Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Skenování animací lidských obličejů zařízením MS Kinect

Plzeň 2018 Dominik Poch

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 28. června 2018

Dominik Poch

Abstract

The bachelor thesis discusses an issue of creation of 3D animations. It describes basic principles and methods used throughout the entire animation process. The main goal of the thesis is to find a solution which requires to put minimal amount of expenses in each substep of the animation process. For this reason, different techniques for scanning in threedimensional space will be tested with respect to the best possible price/performance ratio. The thesis is deeply focused on Microsoft Kinect, which is widely used in households, mostly as part of Xbox game consoles. The next part of the thesis desings and implements programs which lead a user through each single stage of creation of three-dimensional animations to test the principles and methods used in the animation process. Finally, a player, which processes compressed animations using a created compression function, will be introduced.

Abstrakt

Bakalářská práce pojednává o problematice tvorby 3D animací. Popisuje základní používané principy a metody v celém procesu. Cílem práce je nalézt řešení, které bude vyžadovat minimální vložené náklady na jednotlivé dílčí kroky animačního procesu. Z tohoto důvodu budou vyzkoušeny různé techniky pro snímání objektů v trojrozměrném prostoru s ohledem na poměr cena/výkon. Zaměřeno je především na zařízení Microsoft Kinect, které je hojně používané i v domácnostech, majoritně jako součást herních konzolí Xbox. Z důvodu vyzkoušení principů a metod používaných při tvorbě animací je součástí práce návrh a implementace programů, které provedou uživatele jednotlivými fázemi tvorby trojrozměrných animací. Na závěr bude představen přehrávač umožňující zpracování komprimované animace pomocí vytvořené kompresní funkce.

Obsah

1 Úvod

3D animace se v dnešní době využívá v širokém spektru odvětví lidské činnosti, od tvorby filmů a her, až po simulace fyzikálních pokusů, průběhu medicínských zákroků a zobrazení biologických procesů v lidském těle. Speciálním případem, na který se 3D animace zaměřují, převážně u filmů a her, je pohyb člověka nebo jiných živých tvorů a jednotlivých jejich částí. Náklady na zachycení pohybů objektu pomocí statických obrazů bývají bohužel poměrně vysoké. Je k tomu zapotřebí množství specializovaného hardwaru, který mnohdy ani nestačí na správnou reprezentaci skutečného pohybu. Příkladem za všechny budiž animace lidských rukou a prstů při motion capture, kde neexistují dostatečně kvalitní rukavice, které by tak jemné pohyby dokázaly zachytit a jiné přístupy také nedávají nikterak perfektní výsledky. Pro omezení těchto nákladů lze ušetřit na hardwaru, například levnějších 3D kamerách pro snímání, ale s tím většinou přichází ruku v ruce i zhoršení kvality nasnímaných dat a s tím spojené problémy při jejich zpracování.

Cílem bakalářské práce je snížení těchto nákladů v oblasti skenování a animace lidského obličeje. Docílit toho lze navržením vhodného postupu pro získání a zpracování 3D dat z levného, široce dostupného zařízení, jakým je například MS Kinect for Windows 2.0. Potíže, které toto specifické zařízení provázejí jsou především spojené s jeho nepříliš kvalitní kamerou. Důsledkem je řidší point cloud plynoucí z malého rozlišení kamery a zašumění získaných dat kvůli nedostatečně přesnému snímání hloubek bodů v prostoru. Řešením, které by tato práce měla přinést, bude algoritmus, který, i přes výše jmenované nedostatky hardwaru, zvládne vhodně deformovat řídící objekt, vypořádat se se zašuměním dat a vytvořit animace odpovídající skutečným pohybům v obličeji.

Cestou za splněním vytyčeného cíle bude nutné nalézt vhodný způsob pro získání kvalitního 3D objektu, který bude sloužit jako základní kámen pro jednotlivé snímky v průběhu animace. V kapitole 4 bude vytvořeno několik aplikací pro různá zařízení a vyzkoušeno i pár existujících přístupů, které by mohly být schopny tento akt vykonat s důrazem na základní myšlenku celé práce. Nicméně je nutné předvést i opačný konec spektra a z důvodu posouzení kvalit přístupu, jenž práce předkládá, ukázat profesionální zařízení používané pro skenování 3D objektů. S existujícím základním modelem nebude problém nasnímat, pomocí zařízení Microsoft Kinect, částečné skeny a vytvořit jednotlivé snímky animace. Předpokladem ovšem je, že výstup z Kinectu nebude dosahovat takových kvalit, aby mohl být ve své syrové verzi

