu 15°. Po odečtení těchto hodnot vyjde výsledná rotace 15°, která je brána jako offset. Ten bude uložen v proměnné `initialRotation`.

V dalším kroku, při počítání rotací ve hře budeme číst hodnotu řídícího kloubu, který je na začátku 30° (z důvodu přenesení hodnoty *joint orient*). Tuto hodnotu přeškálujeme na základě poměru, s kterým budeme kloub rotovat, takže znova vyjde 15° a tuto hodnotu přičteme k původně vypočtené iniciální rotaci, jak znázorňuje výpis 6.3. Tím pádem vyjde hodnota 30°. Ta bude korespondovat s iniciální rotací v programu Maya.

```
scaledEulers = scaledEulers + initialRotation;
```

Listing 6.3: Přičtení iniciální rotace k výsledné rotaci

Pro *twist joints* je situace obdobná. Jen s tím, že pro tento kloub nepotřebujeme nastavit iniciální rotaci stejnou jako má řídící kloub *joint orient*. Ale pro nastavenou *joint orient* hodnotu řídícího kloubu zajistit, aby *twist joint* v Unity měl rotaci rovnou 0°. Proto je uložena hodnota *joint orient* do nadřazené skupiny a do samotného kloubu je uložena opačná hodnota. Při výpočtu se tyto hodnoty odečtou a vyjde jen opačná hodnota rotace řídícího kloubu, která bude přeškálována na základě poměru zadané rotace. Poté při

výpočtu odečteme tento offset od vypočítané hodnoty rotace a tyto hodnoty se odečtou. Tím zajistíme správnou iniciální rotaci pro *twist joints*.

6.5.2 Unity rotace

Jako problematické se ukázalo ovládání jednotlivých kloubů. Myšlenkou je získat hodnoty rotace řídícího kloubu, tyto hodnoty naškálovat na základě popisu propojených kanálů. Nově vytvořené hodnoty rotace poté uložit do kloubů zajišťujících pokročilejší techniku. Je proto potřeba provést 3 kroky, načtení, škálování a ukládání nových hodnot.

V našem případě, kdy získáváme rotace řídícího kloubu, jsou tyto hodnoty reprezentovány za pomoci quaternionu v proměnné `localRotation`. Tento kloub je ovládaný za pomoci Mecanimu. Můžeme proto pouze číst tyto hodnoty. S načtením quaternionu není problém, ale další krok, škálování, je problematické. Je potřeba vzít jednotlivé kanály, tyto kanály samostatně naškálovat podle požadavků. A uložit je jako nový quaternion.

Bylo testováno několik způsobů, jak tento problém vyřešit, ale žádný nevedl ke správným výsledkům. V situacích, když quaternion obsahoval více než dvě hodnoty, se při rotaci jedné z nich ovlivňovaly i další kanály. V případě, když vezmeme quaternion a použijeme jednu z metod *Lerp*, *Slerp*, *Nlerp* nebo *LerpUnclamped*, tak navíc nedokáže škálovat jednotlivé osy zvlášť. Implementovaná verze za pomoci techniky *swing-twist interpolation (Sterp)* tento problém do určité míry řeší, ale také nevedla na správný výsledek.

Dále byl zkoušen přístup rozdělení jednotlivých složek quaternionu na quaternion představující rotaci jen v ose `x, y` a `z`. Nebo pokusy s využitím převodu quaternionu na matici, operace jak v lokálním, tak v globálním prostoru, analytické řešení za pomoci vektorů anebo využití *axis-angle*. Žádná z metod nevedla na správný výsledek. Při všech pokusech bylo různě experimentováno s *rotate order* a různým pořadí násobení quaternionů. Taktéž v průběhu zkoumání byly prováděny testy, kde se porovnával návrat do iniciální rotace po posloupnosti rotací, kdy výsledek rotací na řídícím a řízeném kloubu musel být stejný.

Na oficiálním fóru Unity existuje diskuze [29], ve které jeden z uživatelů naprogramoval aplikaci. Ta zkouší všechny možné *rotate order* v Unity pro zisk úhlů. Při tvorbě zjistil, že převod z Euler úhlů do quaternionu je v pořadí `yxz`. Při tvorbě rotační matice také záleží na pořadí, kde zjistil, že tvorba matice je potřeba provést v pořadí `zxy`. Při získávání hodnot z matice naopak musí být pořadí `zyx`. Ke konci diskuze je pak poskytnutý 270 řádkový soubor, který sliboval funkční řešení pro tyto převody, resp. zisk správných hodnot z okna *inspector* v Unity. Bohužel toto řešení také nefungovalo.

6.5.3 Implementované řešení

Nakonec bylo ponecháno stávající řešení, které získá z quaternionu hodnotu rotace v Eulerových úhlech za pomoci metody `GetPitchYawRollRadYXZ(Quaternion rotation)`. Získané hodnoty naškáluje na základě požadavků. A hodnoty pro osy `x` a `z` uloží do kloubu zprostředkovávající pokročilejší techniku za pomoci metody *Quaternion.Euler()*. A pro uložení hodnoty osy `y` bude využit nadřazený pomocný objekt.

Toto řešení je funkční, ale nevýhodou je, že tato implementace dokáže v ose `x` správně interpretovat úhly v intervalu $(-90°, 90°)$.

6.6 Validace uživatelských vstupů

V aplikacích je poskytována uživateli zpětná vazba. Ať už se jedná o informační texty, které napomáhají s použitím aplikací, tak se také zde nachází velké množství validačních testů. Ty kontrolují uživatelské kroky. V programu Autodesk Maya se jedná zejména o testy zaměřené na uživatelské vstupy, které kontrolují možnost tvorby pokročilejších technik, jako je například správná selekce kloubů, kontrola existence kloubů (jestli uživatel náhodou mezi nakliknutím kloubu v programu a stisknutím tlačítka na tvorbu nevhodně nepozměnil název). V Unity se naopak zaměřují testy na správné načtení pokročilejších technik na klouby.

V programu Maya je pro tyto účely využito vyskakovacích oken a výpisů do konzole. V Unity je s uživatelem komunikováno pomocí stavového textu v `Rig Manager` a konzole.

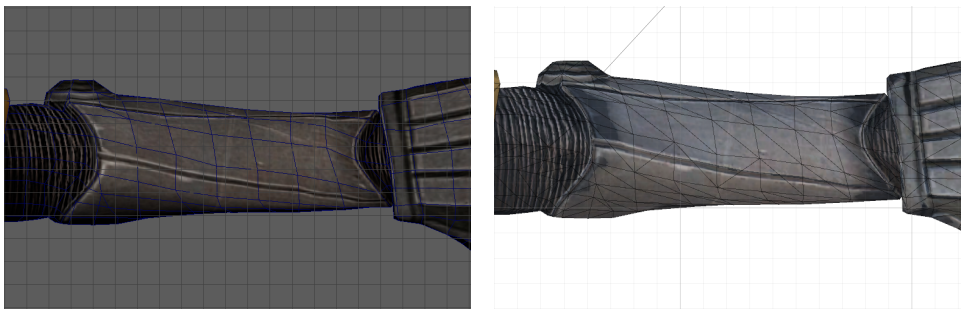
Kapitola 7

Testování

V této kapitole bude popsán způsob testování obou aplikací. Tyto testy mají za cíl odhalit problémy v implementaci, a případně na základě návrhů od samotných testerů zakomponovat jejich poznatky pro budoucí lepší použitelnost.

7.1 Testy zaměřené na správný přenos technik

Prvním typem testu bylo ověření, zda-li dochází ke správnému přenosu technik. Jedná se o porovnání modelů v nějaké póze, kde v obou dvou programech by měl model vypadat identicky. Tímto se také ověří i správný přenos iniciálního nastavení, protože změna rotace je v zásadě jen rozdíl (offset) od iniciální pózy. Na přiloženém obrázku 7.1 se nachází jeden z těchto testů.



(a) Rotace zápěstního kloubu v programu Maya

(b) Rotace zápěstního kloubu v programu Unity

Obrázek 7.1: Ověření přenosu techniky *twist joints*. Rotace zápěstního kloubu je zde 80° v ose `x`

Taktéž byly provedeny numerické testy, při kterých se porovnávaly hodnoty rotace v Unity vůči hodnotám rotace a *joint orient* v programu Maya.

7.2 Testovací scénář

Do testování se zapojili celkem 3 testeři, kteří prošli celý řetězec od začátku až do tvorby výsledné hry. Dále bylo využito dalších 8 testerů. Ti zkoušeli jen určité kroky (otestování aplikace na MacOS, funkčnost Unity balíčku na předem předpřipravenou postavičku s pokročilými technikami atp.). Vzhledem k náročnější časové zátěži pro nastavování kloubů a skinování testeři tento krok neprováděli pro všechny klouby. Kvůli úspoře času byla testerům s mou asistencí provedena instalace aplikací do Unity a Autodesk Maya. Testování probíhalo na počítačích uživatelů za kontroly pomocí sdílené obrazovky přes programy Skype[36] a Discord[37]. Tito uživatelé měli nainstalované aplikace ve verzích:

- **Autodesk Maya** : 2017 (update 5), 2018, 2020,
- **Unity** : 2018.4.13f1, 2019.1.1f1, 2019.2.10.f1, 2019.2.17f1.

Testeři byli znalí programů Autodesk Maya a herního engine Unity. Všichni měli základní přehled o rigování, ale nebyli to žádní pokročilí rigeři. Bylo proto uživatelům napovídáno při hledání některých vestavěných nástrojů, jako je například *Orient joint*.

7.3 Struktura testu

Testovací scénář byl sestaven tak, aby pokryl všechny funkční požadavky. A také otestoval, jestli uživatelé dokážou při prvním použití programu provést úspěšný export technik.

Protože testeři nejsou zběhlí rigeři, byly na začátku představeny některé pojmy ze světa rigování. A také popsán cyklus tvorby rigu krok po kroku, aby testeři byli v obraze a věděli, které kroky jsou potřeba, kde případně zařadit jednotlivé techniky.

Testování začínalo po prvotním nastavení programů. Skládalo se z 12 úkolů.

1. Načtení modelu a kostry do programu Autodesk Maya.
2. Naškálování kostry a napozicování hlavních kloubů do modelu (stačilo jen přibližně a od zápěstí ke konečkům prstů nebylo třeba řešit).
3. Správné nastavení lokálních os rotace pro klouby řízené pomocí pokročilejších technik (zejména pro ten, které si tester vybere k následnému skinování).

