VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POSTUPNÁ DEFORMACE 3D MODELU

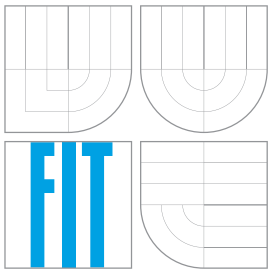
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MAREK ZOUHAR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POSTUPNÁ DEFORMACE 3D MODELU

3D SHAPE INTERPOLATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MAREK ZOUHAR
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL ŠPANĚL, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabývá tvorbou nástroje pro postupnou změnu tvaru trojrozměrného polygonálního modelu. Použité metody se inspirují zejména technikou Manual Landmarks a využívají parametrizace objektu na kouli. Výsledkem práce je zásuvný modul do volně šiřitelného modelovacího programu Blender, který automaticky vytvoří animaci změny tvaru ze dvou vstupních modelů.

Abstract

This thesis deals with creation of a tool for progressive interpolation of shape of a 3D polygonal mesh. Used methods take inspiration mainly in technique Manual Landmarks and use parametrization of an object to a sphere. The result of this thesis is an addon for an open-source modelling program Blender, which automatically creates an animation of shape deformation based on two input models.

Klíčová slova

3D model, polygonální model, interpolace tvaru, změna tvaru, deformace, animace, parametrizace na kouli, mapování, Blender

Keywords

3D model, polygonal model, shape interpolation, mesh morphing, deformation, animation, spherical parametrization, mapping, Blender

Citace

Marek Zouhar: Postupná deformace 3D modelu, bakalářská práce, Brno, FIT VUT v Brně, 2015

Postupná deformace 3D modelu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Zouhar
16. května 2015

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Michalu Španělovi Ph.D. za vedení kvalifikační práce a své rodině a přátelům za podporu.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Počítačová grafika má v dnešním světě velmi široké využití. Nikoho dnes nepřekvapí, že trojrozměrné modely a jejich animace se ve velké míře využívají v zábavním průmyslu, ať už jde o ten filmový či herní. V posledních letech byla ale rovněž odhalena jejich moc například i v medicíně, kde se využívají jak voxelové, tak polygonální modely.

Ačkoli animace nám v lékařství může sloužit často jen k větší názornosti, změna tvaru modelů má daleko širší využití. Například I. A. Sigal et al. [7] používá změnu tvaru existujícího modelu myšího obratle pro generování dalších modelů specifických pro jedince, což se podle očekávání ukázalo být mnohem efektivnější metodou, než dříve užívaná tvorba vždy nového modelu. Dalším příkladem může být práce M. E. Biancoliniho et al. [2], jež používá postupnou změnu tvaru polygonálního modelu pro optimalizaci pozice a natočení plachet při plachtění. Ve filmovém průmyslu jsou dnes běžně do reálné scény přidávány počítačové modely a někdy jsou filmy, podobně jako počítačové hry, kompletně produktem počítačové grafiky.

Cílem této práce bylo vytvořit nástroj pro postupnou deformaci tvaru trojrozměrného objektu. Práce se věnuje několika základním aspektům: zpracování vstupu uživatele, jímž jsou jednak dva modely a jednak definice jejich vzájemných vazeb; vytvoření mapování mezi vstupními modely a tvorba animace změny tvaru vstupního modelu. Práce se soustředí na zpracování modelů s podobným počtem vertexů, snaží se ale také rozšířit své techniky pro vstupní objekty, které se podobají méně. Řešení vychází především z techniky Manual Landmarks a využívá parametrizace na kouli s Laplaceovým vyhlazováním. Nástroj byl implementován v podobě zásuvného modulu do modelovacího programu Blender.

Kapitola 2 čtenáře nejprve seznámí s nejnutnějšími termíny, jež jsou v práci použity a následně se věnuje stručnému shrnutí technik řešení problémů blízké této práci, které se dnes používají a ze kterých jsem vycházel. Kapitola 3 představí plán, podle nějž byl konečný nástroj tvořen a kapitola 4 potom rozvádí konkrétnější aspekty řešení. Další kapitola 5 uvádí některé experimenty, které s řešením byly prováděny a dává příklad výsledků, jež nástroj produkuje. V poslední kapitole 6, která je shrnutím a zhodnocením práce si můžeme přečíst návrhy na možná budoucí rozšíření.

Kapitola 2

Reprezentace modelu a současné metody problematiky

Pro reprezentaci trojrozměrných objektů se nejčastěji používají následující tři způsoby: hraniční, voxelová a CSG reprezentace [9].

Voxelová reprezentace je ve své podstatě velmi podobná rastrovému obrázku, pojem *voxel* vznikl sloučením slov *volumetric element* – objemový prvek, podobně jako *pixel* vzniknul složením slov *picture element* – prvek obrázku. Již z tohoto jednoduchého vysvětlení vyplývá, že *voxel* je hodnotou (například údajem o barvě) ve 3D mřížce, což se také odrazí na vzhledu voxelového modelu. Tato reprezentace je vhodná zejména pro ukládání některých lékařských či jiných vědeckých dat, můžeme se s ní ale setkat například i v herním průmyslu především pro reprezentaci terénu.

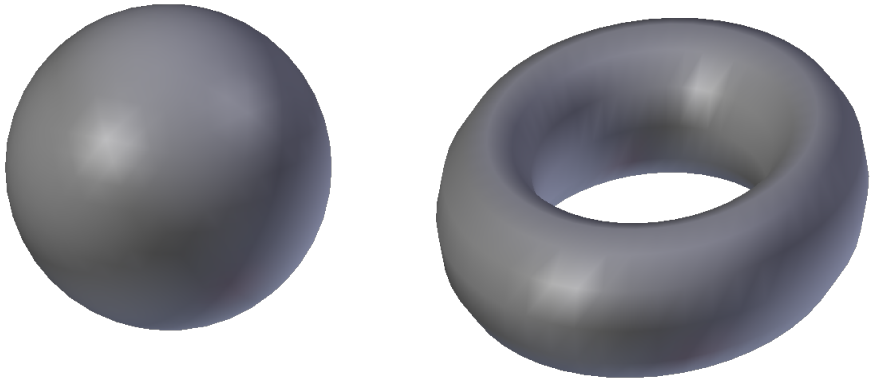
CSG reprezentace (Constructive Solid Geometry) spočívá ve tvorbě modelů pomocí základních tvarů a operací. Základními tvary mohou být například krychle, kvádr, koule, válec, kužel apod. Těm může být dále změněna velikost nebo orientace v prostoru. Operace, které můžeme použít na dva či více objektů, jsou sloučení, průnik a rozdíl. Z podstaty tohoto přístupu je zřejmé, že by bylo extrémně náročné tímto způsobem vytvářet některé modely. Síla této reprezentace ale spočívá v tom, že základní tvary jsou přesně matematicky definovány. Z tohoto důvodu je CSG reprezentace vhodná pro automatizovanou výrobu předmětů nebo procedurální generování modelů.

Hraniční reprezentace je asi nejrozšířenější variantou a z ní také vychází tato práce. Objekty této reprezentace jsou definovány mnoha body v prostoru – *vertexy* – které mezi sebou mohou mít spojnice – *hrany* – čímž utváří *stěny* modelu. Jedná se tedy vlastně jen o dutý plášť. Ze zmíněných reprezentací je tato nejvhodnější pro animaci. To je jeden z důvodů proč se využívá nejvíce ať už v zábavním průmyslu nebo pro medicínské a vědecké účely. Objekty v této prezentaci můžeme získat vymodelováním v jednom z mnoha specializovaných editorů, nebo můžeme například naskenovat skutečný objekt[1].

[1] Skenováním skutečného objektu je myšleno například použití 3D skeneru pro naskenování nějakého předmětu anebo využití údajů počítačové tomografie či magnetické rezonance pro vytvoření modelů orgánů či kostí pacienta.

2.1 Metody mapování

Aby bylo možné vytvořit animaci změny tvaru objektu na tvar objektu druhého, je nutné mezi oběma objekty vytvořit mapování. Obecně se však jedná o velice komplexní problém a proto se tato kvalifikační práce, podobně jako řada jiných prací vědeckých, zaměřuje pouze na jistou podmnožinu takzvaných **Genus 0** objektů. Jednoduše řečeno se jedná o tvary, jež neobsahují otvory (viz 2.1).



Obrázek 2.1: Příklad objektů klasifikovaných jako genus 0 (vlevo) a genus 1 (vpravo)

Jak již bylo zmíněno v úvodu, I. A. Sigal et al. [7] pracoval na tvorbě nových modelů myších obratlů ze stávajících modelů pomocí změny tvaru. Ve své práci k tomu používá dva algoritmy: Automated Wrapping (volně přeloženo jako Automatizovaná obálka) a Manual Landmarks (Ruční značky).

Podstata algoritmu **Automated Wrapping** vychází z toho, že je mnohem jednodušší nalézt mapování objektu na jednoduchý pomocný tvar, než najít mapování mezi dvěma složitými objekty. Pro příklad obratlů tedy Sigal zvolil tvar kapsle, na niž se zdrojový i cílový obratel mapovaly snáz (viz obrázek 2.2). Pro parametrizaci na kapsli byla užita metoda Minimalizace energií, jež je popsána v další podkapitole.

Metoda **Manual Landmarks**, kterou ve své práci používala i Z. Salo et al. [5] pro změnu tvaru modelu pánevní kosti, spočívá v umístění řady bodů na zdrojový i cílový objekt tak, aby si body vzájemně odpovídaly. Tyto vertexy byly voleny tak, aby se snadno hledal jejich protějšek na druhém modelu. Sigal uvádí, že Automated Wrappings algoritmus sice vytvářel modely, které více odpovídaly tvaru očekávaného výsledku, ale Manual Landmarks na druhou stranu zajistí, že si vstupní definované vertexy zdrojového i cílového modelu odpovídají. Experimenty s oběma algoritmy dále ukázaly, že odchylka v přesnosti obou je zanedbatelná.

T. Athanasiadis et al. [1] se věnoval přetvarování modelů, které prezentoval například na lidské tváři. Jeho algoritmus využívá přizpůsobeného mapování na kouli a následně vytváří na modelech *oblasti rysů*, jak je nazývá (feature regions), které si vzájemně odpovídají, podobně jako v metodě Manual Landmarks.

M. E. Biancolini popisuje několik dalších metod pro změnu tvaru objektu [2]. **Free Form Deformation** je poměrně rozšířená technika, nicméně svým principem práce bez cílového tvaru, je pro tvorbu animace v této práci nevhodná. Ze stejného důvodu není optimální ani algoritmus využívající **Radial Basis Functions**, který kombinuje předchozí metodu s přesnější deformací, navíc je tento způsob prý značně výpočetně náročný.

2.2 Metody parametrizace

Řada výše zmíněných metod pro deformaci tvaru modelu předpokládá schopnost model vhodně parametrizovat. Objekty typu genus 0 jsou topologicky totožné s koulí a tak se právě jednotková koule nabízí jako přirozené těleso na které objekt parametrizovat. C. Wang et al. [8] i L. Ying et al. [17] uvádí, že poměrně častým postupem pro parametrizaci objektu na kouli zjednodušení problému na **parametrizaci do roviny** a následné použití stereografické projekce pro namapování na kouli. Asi nás ale napadne, že je poněkud zbytečné přecházet mezi výchozím objektem, rovinou a koulí, a že přirozenější je model parametrizovat na topologicky ekvivalentní kouli rovnou.

Parametrizace výchozího modelu na kouli je v zásadě nalezení mapování jedna ku jedné výchozího povrchu na povrch koule. Při tom však zpravidla vznikají různé druhy zkreslení, které se snažíme během tohoto procesu minimalizovat. L. Shen a F. Makedonová shrnují tři druhy zkreslení [6]:

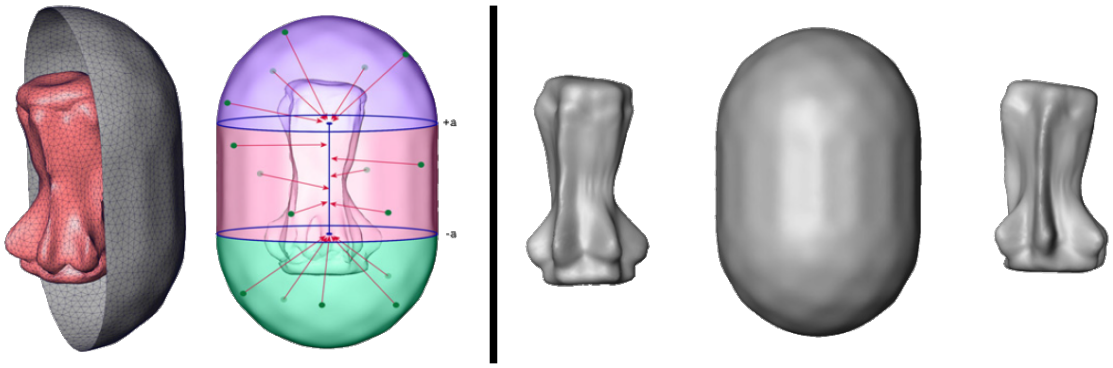
- **Izometrické mapování**: mapování je izometrické, pokud každý oblouk výchozího povrchu je namapován na oblouk povrchu cílového o stejné délce (zachovává délky)
- **Konformní mapování**: mapování je konformní, pokud velikost úhlu, jenž svírá každý pár protínajících se oblouků výchozího povrchu, je rovna velikosti úhlu odpovídající dvojice protínajících se oblouků na povrchu cílovém (zachovává úhly)
- **Equiareální mapování**: mapování je equiareální, pokud každá část vzorového povrchu je namapována na část povrchu cílového o stejné ploše (zachovává plochu)

Mapování je izometrické, pouze když je současně konformní a equiareální. Je zřejmé, že ideální je mapování izometrické, jenže takové nemusí existovat. V praxi se proto požívají algoritmy, které se snaží různé typy zkreslení minimalizovat.

Pokud bychom jednotlivé vertexy modelu pouze jednoduše promítli na kouli, u obecného modelu by s největší pravděpodobností došlo k překrytí hran objektu. Tento problém ve své práci například Sigal et al. [7] řešil použitím procesu **Minimalizace energií** (Energy minimization process).