použit v reálné animaci. Proto bude nejspíše nezbytné data vzít a opravit je pomocí různých postupů, které by měly zlepšit kvalitu natolik, aby bylo možné animace promítnout na kvalitní 3D model. 3D objekt bude potřeba deformovat tak, aby odpovídal upraveným datům získaným ze skenování. Pro tento účel bude využita rigidní a nerigidní registrace, které by společně měly upravit kvalitní 3D objekt a převést tak naskenovanou animaci z Kinectu do mnohem kvalitnější podoby. Popis registrací, jejich kroků a použitých algoritmů bude dále popsán v kapitole 3 a program, který dokáže tyto teoretické znalosti převést do praxe a ukázat jejich využití lze dohledat v kapitole 5.

Vytvoření správné registrace a následné deformace nasbíraných dat z jednotlivých pohybů na řídící model není finálním krokem, který by dokázal správnost a ověřil funkčnost vytvořených postupů. Důležité je také zobrazení výsledného produktu, a proto bude součástí práce také návrh a implementace webového přehrávače (viz kapitola 6), který umožní nahrání animovaného 3D modelu, jeho dekompresi a následné zobrazení na webové stránce.

2 Oprava 3D dat

Při sběru 3D dat působí velké množství faktorů, a tak není divu, že data bývají často zatíženy různými nedostatky, převážně z důvodu nedostatečně kvalitního skenovacího zařízení. Proto je vhodné před samotným finálním využitím nasbíraná data opravit.

2.1 Vyplnění děr

Aby bylo možné díry v 3D modelu zaplnit je nejprve nutné jejich vyhledání. Běžný způsob, která se pro tuto činnost využívá hledá strany, jež přísluší stejnému mnohoúhelníku. Ten následně vytvoří souvislý útvar uvnitř trojúhelníkové sítě, čímž jasně vymezí oblast, ve které se díra nachází. Tento přístup využívá například [5]. Po uskutečnění lokalizace přijde na řadu vyplnění nalezených cyklů trojúhelníkovou sítí. V průběhu let se objevovaly různé postupy jak problém vyplnění děr řešit [12].

Od počátečních heuristických přístupů, přes zjemnění přechodů naivní triangulace pomocí postprocessingu nebo metody nejmenších čtverců a radiálních bázových funkcí, až po metody, jež využívají příkladů podobných zpracovávanému objektu. Tyto příklady vycházejí z jiné části aktuálně zpracovávaného nebo ze základního velmi podobného objektu.

Zmíněné metody zpracovávají statické modely s velkým přehledem, ale u dynamických modelů, kde je nutné zaplnit díry na všech jednotlivých snímcích, mohou vést k nesprávné topologii nebo částečně nesouvislým povrchům. To je v případě animací, jimiž se tato práce zabývá, velká potíž. Tento problém naštěstí řeší například algoritmus dočasně souvislého vyplnění děr [17].

Algoritmus začíná vyplněním děr na jednotlivých snímcích animace pomocí vizuálního obalu meshe a jeho spojením s body objektu na daném snímku. Spojení probíhá pomocí Poissonovské rekonstrukce, blíže popsané v kapitole 4.2, za použití různých vah pro obal a původní body. Využití Poissonovské rekonstrukce může způsobit vytvoření různých nežádoucích artefaktů, jejichž odstranění má na starosti další krok algoritmu, který zjemňuje přechody mezi původním povrchem meshe a vyplněnými dírami. Dále se může na rekonstruované meshi nacházet flickering, který autoři odstraňují zkombinováním aktuálního snímku s jeho sousedy, které na sebe naregistrují a opětovným použitím Poissonovské rekonstrukce s různými váhami pro sousedy a zrekonstruované oblasti meshí flickering odstraní. Nakonec obnoví detaily z původních snímků a objekty jsou připraveny pro další práci.

3 Registrace

Registraci mezi dvěma různými 3D plochami lze definovat jako relaci T, která určuje mapování (transformace) mezi jednotlivými body A a A'.

3.1 Rigidní registrace

Rigidní registrace [4, 21] využívá pouze dvou různých transformací, translace a rotace. Pro mapování bodu (x, y, z) na (x', y', z') platí

$$x' = r_{11}x + r_{12}y + r_{13}z + t_1$$
$$y' = r_{21}x + r_{22}y + r_{23}z + t_2$$
$$z' = r_{31}x + r_{32}y + r_{33}z + t_3$$

což lze zapsat vektorově jako

$$\mathbf{a}' = \mathbf{R}\mathbf{a} + \mathbf{t} \tag{3.1}$$

kde $\mathbf{R}$ je matice rotace, $\mathbf{t}$ translační vektor a $\mathbf{a} = (x, y, x)$, resp. $\mathbf{a}' = (x', y', z')$ jsou souřadnice bodu A, resp. A'.