[36] `https://www.skype.com/cs/`
[37] `https://discordapp.com/`

4. Přidání pokročilejších technik.
    - *Twist joints* od ramena k zápěstí (na jedné z ruk) s 2ma klouby. Se správně nastavenými lokálními osami rotace.
    - *Interpolation joints* na loktu druhé ruky.
    - Přidání *interpolation joints* na dalších dvou místech, které uživatel uzná za vhodné.
    - Přidání *twist joints* se správně nastavenou primární osou, na dalších dvou místech, které uživatel uzná za vhodné.
5. Dalším krokem je skinování. Zde bylo testerům dáno za úkol použít základní skinning, následovat uživatelskou příručku, naskinovat jednu sadu *twist joints* a jedenkrát *interpolation joints* dle výběru.
6. Po naskinování výsledek exportovat.
7. V Unity načíst exportovaný model.
8. Nastavit typ rigu na `humanoid` a zkontrolovat kostru.
9. Přidat `Rig Manager`.
10. Načíst správně klouby.
11. A nakonec výsledek spustit a zkontrolovat chování *twist joints* a *interpolation joints* v Unity.
12. Za mojí asistence bylo posledním krokem vytvoření a spuštění výsledné hry, kde se testovalo, zda-li došlo ke správnému načtení technik i ve vytvořené hře.

7.4 Výsledky testování

Největší potíže uživatelům dělal krok číslo tři, nastavování lokálních os rotace, a to zejména v místě zápěstí. Byl proto v uživatelské příručce tento krok podrobněji rozebrán, aby budoucím uživatelům nečinil již takové potíže. Všichni správně poznamenali, že aplikace se používá odshora dolů. Taktéž po chvilce všichni dokázali správně vybrat jednotlivé klouby při tvorbě pokročilejších technik. Potíže se však znovu dostavily v případě skinování. Testerům nedělalo obtíže správně pochopit a zkusit replikovat návod v uživatelské příručce, ale činilo jim potíže použití nástroje na kreslení vah. Věděli, čeho chtěli dosáhnout, ale nedařilo se jim nástroj použít. Export z programu Maya a následný import technik v Unity proběhl naprosto bez problémů.

U některých testerů byly objeveny problémy v programu Autodesk Maya, kde Python nebyl schopný importovat některé knihovny ve složce **utils** a soubor `errors.py`. Ukázalo se, že dochází k jejich zastínění interními knihovnami

v programu Autodesk Maya. Byly proto upraveny názvy souborů, a problém se vyřešil. V programu Autodesk Maya ve verzi 2020 bylo možné spustit skripty jen přes *shelf*, protože tlačítko **Execute** nefungovalo. Navíc v testované verzi se nepodařilo zprovoznit *outliner*.

7.5 Kroky učiněné na základě výsledků testování

Na základě testování by se dalo konstatovat, že aplikace jako taková (posloupnost jednotlivých kroků, výběr kloubů, export a import) nedělala testerům obtíže. Problémové kroky se týkaly nastavení lokálních os rotace a skinování. Toto jsou zrovna kroky, které by měl riger ovládat. Bylo ale nakonec investováno více času do zlepšení těchto kroků. Ty pak byly více popsány v uživatelské příručce, aby méně zkušenějším rigerům již nečinily takové obtíže.

Kapitola 8

Naměřené výsledky

V této sekci budou rozebrány výsledky. A to jak z vizuálního hlediska, tak z výpočetní náročnosti.

8.1 Porovnání výpočetní náročnosti

Pro porovnání výpočetní náročnosti bylo implementováno několik dalších skriptů. Byl vytvořen build[38], který obsahuje testovací scény se schopností ukládání rychlosti běhu aplikace v čase. Výsledek se pak vždy nachází ve složce **FpsTest** ve formátu `.csv`.

8.1.1 Metodika testování

Testování probíhalo ve vytvořené hře na přenosném počítači s parametry:

- CPU: i7-6700HQ 2,6 GHz (TB 3,5 GHz),
- GPU: NVIDIA GeForce GTX 960M (4 GB GDDR5, 1096 MHz),
- RAM: 24 GB (DDR3 - 1600 MHz),
- SSD: Samsung 860 EVO M.2 500 GB,
- monitor: Full HD (1920x1080),
- Windows 10 Pro (64 bit).

Data byla měřena na 6 scénách. Všechny scény mají stejné nastavení pozice kamery, stejné rozestavění postav, takže se liší jen použitým typem modelu.

[38] Vytvořenou hru se dá i ovládat. Pohyb je možný pomocí kláves `W,A,S,D`, pravého tlačítka na myši a kolečka. Klávesa `Z` zapíná a vypíná wireframe model, `P` pro pauzu nebo zpomalení hry a klávesy `N,M` pro přechod mezi scénami.

Nachází se v nich 96 modelů postavy Eve od grafika J. Gonzales[39] s využitím animace běhu z balíčku Everyday Motion Pack Free[40], který se dá pohodlně stáhnout do projektu z *Unity Asset Store*. Jednotlivě tyto scény obsahují modely, kterými jsou:

- základní manuálně naskinovaný,
- základní s výchozím Maya skinováním (maximálně 4 klouby na jeden vrchol),
- s pokročilými technikami (15 řídících kanálů na 11 kloubech), který je manuálně naskinovaný,
- s pokročilými technikami (15 řídících kanálů na 11 kloubech) s výchozím Maya skinováním (maximálně 4 klouby na jeden vrchol),
- s pokročilými technikami (30 řídících kanálů na 26 kloubech) s manuálním skinovaním, kde následně byly zjemněny přechody,
- s pokročilými technikami (30 řídících kanálů na 26 kloubech) s manuálním skinovaním, bez zjemněných přechodů.

První, třetí a pátá scéna porovnává situace mezi základním modelem a modelem s pokročilejšími rigovacími technikami po manuálním skinování, což je přesně případ, který uživatelé budou používat. Do porovnávání byly zakomponovány ještě scény s výchozím skinováním v programu Autodek Maya. Pro porovnání, jestli nemá na výsledek vliv manuální skinování. Z důvodu, že v některých místech jsou na polygonální síti ostřejší přechody při skinování, kde jsou jednotlivé vrcholy ovlivněné jen jedním kloubem. A na některých zase zjemněné přes více kloubů.

8.1.2 Výsledky testování

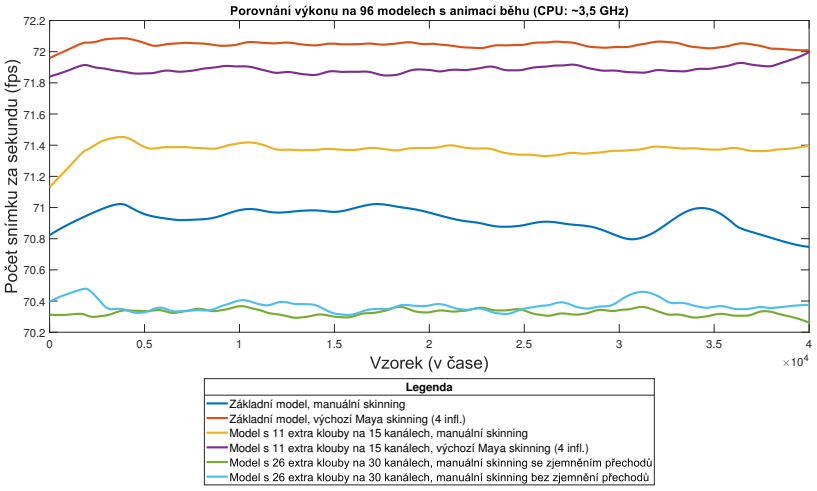
Bylo provedeno několik testů. Všechny grafy byly generovány za použití programu Matlab[41]. Tato data poté byla upravena do konečné podoby za pomocí algoritmu *Locally Weighted Scatterplot Smoothing (LOWESS)*, protože původní naměřená data vykazovala pro každý vzorek odchylku v rámci několika jednotek *fps* a výsledek byl nečitelný. V prvním testu byla pro každou scénu sbírána data po dobu 10 minut. Graf se nachází na obrázku 8.1.

Z naměřených výsledků lze vypozorovat, že modely s pokročilejšími rigovacími technikami jsou mírně výkonově náročnější. Z obrázku lze také vypozorovat, že rozdílné rozdělení vah při skinování má na rychlost také vliv. V obrázku

[39] Postavička Eve se dá stáhnout ze serveru `mixamo.com`. Má 14241 vrcholů a 27207 trojúhelníků.

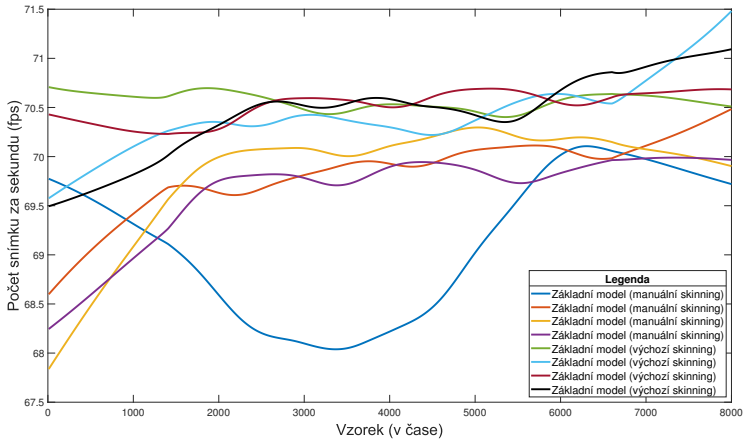
[40] `https://assetstore.unity.com/packages/3d/animations/everyday-motion-pack-free-115067`

[41] `https://www.mathworks.com/products/matlab.html`



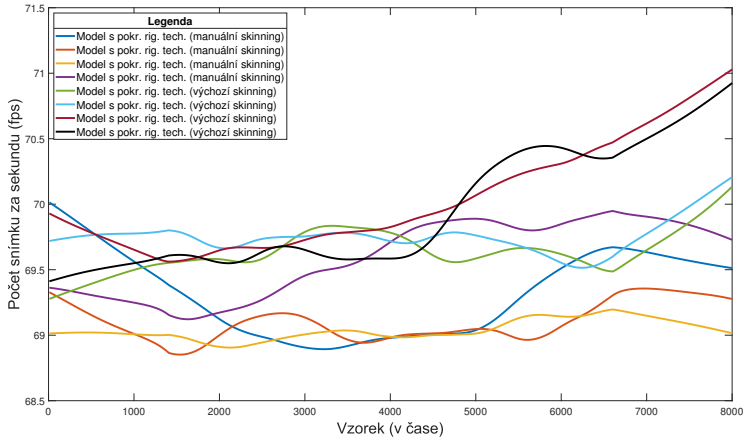
Obrázek 8.1: Výkonnostní porovnání technik při taktu procesoru přibližně 3,5 GHz. Bylo použito techniky *LOWESS* s velikostí rozsahu 10 %

vznikají rozdíly mezi modely s ručním a s výchozím skinováním. Vznikly proto ještě další testy, které porovnávají modely s manuálním skinováním a výchozím skinováním z programu Autodesk Maya (s nastavením maximálního počtu ovlivňovaných kloubů pro jeden vrchol na hodnotě 4). Tyto testy probíhaly opakovaně po dobu 2 minut. Výsledky lze vidět na grafech číslo 8.2 a 8.3.