Během něj je výchozí model i cílový povrch pro parametrizaci chápán jako množina bodů, přičemž pozice bodů výchozího modelu jsou uzamčené. Pozice bodů cílového povrchu začaly na jednoduchém tvaru a bylo jim umožněno se pohybovat, dokud nebylo nalezeno jejich umístění s nejnižší hladinou energie. Funkce pro výpočet energie měla tři složky. První byla energie dlouhé vzdálenosti, která usměrňovala počáteční rozložení vertexů vzhledem k výchozímu tvaru, aby bylo jejich rozdělení pokud možno homogenní. Druhá složka energie byla nelineární funkce, jež upřesňovala pozicování cíle, a poslední třetí složka zajišťovala, aby nedošlo k překrývání hran na cílovém tvaru. Tento proces si můžeme představit jako ustálení bodů modelu, který má své body propojené pomocí nelineárních pružinek, přičemž na každé působí přitažlivé a odpudivé síly s minimem ve výchozí pozici. Přitažlivé síly zajišťují aby celková struktura povrchu nebyla porušena a aby byla zachována rovnoměrná distribuce bodů, odpudivé síly zase brání kolapsu bodů vlivem sil přitažlivých. Výsledek parametrizace můžeme vidět na obrázku 2.2. Také Ola Weistrand [10] zmiňuje, že Kent et al. používal podobnou metodu, založenou na fyzikálních vlastnostech pružin napnutých mezi vertexy modelu zatímco na ně působí síla snažící se objekt „nafouknout“.

S jistou funkcí energie pracuje rovněž metoda založená na takzvané **ARAP (As-Rigid-As-Possible)** energii, z níž vycházel C. Wang et al. [8]. Algoritmus pracuje v několika krocích, přičemž zajímavý je jistě ten, skládající se z lokální a globální fáze. V lokální fázi



Obrázek 2.2: V levé části příklad namapování obratle na kapsli. V pravé části výchozí a cílový obratel s parametrizací. (Z práce Sigal et al. [7])

tohoto kroku jsou trojúhelníky objektu promítnuty na kouli tak, že je zachován jejich vzájemný poměr velikostí, což je usnadněno tím, že je mezi nimi povoleno vytvořit úzké mezery. V globální fázi jsou potom tyto mezery odstraněny „sešíváním“ jednotlivých trojúhelníků zpátky dohromady. Jak již však bylo zmíněno, tento optimalizační algoritmus vyžaduje před vlastním spuštěním dostupnou jinou metodu parametrizace na kouli.

Shen a Makedonová zmiňují [6], že A. Sheffer s kolektivem vytvořili plochu zachovávající algoritmus, který dává dobré výsledky, ale je údajně značně pomalý a nevhodný pro modely o více než pár stech bodech. Obdobné nedostatky ve své práci připsal C. Wang [8] algoritmu Gotsmana et al., který měl umožnit parametrizovat jednoduché objekty na kouli vyřešením kvadratického systému rovnic, přičemž však nebyla příliš rozvinuta analýza na jaké modely lze algoritmus použít. Dále Wang et al. také zmiňuje několik algoritmů (A. Sheffer et al., H. Li a R. Hartley, X. Gu a S.-T. Yau), které se sice snaží zachovávat úhly – být konformní, ale značně zkreslují plochu.

Vlastní postup Shena a Makedonové [6], který nazývají **CALD** neboli **Control of Area and Length Distortions**, definuje parametrizaci *vysoké kvality* jako takovou parametrizaci, která má minimální zkreslení velikosti ploch a délek. Jejich algoritmus na výchozí parametrizaci iterativně aplikuje vyhlazovací procedury, čímž mění pozice vertexů objektu a zvyšuje kvalitu jeho parametrizace, aniž by měnil jeho topologii. Shen a Makedonová uvádí dva typy vyhlazovacích postupů: Laplaceovo a optimalizačně-zaměřené vyhlazování.

Laplaceovo vyhlazování jednoduše určuje novou pozici vertexu jako geometrický střed jeho sousedů. Tato metoda je sice jednoduchá a rychlá, ale negarantuje zkvalitnění parametrizace. Optimalizačně-zaměřené vyhlazování má zase daleko vyšší výpočetní náročnost, ale přináší lepší výsledky.

Ani s jednou z těchto dvou metod vědci však nebyli spokojeni a tak pro CALD vytvořili dvě nové metody a to *Lokální vyhlazování*, které vychází z tvorby podskupin vertexů, a *Globální vyhlazování*, jež se snaží zkreslení plochy vhodně rozptýlit po celé kouli.

V řadě prací narážíme na zmínky o práci pánů Prauna a Hoppa [10] [8] [6] [17], kteří používali strategii „hrubý-na-hladký“ (coarse-to-fine strategy). Během tohoto postupu tvoří nový mesh (model) pomocí algoritmu, jež původní objekt zjednodušuje odebíráním vertexů až do okamžiku, kdy získáme čtyřstěn. Ten je bijektivně namapován na kouli a následně jsou dříve odebrané vertexy rekurzivně znovu-vkládány, přičemž jsou jejich nové pozice optimalizovány vhodnými metodami pro kontrolu napětí.

Kapitola 3

Návrh řešení

Tato kapitola popisuje obecný přístup k řešení a algoritmy či metody, které byly použity. Cílem práce je vytvořit nástroj pro postupnou změnu tvaru modelu (který bude dále označován jako *hlavní*) na základě modelu druhého (dále označovaného jako *vedlejší*). Z důvodů, které jsou blíže rozepsány v kapitole 4, jsem se rozhodl tomuto nástroji dát podobu pluginu do programu Blender a to také silně ovlivnilo návrh jeho řešení. Při návrhu jsem bral v potaz účel práce a rozsah v jakém je možné se problematice věnovat. Jak již bylo naznačeno v kapitole 2, vytvořit nástroj pro bezchybnou interpolaci tvaru obecného modelu na libovolný jiný tvar, je extrémně komplexní problém a je tedy nutné si stanovit reálně splnitelný cíl vybráním podskupiny problému. Zvolenou podskupinou pro tuto práci jsou modely s obdobným počtem vertexů, ale bylo navrženo řešení, které se snaží nalézt vhodné výsledky i pro složitější vstupní objekty.

Většina metod pro parametrizaci a mapování modelů popsaných v předchozí kapitole 2 pracovala především s objekty získanými skenováním reálných objektů. Takto získané modely se vyznačují trojúhelníkovým tvarem svých stěn a někdy poměrně rovnoměrným rozložením vertexů po celém povrchu. Já jsem se rozhodnul zaměřit spíše na modely vytvářené ručně pomocí modelovacích nástrojů, které obvykle mívají stěny definovány čtyřmi vertexy. Při vykreslení takového objektu jsou stěny modelu samozřejmě pro účely renderování automaticky rozděleny na trojúhelníky, nicméně vnitřní reprezentace modelu stále sestává z původně definovaných stěn. Tento přístup umožňuje při modelování používat pokročilejší techniky, které práci urychlují a usnadňují.

Fakt, že jsem se soustředil na tento typ modelů, ale samozřejmě neznamená, že by mé řešení bylo pro modely první skupiny nepoužitelné. Jedním z faktorů proč vycházet z objektů druhé kategorie ale rozhodně je jejich dostupnost a možnost si sám vytvořit kontrolní skupinu modelů.

Dříve než začnu podrobněji popisovat jednotlivé části návrhu, zmíním základní myšlenku postupu. Uživatel nejprve nástroji předá všechna důležitá data a následně zvolí, který z implementovaných operátorů chce použít. Možnosti budou následující:

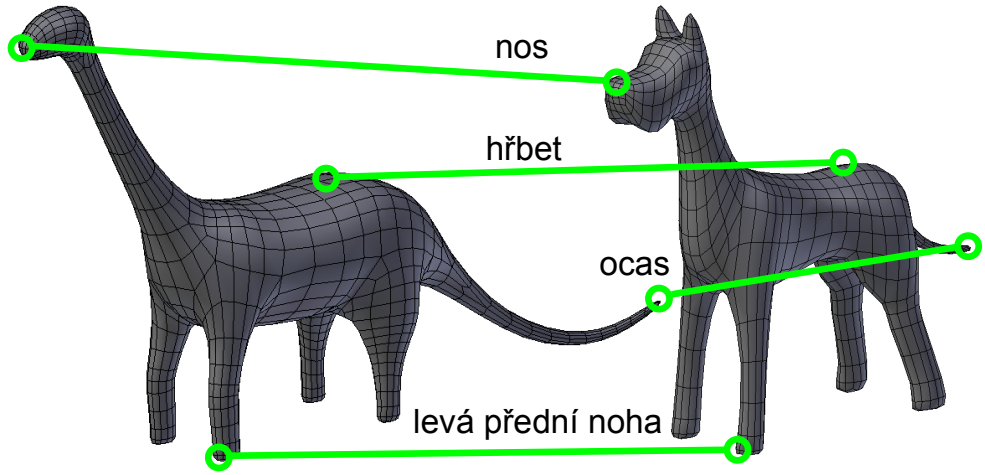
- Spustit základní verzi algoritmu pro vytvoření animace
- Spustit rozšířenou verzi s parametrizací na kouli a následným namapováním a vytvořením animace
- Spustit rozšířenou verzi s parametrizací využívající definovanou hranici a následným namapováním a vytvořením deformační animace
- Vytvořit pouze animaci z již vytvořených *tvarových klíčů* objektu

Jednotlivé možnosti budou dále rozepsány v nadcházejících kapitolách. Dalším krokem je zpracování zmíněné hranice, pokud byla zvolena tato možnost, a dále provedení parametrizace na kouli. Následně je provedeno namapování hlavního modelu pomocí parametrizovaných dat, případně s využitím skutečných pozic vertexů, pokud nebyla zvolena možnost s parametrizací. Po namapování je konečně vytvořena výsledná animace změny tvaru.

3.1 Uživatelský vstup a rozhraní

Vstupem nástroje mají být dva modely, přičemž tvar prvního má být deformován na tvar druhého. Uživateli tedy bude umožněno vybrat dva objekty, se kterými se bude pracovat, přičemž pro hlavní objekt bude vytvořena animace změny tvaru na tvar objektu pomocného.

Vzhledem k použitým algoritmům, které jsou popsány v dalších podkapitolách, musí být uživateli také umožněno vytvořit vazby mezi oběma modely, jež budou určovat, které vertexy hlavního modelu odpovídají kterým vertexům modelu pomocného. Uživatel tedy musí mít možnost na hlavním objektu například označit vertex jako *nos* a na pomocném modelu potom označit odovídající vertex stejným způsobem. Příklad několika žádoucích vazeb mezi modely můžeme vidět na obrázku 3.1.



Obrázek 3.1: Příklad několika definovaných vazeb mezi dvěma modely vytvořených pomocí stejně pojmenovaných *Skupin vertexů*

Podkapitola 3.2 popisující princip parametrizace modelu uvádí, že použitý algoritmus dokáže pro zvýšení kvality parametrizace využít uzavřeného řetězu vertexů na hranici souměrnosti, ale že takový řetěz vertexů je často možné nalézt i na nesouměrných modelech. Proto musí uživatelské rozhraní obsahovat možnost jak označit vertexy, které takový řetěz „souměrnosti“ tvoří, pokud se uživatel rozhodne aplikovat parametrizaci využívající této vlastnosti. Jestliže však uživatel manuálně definuje hraniční vertexy, je nutné také určit bod, od kterého má jejich zpracování začít. Pro zachování jednotnosti uživatelského rozhraní by bylo vhodné pro označení vertexů hranice i pro označení jejího prvního bodu použít stejný princip, jaký byl aplikován na tvorbu vazeb mezi modely.

Pokud uživatel již definoval všechny vzájemné vstupní vztahy a označil oba modely, je připraven spustit zpracování. To mu může být umožněno několika způsoby, jako například vyhledáním názvu operátoru, jež se chystá použít, proklikáním se k danému operátoru skrz menu, nebo použitím klávesové zkratky, která dané podmenu zpřístupní přímo.

Jak již bylo zmíněno, při používání ručně definované parametrizační hranice je nutné určit i její „počáteční“ bod. Pokud je ale model skutečně rovinně souměrný, bude v rovině souměrnosti existovat uzavřený řetězec vertexů představující právě požadovanou hranici. V takovém případě lze vyjít ze skutečnosti, že tyto vertexy mají nulovou X-ovou složku svých souřadnic a proto není nutné je definovat manuálně. Pro maximální komfort v podkapitole 3.2 popisuji postup, jak algoritmicky odhadnout počáteční bod této hranice i bez jeho explicitní definice. Jestliže se tedy uživatel snaží vytvořit animaci změny tvaru souměrného modelu a přitom využít parametrizace na kouli s využitím vertexů v rovině souměrnosti, není nutné tuto hranici ani její počátek explicitně definovat.

Vzhledem k tomu, že souřadnice vertexů používají čísla s plovoucí řádovou čárkou, se může stát že náš model, který je rovinně souměrný, nemá X-ovou složku svých souřadnic přesně nulovou a je prakticky nemožné tuto složku nastavit přesně na nulu bez přichycení vertexů k mřížce. Proto navrhuji ještě jeden pomocný operátor, který projde všemi vertexy a u uživatelem vybraných nastaví X-ovou souřadnici vertexů přesně na nulu. Tím se ještě zvýší využitelnost automatického nalezení hranice.

3.2 Parametrizace

Algoritmus, který jsem navrhnul, vertexy hlavního a pomocného modelu páruje na základě podobnosti úhlů mezi hranami. Proto není nutné parametrizaci komplikovat equiareálností. Je tedy využita jednoduše implementovatelná metoda **Laplaceova vyhlazování** [6] a průběžného promítání na jednotkovou kouli. Parametrizaci na kouli ve svém řešení používám ve snaze rozšířit své řešení i na obtížnější vstupy, neboť jsem využil poznatky nabyté například při studiu metody **Automated Wrapping** [7], v tom ohledu, že oba modely před vzájemným mapováním parametrizuji na jednodušší topologicky ekvivalentní objekt, abych se zbavil přehybů. V Sigalově práci, kde jsem se s touto metodou setkal poprvé, se jednalo o kapsli, neboť to byl tvar vhodný pro myší obratel. Pro poněkud obecnější model, který je však stále typu **genus 0**, dává smysl provádět parametrizaci na kouli.