Cílem rigidní registrace je najít transformace $\mathbf{R}$ a $\mathbf{t}$, které minimalizují vzdálenost mezi korespondujícími body na mapovaných plochách

$$\min_{\mathbf{R},\mathbf{t}} \sum_{i=1}^{n} \|\mathbf{x_i'} - \mathbf{R}\mathbf{x_i} + \mathbf{t}\|^2$$

kde $\mathbf{x_i'}$ je i-tý bod na cílové a bod $\mathbf{x_i}$ je i-tý bod na mapované (zdrojové) ploše. Algoritmus, který se při hledání transformací využívá se nazývá **Iterative closest point**.

3.1.1 Iterative closest point

Řekněme, že máme dvě plochy K a F, přičemž první zmiňovaná je plocha referenční (cílová, pevná), na kterou budeme registrovat druhou počáteční (zdrojovou, mapovanou) plochu. Každá z ploch obsahuje množinu bodů $Q \in K$ a $P \in F$. ICP přiřadí každému bodu p_i nejbližší bod z množiny Q a vyhledá translační vektor $\mathbf{t}$ a matici rotace $\mathbf{R}$. Nakonec aplikuje nalezené transformace na zdrojovou plochu. Nutno podotknout, že plochy by od sebe neměly být příliš vzdáleny, jinak se může vyskytnou nestabilita algoritmu

a registrace selže. Metod používaných k hledání vhodné translace a rotace existuje více, nicméně zde budou ukázány pouze ty nejčastější.

Hledání translace je velmi jednoduché, a to pomocí rozdílu centroidů C_Q a C_P jednotlivých ploch

$$\Delta x = C_{Q,x} - C_{P,x}$$
$$\Delta y = C_{Q,y} - C_{P,y}$$
$$\Delta z = C_{Q,z} - C_{P,z}$$

Výpočet centroidů lze provést například pomocí průměru bodů v daných množinách pro

$$C_Q = \frac{q_1 + q_2 + q_3 + \cdots + q_n}{n}$$

$$C_P = \frac{p_1 + p_2 + p_3 + \cdots + p_n}{n}$$

Výsledný translační vektor se bude poté rovnat

$$\mathbf{t} = (\mathbf{\Delta x}, \mathbf{\Delta y}, \mathbf{\Delta z} : \mathbf{1})$$

Nalezení matice rotace není tak přímočaré jako hledání matice translace a je zapotřebí rozsáhlejších úprav. Mezi způsoby, jak spočítat matici rotace se řadí Kabschův algoritmus, využívající singulárního rozkladu, nebo reprezentace a výpočet rotace pomocí kvaternionů.

Po vypočtení matic začíná ICP novou iteraci opětovným přiřazením nejbližších bodů, dokud se obě plochy neshodují. To není v praxi vždy zcela dosažitelné, a tak dochází pouze k minimalizaci vzdáleností mezi body nebo k zadefinování maximální přípustné chyby vzdálenosti ϵ, která dokáže pokrýt chybu při získávání dat a vzdálenost mezi body na plochách K a F bude ve výsledku menší nebo rovna ϵ. Další možností, jak zastavit algoritmus ICP je pomocí maximálního počtu iterací. Volba ukončovací podmínky záleží na konkrétní implementaci.

3.1.2 Kabschův algoritmus

Kabschův algoritmus [13] se využívá pro hledání matice rotace. V průběhu algoritmu jsou využívány centroidy C_Q z cílové množiny bodů a C_P ze zdrojové množiny bodů, pojmenovaných stejně jako v předchozí kapitole, $Q \in K$ a $P \in F$.

Centroidy jsou přesunuty do počátku soustavy souřadnic a odečtou se od všech bodů $q'_i = q_i - C_Q$ a $p'_i = p_i - C_P$. Tyto body se uloží do matic