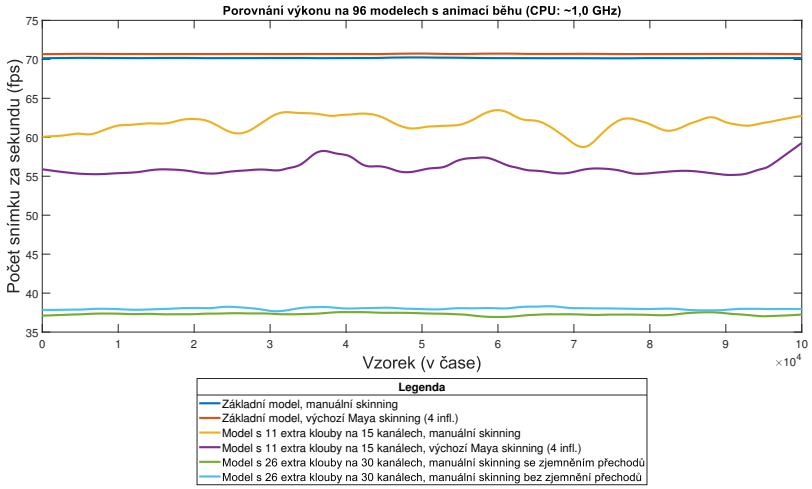
Obrázek 8.2: Výkonnostní porovnání skinování pro základní modely. Bylo použito techniky *LOWESS* s velikostí rozsahu 35 %

Z grafů je patrné, že všechny modely s výchozím skinováním jsou o přibližně 0,5 až 1 snímek za sekundu rychlejší.



Obrázek 8.3: Výkonnostní porovnání skinování pro modely s pokročilejšími technikami. Bylo použito techniky *LOWESS* s velikostí rozsahu 35 %

Nakonec byl proveden ještě jeden test, kde se porovnával výkon na podtaktovaném procesoru (CPU). Ten probíhal po dobu jedné hodiny pro každou scénu a takt procesoru se držel přibližně kolem hodnoty 1,0 GHz. Výsledky znázorňuje graf 8.4.



Obrázek 8.4: Výkonnostní porovnání technik při taktu procesoru 1,0 GHz. Bylo použito techniky *LOWESS* s velikostí rozsahu 10 %

Z tohoto grafu je patrné, že již dochází k úbytku výkonu. V Unity profileru při spuštěné hře v editoru bylo viditelné, že pro ovládání scény s 96 základními modely potřebuje Unity smyčka pro výpočet 5-6 *ms*. Zbytek je čekání na

vykreslení od grafické karty (GPU). GPU ve všech případech vytváří nový snímek přibližně každých 14 *ms*. Byl proto vytvořen build, kde probíhalo měření a kontrola dat z profileru. Bohužel Unity nedisponuje jednoduchou možností zisku všech dat z kódu a nad ní možnost provést analýzu. Bylo proto nutné tato data procházet manuálně v reálném čase.

1,0 GHz	**Základní model**	**Model s 11 extra klouby**	**Model s 26 extra klouby**
Naměřená hodnota *fps*	70	61	38
Spočtená průměrná délka rámce v *ms*	14,2	16,39	26,31
Průměrná délka herní smyčky v *ms*	11-16	14-25	20-40
Čas pro ovládání extra kloubů v *ms*	0	7-13	10-30

Tabulka 8.1: Tabulka získaných dat z profileru při běhu aplikace s taktem procesoru na přibližných 1,0 GHz

3,5 GHz	**Základní model**	**Model s 11 extra klouby**	**Model s 26 extra klouby**
Naměřená hodnota *fps*	70	71	70
Spočtená průměrná délka rámce v *ms*	14,20	14,01	14,20
Průměrná délka herní smyčky v *ms*	11-16	11-16	11-16
Čas pro ovládání extra kloubů v *ms*	0	2-3	5

Tabulka 8.2: Tabulka získaných dat z profileru při běhu aplikace s taktem procesoru na 3,5 GHz

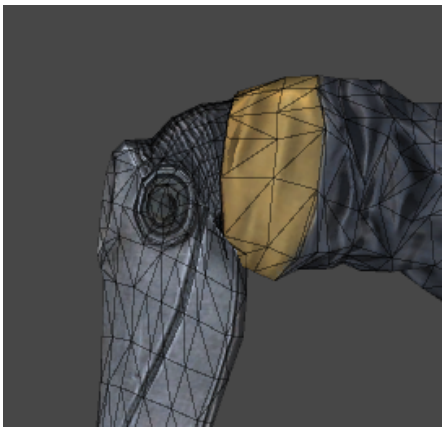
Tabulky 8.1 a 8.2 obsahují přibližné informace o běhu hry s taktem procesoru na 1,0 GHz a 3,5 GHz. V druhé tabulce je viditelné, že počet průměrných snímků za sekundu se neměnil. V této fázi CPU stíhá počítat všechny informace a čeká se na vykreslení ze strany grafické karty. Je pravděpodobné, že výkon by se měnil až v případě, když by čas výpočtu pozic nových kloubů zabral v součtu s naměřenou hodnotou hry bez prokročilejších kloubů (která byla 5-6 *ms*) větších hodnot, než je průměrná spočtená délka rámce, která je 14 *ms*. V první tabulce je viditelný úpadek výkonu, ten je způsoben výpočtem pozic nových kloubů.

8.2 Porovnání vizuálních výsledků

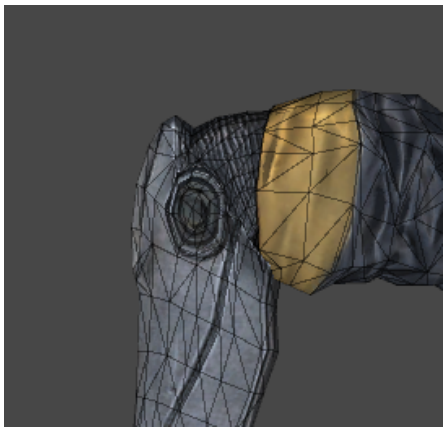
Druhá část porovnání se bude zabývat vizuálním výsledkem. V této kapitole tedy najdeme odpověď na otázku, zda-li má smysl tyto techniky převádět do Unity.

Porovnání bude probíhat na dvou postavičkách. Jednou z nich je Eve, pro kterou byly vytvářeny výkonostní testy. Má nižší počet trojúhelníku, 27207. A druhá postavička, Max[42], která má 220672 trojúhelníků.

Porovnání bude provedeno zejména v místech, kde jsou tyto techniky zpravidla nejčastěji používané. Na levé straně stránky se bude nacházet model bez pokročilejších technik a na straně pravé model s pokročilejšími technikami.



(a) Model bez *interpolation joint*, kde dochází k úbytku objemu na vnější straně lokte



(b) Model s *interpolation joint*, který rotuje s poměrem 30 % rotace loketního kloubu, tím zajistí menší úbytek objemu

Obrázek 8.5: Porovnání deformace v oblasti lokte

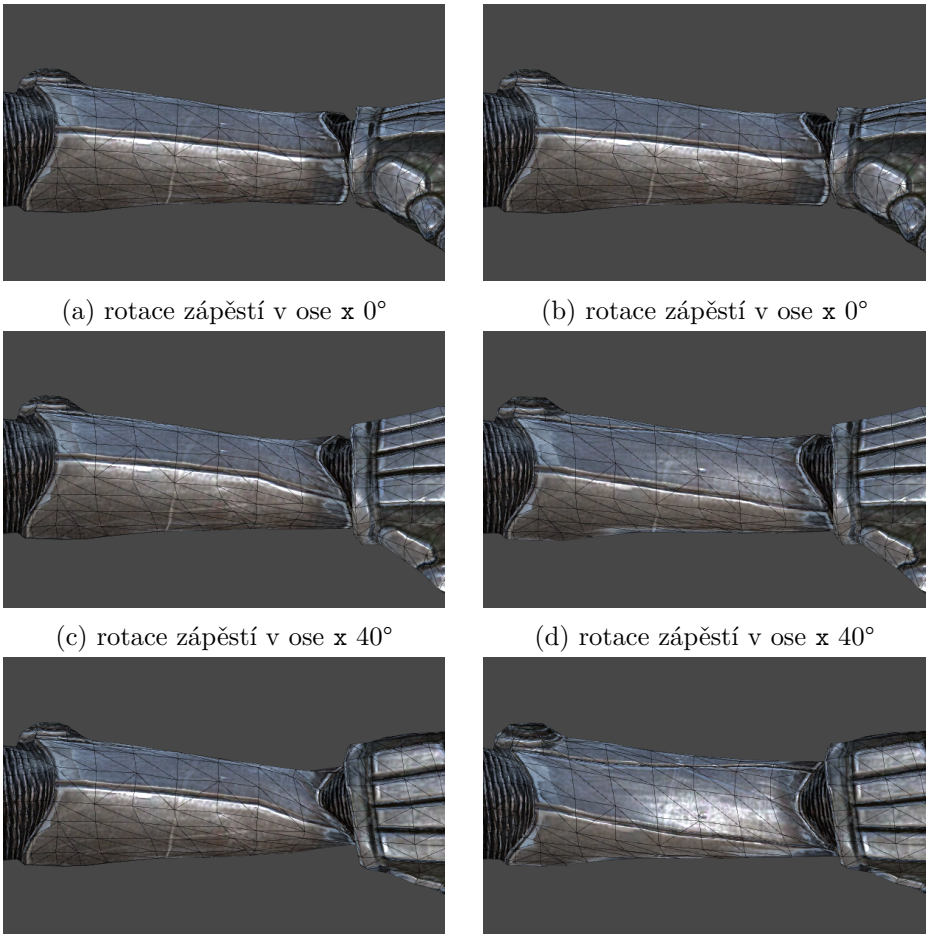
8.3 Unity Animation Rigging

Zde mělo být ještě srovnání s Unity Animation Rigging. Což, jak již bylo avizováno, je nový balíček od Unity. Ten je ale v předběžné verzi a neexistují k němu téměř žádné návody, mimo základních ukázek od Unity.

První problém je se samotným typem rigu. Existuje testovací scéna, která byla představena na konferenci SIGGRAPH s modelem Ninja. Jak si někteří uživatelé všimli [30][43], tak tento model využívá rig *Generic*. S rozběháním humanoid rigem měli problémy. V mých testech se nepovedlo dostat do fáze,

[42] Max byl stažen ze stránky: `https://www.turbosquid.com/FullPreview/Index.cfm/ID/530373`, je zde pod *Royalty Free licencí*, která dovoluje použití i pro školní účely.