Pro parametrizaci samotnou navrhuji dva přístupy. Jedním je prosté promítnutí všech vertexů na jednotkovou kouli pomocí rovnice

$$\mathrm{v}_{\mathrm{novy}} = \mathrm{v}_{\mathrm{stary}} * \sqrt{\frac{1}{\mathrm{v}_{\mathrm{stary}}.x^2 + \mathrm{v}_{\mathrm{stary}}.y^2 + \mathrm{v}_{\mathrm{stary}}.z^2}}$$

(kde „v“ je poziční vektor vertexu) a následně iterativní odstranění všech překryvů pomocí Laplaceova vyhlazování, které určuje novou pozici bodu jako geometrický střed všech jeho sousedů.

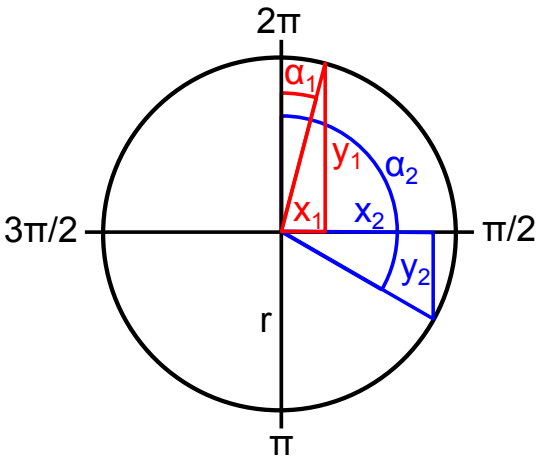
Druhý přístup, jak již zmínila minulá podkapitola, využívá skutečnosti, že mnoho trojrozměrných modelů je zrcadlově souměrných. Navíc, jak již bylo předesláno, lze tento postup rozšířit na modely, které sice nejsou souměrné, ale lze u nich nalézt uzavřený řetězec vertexů, jež by bylo možné rovněž použít jako pomyslnou hranici souměrnosti. Na této hranici se často vyskytují překryvy, které komplikují vyhlazovací část postupu a proto jsem navrhnul algoritmus, který z řetězu vertexů na hranici souměrnosti (případně, jak bylo zmíněno, z uživatelem definovaných bodů) vytvoří před spuštěním parametrizace kruh a zachová při tom poměr vzdáleností mezi hraničními vertexy.

Tento algoritmus nejprve musí určit, od kterého bodu má začít pracovat. Pokud si hraniční vertexy modelu představíme jako kruh a vsadíme do něj ciferník hodin, pak zmíněný počáteční bod se vyskytuje u čísla dvanáct. Jestliže uživatel tento bod nezadal manuálně,

vertexy hranice mají nulovou X-ovou složku a uživatel spoléhá na automatické nalezení tohoto bodu, pak algoritmus mezi hraničními vertexy najde několik bodů s Ypsilonovou souřadnicí co nejbližší nule a z těch poté vybere ten nejvýše položený (maximální Zetová souřadnice).

Jakmile získáme počáteční bod, je nutné ostatní vertexy hranice vhodně seřadit a to po směru hodinových ručiček. To provedeme srovnáním sousedů aktuálně zpracovávaného bodu s vertexy, které mají být na hranici. Kromě počátečního bodu dostaneme vždy pouze jeden vertex, který dosud nebyl využit a je průnikem obou těchto množin, a v případě počátečního bodu můžeme jednoznačně rozhodnout, který vertex je ten správný, podle Ypsilonové souřadnice obou favoritů. Postupujeme tedy po hranici, řadíme vhodně do seznamu její body a počítáme uraženou vzdálenost po hranách mezi vertexy. Tato vzdálenost totiž bude hrát klíčovou roli při tvorbě kruhu z hraničních bodů.

Po seřazení bodů a spočtení celkové délky hranice po ní postupujeme a počítáme pro jednotlivé vertexy nové parametrizační pozice. To ilustruje obrázek 3.2. Dosud uraženou vzdálenost po hranici využijeme spolu s její celkovou délkou pro výpočet úhlu, jež svírá právě zpracovávaný bod s bodem počátečním. Tento úhel nám řekne, ve kterém kvadrantu se právě nacházíme a podle toho zvolíme způsob výpočtu obou souřadnic, ve kterém spočtený úhel opět využíváme. Tímto způsobem bude mít parametrizovaná hranice tvar kruhu a vzdálenosti mezi jednotlivými body budou ve stejném poměru jako jejich vzdálenosti na parametrizovaném objektu. Jakmile byly hraniční vertexy parametrizovány, jejich parametrizační pozice je konečná a je tedy uzamčena. Při hledání parametrizačních souřadnic ostatních vertexů pomocí Laplaceova vyhlazování jsou tedy hraniční vertexy přeskočeny.



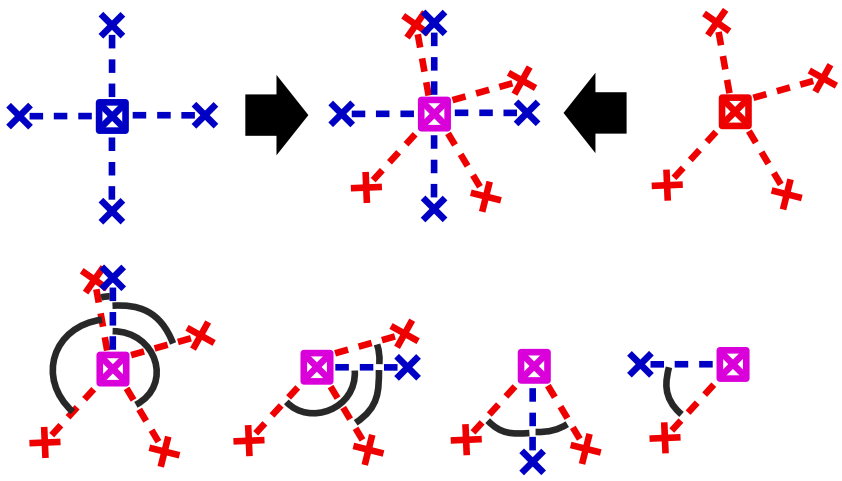
$$
\begin{aligned}
r &= 1 \\
\alpha &= \frac{2\pi * \text{urazena\_vzdalenost}}{\text{celkova\_delka}} \\
&\\
&\text{1. kvadrant :} \\
x_1 &= r * \sin \alpha_1 \\
y_1 &= r * \cos \alpha_1 \\
&\\
&\text{2. kvadrant :} \\
x_2 &= r * \sin(\pi - \alpha_2) \\
y_2 &= r * \cos(\pi - \alpha_2) * (-1)
\end{aligned}
\tag{3.1}
$$

Obrázek 3.2: Příklad výpočtu souřadnic hraničních vertexů. Kružnice je hranicí a x a y jsou souřadnicemi bodů.

3.3 Mapování

Pro vytvoření výsledné animace je nutné pro každý vertex hlavního modelu nalézt novou pozici, která bude odrážet tvar modelu pomocného. Jelikož, jak již bylo zmíněno, se tato práce zaměřuje především na objekty, které mají obdobný počet vertexů, byl zvolen přístup pro mapování, který se snaží vertexy hlavního modelu namapovat na body modelu pomocného. Teprve v okamžik, kdy vertexy již není možné dál párovat se pokusí najít vhodnou novou pozici pro vertex jiným způsobem. Stejný algoritmus je použit i v případě, že uživatel zvolil využití parametrizace na kouli. Jediný rozdíl spočívá v tom, že v takovém případě bude algoritmus vycházet z parametrizovaných pozic vertexů namísto skutečného tvaru modelu.

Při tvorbě mapovacího algoritmu jsem se inspiroval metodou **Manual Landmarks**, kterou ve své práci použil I. A. Sigal et al. [7] a metodou T. Athanasiadise [1] využívající oblasti rysů. V části algoritmu, která mapuje vertexy hlavního modelu na body objektu pomocného začneme s tímto párováním od uživatelem definovaných vazeb. U každého zadaného vertexu pak vytvoříme seznam úhlů, které by svíraly hrany mezi ním a jeho sousedy s obdobnými hranami na modelu pomocném, kdybychom spárované body překryly. Tento stav ilustruje obrázek 3.3. Po spárování dvou bodů jsou oba z výběru odstraněny a postup opakujeme. Tímto způsobem tedy spárujeme vždy takové body, které se z dostupných možností nejvíce pozičně podobají.



Obrázek 3.3: Postup mapování: Body hlavního (modrého) a pomocného (červeného) modelu jsou srovnány (nahoře) a následně jsou párovány vzestupně podle velikosti úhlů, jež svírají jejich hrany (dole).

Vždy, když spárujeme dva body, je nutné si udělat poznámku, který vertex již na pomocném modelu nemáme pro párování používat a který vertex na hlavním modelu už máme namapován a máme se tedy v budoucnosti soustředit na mapování jeho sousedů, jenž touto procedurou ještě neprošly. Body jejichž sousedy je nutné spárovat vybíráme v takovém pořadí, aby se mapování z uživatelem definovaných ohnisek, šířilo napříč celým modelem pokud možno rovnoměrně.

Jakmile není možné dále postupovat s párováním bodů, například protože pomocný model měl méně vertexů než hlavní, přejdeme k druhé části algoritmu. Ta prochází dosud nenamapované body a kontroluje počet jejich již namapovaných sousedů. Pokud takový bod nemá ani jednoho, je prozatím přeskočen. Pokud má právě jednoho, je k němu připnut a pokud má více než jednoho, je umístěn do jejich geometrického středu.

3.4 Animace

Po dokončení mapování máme pro každý vertex hlavního modelu dostupné souřadnice, které odráží tvar pomocného modelu. Návrh této části práce bylo vhodné založit na zvoleném implementačním prostředí a počítal tedy s využíváním *tvarových klíčů*, které jsou vytvořeny po namapování hlavního objektu na pomocný, a následnou prací s nimi. Během ní jsou do snímků na časové ose animace s dostatečnými rozestupy vkládány klíčové snímky, které aktuálním tvarovým klíčům určují váhu s jakou se promítnou na tvar modelu.

Na prvním snímku nově vytvořené animace tedy bude klíč, který určí, že tvarový klíč vycházející z původního tvaru modelu má plnou váhu a tvarový klíč vytvořený z pomocného modelu má váhu nulovou. O jistý časový interval dále se na časové ose vytvoří další klíč s opačnými hodnotami u obou tvarových klíčů a za další časový interval bude klíč totožný s tím prvním, což zajistí cyklickou návaznost animace.

Mezi jednotlivými klíčovými snímky se každý vertex hlavního objektu postupně přesouvá ze své výchozí pozice na pozici, kterou má uloženou v právě animovaném tvarovém klíči, tedy na pozici jež mu byla přidělena při mapování.

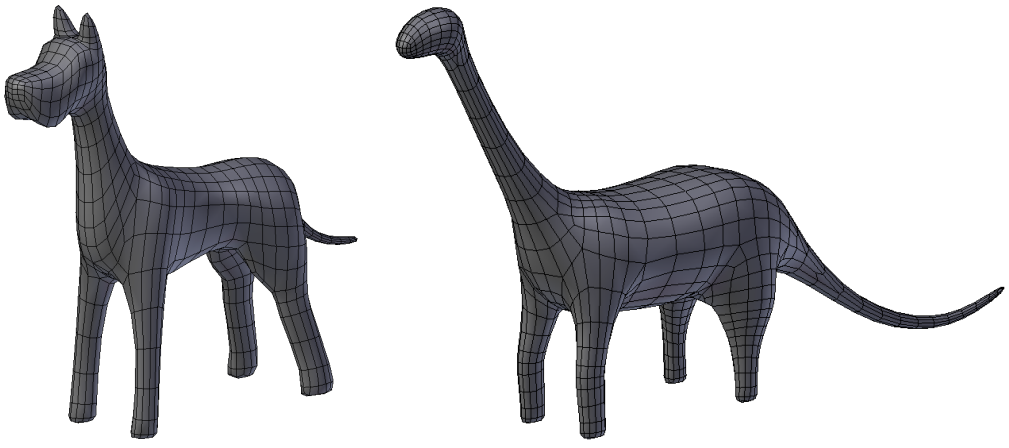
Kapitola 4

Implementace

Tato kapitola popisuje, jaké aspekty návrhu byly vypracovány a jakým způsobem. Projednává konkrétněji provedení některých úkonů a uvádí příklady. Pro vývoj a testování byla vytvořena sada objektů, ze které můžeme vidět několik objektů na obrázku 4.1. Jednotlivé modely se liší buď tvarem, počtem vertexů, nebo obojím a to tak, aby bylo možné experimentovat s použitými metodami a uživatelským vstupem.

Během tvorby nástroje byla využita dokumentace jazyka Python 3 [16] a nesmírnou pomoc poskytnula rovněž dokumentace Blenderu [12] stejně jako program samotný, který je navržen takovým způsobem, aby byl snadno rozšiřitelný, což je například možné demonstrovat na faktu, že pokud uživatel namíří kurzor myši na libovolné tlačítko uživatelského rozhraní, po chvíli se zobrazí název prvku a operátor se svými parametry, jež bude kliknutím spuštěn.

Implementace se drží zásad objektově orientovaného přístupu k programování a tedy vytváří vlastní novou třídu *SKAObject* obsahující odkaz na objekt Blenderu k němuž se vztahuje a definuje nové atributy, kterými jsou například struktury obsahující data o parametrizaci a mapování, a metody, jež veškerou funkčnost zapouzdřují. Rovněž samotné výsledné operátory addonu jsou implementovány jako třídy, jež dědí z tradičních tříd Blenderu, což umožnilo správnou integraci funkčnosti do uživatelského rozhraní.