3xN, kde N je počet bodů, pojmenovaných například $\mathbf{Q}'$ a $\mathbf{P}'$, jenž budou ve tvaru

$$\mathbf{Q}' = \begin{pmatrix} x_1 & x_2 & \dots \\ y_1 & y_2 & \dots \\ z_1 & z_2 & \dots \end{pmatrix}$$

a obdobně pro matici $\mathbf{P}'$. Následně se spočítá jejich kovarianční matice a rozloží se pomocí singulárního rozkladu

$$\mathbf{C} = \mathbf{Q}'\mathbf{P}'^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$$

kde matice $\mathbf{U}$ a $\mathbf{V}$ jsou unitární a představují matice rotace a $\mathbf{\Sigma}$ je diagonální matice rovná matici používané pro změnu měřítka. Výsledná matice rotace $\mathbf{R}$ je následně spočtena pomocí vztahu

$$\mathbf{R} = (\mathbf{U}\mathbf{V})^T$$

3.2 Nerigidní registrace

Rigidní registrace, jak ukázala předchozí kapitola, využívá pouze translačního a rotačního pohybu k registraci jedné plochy na druhou. Nerigidní registrace se využívá v případě, kdy je kromě změny polohy a natočení potřeba ještě navíc objekt deformovat, aby bylo možné registraci uskutečnit. Metod, jak provést nerigidní registraci je mnoho. Liší se svojí obtížností, obecností či přesností mapování. Lze je aplikovat buď na celý objekt, či pouze na jeho část a transformovat tak pouze místo, které je třeba. V tom se liší od rigidních registrací, které lze aplikovat pouze na celý objekt. Díky tomu často dochází k využití rigidní registrace a následné lokální nerigidní registrace k doladění nepřesností. Mezi první metody používané pro nerigidní registraci přichází v úvahu nejjednodušší metoda a to pouhé zobecnění rigidní registrace pomocí afinních transformací [8] a postupně se přejde k obtížnějším a propracovanějším technikám, mezi než lze zařadit například RBF interpolace s různými typy bázových funkcí [4, 21] a interpolovaných hodnot či algoritmus Hao Li [16].

3.2.1 Afinní transformace

Rigidní registraci lze na nerigidní převést zobecněním používaných transformací. Zatímco rigidní registrace využívá pouze lineárních transformací, jak lze vidět v rovnici 3.1, nerigidní registrace by pro převedení bodu $\mathbf{x} =$

(x, y, z) do bodu $\mathbf{x}' = (x', y', z')$ využívala afinních transformací, definovaných jako:

$$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{b}$$

u nichž platí, že $\mathbf{b}$ je translační vektor a $\mathbf{A}$ je matice 3×3. Afinní transformace nezachovávají úhly ani vzdálenosti, ale udržují rovnoběžnost, a tak umožňují provádět deformace objektů.

Robustní algoritmus pro nerigidní registrace pomocí afinních transformací vytvořili Feldmar a Ayache ve své práci [8]. Zabývají se zde zobecněním iterativního algoritmu rigidní registrace pomocí afinních transformací a jeho použití u nerigidního přístupu. Využívají ideu hledání bodů na podobných plochách. Nehledají tak pouze nejbližší bod ve smyslu polohy v prostoru, nýbrž uvažují i další parametry, jimiž jsou normála a hlavní křivost[1] v daném bodě. Počítají tedy vzdálenost mezi vektory o složkách $(x, y, z, n_x, n_y, n_z, k_1, k_2)$ a to pomocí struktury zvané KD-strom[2]. Dále hledají podmínky pro stabilitu tohoto algoritmu, jehož primitivní řešení vede až k vymizení mapované meshe. Toto řešení umožňuje mapovat povrchy, ale pro přesné deformace se úplně nehodí, a tak představují algoritmus pro lokální afinní deformace. Algoritmus pracuje iterativně, nejprve zvolí bod M_k na mapované ploše a pro všechny jeho body v okolí o poloměru r spočítá rigidní registrace. Tyto registrace následně vyhladí a aplikuje jednu vyhlazenou registraci na celé okolí.

3.2.2 Interpolace pomocí radiálních bázových funkcí

Interpolace pomocí radiálních bázových funkcí (RBF interpolace) je jednou z interpolačních technik používaných pro interpolace neuspořádaných dat, kdy existuje vstup souřadnic bodů $\mathbf{x_i^n}$ a hodnot $\mathbf{h_i} \in \mathbf{E^k}$, které je nutno interpolovat. Výsledná hodnota interpolace je definována jako

$$f(\mathbf{x}) = \sum_{\mathbf{i=1}}^{\mathbf{N}} \lambda_\mathbf{i} \phi(\|\mathbf{x} - \mathbf{x_i}\|)$$

kde $\lambda_i \in \boldsymbol{\lambda}$ jsou váhy jednotlivých bázových funkcí, $\phi(\|\mathbf{x} - \mathbf{x_i}\|)$ jsou jejich hodnoty, $\mathbf{x}$ je bod pro který interpolujeme a $\mathbf{x_i}$ jsou souřadnice i-tého bodu. Pokud interpolujeme vícero bodů vzniká tak soustava rovnic:

[1] Hlavní křivost [11] je minimální a maximální normálová křivost v daném bodě. Normálová křivost je součin druhé derivace křivky procházející daným bodem a jednotkovým normálovým vektorem plochy v daném bodě.