[43] Jedná se o odkaz na celou sekci, která se zabývá tímto balíčkem. Odkaz, kde se přímo o tomto problému mluví je např. `https://forum.unity.com/threads/my-feedback-on-new-animation-rigging-package.728597/`.



(a) rotace zápěstí v ose `x` 0°

(b) rotace zápěstí v ose `x` 0°

(c) rotace zápěstí v ose `x` 40°

(d) rotace zápěstí v ose `x` 40°

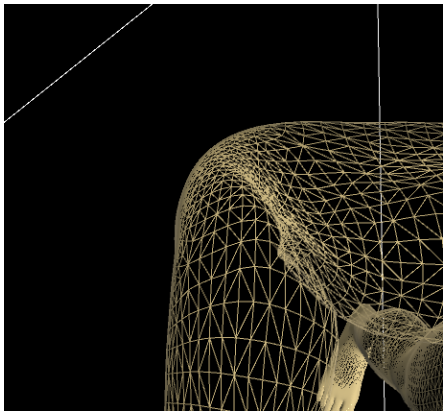
(e) rotace zápěstí v ose `x` 80°

(f) rotace zápěstí v ose `x` 80°

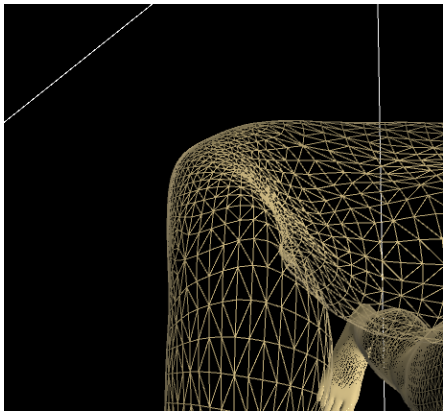
Obrázek 8.6: Porovnání deformace v oblasti zápěstí. Na levé straně model bez pokročilejších technik, na straně pravé model obsahující 3 *twist joints*, kde je znatelně viditelná propagace deformace vzniklá rotací zápěstí. Na straně levé, v případě základního modelu žádná taková propagace deformace neprobíhá, navíc je viditelná nepřirozená deformace v oblasti zápěstí

abych byl schopný ve hře při rotaci ruky rozhýbat *twist joints*. Ty fungovaly jen v případě předaného objektu *Animator* s animací vytvořené přímo pro tento model.

Zkusil jsem proto na základě nastavení z oficiální dokumentace, vzorového příkladu a na základě nastavení Ninja postavičky z přednášky nastavit ovládání *twist joints* pro můj model. Po nastavení přesně jako v předchozích 3 případech se povedlo nejdále dostat do fáze, kdy zápěstí ovládá 2 ze 3 *twist joints*, z toho jeden rotuje podivným způsobem na opačnou stranu. Vlastní příklad s rotací kostiček na základě dokumentace ale fungoval bez problémů.



(a) Model bez *interpolation joint*, kde dochází k úbytku objemu na vnější straně lokte



(b) Model s *interpolation joint*, který rotuje s poměrem 30 % rotace loketního kloubu, tím zajistí menší úbytek objemu

Obrázek 8.7: Porovnání deformace v oblasti lokte na modelu Max

8.4 Zhodnocení výsledků

Z přiložených obrázků je na první pohled patrné zlepšení výsledných deformací. Ať už se jedná o *interpolation joints*, které zajišťují kompenzaci úbytku objemu anebo hlavně *twist joints*, které dokážou rovnoměrně propagovat deformaci způsobenou rotací řídícího kloubu.

Co se týče výpočetní složitosti, tak zde dochází k úbytku výkonu v případě, když je zatížené CPU. V případě 96 postav, na kterých je 26 speciálních kloubů, dochází k 2496 voláním výpočtu pozice nového kloubu. Což v případě procesoru, na kterém bylo testováno, vychází na průměrných 500 vypočtených pozic nových kloubů za 1 *ms*. Při modelové situaci, kdy hra běží na 100 *fps* a procesor běží na plný výkon. Tak pro pokles z 100 *fps* na 90 *fps* by bylo potřeba 500 kloubů, které počítají novou pozici v jedné chvíli ve scéně. V případě, že hra bude čekat na grafiku, tak zde k úbytku výkonu téměř nedochází. Navíc v jedné chvíli nebude na scéně tolik postav najednou, takže reálně pro koncového uživatele nebude rozdíl tak znatelný. A minimálně např. pro hlavní postavu, nebo postavy, které budou zaměřené ve filmových scénách se vyplatí tyto techniky použít.

Je jen škoda, že se nepovedl rozběhat balíček Unity Animation Rigging a porovnat výsledky (zejména výkonnostní). Do budoucna má tento balíček na základě toho, co slibuje, jistě velký potenciál, který bohužel teď není příliš uživatelsky přívětivý.

Závěr

Cílem této diplomové práce bylo vytvoření nástrojů, které umožní exportovat dvě pokročilejší rigovací techniky z programu Autodesk Maya do Unity. Vybrány byly techniky *twist joints* a *interpolation joints*. Pro splnění tohoto cíle bylo nutné se nejprve seznámit s problematikou kolem rigování, následně prozkoumat možnosti využití skriptování pro automatizaci jednotlivých technik. A nakonec vymyslet, jak tyto techniky přenést a následně je v Unity ovládat. Na základě analýzy bylo navrhnuto řešení, které všechny tyto požadavky splňuje.

Pro tyto techniky byla v programu Autodesk Maya vytvořena knihovna. Ta pomůže uživateli tyto techniky vytvořit a poté exportovat. V herním engine Unity jsou načteny a aplikovány na model, který lze potom libovolně používat. Bylo dbáno na co největší zjednodušení v Unity, aby bylo zajištěno snadné použití. Díky této aplikaci je možné v Unity mít stejné deformace jako v programu Autodesk Maya. S použitím typu rigu humanoid je poté možné využít širokou škálu animací za pomoci techniky *animation retargeting*.

Jedinou nevýhodou práce je omezená rotace kloubu v ose `x`, která zvládá rotace v intervalu $(-90°, 90°)$.

Budoucí práce

I když je umožněno uživatelům přenos pokročilejších technik, tak stále je možnost aplikaci vylepšit, resp. rozšířit.

- Prvním krokem může být upravení počítání rotací, aby bylo možné získávat rotace v ose `x` i ve větším intervalu než $(-90°, 90°)$.
- Další možností je rozšíření sběru a reprezentace dat z grafu závislostí. Tím pádem by se dalo přenášet mnohem více prvků z programu Autodesk Maya do Unity.

- Přidáním podpory i pro další pokročilejší rigovací techniky, jako např. *volume joints* nebo *stretchy joints*.

Literatura

[1] UV mapping. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2019-12-08]. Dostupné z: https://en.wikipedia.org/w/index.php?title=UV_mapping&oldid=871387795

[2] GEND, Peter. Maya Rigging Fundamental. Pluralsight [online]. Pluralsight, 2017 [cit. 2019-12-08]. Dostupné z: https://www.pluralsight.com/courses/maya-rigging-fundamentals

[3] CABRERA, Cheryl. An Essential Introduction to Maya Character Rigging. Focal Press, 2008. ISBN 978-0240520827.

[4] RITCHIE, Kiaran, Jake CALLERY a Karim BIRI. The Art of Rigging (A Definitive Guide to Character Technical Direction with Alias Maya, Volume 1). CG Toolkit, 2005. ISBN 0976800306.

[5] ASSAF, Eyal. Rigging for games: a primer for technical artists using Maya and Python. Burlington, MA: Focal Press, 2016. ISBN 978-0415743051.

[6] DERAKHSHANI, Dariush. Introducing Maya 7: 3D for beginners. Indianapolis, IN: Wiley Pub., 2006. ISBN 0782144349.

[7] Why movies like The Hobbit are moving from 24 to 48 fps. In: ExtremeTech [online]. 2013 [cit. 2020-01-06]. Dostupné z: https://www.extremetech.com/extreme/128113-why-movies-are-moving-from-24-to-48-fps

[8] SKOVBO JOHANSEN, Rune. Automatic Setup of a Humanoid. In: Unity Blog [online]. 2013 [cit. 2019-11-24]. Dostupné z: https://blogs.unity3d.com/2013/02/07/automatic-setup-of-a-humanoid/

[9] 3D production pipeline. In: VFX movies upcoming [online]. 2014 [cit. 2019-11-27]. Dostupné z: http://www.upcomingvfxmovies.com/2014/03/3d-production-pipeline-pixar-vs-dreamworks/

[10] Judd Simantov on character rigging and modeling in Naughty Dog's The Last of Us. In: Youtube.com [online]. 2013 [cit. 2019-12-01]. Dostupné z: https://youtu.be/myZcUvU8YWc

[11] Unity 2019.3. In: Unity3D [online]. 2019 [cit. 2019-12-02]. Dostupné z: https://unity3d.com/beta/2019.3

[12] Unity: Auto Rigger - Runtime Joints. In: Vimeo [online]. 2018 [cit. 2019-12-02]. Dostupné z: https://vimeo.com/301451175

[13] IK solvers. Autodesk [online]. [cit. 2019-12-08]. Dostupné z: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/Maya/files/GUID-952FC4B3-19A6-4055-B034-3A7D15EC66D6-htm.html

[14] Everything you thought you knew about Maya Joint Orient is wrong! Rigging Dojo [online]. 2014 [cit. 2019-12-08]. Dostupné z: https://www.riggingdojo.com/2014/10/03/everything-thought-knew-maya-joint-orient-wrong/

[15] Baking simulations. Autodesk Maya [online]. 2018 [cit. 2019-12-08]. Dostupné z: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/Maya-Animation/files/GUID-56C1AA69-D0DD-42F6-BC85-931AFBDE70D4-htm.html

[16] Dependency graph. Autodesk Maya [online]. 2019 [cit. 2019-12-08]. Dostupné z: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/Maya-Basics/files/GUID-51096BC4-32B7-4391-BE39-21641B374745-htm.html

[17] MECHTLEY, Adam a Ryan TROWBRIDGE. Maya Python for games and film: a complete reference for the Maya Python and the Maya Python API. Waltham, MA: Morgan Kaufmann, c2012. ISBN 978-0123785787.