Obrázek 4.1: Příklad modelů použitých při vývoji nástroje (Pes a Brontosaurus)

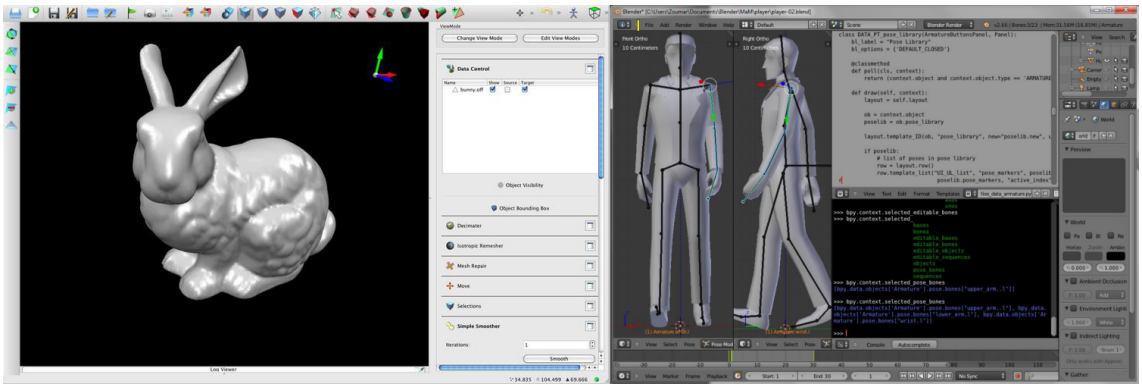
4.1 Vývojové prostředí

Zásadním rozhodnutím, které ovlivnilo celou implementaci byla volba vývojového prostředí. Zdá se mi pouze logické na tomto místě uvést kromě své konečné volby rovněž alternativy, mezi kterými jsem se rozhodoval. Největšími favority se mi jevily knihovna OpenMesh [15] a program Blender [11]. OpenMesh je obecná a efektivní datová struktura pro uložení a práci s polygonálními modely, zatímco Blender je open-sourcový program pro modelování 3D objektů. Po průzkumu svých možností jsem výběr zúžil na následující tři alternativy:

- OpenMesh a OGRE
- OpenFlipper
- Blender

První možnost, která spojuje datovou reprezentaci OpenMeshe s renderovacím enginem OGRE (Object-Oriented Graphics Rendering Engine) [13], by mi poskytnula absolutní kontrolu nad výslednou aplikací. Značnou nevýhodou tohoto řešení je však nadměrná pracnost implementace.

Tento nedostatek značně minimalizuje možnost druhá. OpenFlipper [14] je totiž open-sourcový multiplatformní aplikační a programovací framework navržený pro zpracování, modelování a vykreslení geometrických dat. Vychází z OpenMeshe a umožňuje rozšíření pomocí zásuvných modulů – pluginů. Nabízela se tedy možnost vytvořit plugin do OpenFlipperu.



Obrázek 4.2: Programy OpenFlipper (vlevo) a Blender (vpravo)

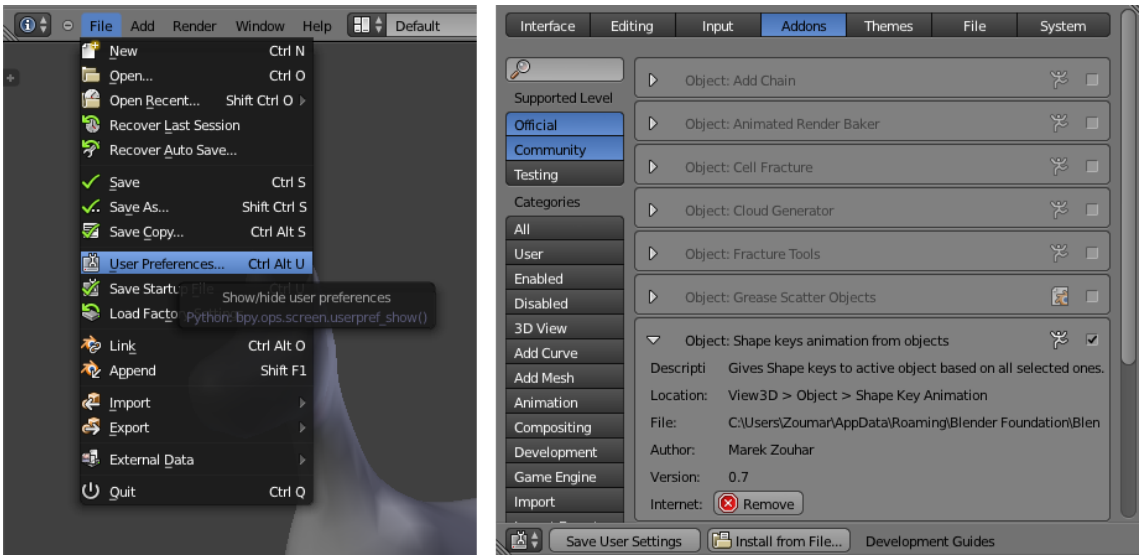
Podobně jako lze rozšířit OpenFlipper, je možné vytvořit zásuvný modul i do Blenderu. (4.2) Blender je velmi snadno rozšiřitelný oficiálními, komunitními, nebo i uživatelskými pluginy ve formě skriptů v jazyce Python 3. Pro tvorbu trojrozměrných modelů poskytuje celou řadu pokročilých operací, ale mimo modelování podporuje rovněž například texturování, tvorbu animací a obsahuje i jednoduchý herní engine a řadu editorů pro úpravu obrázků, videosekvencí a textu. Posledně jmenovaný editor je užitečný zejména kvůli možnosti z něj přímo spouštět Pythonovské skripty pro plynulejší vývoj doplňků, což dále doplňuje možnost Pythonovské konzole. Kolem tohoto přívětivého a mocného nástroje existuje rozsáhlá uživatelská a vývojářská komunita, což by mohlo značně zvýšit význam této práce tím, že by byl výsledný plugin snadno rozšiřitelný a použitelný velkým množstvím uživatelů.

Po zvážení všech faktorů jsem se nakonec rozhodl, že nástroj pro postupnou deformaci modelu vytvořím ve formě doplňku do programu Blender. To mi umožnilo využít

vykreslovací síly jeho renderovacího enginu, struktur do kterých ukládá data, rozšířit jeho uživatelské rozhraní o prvky svého doplňku a nakonec umožnit nástroj jednoduše používat komukoli, kdo by o něj mohl mít zájem.

4.2 Integrace doplňku do programu

Pokud jsou při tvorbě doplňku dodrženy jisté zásady, je možné jej snadno importovat přímo jako Pythonovský skript, nebo .zip archiv. Import doplňku je možné provést otevřením příslušné obrazovky pomocí *File (Soubor)* → *User Preferences (Uživatelská nastavení)*, nebo klávesovou zkratkou *Ctrl + Alt + U*. Následně v záložce *Addons (Doplňky)* je možné dole kliknout na tlačítko *Install from file... (Instalovat ze souboru...)* a dále nalézt požadovaný plugin. Pokud tento proces již uživatel vykonal dříve, bude addon už v seznamu doplňků obsažen. Konkrétně mnou vypracovaný plugin patří do kategorií *Object (Objekt)* a *User (Uživatelské)*. Po vybrání pluginu si můžeme prohlédnout informace o něm a aktivovat ho zakliknutím boxu vpravo. Stejně tak je možné ho stejným způsobem deaktivovat, nebo zcela odebrat ze seznamu kliknutím na *Remove (Odebrat)*. Nastavení uložíme vybráním volby *Save User Settings (Uložit uživatelská nastavení)* dole. Celý postup shrnuje obrázek 4.3.

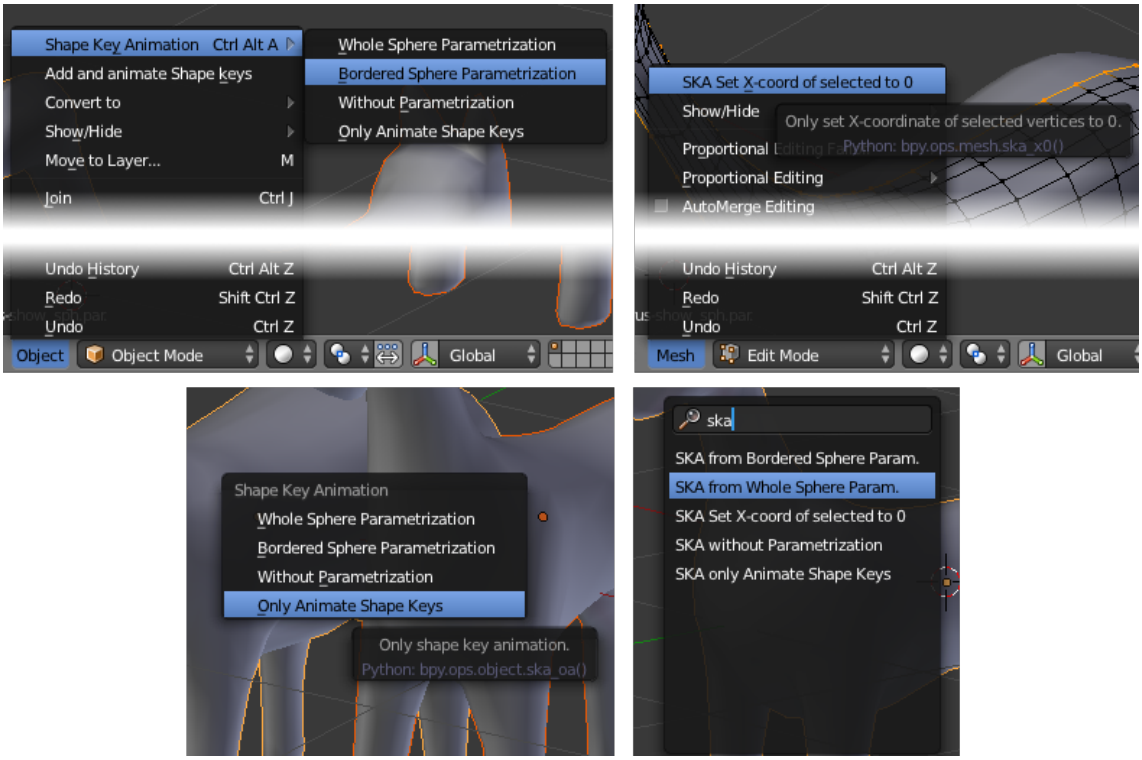


Obrázek 4.3: Načtení a aktivace zásuvného modulu

Aktivací pluginu spustíme automatickou registraci všech jeho komponent. Bylo tedy nutné vytvořit vlastní interní třídu pro každý nově vytvořený operátor (jejichž funkce a implementace bude rozvedena později) a nové podmenu, které tyto operátory zpřístupňuje.

Registrační funkce addonu tedy registruje každý z pěti operátorů i nově vytvořené podmenu, které čtyři z nich zpřístupňuje, a dále rozšíří již existující prvky uživatelského rozhraní jednak o položku podmenu a jednak právě o pátý operátor. Nové podmenu je přístupné v *Objektovém režimu* navigací v menu *Object (Objekt)* → *Shape Key Animation (Animace tvarových klíčů)* a pomocný operátor je dostupný v *Editovacím módu* pod umístěním *Mesh* → *SKA Set X-coord of selected to 0 (SKA nastavit X-ovou souřad. na nulu)*. Poslední akcí, kterou registrační funkce vykoná je vytvoření klávesové zkratky *Ctrl + Alt + A*, která vykreslí nové podmenu a tak operátory zpřístupní komfortnějším způsobem. Po registraci

operátoru je možné jej vyhledat v globálním prostoru všech operátorů. Stačí stisknout *Mezerník* a začít psát název operátoru. Jednotlivé možnosti přístupu jak k funkcím addonu přistoupit shrnuje obrázek 4.4.

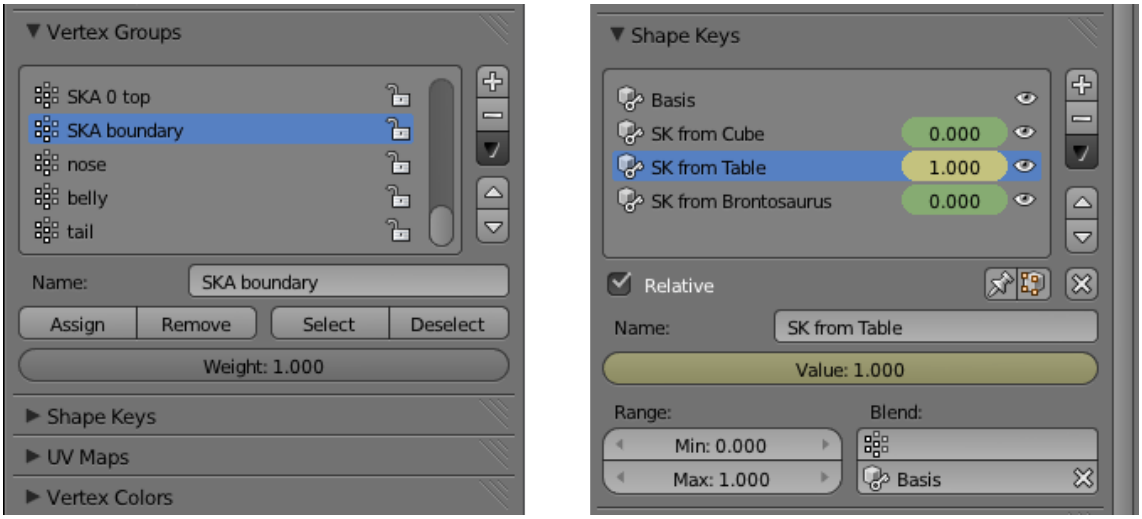


Obrázek 4.4: Integrované podmenu zpřístupňující hlavní operátory (vlevo nahoře); Pomocný operátor integrovaný do existujícího menu (vpravo nahoře); Podmenu otevřené klávesovou zkratkou (vlevo dole); Nalezení nových registrovaných operátorů globálním vyhledáváním (vpravo dole)

4.3 Uživatelský vstup

Definice propojení modelů byla implementována v souladu s návrhem. Uživatel tedy v *Objektovém módu* vybere *pravým tlačítkem myši* se kterým modelem chce pracovat a klávesou *Tab* se přepne do *Editačního módu*, kde může označovat jednotlivé vertexy modelu. V pravé části obrazovky vidí okénko se *Skupinami vertexů (Vertex Groups)* (obrázek 4.5), kde musí pro každou vstupní vazbu vytvořit novou skupinu tlačítkem „+“ a poté v kolonce *Name (Jméno)* skupinu pojmenovat. Následně na modelu označí vertex, který chce do skupiny *Přiřadit (Assign)* a akci samotnou provede kliknutím na příslušné tlačítko. Pokud byl do skupiny přidán více než jeden vertex, nebo pokud do ní byl přiřazen bod omylem, je možné ho odstranit tlačítkem *Remove (Odstranit)*. Vazba mezi hlavním a pomocným modelem bude vytvořena tak, že i v pomocném modelu vytvoříme skupinu vertexů, kterou pojmenujeme totožně a do které vložíme vertex, jež tomu na hlavním modelu odpovídá.