[2] Ukládá vrcholy do stromu podle velikosti souřadnice, přičemž souřadnice se mění s aktuální úrovní ve stromu. Vyhledává nejbližšího souseda dle hodnoty souřadnic.

$$f(\mathbf{x_1}) = \sum_{\mathbf{i=1}}^{\mathbf{N}} \lambda_\mathbf{i} \phi(\|\mathbf{x_1} - \mathbf{x_i}\|)$$
$$f(\mathbf{x_2}) = \sum_{\mathbf{i=1}}^{\mathbf{N}} \lambda_\mathbf{i} \phi(\|\mathbf{x_2} - \mathbf{x_i}\|)$$
$$\cdots$$
$$f(\mathbf{x_m}) = \sum_{\mathbf{i=1}}^{\mathbf{N}} \lambda_\mathbf{i} \phi(\|\mathbf{x_m} - \mathbf{x_i}\|)$$

soustavu lze samozřejmě přepsat do maticového tvaru:

$$\boldsymbol{\Phi\lambda} = \boldsymbol{f}$$

a RBF interpolace se stane lehce vypočitatelnou úlohou řešení soustavy lineárních rovnic.

RBF se při nerigidní registraci nikdy nevyužívá sama. Vždy je nutné data nějakým způsobem předzpracovat a získat z nich hodnoty pro interpolaci. Častým způsobem předzpracování je zvolení deformačního grafu, jehož vrcholy představují body na objektu, pro nějž se bude interpolovat. Okolí těchto řídících bodů se rigidně registruje na cílový objekt a dílčí výsledky těchto registrací se stanou interpolačními hodnotami.

Bázové funkce

Existují různé typy bázových funkcí $\phi(\|\mathbf{x} - \mathbf{x_i}\|)$, které se využívají. Toho dokáže zužitkovat nerigidní registrace jejíž metody, ve většině případů, RBF využívají a liší se v použité bázové funkci. Používané funkce se dělí do několika skupin. Lze je rozdělit podle typu, na polynomiální či spline, nebo rozsahu jejich vlivu, na globální a lokální.

Polynomiální funkce Jedná se o případ, kdy za bázové funkce volíme polynomiální funkce [4, 21] o řádu většinou 2 až 5. Není příliš vhodné volit vyšší řády bázového polynomu, jelikož mohou vést k nepředvídatelnému chování celého procesu. Polynomiální bázové funkce fungují lépe než-li registrace pomocí afinních transformací.

Spline funkce Spline funkcí nazýváme aproximaci křivky, která již nemusí být polynomiální. Roku 1977 přišel Jean Duchon s funkcí $\phi(r) = r^2 \log r$, $r = \|\mathbf{x} - \mathbf{x_i}\|$ zvanou **Thin-plate spline** [7], jež je jednou z nejčastěji používaných bázových funkcí [21] a našla si cestu i do oblasti registrací dvou ploch. Další používanou spline funkcí je **B-spline** (basis spline).

Globální funkce Globální funkce jsou funkce, které jsou použitelné na celý interval vstupních hodnot r. Z toho důvodu je matice $\mathbf{\Phi}$ hustě zaplněna nenulovými hodnotami. Globální funkce lze dále rozdělit na funkce, jejichž hodnota

- roste s zvětšující se hodnotou r - jedná se například o funkce $\phi(r) = r^3$ nebo $\phi(r) = r^2 \log r$
- klesá s zvětšující se hodnotou r - např. $\phi(r) = 1/r^2$ nebo $\phi(r) = e^{-\epsilon r^2}$, kde ϵ je konstanta, kterou je nutno pokusně určit.

Lokální funkce Jedná se o funkce, jejichž hodnota je nenulová pouze na intervalu $r \in \langle 0, 1 \rangle$, což způsobí, že matice $\mathbf{\Phi}$ obsahuje málo nenulových hodnot, čímž se sníží výpočetní náročnost soustavy lineárních rovnic.

Interpolovaná hodnota

V nerigidní registraci lze v principu interpolovat tři hodnoty. Translační korekční vektory, matice rigidní transformace a translační vektor společně s kvaternionem.

Translační korekční vektory Translační korekční vektor je výsledek rozdílu souřadnic řídícího bodu po registraci a jeho původní hodnoty. Představuje posun bodu po aplikaci translace a rotace v průběhu rigidní registrace.