[18] Maya API Reference. Autodesk Maya [online]. 2017 [cit. 2019-12-09]. Dostupné z: https://help.autodesk.com/view/MAYAUL/2017/ENU/?guid=__cpp_ref_index_html

[19] Animation expressions. Autodesk Maya [online]. 2018 [cit. 2019-12-09]. Dostupné z: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/Maya-Scripting/files/GUID-5F75516C-403A-4CC3-A118-AEDEFDE970B4-htm.html

[20] Importing Objects From Maya. Unity [online]. 2017 [cit. 2019-12-09]. Dostupné z: https://docs.unity3d.com/560/Documentation/Manual/HOWTO-ImportObjectMaya.html

[21] Twist Corrective. Unreal Engine [online]. [cit. 2019-12-09]. Dostupné z: https://docs.unrealengine.com/en-US/Engine/Animation/NodeReference/SkeletalControls/TwistCorrective/index.html

[22] Animation System Overview. Unity Documentation [online]. 2019 [cit. 2019-12-09]. Dostupné z: https://docs.unity3d.com/Manual/AnimationOverview.html

[23] System Requirements. Unity [online]. 2019 [cit. 2019-12-11]. Dostupné z: https://unity3d.com/unity/system-requirements

[24] O'HAILEY, Tina. Rig it right!: Maya animation rigging concepts. New York: Focal Press, Taylor & Francis Group, 2013. ISBN 9780240820910.

[25] KAVAN, Ladislav, Steven COLLINS, Jiří ŽÁRA a Carol O'SULLIVAN. Geometric skinning with approximate dual quaternion blending. ACM Transactions on Graphics [online]. 2008, 27(4), 1-23 [cit. 2019-12-12]. DOI: 10.1145/1409625.1409627. ISSN 07300301. Dostupné z: http://portal.acm.org/citation.cfm?doid=1409625.1409627

[26] How It Works | Linear Blend Skinning Part 1. In: Youtube [online]. Pixel Fondue, 2017 [cit. 2019-12-13]. Dostupné z: https://www.youtube.com/watch?v=QDXG4wNzkOE

[27] Importing skinned Meshes. In: Unity Documentation [online]. 2019 [cit. 2019-12-13]. Dostupné z: https://docs.unity3d.com/es/2019.2/Manual/ImportingSkinnedMeshes.html

[28] Script Serialization. In: Unity Manual [online]. Unity Technologies, 2019 [cit. 2019-12-23]. Dostupné z: https://docs.unity3d.com/Manual/script-Serialization.html#FieldSerliaized2

[29] [SOLVED] How to get "Rotation"value that is in the inspector? In: Unity Forum [online]. 2018 [cit. 2019-12-30]. Dostupné z: https://forum.unity.com/threads/solved-how-to-get-rotation-value-that-is-in-the-inspector.460310/#post-3956854

[30] Animation Previews. Unity Forum [online]. [cit. 2020-01-02]. Dostupné z: https://forum.unity.com/forums/animation-previews.141/

Příloha **A**

Seznam použitých zkratek

2D Dvoudimenzionální prostor

3D Trojdimenzionální prostor

CPU Procesor

DA *Dependency graph* - graf závislostí

DAG *Directed acyclic graph* - orientovaný acyklický graf

FK *Forward Kinematics* - dopředná kinematika

fps *Frames per second* - udává rychlost změny snímku za jednu sekundu

GPU Grafická karta

IK *Inverse Kinematics* - inverzní kinematika

LRA *Local rotation axis* - lokální osa rotace

MEL *Maya Embedded Language* - skriptovací jazyk v Autodesk Maya

Příloha B

Instalační příručka

V této příručce bude popsáno, jaké kroky jsou potřeba pro spuštění a použití vytvořených knihoven v programech Autodesk Maya a Unity.

B.1 Nastavení programů

- **Autodesk Maya**

  Pro možnost exportu formátu `.fbx` je nutné v nastavení povolit načtení zásuvného modulu do programu Maya. V menu liště proto následujeme: *Windows -> Settings/Preferences -> Plug-in Manager*. Kde vyhledáme a povolíme načítání modulu `fbxmaya.*`. Dále je vhodné pro správné přenesení modelu nastavit správné jednotky[44] *Windows -> Settings / Preferences -> Preferences*, v levé nabídce vybereme *Settings* a pokračujeme *Working Units -> Linear*, kde vybereme jednotky v metrech `(meter)`. Pro testování aplikace není nutné tento předchozí krok dělat. Zato ale musíme nastavit *Up axis* na osu `y`.

- **Unity**

  V práci je použita externí knihovna pro manipulaci s JSON soubory, která vyžaduje framework .NET ve verzi `4.4`. Pro nastavení následujeme: *File -> Build Settings -> Player Settings*. Zde v nabídce jdeme do *Other settings -> Configuration*, kde nastavíme *Scripting Runtime Version:* `.NET 4.x Equivalent` a *Api Compatibility Level:* `.NET 4.x`[45].

[44] Tyto jednotky se dají nastavit i v Unity, ale je lepší je nastavit v programu Maya, kde máme jen model, než v Unity rozpracované hře, kde to ovlivní i ostatní assety.

[45] V některých verzích Unity se položka *Api Compatibility Level* nenachází. Mělo by proto stačit nastavit jen první položku (testováno třemi uživateli).

B.2 Spuštění aplikace v programech

- **Autodesk Maya**

  V programu Maya je zapotřebí spustit napsaný kód v jazyku Python. Otevřeme proto *Script Editor*, který můžeme najít v menu *Windows -> General Editors -> Script Editor* anebo v pravém dolní rohu pod ikonkou `{;}`. Na výběr máme prostředí pro MEL nebo Python. Vybereme záložku Python a do okna zkopírujeme kód obsažený na výpise B.1, kde je navíc nutné zaměřit správnou cestu ke složce obsahující skripty pro program Maya.

```
import sys

# You need to change this path
rig_dir = 'D:/Maya/code'

# add it into path
if not rig_dir in sys.path:
   sys.path.append(rig_dir)

rt import_all
reload(import_all)

import_all.run(rig_dir)
```

Listing B.1: Popis spuštění skriptu v prostředí Autodesk Maya

  Pak už jen spustíme tento kód a mělo by se nám objevit hlavní menu aplikace.

  V případě opakovaného spouštění je pro ulehčení práce možnost přidat kód do lišty `(shelf)` *File -> Save script to shelf...*. Ten pak spouštět přes jedno kliknutí na myši.

- **Unity**

  V programu Unity je situace o mnoho jednodušší. Stačí pouze přidat balíček `advanced_rig.unitypackage` (implementovaná knihovna v rámci této práce) do projektu. Uživatel tento balíček importuje do projektu pomocí *Assets -> Import Package -> Custom Package* a vybere na disku balíček `advanced_rig.unitypackage`. Po odsouhlasení importu pak uživatel může spustit aplikaci z lišty v menu, kde se objeví položka *Advanced Rig -> Rig Tools*.

Příloha C

Uživatelská příručka

V této příručce bude popsáno, jak z uživatelského hlediska použít tyto knihovny pro přenesení speciálních technik z programu Autodesk Maya do herního engine Unity.

Bohužel není v rozsahu této příručky popsat dopodrobna, jak správně rigovat jednotlivé části těla krok za krokem. Kdyby byl v této práci uveden podrobný postup, obsahovala by tato část mnohem více textu a obrázků než je celá diplomová práce. Proto se zde zaměříme jen na nejnutnější kroky.

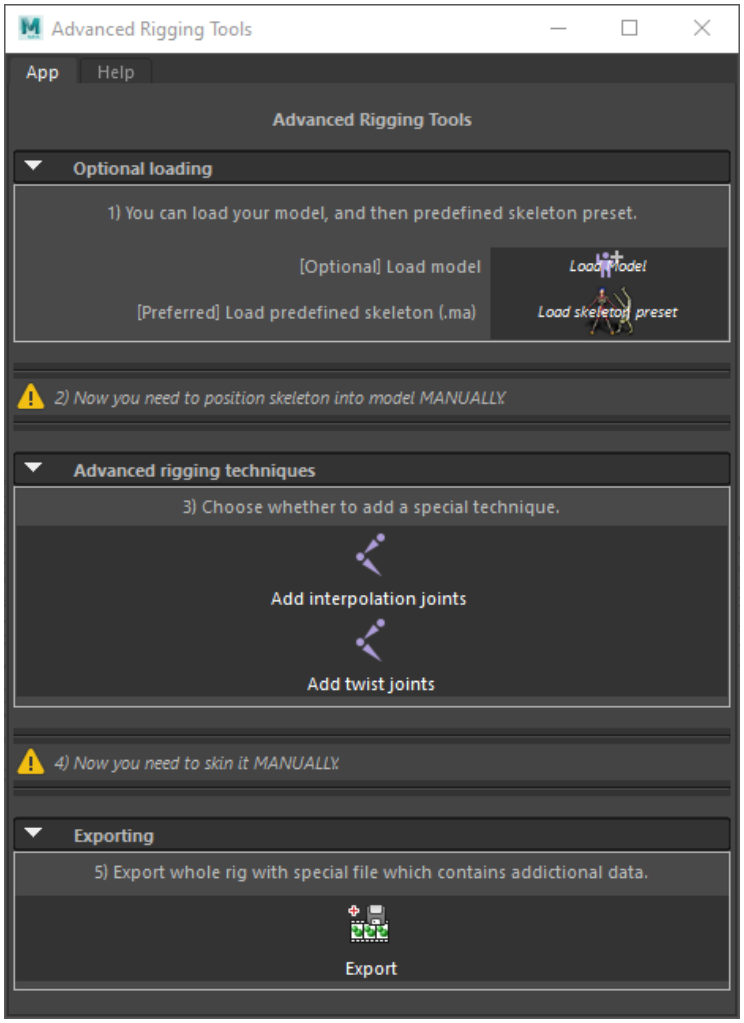
C.1 Obecné informace

Převod pokročilejších technik se skládá z několika kroků. Prvním krokem je vytvoření těchto technik v programu Autodesk Maya. Následuje export z tohoto programu. Exportuje se naskinovaný model ve formátu `.fbx` a navíc popis těchto pokročilejších technik ve formátu `.JSON`. Uživatel dále pokračuje do Unity, kde importuje tuto postavičku a vytvoří pro ni tzv. `Rig Manager`. Ten pomáhá sestavovat rig. `Rig Manager` přetáhneme do modelu postavičky ve scéně a načteme do něho exportovaný soubor ve formátu `.JSON`, který jsme vytvořili v programu Maya. Pokud import proběhne bez chyb, tak je rig importovaný a připraven k použití.

C.2 Příprava postavičky v programu Autodesk Maya

Po spuštění aplikace (jak je představeno v instalační příručce v kapitole B.2) můžeme vidět hlavní obrazovku, viz obrázek C.1. Aplikace má zde dvě záložky. Jedna poskytuje ovládání aplikace a druhá, kde je popsán základní návod, jak aplikaci používat.