Pokud chce uživatel využít parametrizace na kouli a rád by si definoval vlastní hraniční vertexy, může to provést obdobně jako definoval vazby mezi modely. Vytvoří skupinu vertexů, kterou pojmenuje „SKA boundary“ a vloží do ní všechny vertexy, které mají tvořit



Obrázek 4.5: Okénko se *Skupinami vertexů (Vertex groups)* (vlevo) a s *Tvarovými klíči (Shape Keys)* (vpravo)

hranici. Je důležité aby tvořily uzavřený řetězec. Pro definici „počátku“ této hranice vytvoříme skupinu vertexů s názvem „SKA 0 top“ a vložíme do ní odpovídající jediný vertex, který se rovněž nacházel mezi těmi, které byly přiřazeny do vertexů ve skupině „SKA boundary“. Skupiny vertexů definující vazby mezi modely tedy lze pojmenovat libovolně. Stačí aby byla dodržena zásada zachování stejného názvu odpovídajících si skupin u obou objektů a aby pro tento účel nebyl použit ani jeden ze dvou rezervovaných řetězců „SKA boundary“ a „SKA 0 top“.

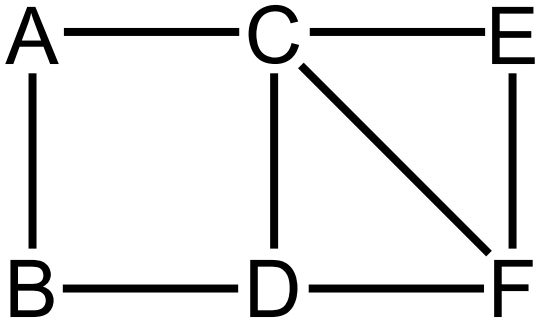
Výběr modelů, ze kterých bude plugin vycházet je vyřešen v souladu s filozofií Blenderu. V *Objektovém módu* uživatel vybere *pravým tlačítkem myši* nejprve pomocný objekt a následně při současném držení klávesy *Shift* klikne opět *pravým tlačítkem myši* na objekt hlavní. Tím vybere oba dva a protože hlavní objekt byl vybrán jako poslední, stane se tzv. *aktivním objektem*. Tento objekt bude deformován podle tvaru modelu pomocného. Po vybrání vstupních modelů je možné spustit některou z hlavních funkcí addonu libovolným výše popsaným způsobem. Pro pohodlnější práci v případě komplikací ještě zmíním, že pokud je ve scéně vybrán alespoň jeden objekt, stisknutím klávesy „A“ se provede odznačení všech objektů.

4.4 Načtení vstupů

Prvním krokem spuštěného skriptu je zpracování vstupních informací zadaných uživatelem. V kontextu 3D scény jsou tedy nalezeny reference na uživatelem označené objekty, přičemž je rozlišen objekt aktivní (poslední vybraný) a objekt pouze vybraný (těch může být více), což vychází z přístupu k výběru objektů programu Blender. Poté se skript soustředí na svázání vertexů pomocí pojmenování *Skupin vertexů*. Vytvoří si tedy seznam všech jmen skupin u obou modelů a nalezne jejich průnik. Tím získá body, které si při mapování mají vzájemně odpovídat, a spárované je uloží do příslušného atributu objektu *SKAObject*.

4.5 Nalezení sousedů vertexů

Při parametrizaci i mapování modelů je klíčové znát sousední body každého vertexu. Během implementace se ale ukázalo, že Blender překvapivě sám o sobě neposkytuje informace o sousedství vertexů. Proto byla implementována metoda (*find_vertex_neighbours()*) pro jejich nalezení a vytvoření pole, které na každém svém indexu, jenž odpovídá indexu vertexu v poli všech vertexů, má další malé pole obsahující indexy vertexů, se kterými vertex reprezentován svým indexem v poli sousedí (viz obrázek 4.6).



Index v poli	0	1	2	3	4	5
Pole vertexů	A	B	C	D	E	F
Pole sousedů	1 2	0 3	0 3 4 5	1 2 5	2 5	2 3 4

Obrázek 4.6: Příklad utvořeného pole sousedních vertexů

Pole se sousedy je tvořeno tak, že postupně od nultého indexu procházíme přes všechny vertexy. Pro každý bod pak proiterujeme všemi hranami modelu, přičemž srovnáváme body, jež spojuje, s indexem právě zpracovávaného vertexu. V případě shody rozšíříme pole sousedů aktuálního bodu o druhý bod hrany. Po proiterování přes všechny vertexy je odkaz na pole sousedů uložen jako atribut objektu. Je pravda, že u rozsáhlejších modelů tato procedura může zabrat delší dobu, ale skutečnosti, že je nutné ji provést pouze jednou a že informace o sousedech jsou nezbytně vyžadovány, tuto nevýhodu plně vyváží.

4.6 Parametrizace

Tři z pěti implementovaných operátorů mají za úkol mimo jiné vytvořit mapování mezi dvěma vstupními objekty. První operátor (*Shape Key Animation (Animace tvarových klíčů)* → *Without Parametrization (bez parametrizace)*) mapování provádí pouze na základě tvaru hlavního a pomocného modelu, druhý operátor (*Shape Key Animation (Animace tvarových klíčů)* → *Whole Sphere Parametrization (parametrizace na kouli)*) před spuštěním mapování oba objekty namapuje na kouli a třetí operátor (*Shape Key Animation (Animace tvarových klíčů)* → *Bordered Sphere Parametrization (parametrizace na kouli s hranicí)*) provede parametrizaci na kouli, při které využije vertexy, které jsou definované jako součást hranice. Ve všech třech případech je mapovací algoritmus totožný, jen vychází z rozdílných dat.

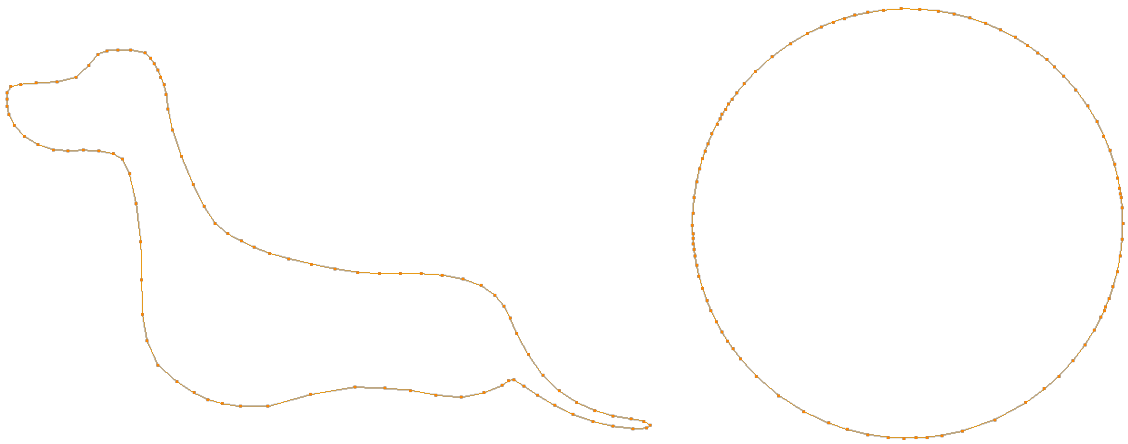
Část vstupních dat pro mapování je uložena v atributu objektu *SKAObject* zvaném *prarametrized (parametrizováno)*. Jedná se o slovník, jehož klíče jsou indexy vertexů, a jehož hodnoty jsou vektory určující parametrizované pozice daného vertexu. Z této struktury mapovací metoda vždy čerpá informace o pozicích bodů, takže i když používáme operátor, který pro mapování nevyužívá parametrizaci, ale dívá se pouze na skutečné pozice bodů, je nutné tyto pozice překopírovat do atributu *parametrized* (metoda *parametrize_onto_self()*).

Parametrizace na kouli je implementována podle návrhu v kapitole 3.2 a tak probíhá mírně odlišně podle toho, zda využíváme uzamčenou kruhovou hranici. Pokud ano, je nej-

prve třeba nalézt bod označovaný jako „**SKA 0 top**“ (metoda *find_boundary_start()*). Ten může být definován uživatelem pomocí *Skupin vertexů*, ale pokud chce uživatel využít automatického nalezení hraničních vertexů (podle jejich nulové X-ové souřadnice), je jeho explicitní definice dobrovolná. V takovém případě jsou ze všech bodů modelu vybrány ty s nulovou X-ovou souřadnicí a mezi nimi je nalezeno pár vertexů nejbližších ose Y (co nejnižší Ypsilonová souřadnice). Z těch je pak zvolen ten nejvýše položený (co nejvyšší Zetová souřadnice). Ať už jsme výsledný bod získali zmíněným algoritmem anebo ho dostali zadaný od uživatele, jeho index je uložen jako atribut objektu *SKAObject*.

Jakmile je nalezen „začátek“ kruhu, pokračujeme hledáním **hraničních bodů** (metoda *find_boundary()*). Při tom se orientujeme buď podle X-ové souřadnice podobně jako při hledání počátku, nebo, pokud je zadána, pomocí skupiny vertexů „**SKA boundary**“. Vzhledem k příští části algoritmu parametrizace je v souladu s návrhem v kapitole 3.2 žádoucí, aby byly hraniční body uloženy ve smysluplném pořadí. Postupujeme tedy od počátečního bodu po směru hodinových ručiček a stavíme pole hraničních bodů, které se stane atributem našeho objektu, a současně počítáme uraženou vzdálenost po hranách mezi vertexy. Tuto vzdálenost si po dokončení uložíme pro pozdější použití neboť, jak je podrobně popsáno v kapitole 3.2 je tato informace klíčová pro výpočet parametrizačních souřadnic hraničních vertexů. Dalším krokem je **sestavení kruhové hranice** (metoda *parametrize_boundary()*). Ten byl proveden přesně podle návrhu, takže z celkové délky hranice a dosud uražené vzdálenosti po ní počítáme úhel, který je využíván při výpočtu souřadnic každého vertexu.

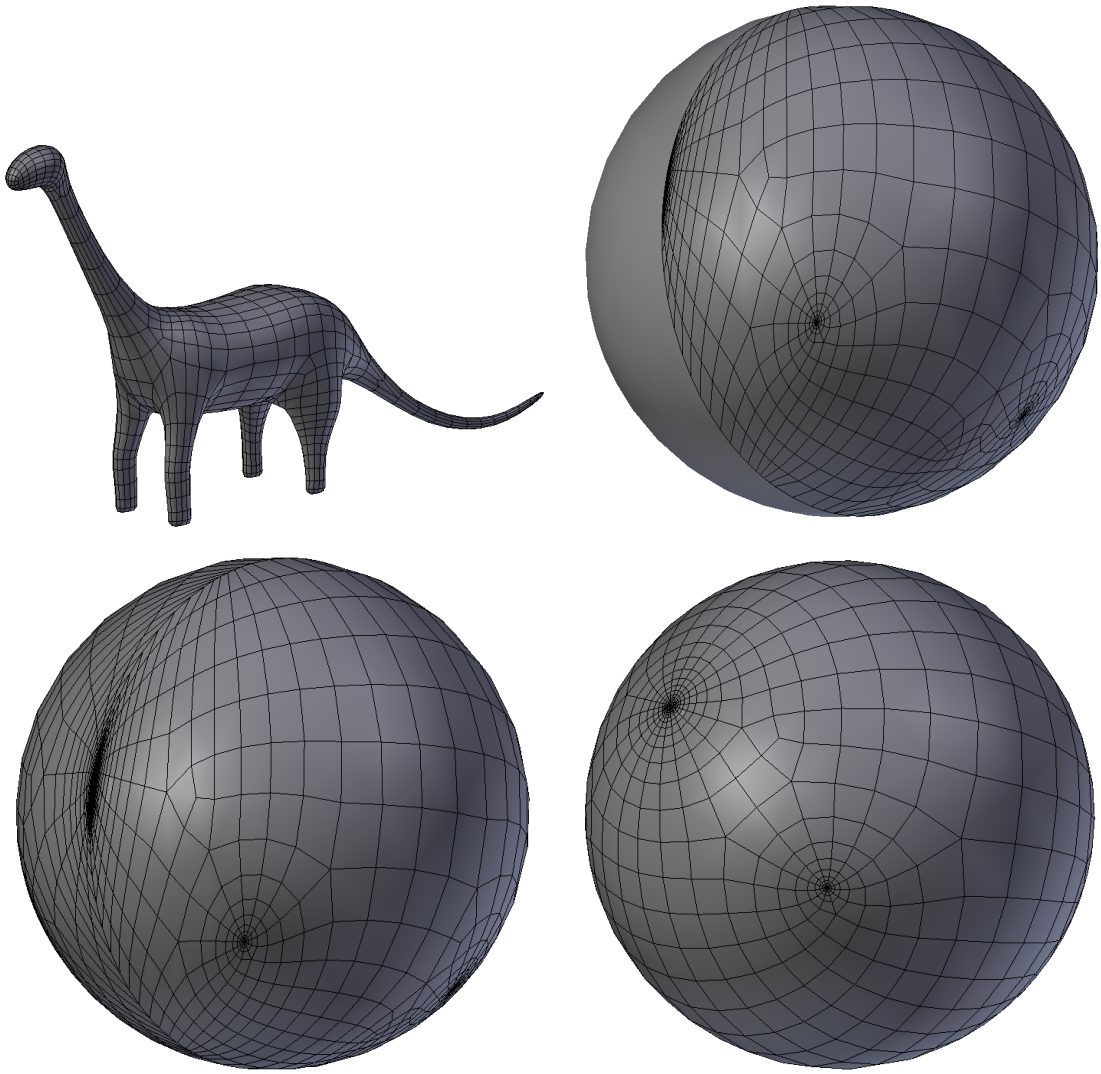
Jak již bylo zmíněno, použití parametrizace s hranicí může být žádoucí i z toho důvodu, že modely často mívají na pomyslné hranici poměrně komplikované převisy, které by se při pouhém promítnutí na kouli značně překrývaly a Laplaceovo vyhlazování by pak bylo nutné provést v podstatně více iteracích. Na příkladu v obrázku 4.7 jsou taková problematická místa zejména hlava a ocas pejska.