Matice rigidní registrace Teoreticky lze interpolovat také výslednou matici rigidní registrace. V praxi ovšem interpolace matic nefunguje tak, jak bychom čekali a matice obsahující rigidní transformace již nemusí být po interpolaci rigidní a aplikace takovéto matice v našem případě zcela postrádá smysl.

Kvaterniony Při použití kvaternionů se odděleně interpoluje translační vektor a kvaternion, jenž představuje jednu z možných vyjádření rotací v trojrozměrném prostoru. Ze všech zmíněných interpolačních hodnot se kvaternion interpoluje nejlépe. Kvaterniony se začal zabývat Sir William Rowan Hamitlon (1805 - 1865), jenž strávil formulací konceptu čtyřrozměrné notace pro trojrozměrnou rotaci 15 let. Nakonec dokázal stanovit základní pravidla kvaternionu definovaného jako vícerozměrné komplexní číslo následovně:

$$i^2 = j^2 = k^2 = ijk = -1$$

$$ij = k \qquad jk = i \qquad ki = j$$

$$ji = -k \qquad kj = -i \qquad ik = -j$$

Nicméně až Joshian Gibbs (1839-1903) dokázal uvést kvaterniony k širšímu použití, do té doby nebyly kvaterniony plně akceptovány. Gibbs nalezl nový způsob reprezentace komplexních členů $ib + jc + kd$, kde i, j, k představují komplexní složky, pomocí trojrozměrného vektoru. $\mathbf{i}, \mathbf{j}, \mathbf{k}$ se poté stanou jednotkovými vektory v Kartézském souřadném systému. Značení kvaternionů vycházející z Gibbsovi reprezentace vypadá takto:

$$\mathbf{q} = [\mathbf{s}, \mathbf{v}] = [\mathbf{s}, \mathbf{xi} + \mathbf{yj} + \mathbf{zk}]$$

kde s,x,y,z jsou reálná čísla.

Vektory lze převést na kvaterniony velmi jednoduše. Například bod $P(x, y, z)$ lze přepsat do tvaru:

$$\mathbf{p} = [\mathbf{0}, \mathbf{xi} + \mathbf{yj} + \mathbf{zk}]$$

Dle Eulera je rotace určena pomocí směrového vektoru $\mathbf{u}$ reprezentující osu rotace a úhlem δ, jenž určuje velikost rotace. Pomocí kvaternionu se rotace zapisuje takto

$$\mathbf{q} = [\cos \delta/\mathbf{2}, \mathbf{u} \sin \delta/\mathbf{2}]$$

a rovnice určující vztah mezi počátečním bodem $\mathbf{p}$ a bodem po rotaci $\mathbf{p}'$ bude

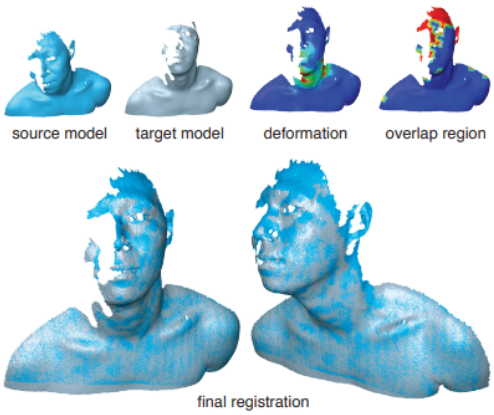
$$\mathbf{p}' = \mathbf{qpq^{-1}}$$

Aproximace

Při výpočtu interpolace hodnoty bodu $\mathbf{x_i}$ pomocí RBF dochází k výpočtu bázové funkce se všemi ostatními body, což vede k maticím velkých řádů. Matice lze zmenšit pomocí aproximací hodnot λ_i. Řekněme, že N naměřených bodů aproximujeme nějakou množinou bodů Ω o velikosti M, přičemž $M \ll N$. Při aplikaci RBF aproximace získáme vztah:

$$f(\mathbf{x}) = \sum_{\mathbf{j=1}}^{\mathbf{M}} \lambda_{\mathbf{j}} \phi(\|\mathbf{x} - \omega_{\mathbf{j}}\|)$$

který lze, pro aproximaci všech bodů, opět převést do maticového tvaru. Výsledné dimenze matice Φ by se poté rovnaly $N \times M$ a délka vektoru λ by byla rovna M, což směřuje k přeurčené soustavě rovnic s menším počtem neznámých než rovnic a je tak vhodná pro globální registraci, zatímco v



Obrázek 3.1: Hao Li algoritmus [16]

lokálním měřítku dosahuje lepších výsledků interpolační metoda radiálních bázových funkcí.