Okno aplikace je rozděleno horizontálně na 5 částí, které představují jednotlivé kroky. Ty jsou potřeba splnit pro přidání a exportování pokročilejších



Obrázek C.1: Hlavní okno v programu Autodesk Maya

technik. Tři z nich jsou za asistence aplikace, kdy postupujeme podle pokynů na dalších obrazovkách. A dvě z nich jsou potřeba udělat manuálně. Proto jsou zvýrazněny žlutým trojúhelníčkem. Při používání aplikace uživatel postupuje odshora dolů, kdy je nutné dodržet následující sekvenci úkonů.

Těmito kroky jsou:

1. načtení modelu a kostry,
2. ruční napozicování kostry do modelu a úprava lokálních os rotace,
3. přidání pokročilejších technik,
4. manuální skinování,
5. exportování.

C.2.1 Načtení modelu a kostry

Prvním krokem je načtení modelu a jedné z předpřipravených koster. Jak model, tak kostru si uživatel může vložit do scény vlastní. Nemusí pro tento účel využívat aplikaci. Je ale doporučené využít předpřipravenou kostru, protože je plně kompatibilní s Unity animačním systémem a má předpřipravené lokální osy rotace na končetinách. Druhou možností je vytvořit základní kostru pomocí `HumanIK`. Pokud žádné kostry nebo modely nevidíme, buď jsme v jiné složce, nebo špatně filtrujeme soubory. Přepneme proto filtrování souborů na *All Files*. Pro správnou funkčnost by model měl být v T-Pose, tak jak je popsáno v kapitole 4.4.4. Unity tuto kostru požaduje. Tato aplikace s tímto nastavením počítá. Navíc v případě nedodržení T-Pose nebude správně fungovat *animation retargeting*. Možností je narigovat a naskinovat postavičku v A-Pose bez speciálních kloubů. Tento model poté narovnat. Přidat pokročilejší techniky a pomocí *Skin -> Edit influences -> Add influences* přidat nově vytvořené klouby a přidat jim váhy. Tato možnost je ale doporučená pro zkušenější rigery.

C.2.2 Posunovaní kloubů kostry na správná místa

Následuje první manuální krok. Je jím pozicování kloubů na jejich správné místo v modelu. Při pozicování je nutné dodržet několik zásad, nejen aby výsledek byl co nejlepší, ale hlavně, aby rig správně fungoval.

Prvním krokem bude naškálování celé skupiny kloubů, to provedeme škálováním transformačního uzlu `rig_group`. Poté je doporučené použít nástroj *Freeze Transformations*[46] na tuto hlavní skupinu, aby škálování bylo v jednotlivých osách $[1, 1, 1]$. Protože při následném pozicování kloubů, pokud budeme různě odpojovat klouby a zase napojovat[47], budou vznikat prázdné transformační uzly.

C.2.2.1 Postup při rigování

- Po načtení kostry pozicujeme jednotlivé klouby na správné místo. Při pozicování jdeme od *Root/Hips* kloubu až ke koncům končetin. Ideální je proto naškálovat celou kostru. Tu vložit přibližně do modelu, a pak jen jednotlivé klouby dolaďovat. Pro pozicování končetin je vhodné využívat zejména rotování rodičovských kloubů a délku kostí prodlužovat jen v jedné ose.

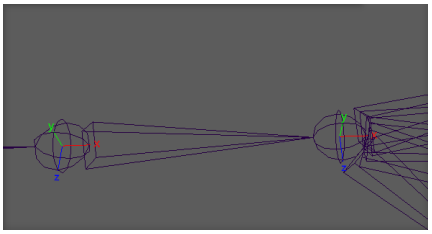
- Až jsme dopozicovali danou končetinu, použijeme na ni nástroj *Freeze Transformations*, abychom přesunuli hodnoty z *rotate* do *joint orient*.

[46] Nástroj *Freeze Transformations* se nachází v menu v záložce *Modify*.

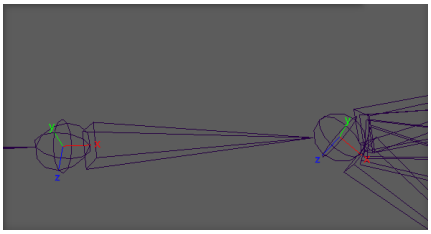
[47] Tato situace může třeba vzniknout při nastavování LRA u ruky, kdy odpojíme všechny prsty. Nastavíme LRA a znovu připojíme. Nebo při zrcadlení části rigu z jedné strany na druhou.

A následně použijeme nástroj *Orient Joint*, který nám nastaví lokální osy rotace tak, aby primární osa rotace (doporučená osa `x`) vždy mířila na další kloub v řetězci (mimo výjimky jako například zápěstí). Tento krok je velmi důležitý. Je potřeba ho udělat správně. V opačném případě nejen, že nebude rig dávat hezké výsledky, ale dokonce nebude fungovat vůbec. A v programu Unity bude dávat špatné výsledky. Situace, kdy jsou osy špatně nastavené, vidíme na obrázcích C.2 a C.4.

- Tento postup se snažíme dodržet pro všechny končetiny.

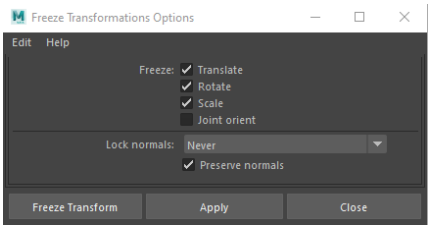


(a) Dobře nastavené LRA

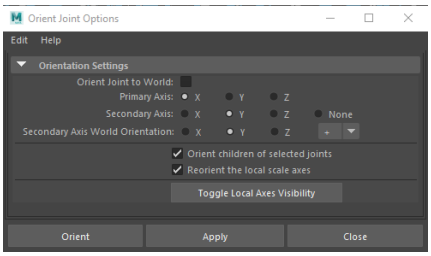


(b) Špatně nastavené LRA

Obrázek C.2: Zde můžeme vidět dvě konfigurace nastavení LRA. V případě napravo je zřejmé, že při rotaci po ose `x` bychom nepřirozeně rotovali zápěstím pod úhlem. Navíc v případě zakomponování pokročilejších technik bychom dostávali nesmyslná řešení. Proto je nutné napozicovat klouby tak, aby dávaly smysl a rozmyslet si, jak tyto LRA nastavit



(a) Nástroj *Freeze Transformation* inicializuje všechny transformace. U kloubů ponechá hodnoty posunu, protože na základě nich je definována délka kosti



(b) Nástroj *Orient Joint* pro orientaci kloubů. Občas nedává nejlepší výsledky, je proto po nástroji potřeba zkontrolovat výsledky

Obrázek C.3: Ukázka nástrojů *Freeze Transformation* a *Orient Joint*

C.2.2.2 Zásady správného rigu

Pro úplnost je vhodné ještě zmínit 4 zásady, které jsou uveřejněné na webu RiggingDojo [14].

- Všechny rotace v položce *rotate* musí být nastavené na nulu, tyto rotace budou v *joint orient*[48].

- U ruky a nohy, zejména v místě kolenního kloubu a lokte, mít sousední klouby v jedné rovině (obecně to jsou ta místa, která můžou být ovládaná pomocí IK). Navíc je nutné vždy zachovat vyplněnou pouze jednu hodnotu v *translate* a jednu hodnotu v *joint orient*.

- První kloub v řetězci kloubu (jedná se např. o kloub *LegUp* nebo *Shoulder*) může mít *joint orient* nastavený pro všechny osy.

- Poslední klouby v řetězci (palec na noze, poslední článek na ruce) se ze zvyklosti nastavují všechny *joint orient* hodnoty na 0. Tyto klouby nemají při skinnovaní žádné váhy. Jsou zde jen pro vizuální ukončení řetězce.

C.2.3 Pokročilejší techniky

C.2.3.1 Kontrola LRA

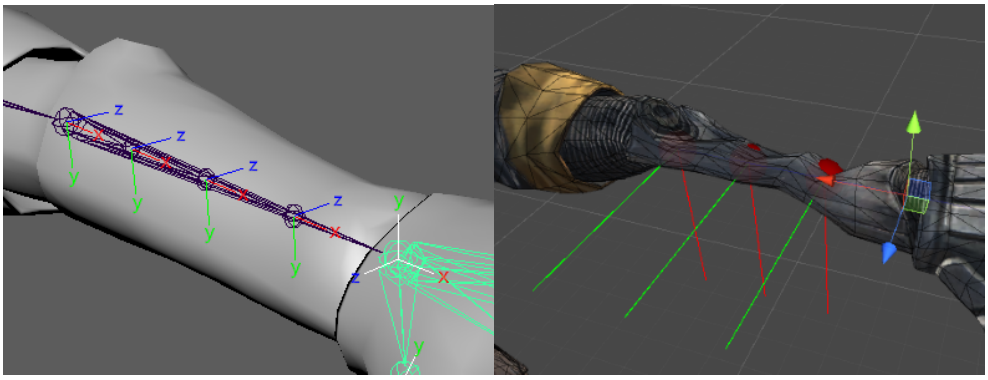
Až máme všechny klouby na svém místě, můžeme přidat pokročilejší techniky. Pro správnou funkci je ještě jednou vhodné upozornit, že je nutné zkontrolovat lokální osy rotace. Zejména na kloubech, které budou řídit speciální techniky. Pro *twist joints* je to druhý kloub, v menu pojmenovaný jako „*driver joint*“. Pro *interpolation joints* je to ten stejný kloub, který jsme vybrali. *Orient Joint* nástroj totiž velmi často při nastavování lokálních os prohodí[49] osy a na kloubech, kde bychom neměli mít žádné rotace, jsou najednou hodnoty 180° nebo 90°. Je třeba se takovým situacím vyvarovat. Zejména u kloubů řídící pokročilejší techniky. Situace na obrázku C.4 zobrazuje 180° rotaci v ose `x` u jednoho z řídících kloubů. Díky této iniciální rotaci výsledný program nebude fungovat, protože po exportu do Unity na tomto kloubu bude v ose `x` rotace 180°, která se bude brát jako výchozí rotace pro klouby obsahující pokročilejší techniky.

C.2.4 Přidání pokročilejších technik

V aplikaci je možné přidat dvě pokročilejší techniky, jejich tvorba je jednoduchá. Od této chvíle je nutné ukládat soubor v souboru `.ma` nebo `.mb`, jinak ztratíme nastavení pokročilejších technik. Pokud chceme přidat na stejný kloub obě dvě techniky, tak první přidáme techniku *interpolation joints*.

[48] Unity dokáže brát rotace kloubů, jak z položky *rotate*, tak z položky *joint orient*. Z důvodu čistoty rigu je ale dobré je mít v *joint orient* a *rotate* ponechat čistý, bez transformací.