Obrázek 4.7: Příklad vytvoření kruhu z hraničních vektorů se zachováním poměru délek

Jakmile byly hraniční vertexy parametrizovány, jejich parametrizační pozice je konečná a je tedy uzamčena. Například při cyklickém průchodu přes všechny body modelu za účelem odstranění překryvů jsou tedy tyto vertexy přeskočeny. Výše popsané úkony se, jak bylo zmíněno, provádí pouze pokud uživatel zadal práci s kruhovou hranicí. V každém případě jádro parametrizace spočívá v mnohonásobně vykonávaném cyklu, během něhož se provádí **Laplaceovo vyhlazování**. Před tím, než je spuštěno, jsou všechny vertexy promítnuty

na kouli. Samotné vyhlazování probíhá tak, že procházíme v mnoha cyklech všemi vertexy a pro každý spočteme novou parametrizační polohu jako geometrický střed jeho sousedních vertexů. Stačí tedy sečíst vektory představující jejich souřadnice a každou složku výsledného vektoru pak podělit počtem sousedů. Počet cyklů se odvíjí od složitosti modelu a může řádově dosáhnout desetitisíců cyklů. Tato fáze je tedy značně výpočetně náročná a může trvat pro velmi složité modely a pomalejší stroje i několik minut.



Obrázek 4.8: Příklad parametrizace modelu: Výchozí model (vlevo nahoře); Parametrizace na polokouli při použitém modifikátoru zrcadlení modelu (vpravo nahoře); Parametrizace na kouli s využitím hranice (vlevo dole); Prostá parametrizace na kouli (vpravo dole)

Příklady výsledků parametrizace si můžeme prohlédnout na obrázku 4.8. Parametrizace bez využití hranice sice na pohled možná vypadá nejpříjemněji, ale využití parametrizační hranice často přináší kvalitnější výsledky. Pro jistotu ještě zdůrazním, že parametrizace modelu nemá žádný skutečný vliv na tvar modelu. Veškeré výpočty jsou prováděny pouze interně a tvar modelu je modifikován až v úplně poslední fázi při přehrávání výsledné animace, po spárování vstupních vertexů, parametrizaci, namapování obou modelů a vytvoření animace.

4.7 Mapování

Mapovací algoritmus je implementován v souladu s návrhem, probíhá tedy ve dvou fázích. V první se snaží spárovat vertexy hlavního a pomocného modelu a ve druhé hledá vhodné pozice bodům, které se spárovat nepodařilo.

Před první fází jsou nejprve namapovány vertexy definované uživatelem pomocí *Skupin vertexů*. Záznamy o mapování jsou ukládány do atributu objektu *SKAObject* a mají podobu slovníku, ve kterém jsou klíči indexy vertexů hlavního objektu a hodnotou je buď odkaz na vertex pomocného modelu, na nějž je vertex namapován, nebo vektor představující souřadnici, na níž byl bod namapován v případě, že nebyl nalezen ekvivalentní bod v pomocném objektu. Před spuštěním cyklu **první fáze** jsou také vytvořeny dvě pole, přičemž jedno obsahuje indexy vertexů pomocného modelu, které už byly využity k mapování, a druhé indexy vertexů hlavního modelu určené ke zpracování. Těmi jsou před spuštěním pouze uživatelem zadané body. Ty jsou sice již namapovány, ale v tomto kontextu se zpracováním myslí namapování jejich sousedů. Jakmile je vše připraveno, je možné spustit mapovací cyklus.

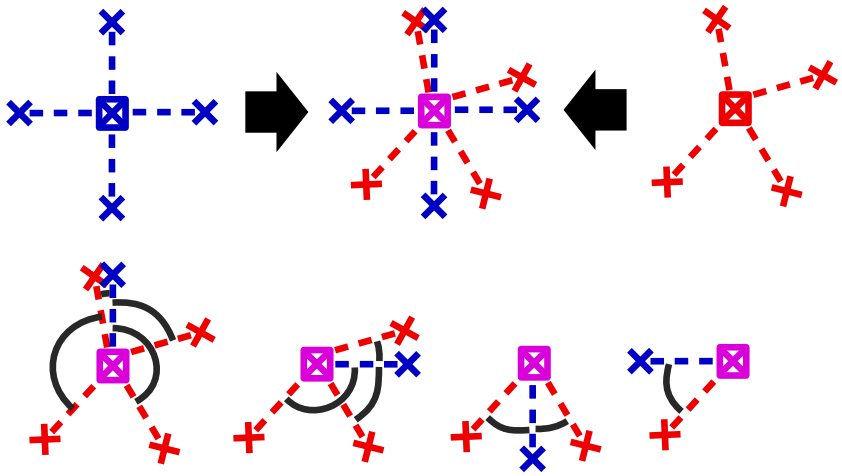
V každém průchodu cyklem se stane aktivním vertexem ten, který je v poli bodů pro zpracování na nejmenším indexu a ten je rovnou odstraněn ze seznamu bodů pro zpracování. Je vytvořeno pomocné pole s indexy sousedů tohoto bodu, ze kterého jsou případně odstraněny vertexy, které již byly namapovány. Dále nalezneme vertex, na který je aktuálně zpracovávaný bod namapován, a vytvoříme obdobné pomocné pole i pro něj. Budou v něm tedy indexy vertexů pomocného modelu a budou odstraněny body, na něž již byl některý vertex hlavního objektu namapován. Pamatujme, že toto je možné, protože zpracovávané vertexy již byly namapovány (ty první přímo vstupem uživatele) a protože si udržujeme přehled o již využitých bodech v obou objektech.

Další krok je o něco komplikovanější. Jsou totiž vytvořeny dvě skupiny vektorů, které představují hrany vedoucí z právě zpracovávaného vertexu do jeho dosud nenamapovaných sousedů (respektive hrany vedoucí z bodu, na nějž byl aktuální vertex namapován, do jeho sousedů ještě nevyužitých pro namapování). Poté jsou vektory v obou skupinách normalizovány, neboli jejich délka je nastavena na hodnotu jedna a to bez ovlivnění jejich orientace, abychom dále nebyli svedeni z pravé cesty délkou vektoru, ale ohlíželi se pouze na úhly, jež vzájemně svírají.

Následně jsou vypočteny vzájemné vzdálenosti mezi konci všech vektorů obou skupin, přičemž je uloženo, která vzdálenost odpovídá které dvojici vektorů. Pro větší názornost znovu uvádím obrázek 4.9. Mezi všemi vzdálenostmi je následně nalezena ta nejmenší a odpovídající vertexy obou modelů jsou spárovány. Vzdálenost mezi vektory poměrně přesně zastupuje velikost úhlu mezi nimi, protože vektory byly normalizovány.

Při párování jsou ze struktury vzdáleností mezi vektory odstraněny všechny vstupy, kde figuroval alespoň jeden z obou vertexů. Dále je nově spárovaný bod hlavního objektu přidán na konec pole s vertexy určenými pro zpracování (takže v budoucnu budeme párovat jeho sousedy) a nově spárovaný vertex pomocného objektu je přidán do pole vertexů, které již byly využity k mapování. Samozřejmostí je vytvoření záznamu do atributu objektu *SKAObject*, který schraňuje mapování bodů.

Dalším krokem je opětovné nalezení nejmenší vzdálenosti mezi vektory obou skupin. Tentokrát dostaneme jiný výsledek, neboť jsme právě spárované vertexy ze struktury obsahující vzdálenosti vektorů odstranili. Tímto způsobem postupujeme, dokud jsou ve struktuře nějaké hodnoty. Jakmile je vyčerpáme, můžeme se pustit do dalšího cyklu první fáze a zpracovat tedy další vertex. Tím, že body zpracováváme z jednoho konce pole a nově na-



Obrázek 4.9: Postup mapování: Body hlavního (modrého) a pomocného (červeného) modelu jsou srovnány (nahoře) a následně jsou párovány vzestupně podle velikosti úhlů, jež svírají jejich hrany (dole).

mapované vertexy k dalšímu zpracování přidáváme na konec druhý, zaručujeme postupné a rovnoměrné mapování bodů.

V okamžik, kdy se pole obsahující vertexy ke zpracování vyprázdní a tedy doběhne poslední cyklus první fáze, můžeme přistoupit k **fázi druhé**, kde budeme mapovat zbylé body. Ty už nemůžeme mapovat na vertexy pomocného modelu a tak si budeme muset vystačit s modelem hlavním. Nejprve tedy vytvoříme seznam všech bodů, které ještě nebyly namapovány a dále se můžeme pustit do cyklu, jež se bude opakovat, dokud tento seznam bude obsahovat nějaké prvky.

Cyklus prochází všechny body a počítá kolik sousedů aktuálního vertexu již bylo namapováno. Pokud nebyl namapován žádný soused, je bod prozatím přeskočen, pokud bylo namapováno několik sousedů, je bod umístěn do jejich geometrického středu, a pokud byl namapován právě jeden soused, je k němu aktuální vertex přichycen. Vzhledem k potenciální nekonečnosti tohoto cyklu v případě, že by některé vertexy nebyly spojeny se zbytkem modelu, byl implementován čítač cyklů, který tuto fázi ukončí v případě, že bychom se ocitli v nekonečné smyčce.

4.8 Animace

Po dokončení mapování skript přistoupí ke tvorbě animace. Nejprve tedy z časové osy odstraní klíčové snímky případné minulé animace a přistoupí ke tvorbě *tvarového klíče* ze struktury s mapovacími souřadnicemi. Vytvoří tedy prázdný tvarový klíč a spustí cyklus, v němž projde všemi vertexy a pro každý z nich v klíči vytvoří záznam o nové pozici. Nově tvořený tvarový klíč je pojmenován podle formule „SK object_name“, kde object_name je název objektu, na který jsme aktivní objekt mapovali, a SK je zkratka pro *Skape key (Tvarový klíč)*. Kromě tvarového klíče vytvořeného podle mapovacích souřadnic všech bodů, ale rovněž vyrobí další klíč pojmenovaný „Basis“, který je výchozím tvarem objektu a který bude velmi užitečný při tvorbě animace samotné.

Pokud je při tvorbě tvarového klíče nalezen klíč se stejným názvem, znamená to zřejmě, že jsme již stejný model pro deformaci tvaru dříve použili, a tak nově tvořený tvarový klíč ten starý aktualizuje (přepíše).

Jakmile jsou vytvořeny tvarové klíče a je vyčištěna časová osa započne tvorba animace samotné. Začneme tedy procházet seznamem všech tvarových klíčů a pro každý vytvoříme celkem tři klíčové snímky, mezi kterými je vždy stejný rozestup. První klíčový snímek každého tvarového klíče určuje, že jím nejsou pozice bodů modelu ovlivněny. Druhý naopak vynutí použití svých hodnot na souřadnice vertexů no a třetí opět vrací body do původní polohy. Klíčové snímky všech tvarových klíčů jsou rozmístěny tak, aby ve výsledné animaci se model z původního tvaru deformoval na svůj první tvarový klíč, poté změny navrátil do původního stavu a následně stejný proces proběl s dalším tvarovým klíčem. To je ostatně také důvod proč jsme při přípravě tvarových klíčů vytvořili jeden (pojmenovaný „Basis“), který odráží výchozí pozice všech bodů. Pokud vytvoříme animaci ze tří tvarových klíčů, bude její časová osa vypadat jako na obrázku 4.10.

Obrázek 4.10: Časová osa po vytvoření animace ze tří tvarových klíčů

Tento přístup umožňuje, abychom nejprve pomocí doplňku vytvořili z několika různých modelů vytvořili několik mapování a tvarových klíčů a poté vytvořili animaci ze všech najednou. Ostatně operátor vytvářející pouze animaci z tvarových klíčů je dostupný v *Objektovém režimu* v menu *Object (Objekt)* → *Shape Key Animation (Animace tvarových klíčů)* → *Only Animate Shape Keys (Pouze animovat tvarové klíče)*. Připomínám, že k tomuto operátoru je možné rovněž přistoupit pomocí klávesové zkratky *Ctrl + Alt + U* nebo vyhledáním jeho názvu (*SKA only Animate Shape Keys*) v databázi všech operátorů po stisknutí *Mezerníku*.

Několik příkladů výsledné interpolace tvaru můžeme vidět v příloze C.

4.9 Pomocné funkce

Zbývá už jen popsat pár posledních implementovaných funkcí, z nichž je rozhodně nejpodstatnější operátor *SKA Set X-coord of selected to 0 (SKA nastavit X-ovou souřad. na nulu)*, který je dostupný v *Editačním módu* v menu *Mesh* → *SKA Set X-coord of selected to 0 (SKA nastavit X-ovou souřad. na nulu)*. Jak bylo zmíněno výše tato funkce proiteruje všemi vertexy modelu a u těch, které jsou ve scéně vybrány, vynuluje X-ovou složku jejich souřadnic. Tento nástroj je užitečný zejména pokud používáme modifikátor zrcadlení podle osy X nebo chceme využít možnosti automatického nalezení hranice pro parametrizaci na kouli.

Další funkce, které byly implementovány, ale v současném řešení nejsou využity, slouží k převodu všech stěn modelu na trojúhelníky (*triangulate()*) takovým způsobem, aby při běhu skriptu byla tato změna navratitelná (*detriangulate()*). Tyto metody byly používány při experimentování s mapovacím algoritmem, ale nepřispěly ke zkvalitnění výsledků.

Poslední metody, které zmíním, měly spíše význam při vývoji tohoto doplňku Blenderu a hledání chyb v něm. Jedná se o procedury, které promítnou výsledky mapování (*show_mapping()*), nebo dokonce parametrizace (*show_parametrization()*) přímo do tvaru modelu. To při finálním běhu skriptu pochopitelně není žádoucí, ale řízené používání těchto metod umožnilo rychlejší a dynamičtější vývoj.

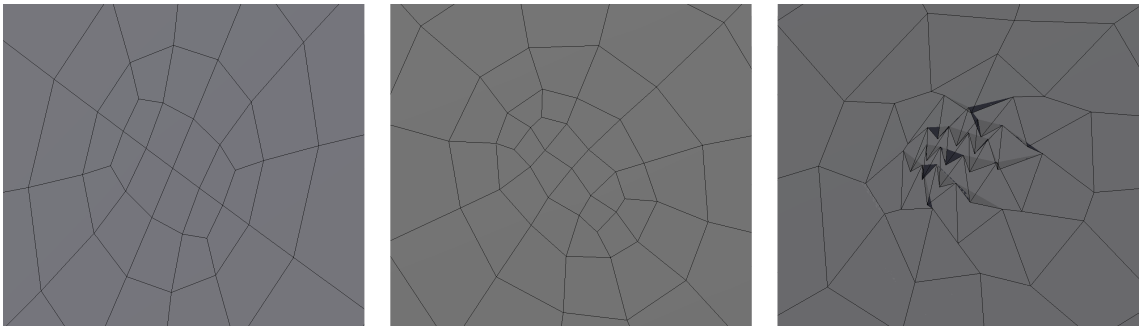
Kapitola 5

Výsledky

Při implementaci a tvorbě algoritmu byly prováděny různé experimenty, které se snažily zlepšit jeho jednotlivé části. Tato kapitola popíše ty, které největším dílem přispěly ke zkvalitnění výsledků a zmíní známé problémy, které při nevhodných vstupech můžou nastat.