3.2.3 Hao Li

Pánové Hao Li, Robert W. Summer a Mark Pauly vytvořili ve své práci [16] algoritmus pro registraci částečných skenů s využitím nerigidní registrace. Hlavním tématem práce byla registrace dvou částečných překrývajících se skenů získaných v různých okamžicích z nějakého deformovaného objektu (viz. Obrázek 3.1). Deformace navržené v algoritmu jsou řízeny příslušností bodů mezi zdrojovým a cílovým skenem, tak aby bylo dosaženo co nejpřirozenější deformace s výsledkem odpovídajícím skutečnosti. Příslušnost bodů je v algoritmu dále vážena v závislosti na vzdálenosti mezi odpovídajícími body k identifikaci překrývajících se částí naskenovaných objektů.

Deformační algoritmus nejprve vytvoří ze zdrojového objektu redukovaný model ve formě deformačního grafu. Vrcholy tohoto grafu vytvoří síť řídících bodů s danou pozicí v prostoru. S každým vrcholem je také sdružena jedna afinní transformace určující deformaci celého blízkého okolí na cílový sken. Dohromady tvoří transformace nerigidní registraci zdrojového skenu a grafu, který mu přísluší. Tento přístup zvládne rigidní i nerigidní registraci najednou, nicméně pro zvýšení efektivity algoritmu navrhli autoři registrace oddělit, a tak jsou nejprve spočítány globální rigidní transformace, translace $\mathbf{t}$ a rotace $\mathbf{R}$ okolo těžiště $\mathbf{g}$, a až následně počítá lokální deformace na deformačním grafu. Celková transformace vrcholu $\mathbf{v}_j$ na $\mathbf{v}'_j$ je tedy rovna

$$\mathbf{v}'_j = \Phi_{global} \cdot \Phi_{local}(\mathbf{v}_j)$$

kde

$$\Phi_{global}(\mathbf{v}_j) = \mathbf{R}(\mathbf{v}_j - \mathbf{g}) + \mathbf{g} + \mathbf{t}$$
$$\Phi_{local}(\mathbf{v}_j) = \sum_{i=1}^{n} w_i(\mathbf{v}_j)[\mathbf{A}_i(\mathbf{v}_j - \mathbf{x}_i) + \mathbf{x}_i + \mathbf{b}_i]$$

$w_i(\mathbf{v}_j), i \in 1 \ldots n$ představují váhy, které jsou nenulové pouze pro k-nejbližších vrcholů grafu, definované jako

$$w_i(\mathbf{v}_j) = \frac{1 - \|\mathbf{v}_j - \mathbf{x}_i\|/d_{max}}{\sum_{p=1}^{k}(1 - \|\mathbf{v}_j - \mathbf{x}_p\|/d_{max})}$$

kde d_{max} je vzdálenost ke (k+1)-nejbližšímu vrcholu. Vypočtené transformace jsou následně v algoritmu optimalizovány pomocí různých přístupů. Optimalizace existují v algoritmu celkem čtyři. První hlídá, respektive penalizuje, odklon afinních transformací od čistě rigidního pohybu, v algoritmu označována jako E_{rigid}. Další optimalizační kritérium E_{smooth} vyhlazuje afinní transformace vrcholů grafu tak, aby odpovídaly svému okolí, čímž zabrání ostrým přechodům. K optimalizaci dochází také v případě mapování mezi zdrojovým a cílovým skenem, kde se k algoritmu hledání nejbližších bodů navíc využívá dat z hloubkových map získaných v průběhu skenování. Tím vzniká kritérium E_{fit}, jež zpřesňuje mapování bodů na cílovou mesh a je ještě dále upraveno pro zlepšení mapování v případě částečných skenů, kde je rovnice E_{fit} doplněna o váhu daného mapování. Nakonec přichází na řadu optimalizace E_{conf}, která určuje spolehlivost mapování. Optimalizační rovnice je rovna součtu jednotlivých optimalizačních kritérií vynásobených různými konstantami určujícími míru jednotlivých složek.

4 Získání kompletního 3D modelu

Získání kvalitního základu je klíčové pro finální animaci lidského obličeje. Na zvolení správného postupu bylo proto vynaloženo velké úsilí a vyzkoušeno bylo několik různých hardwarových zařízení, ale i softwaru k nalezení toho nejlepšího způsobu, který by odpovídal potřebám této práce. Při výběru bylo nahlíženo na několik důležitých vlastností, například kvalita 3D dat a textur, doba nutná pro nasbírání dat nebo jejich zpracování atd.