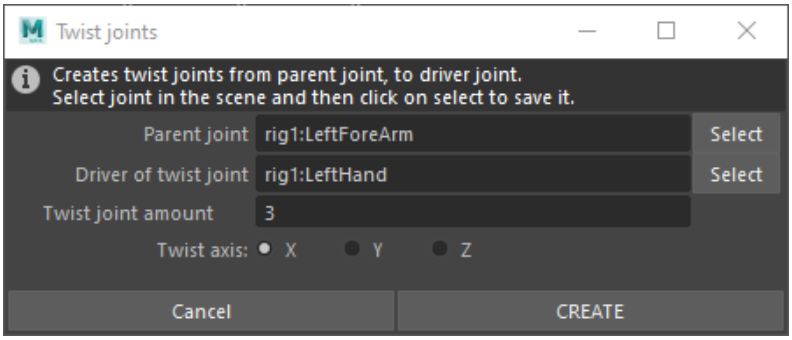
[49] Proto je lepší na tyto operace používat spolehlivějších nástrojů, jako je například `cometJointOrient http://www.comet-cartoons.com/melscript.php`. A po aplikaci ručně zkontrolovat.



Obrázek C.4: Iniciální rotace v zápěstí 180° (v *joint orient*), která způsobí po přenosu nevhodné chování aplikace

- ***Twist joints***

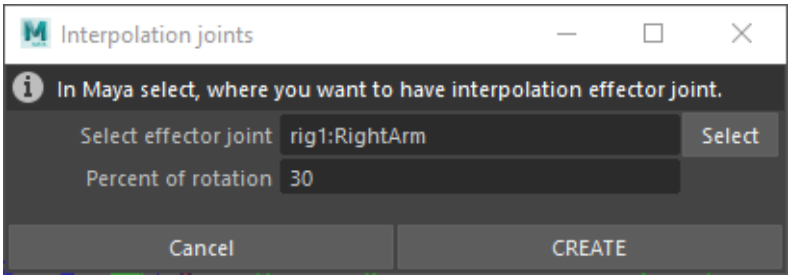
  Pro přidání klikneme v 3. sekci na tlačítko **Add twist joints**. Po stisku tlačítka se nám otevře nové okno, jak zobrazuje obrázek C.5. Pro přidání potřebujeme zadat dva klouby. První rodičovský „*parent joint*“, který bude rodičem nových vytvořených kloubů v hierarchii. A pak druhý, řídící kloub „*driver joint*“, ze kterého budeme číst hodnotu rotace. Tento řídící kloub v hierarchii by měl být dítětem rodičovského kloubu. Vytvořené *twist joints* budou mezi těmito dvěma klouby. Pro zadání je nutné kloub vybrat ve scéně, pak kliknout na tlačítko **Select**. Tím se potvrdí volba. Další nastavení je specifikace počtu vytvořených kloubů. Tyto klouby jsou uniformně vytvořeny a jejich míra rotace je úměrná vzdálenosti mezi řídícím a rodičovským kloubem. Potom vybereme, kolem které osy budou rotovat (v našem případě by to měla být osa `x`, ale v případě jiných konfigurací je zde možnost vybrat jinou osu). Tvorbu potvrdíme tlačítkem **Create**.



Obrázek C.5: Okno pro přidání *twist joints*

- ***Interpolation joints***

  Pro přidání klikneme v 3. sekci na tlačítko **Add interpolation joints** a objeví se nové okno C.6. V tomto případě je potřeba zadat jen jeden kloub. Vybereme proto ve scéně tento kloub a klikneme na tlačítko **Select**. Po jeho stisku bude kloub vybrán. Pak už jen nastavíme kolik % rotace bude tento kloub mít oproti řídícímu. Po stisku tlačítka **Create** budou tyto klouby vytvořeny.



Obrázek C.6: Okno pro přidání *interpolation joints*

C.2.5 Manuální skinování

Až máme pokročilejší techniky přidány, tak bude potřeba postavičku manuálně naskinovat. Vybereme proto všechny klouby (mimo koncových jako je např. *HeadEndJoint* nebo konečky prstů), a potom označíme polygonální síť. Přepneme Mayu do `rigging` módu a následujeme do menu, *Skeleton -> Bind Skin*. Zde vybereme *Skinning method* jako `Classic Linear` (protože Unity zatím neumí `Dual Quaternion`) a nastavíme počet ovlivňujících kloubů pro jeden bod polygonální sítě na 4, *Max influences: 4*[50]. Tímto krokem se udělá základní skinování, které ale není nejpřesnější. Zejména v případě pokročilejších technik, ty je potřeba naskinovat zvlášť.

C.2.5.1 Nástroj *Paint Skin Weights*

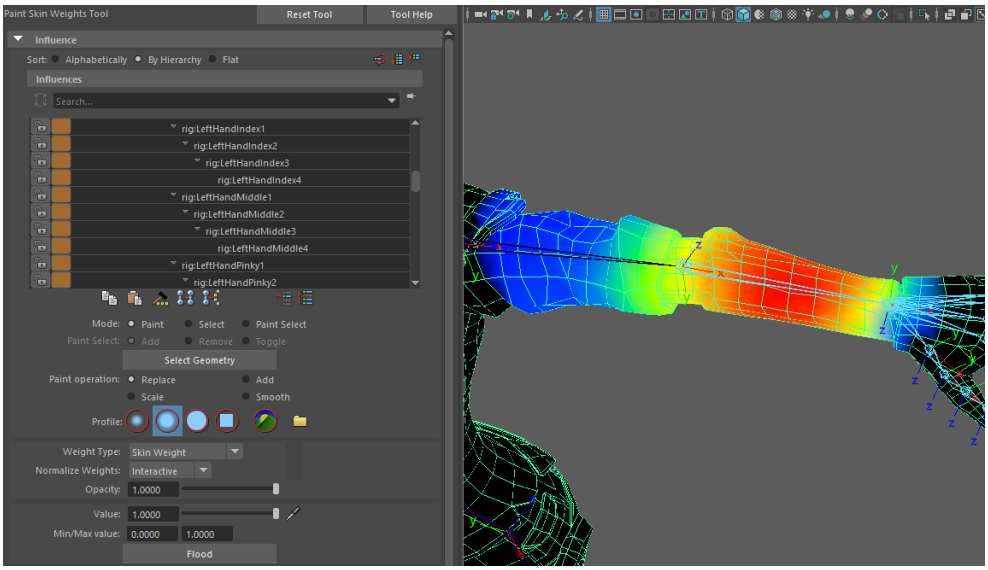
Proto otevřeme nástroj *Skeleton -> Paint Skin Weights*. S pomocí jeho budeme jednotlivým kloubům nastavovat, s jakou váhou budou působit a měnit příslušnou polygonální síť.

Vybereme danou polygonální síť. Po rozkliknutí nástroje uvidíme podobnou situaci, jako na obrázku C.7. V nastavení nástroje (na levé straně obrázku) můžeme vidět jednotlivé klouby. Po vybrání jednoho z nich uvidíme, kde a jakou měrou tuto síť ovlivňuje. Cílem je tedy projít jednotlivé klouby a upravit

[50]Unity donedávna více neumělo a pro podporu bylo nutné přepsat `build-in shadery`. Nově se ale u humanoid postavy objevila možnost navstavit větší hodnotu. Tato volba ale nebyla testována.

rozsah a váhu, kterou ovlivňují síť. Tak aby například při zvedání ramena se neposunovala i půlka trupu.

Každý vrchol polygonální sítě musí mít součet všech vah kostí roven 1. To může být problém v případě, když odebíráme váhy z jedné kosti. Ty se pak můžou vyskytnout na místech, kde je nechceme. Je proto vhodné váhy pouze přidávat, abychom se vyhnuli generování vah na nevhodných místech. Nastavíme tento nástroj na *Mode:* `Paint`, *Pain operation:* `Replace`, *Normalize Weights:* `Interactive`, *Opacity* dáme na nižší čísla, např. `0.25` a *Value* necháme na `1.0`. Zejména pro kreslení pokročilejších technik nám toto nastavení zaručí bezproblémovou tvorbu.



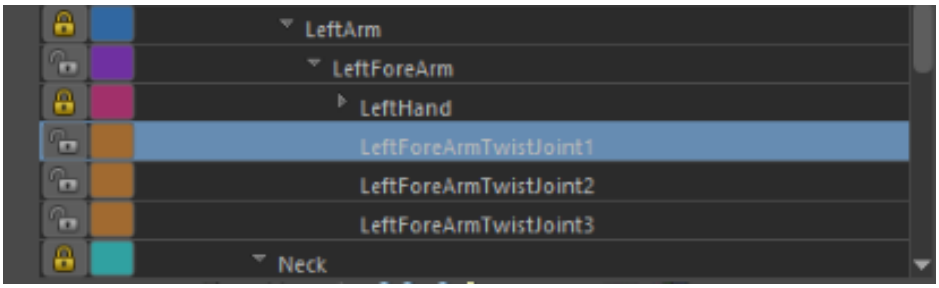
Obrázek C.7: Nástroj *Paint Skin Weights*, s jeho pomocí můžeme nastavit, jak jednotlivé klouby budou ovlivňovat přidruženou polygonální síť

C.2.5.2 Skinování *twist joints*

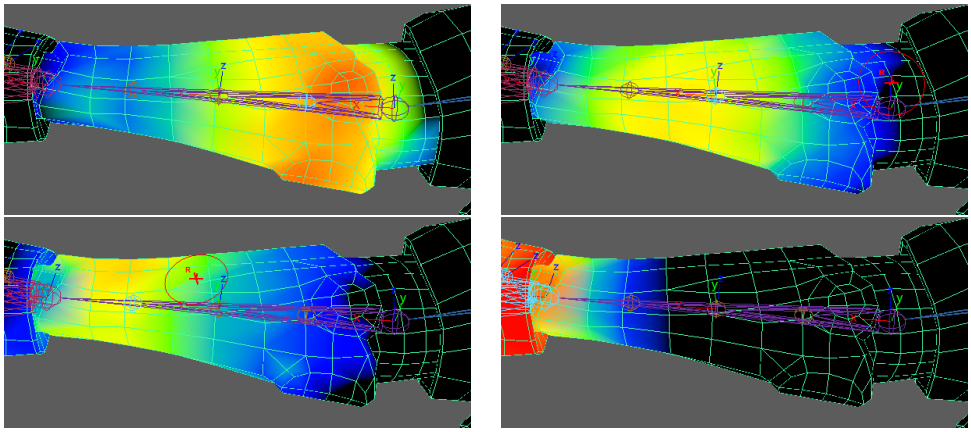
Úkolem této techniky je rovnoměrně rozprostřít deformaci způsobenou rotací primární osy řídícího kloubu do rodičovského kloubu. Představme si situaci, kdy mezi loktem a zápěstím máme další 3 klouby, které se budou rotovat v ose `x` s 75 %, 50 % a 25 % rotace řídícího kloubu (procenta uváděna od zápěstí).