5.1 Experimenty s parametrizací

Při implementaci Laplaceova vyhlazování se ukázalo, že pokud je zvolena parametrizace na kouli bez využití hraničních bodů, může po provedení mnoha tisíců cyklů dojít k mírnému pootočení parametrizace. Docházelo k tomu z toho důvodu, že nově spočtené parametrizační pozice vertexů byly okamžitě ukládány do stejné struktury, ze které byly čerpány informace pro vyhlazování. Během experimentování s řešením, kdy jsem nově spočtené pozice ukládal stranou a aktualizoval všechny najednou až po dokončení průchodu přes všechny vertexy, se ukázalo, že při použití tohoto postupu „globální aktualizace“ může po velmi vysokém počtu cyklů dojít ke tvorbě artefaktů, které můžeme vidět na obrázku 5.1.



Obrázek 5.1: Příklad znehodnocování parametrizace: Ideální případ (vlevo); Počínající problém (uprostřed); Značné zkreslení (vpravo)

Tyto nepřesnosti byly s největší pravděpodobností způsobeny tím, že po několika tisících cyklech se vzdálenosti, o něž se měla parametrizační pozice vertexu změnit, se přiblížila, či dokonce přesáhla, vzdálenost, jaká je mezi sousedními body. K tomuto jevu docházelo především po mnoha tisících cyklech, neboť místa s hustějším osázením vertexy se začínají shlukovat a vzdálenosti mezi vertexy se tedy značně zmenšují, což pro vznik výše zmíněného artefaktu vytváří příznivější podmínky.

Z důvodu výskytu těchto nepřesností nepřicházelo v úvahu „globální aktualizace“ používat a nové parametrizační pozice jsou tedy ukládány okamžitě in situ. To ve výsledku však prakticky nevadí, neboť jediný problém, který toto řešení způsobovalo, bylo pootočení celé parametrizace modelu o pouhých pár stupňů a to pouze po mnoha tisících provedených cyklech, to vše navíc pouze v případě, že byla použita parametrizace na kouli bez použití kruhové hranice. Při jejím využití jsou totiž parametrizační pozice hraničních vertexů uzamčeny a tak pootáčení není umožněno.

Při experimentování s řešením byla také při parametrizaci na kouli bez užití hranice nalezena a několikrát úspěšně reprodukována další komplikace. Při mnohatisícovém cyklení vyhlazováním modelu se totiž mohlo stát, že se celá parametrizace vlivem drobných posunů všech vertexů přemístí poměrně daleko od relativní nulové souřadnice[1] modelu a tak při promítnutí bodů na kouli po nalezení geometrického středu sousedních vertexů dojde k tomu, že se všechny body nachází pouze na jedné polokouli. To má pochopitelně fatální následky. Naštěstí je tomu možné snadno předejít a to tak, že pokud byla uživatelem zvolena parametrizace bez použití hranice, provedeme při cyklení v jistých intervalech následující proceduru: nalezneme geometrický střed všech vertexů modelu (což bude prakticky střed koule, vzhledem k tomu, že již probíhala parametrizace), spočteme vektor, mezi tímto bodem a relativní nulou a následně posuneme každý vertex o vektor opačný tomu spočítanému. Tím si můžeme být jisti, že parametrizace si zachová geometrický střed a ke všem artefaktům tohoto typu pak může docházet už jedině z důvodů abnormálně asymetrického rozprostření bodů modelu nebo jeho chybného počátečního umístění vzhledem k relativní nule.

5.2 Experimenty s mapováním

Při implementaci mapovací funkce byly prováděny experimenty s počtem sousedů vertexů během první i druhé fáze. Byly implementovány metody pro převedení objektu na model složený z výhradně trojúhelníkových polygonů (narozdíl od čtyřúhelníkových typických pro ručně modelované objekty) a zpět. To zvýšilo počet sousedů každého vertexu v průměru o 50 %. Podobně byl algoritmus rozšířen o možnost, kdy mezi sousedy bodu počítal i vertexy, které byly společné pro alespoň dva (prakticky vždy však pro právě dva) jeho skutečné sousedy, což počet vertexů na něž se algoritmus ohlížel v podstatě zdvojnásobilo. Ukázalo se však, že tyto přístupy nezvýšily kvalitu výsledků algoritmu, naopak spíše ve většině případů vedly k méně přesným mapováním, což je vzhledem k principu na němž algoritmus pracuje poměrně očekávatelný výsledek. Je totiž smysluplné aby mapování bylo přesnější, pokud při párování má algoritmus méně možností jak vertexy svázat, a tedy má jednoduše řečeno méně příležitostí udělat chybu.

[1] Relativní nulou je zde myšlena souřadnice [0, 0, 0] při upravování geometrie modelu – tedy v *Editovacím módu*, na rozdíl od nuly absolutní, vzhledem k níž jsou pozicovány jednotlivé objekty ve scéně, tedy v *Objektovém módu*

5.3 Shrnutí výsledků

I poté, co jsem své postupy modifikoval na základě experimentů popsaných výše, mapování pochopitelně nemusí vždy proběhnout podle našich plánů nebo očekávání. To nastává zejména v případě, že nebyly vhodně definovány vstupní vazby, byl uživatelem zvolen nevhodný způsob parametrizace, nebo se v některých částech modelu značně lišila hustota vertexů. Příklad těchto problémů ilustruje obrázek 5.2. Na ocase si můžeme všimnout případu, kdy na hlavním modelu nebyl dostatek vertexů pro plynulé mapování. Na stehně zase můžeme vidět příklad nedostatečného provázání pomocí vstupních definic vazeb. Jistě k nevhodnému namapování v tomto místě přispělo i nesprávné šíření párování vertexů.

Obrázek 5.2: Příklad nepříliš povedeného mapování (uprostřed) psa (vlevo) na brontosaura (vpravo)

Přední nohy se viditelně pokouší příliš brzy napojit na trup a v jejich horní části je do očí bijící zub. Velkou trhlinu můžeme pozorovat i nad pánví a drobné nesrovnalosti jsou vidět i na hlavě. Vina je ale samozřejmě i na straně skriptu. Dokonalý algoritmus by se se všemi těmito nedostatky dokázal vypořádat. Jako možné budoucí rozšíření bych tedy navrhnul několik úprav, které rozvedu v závěru práce.

Navzdory tomu můj postup dával dobré výsledky pro objekty spadající do zvolené podmožiny trojrozměrných modelů. Příklady úspěšné parametrizace jsou k vidění výše na obrázku 4.8 a úspěšného namapování objektů a jejich následné animace jsou zejémna v příloze C.

Kapitola 6

Závěr

Tato práce se zabývala tvorbou nástroje ve formě doplňku do populárního modelovacího programu Blender s otevřeným zdrojovým kódem, který zpracuje uživatelem zadané modely a vytvoří animaci změny tvaru jednoho modelu na tvar druhého. Především vzhledem k omezenému času se plugin soustředí především na modely o podobném počtu vertexů, přičemž se snaží řešení rozšířit i na komplexnější vstupní modely, a tak bylo zadání splněno.

Pro modely s výrazně odlišným tvarem byla implementována možnost parametrizace na kouli s Laplaceovým vyhlazováním. Mapovací algoritmus se inspiruje metodou Manual Landmarks a Feature Regions a vzhledem k možnému využití parametrizace také metodou Automated Wrapping. Doplněk dává dobré výsledky pro zvolenou podskupinu problémů, ale právě proto vidím mnoho příležitostí jak by jej bylo možné v budoucnosti rozšířit a vylepšit.

Vcelku jednoduchá, ale velmi mocná modifikace algoritmu mapování spočívá v možnosti definice toho, jakou váhu by měla každá uživatelem definovaná vstupní vazba mít při mapování. Během šíření párování z těchto uživatelem definovaných ohnisek bychom potom mohli řídit, které rozšiřující se skupiny mají postupovat rychleji a mít prioritu. Tento přístup by také umožnil velice precizně nadefinovat velké množství různě hustě rozmístěných vstupních vazeb.

Další, o něco málo složitější, možná úprava by spočívala v možnosti přeskočit některé vertexy v místech, kde je výrazně vyšší hustota bodů na hlavním modelu, a podobně také přeskočit některé vertexy pomocného modelu, pokud by nastala opačná situace. Rozhodování o tom, zda je při postupování po hranách modelu nutné některý vertex přeskočit, by mohlo probíhat na základě měření vzdálenosti mezi sousedy vertexů a sousedy jejich sousedů... Je však možné, že by (přinejmenším při nedokonalém zpracování této myšlenky) takováto modifikace vedla spíše k horším a v negativním slova smyslu překvapivějším výsledkům.

Třetí navrhovaná úprava by vytvořila vnitřní kopii pomocného modelu a pokusila by se vhodně modifikovat počty vertexů a to buď globálně nebo spíše pouze ve vhodných místech tak, aby bylo možné hlavní model lépe namapovat.

Mým posledním návrhem na úpravu addonu je ho mírně přepracovat tak, aby třída *SKAObject* nebyla nadstavbou nad standardním objektem Blenderu, ale aby tento interní objekt byl rozšířen o všechny atributy a metody nové třídy. To by umožnilo mít interně uloženou parametrizaci bez nutnosti ji počítat znovu při každém spuštění.

Literatura

[1] Athanasiadis, T.; Fudosand, I.; Nikou, C.; aj.: Feature-based 3D morphing based on geometrically constrained spherical parameterization. *Computer Aided Geometric Design*, ročník 29, č. 1, leden 2012: s. 2 – 17, dostupné z `http://www.sciencedirect.com/science/article/pii/S0167839611001075`.

[2] Biancolini, M. E.; Viola, I. M.; Riotte, M.: Sails trim optimisation using CFD and RBF mesh morphing. *Computers & Fluids*, ročník 93, č. 10, duben 2014: s. 46 – 60, dostupné z `http://www.sciencedirect.com/science/article/pii/S0045793014000140`.

[3] Hartmannová, V.; Hartmannová, D.; J.Doleželová; aj.: *Pravidla českého pravopisu*. Brno: Fin publishing, třetí vydání, 1997, ISBN 80-86002-10-1.

[4] Rábová, Z.; Hanáček, P.; Peringer, P.; aj.: Užitečné rady pro psaní odborného textu [online]. `http://www.fit.vutbr.cz/info/statnice/psani_textu.html`, 2008-11-01 [cit. 2008-11-28].

[5] Salo, Z.; Beek, M.; Wright, D.; aj.: Computed tomography landmark-based semi-automated mesh morphing and mapping techniques: Generation of patient specific models of the human pelvis without segmentation. *Journal of Biomechanics*, ročník 48, č. 6, duben 2015: s. 1125 – 1132, dostupné z `http://www.jbiomech.com/article/S0021-9290(15)00025-1/fulltext`.

[6] Shen, L.; Makedon, F.: Spherical mapping for processing of 3D closed surfaces. *Image and Vision Computing*, ročník 24, č. 7, červenec 2006: s. 734 – 761, dostupné z `http://www.sciencedirect.com/science/article/pii/S0262885606000485`.

[7] Sigal, I. A.; Hardisty, M. R.; Whyne, C. M.: Mesh-morphing algorithms for specimen-specific finite element modeling. *Journal of Biomechanics*, ročník 41, č. 7, červenec 2008: s. 1381 – 1389, dostupné z `http://www.sciencedirect.com/science/article/pii/S0021929008000882`.

[8] Wang, C.; Liu, Z.; Liu, L.: As-rigid-as-possible spherical parametrization. *Graphical Models*, ročník 76, č. 5, září 2014: s. 457 – 467, dostupné z `http://www.sciencedirect.com/science/article/pii/S152407031400023X`.

[9] Watt, A.: *3D Computer Graphics*. Addison Wesley, třetí vydání, 2000, ISBN 978-0-201-39855-7.

[10] Weistrand, O.: Parameterizations of digital surfaces homeomorphic to a sphere using discrete harmonic functions. *Pattern Recognition Letters*, ročník 27, č. 16, prosinec

2006: s. 1934 – 1941, dostupné z
http://www.sciencedirect.com/science/article/pii/S0167865506001541.

[11] WWW stránky: Blender. https://www.blender.org.

[12] WWW stránky: Blender Documentation contents.
http://www.blender.org/api/blender_python_api_2_60a_release/.

[13] WWW stránky: OGRE – Object-Oriented Graphics Rendering Engine.
http://www.ogre3d.org/.

[14] WWW stránky: OpenFlipper. http://www.openflipper.org.

[15] WWW stránky: OpenMesh. http://www.openmesh.org.

[16] WWW stránky: Python Documentation. https://docs.python.org/3/.

[17] Ying, L.; Zhou-Wang, Y.; Jian-Song, D.: Spherical Parameterization of Genus-Zero Meshes Using the Lagrange-Newton Method. *Journal of Software*, ročník 18, prosinec 2007: s. 8 – 17, ISSN 1000-9825, dostupné z
http://www.jos.org.cn/1000-9825/18/s8.htm.

Dodatek A

Obsah CD

- Adresář „Blender“ obsahující
  - blender-2.74-linux-glibc211-i686-32bit.tar.bz2
    Instalační soubor Blenderu pro 32bit Linux (vyžaduje glibc 2.11)
  - blender-2.74-linux-glibc211-x86_64-64bit.tar.bz2
    Instalační soubor Blenderu pro 64bit Linux (vyžaduje glibc 2.11)
  - blender-2.74-OSX_10.6-x86_64.zip
    Instalační soubor Blenderu pro OSX (vyžaduje Mac OS X 10.6+)
  - blender-2.74-windows32.exe
    Instalační soubor Blenderu pro 32bit Windows 8/ 7/ Vista
  - blender-2.74-windows64.exe
    Instalační soubor Blenderu pro 64bit Windows 8/ 7/ Vista
- Adresář LaTeX obsahujici veškeré soubory pro překlad Technické zprávy (preložitelné pomocí „make“ na školním serveru Merlin)
- Bakalarska_prace.pdf
  Technická zpráva bakalářské práce
- Manual.txt
  Návod pro zprovoznění práce
- Obsah CD.txt
  Soubor popisující obsah CD
- Plakat.pdf
  Plakát prezentující práci
- Scena.blend
  Soubor programu Blender obsahující scénu pro příklad práce skriptu
- ShapeKeyAnimation.py
  Výsledný skript Bakalářské práce – k načtení do Blenderu jako plugin

Dodatek B

Manuál

B.1 Aktivace doplňku

- Nainstalovat Blender ze složky *Blender* na CD podle cílové platformy
- Otevřít scénu *Scena.blend* v Blenderu
- Aktivovat plugin pomocí
  - *File (Soubor)* → *User Preferences (Uživatelská nastavení)*, nebo klávesovou zkratkou *Ctrl + Alt + U*
  - Vybrat záložku *Addons (Doplňky)*
  - Dole kliknout na *Install from file... (Instalovat ze souboru...)*
  - Navigovat na CD a vybrat soubor *ShapeKeyAnimation.py*
  - Vpravo vybrat kategorii pluginů *User (Uživatelské)*
  - V seznamu položek u *Object: Shape keys animations from objects* vpravo zaklknout checkbox pro aktivaci

B.2 Použití doplňku

- Ujistíme se, že jsme v *Objektovém režimu*
- Vybereme dva objekty ve scéně
- Aktivujeme doplněk jedním z následujících způsobů
  - V menu 3D pohledu pomocí *Object (Objekt)* → *Shape Key Animation (Animace tvarových klíčů)* a vybereme požadovaný operátor doplňku
  - Totožné podmenu otevřeme klávesovou zkratkou *Ctrl + Alt + A* a vybereme požadovaný operátor doplňku
  - Vyhledáním názvu požadovaného operátoru stisknutím *Mezerníku* a napsáním jeho názvu (Názvy operátorů vždy začínají na *SKA*)
- Počkáme na dokončení výpočtů
- Přehrajeme animaci kliknutím na odpovídající tlačítko na časové ose.

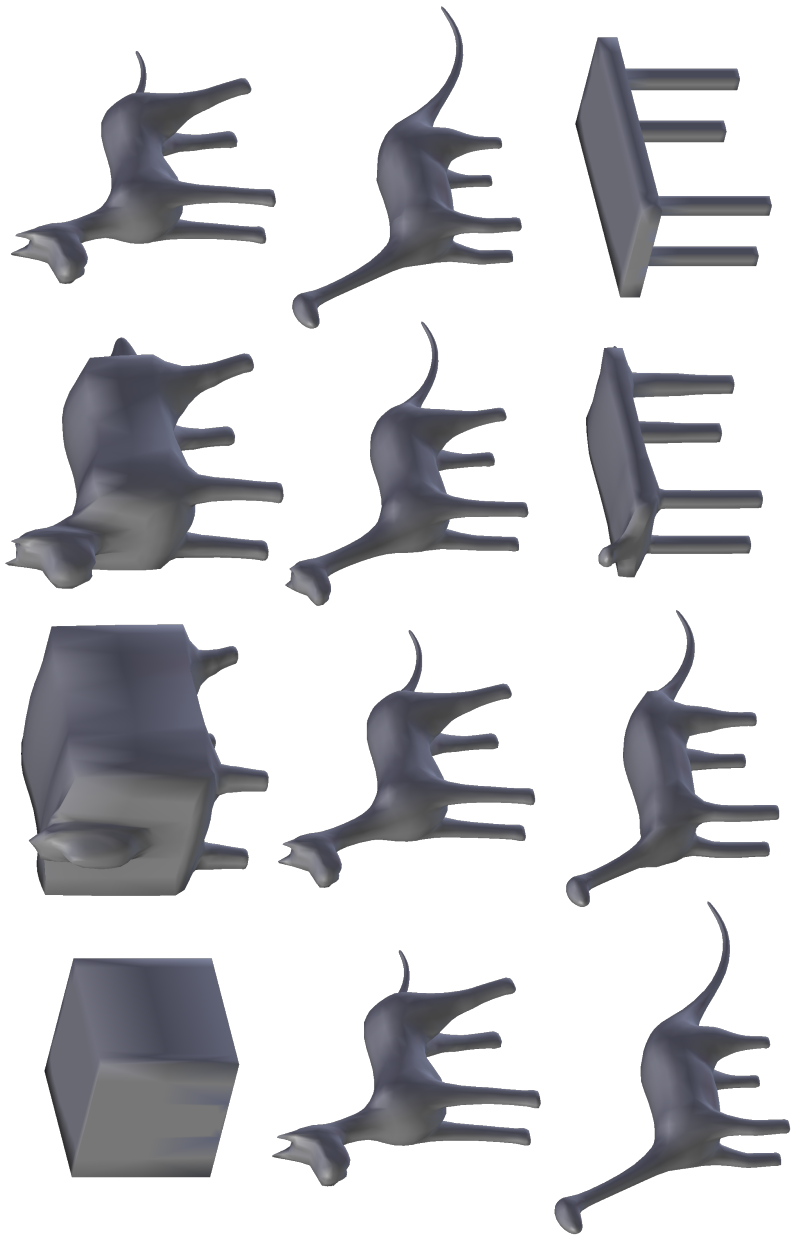
B.3 Další ovládání programu

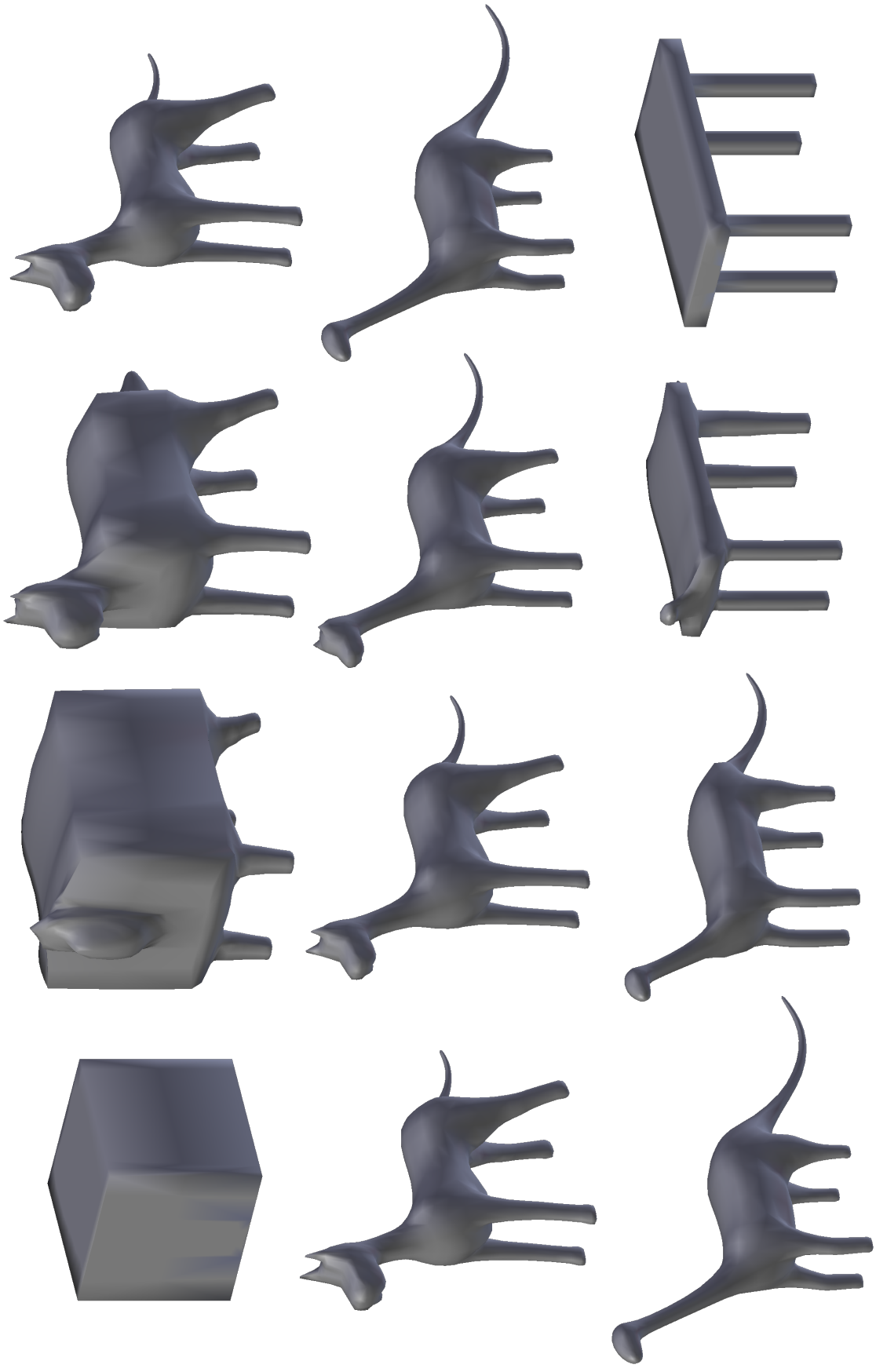
- Mezi *Editovacím* a *Objektovým módem* se můžeme přepínat klávesou *Tab*
- Prvky vybíráme *pravým tlačítkem myši*
- Více prvků vybereme, pokud při vybírání držíme klávesu *Shift*
- Pokud je alespoň jeden prvek vybrán, všechny odznačíme klávesou *A*
- Definovat vazby mezi modely můžeme v pravé části obrazovky využitím *Vertex Groups*[1]

[1] V tomto dodatku jsou zmíněny pouze nejzákladnější úkony. Podrobnější ovládání, jako například definice vazeb modelů je rozepsáno v textu této technické zprávy

Dodatek C

Animace



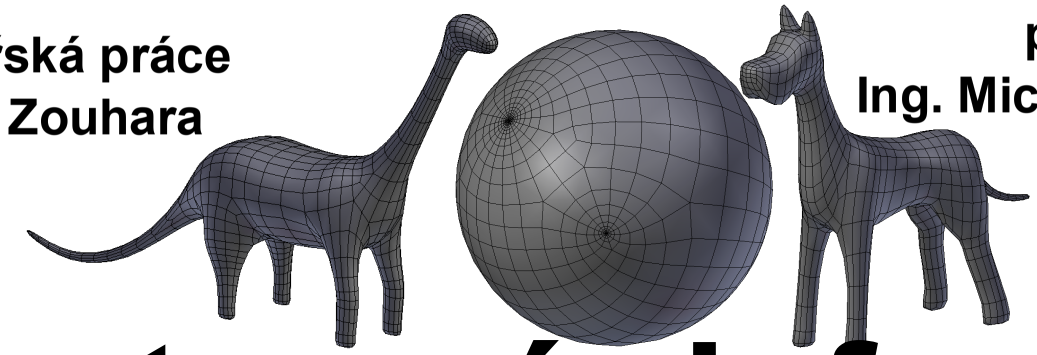


Dodatek D

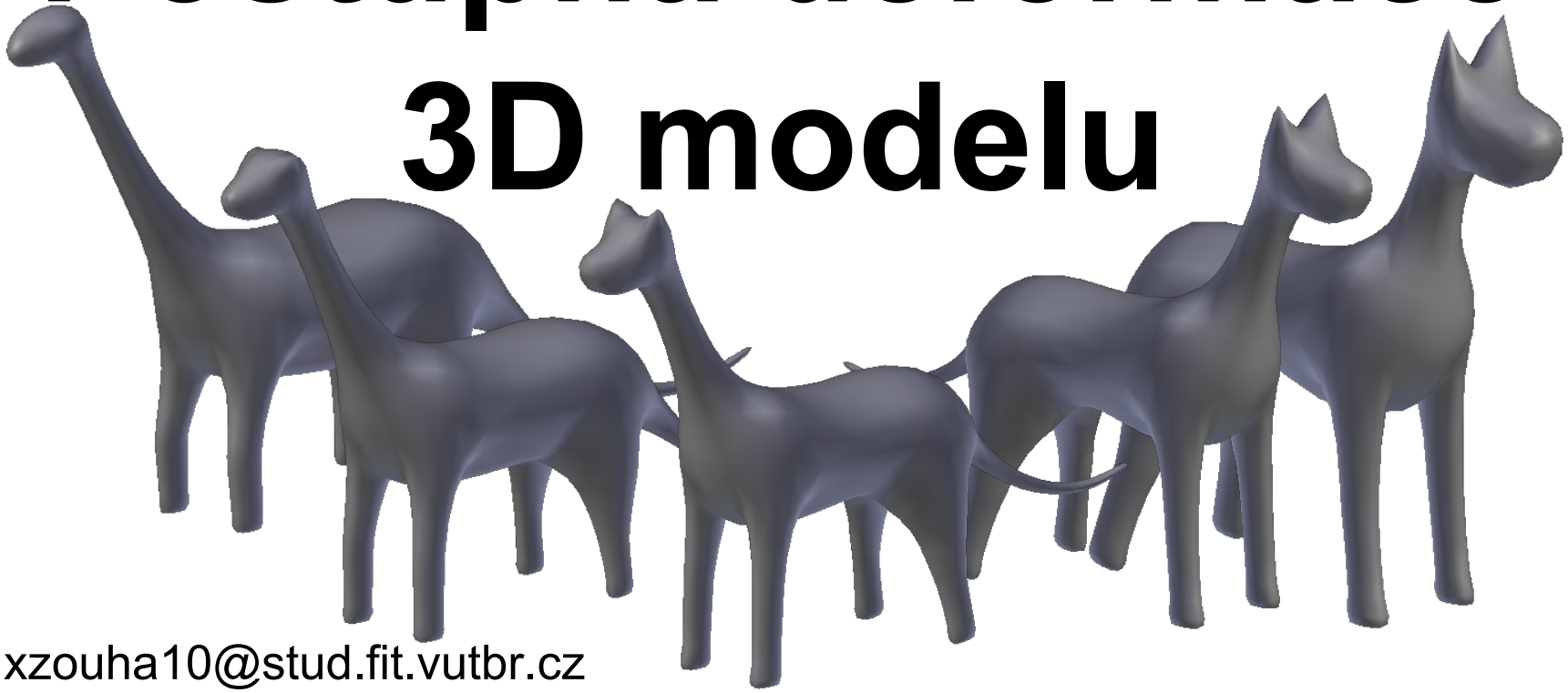
Plakát

Bakalářská práce
Marka Zouhara

pod vedením
Ing. Michala Španěla, Ph.D.



Postupná deformace 3D modelu



xzouha10@stud.fit.vutbr.cz