Při skinování zamkneme všechny ostatní klouby (abychom nepřekreslovali váhy). Necháme otevřené jen *twist joints* a jejich rodičovský kloub, viz obrázek C.8. Postupujeme směrem od rodičovského kloubu (takže první se bude skinovat kloub, který je nejdále od řídícího a má rotaci 25 %). V okolí tohoto kloubu nakreslíme váhy. Ty budou začínat po rodičovském kloubu. Na druhé straně můžeme klidně přetáhnout až k prvnímu kloubu. Díky správnému nastavení nástroje, jak bylo popsáno v posledním odstavci kapitoly C.2.5.1, nás

přetažení trápit nemusí, protože při kreslení následujícího kloubu budou přetažené váhy překresleny. Na konci je vhodné ještě výsledek vyhladit, například pomocí nástroje *Smooth Skin Weights* nebo lépe `ngSkinTools`[51].



Obrázek C.8: Zamykání kloubů v nástroji *Paint Skin Weights*

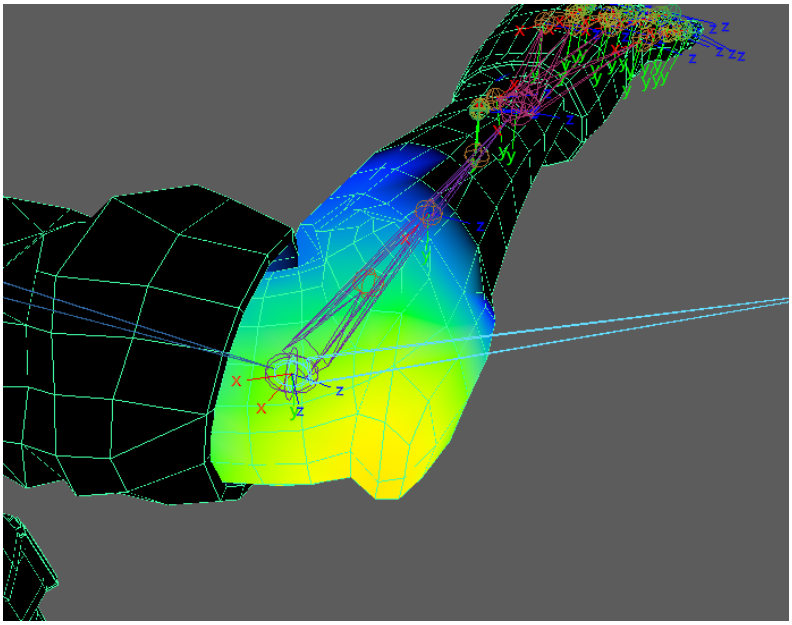


Obrázek C.9: Ukázka naskinovaných *twist joints*. Výsledek byl poté ještě vyhlazen, aby nebyly přítomné žádné ostré přechody

C.2.5.3 Skinování *interpolation joints*

Interpolation joints se zase používají pro zachování objemu v případě velkých rotací kloubu. Jsou to klouby, jako je například loket, kde bude docházet ke ztrátě objemu na vnější straně kloubu. Při tvorbě necháme všechny klouby zamknuté, až na daný *interpolation joint* a kloub, který zdvojuje, případně i jeho rodiče. Při kreslení je vhodné natočit daný kloub do pozice, ve které dochází k úbytku objemu, aby byl tento úbytek viditelný. Nástroj máme nastavený podobně jako v případě *twist joints*. Při kreslení pak přidáváme váhy tak, aby při větších rotacích byl úbytek objemu méně znatelný.

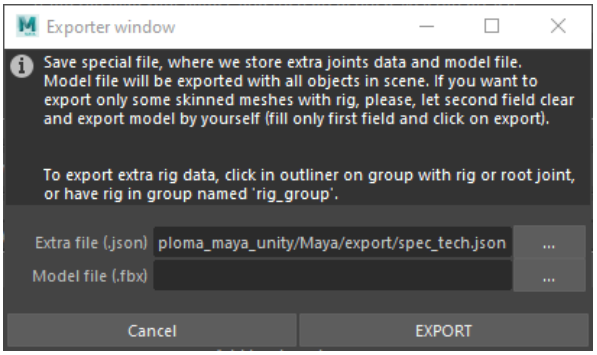
[51] `https://www.ngskintools.com/`



Obrázek C.10: Ukázka naskinovaných *interpolation joints*

C.2.6 Export

Po dokončení skinování je posledním krokem exportování postavičky. V hlavním menu proto klikneme na tlačítko **Export**, které otevře nové okno, viz obrázek C.11. Zde vybereme místo, kam uložíme soubor obsahující popis pokročilejších technik ve formátu `.JSON`. Aby bylo možné exportovat, je nutné mít kostru uloženou v skupině `rig_group` nebo mít označený první kloub v hierarchii *Root/Hips joint*, nebo označenou skupinu, která obsahuje kostru. Druhou položkou je možnost exportování modelu ve formátu `.fbx`. Tento krok ale můžeme vynechat a exportovat model s kostrou a skinováním samostatně.



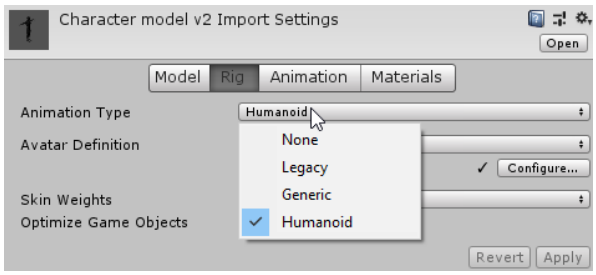
Obrázek C.11: Okno pro exportování modelu s pokročilými technikami

Po stisku tlačítka **Export** dojde k exportování popisu pokročilejších technik a případně modelu na požadované místo.

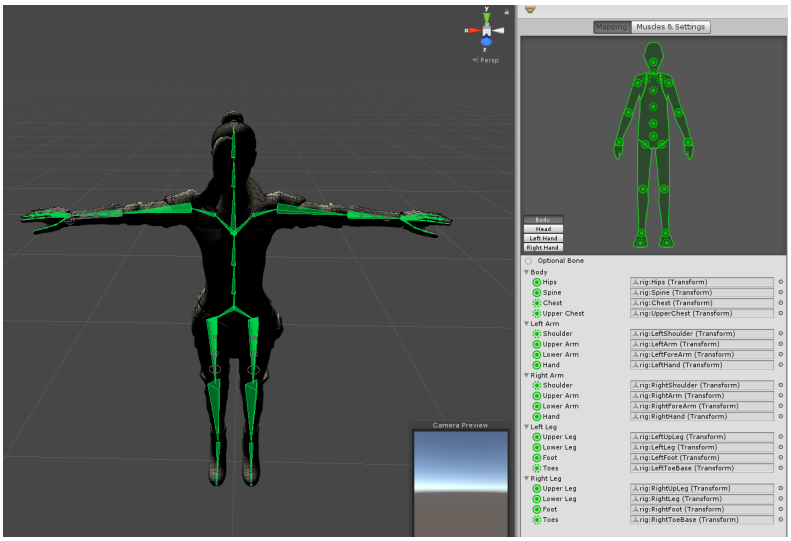
C.3 Přidání pokročilejších technik v Unity

- **Nastavení modelu**

  Před samotným přidáním je potřeba načíst do Unity model, u kterého musíme nastavit typ rigu na `humaniod`. Vybereme proto model v Unity okně `Project` (tam, kde máme všechny assety) a nastavíme mu tento typ rigu. Volbu potvrdíme. Pak stiskneme tlačítko **Configure**, kde zkontrolujeme kostru. Situace by měla vypadat tak, jako na obrázku C.13. Všechny hlavní klouby budou ovládány Mecanimem. Klouby zprostředkovávající pokročilejší techniky zůstanou šedé. Pokud tak není, je potřeba přetáhnout z Unity `Hierarchie` scény správné klouby do příslušných kolonek, jak je zobrazeno na pravé straně obrázku.



Obrázek C.12: Nastavení typu rigu v Unity



Obrázek C.13: Nastavení kostry v Unity

- **Instalace balíčku do Unity**

  V Unity po přidání balíčku, dle kapitoly B.2, navigujeme do záložky v menu *Advanced Rig*, kde vybereme položku *Rig Tools*. Objeví se nám okno, které tvoří speciální objekt ve scéně zvaný `Rig Manager`. Ten má za účel načtení a aplikaci pokročilejších technik do Unity.

- **Aplikace pokročilejších technik**

  Tento objekt vytvoříme pomocí tlačítka **Create Rig Manager gameObject** a přesuneme ve scéně do modelu, pro který chceme načíst pokročilejší techniky (jedná se o ten stejný model, který jsme v programu Autodesk Maya exportovali). Je nutné mít tento objekt na stejné nebo vyšší úrovni v hierarchii (blíže k objektu zaobalující celý model), jako máme kostru. Poté v tomto objektu stiskneme tlačítko **Build it**, kde následně vybereme exportovaný soubor ve formátu `.JSON`, který obsahuje popis pokročilejších technik. V případě úspěšného načtení a aplikace na klouby budeme skrze tento objekt a konzoli informováni. Celý import je tímto krokem hotový.

  Tím je celý přenos ukončen. Je pak vhodné tento model přesunout do Unity okna `Project` a udělat z modelu prefab, abychom mohli poté jen přesunout tento objekt do scény a nemuseli již nic importovat.

Příloha D

Obsah přiloženého DVD

- `readme.txt` ..... stručný popis obsahu DVD
- `app`
  - `Maya` ..... skripty aplikace pro program Autodesk Maya
  - `advanced_rig.package` ..... balíček pro program Unity
- `attachments`
  - `screenshots` ..... obrázky aplikace
  - `video` ..... videa aplikace
  - `others` ..... adresář obsahující uživatelskou a instalační příručku
- `doc` ..... adresář s dokumentací
- `exe` ..... adresář se spustitelnou formou testovací aplikace v Unity
- `project`
  - `Maya` ..... projekt v programu Autodesk Maya
    - `model_data` ..... adresář obsahující rozpracované testovací modely
  - `Unity` ..... projekt v programu Unity
- `src`
  - `Maya` ..... zdrojové kódy implementace pro program Autodesk Maya
  - `Unity` ..... zdrojové kódy implementace pro program Unity
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text`
  - `DP_Tkac_Jakub_2020.pdf` ..... text práce ve formátu PDF
  - `zadani_prace.pdf` ..... zadání práce ve formátu PDF