4.1 MS Kinect

Zařízení MS Kinect for Windows 2.0 bylo první, které přišlo do úvahy pro vytvoření základního 3D modelu hlavy vzhledem k jeho dostupnosti a snadnému použití. Vlastnosti senzorů [20] jsou

- Hloubkový senzor
  - Rozlišení: 512 x 424
  - Frekvence snímků: 30 Hz
  - FOV: 70 x 60°
  - Dosah: 0,5 - 4,5 m
- Kamera
  - Rozlišení: 1920 × 1080 (2,1 Mpx)
  - Frekvence snímků: 30 Hz nebo 15 Hz

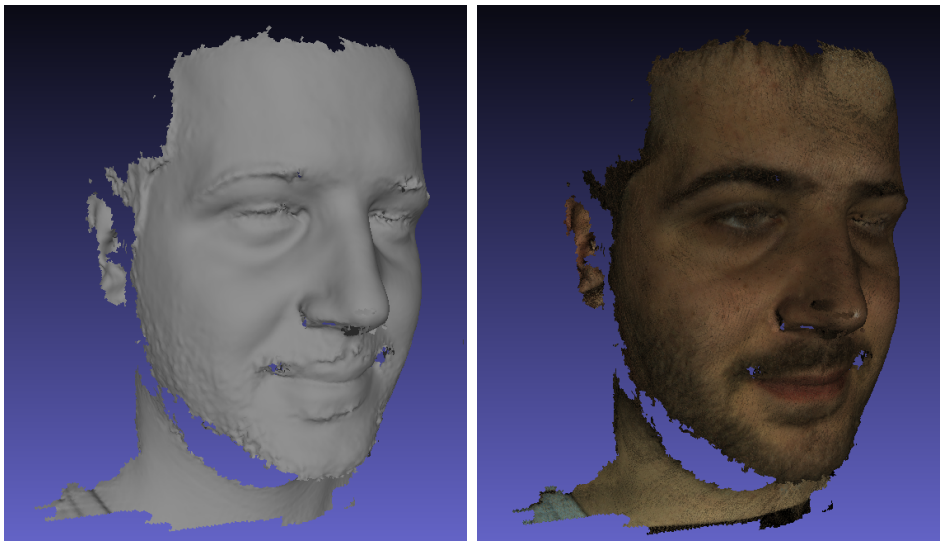
Pro získání 3D modelu byla navržena a implementována utilita, která načte určité množství hloubkových map, zprůměruje je a vytvoří z nich point cloud, jehož body využije pro vytvoření trojúhelníkové sítě. Kvalita výsledného modelu je ale poněkud slabší, jak lze vidět na Obrázku 4.1, především z důvodu velkého zašumění naskenovaných dat.

Obrázek 4.1: MS Kinect

4.2 Poissonovská rekonstrukce

Další způsob, který byl vyzkoušen, je Poissonovská rekonstrukce nad daty z MS Kinect for Windows 2.0. Pro získání těchto dat byly vyvinuty tři drobné programy. První dva slouží k načtení hloubek jednotlivých bodů ze zařízení Kinect a uložení do binárního souboru, přičemž každý z nich využívá rozdílných přístupů. Hlavní myšlenkou těchto programů je v zásadě pouze přesměrování výstupu z Kinectu do binárního souboru a nezabývají se žádnou složitější logikou. Poslední program slouží pro zpracování hloubek a jejich převedení na 3D point cloud. Zpracování je oddělené od načítání, aby programy zvládaly načítat data v reálném čase jakmile je skener vyprodukuje a nedocházelo tak k prodlevám. Načtené point cloudy se na sebe následně v programu třetí strany namapují a provede se Poissonovská rekonstrukce.

Cílem této rekonstrukce [15] je vytvořit voděodolnou aproximaci povrchu objektu z množiny vstupních dat S, přičemž každý vzorek $s \in S$, který se skládá z bodu a normály, leží na nebo poblíž výsledného povrchu. Aproximace probíhá získáním vztahu mezi gradientem vyhlazené charakteristické funkce a integrálem plochy normálového pole. Integrál je následně aproximován sumou vzorků z množiny vstupních dat a nakonec se rekonstruuje charakteristická funkce z gradientního pole ve formě Poissonova problému. Tato metoda se dočkala dalších vylepšení v roce 2013 [14] v oblasti hledání gradientu funkce, která nejlépe aproximuje integrál plochy vektorového pole.



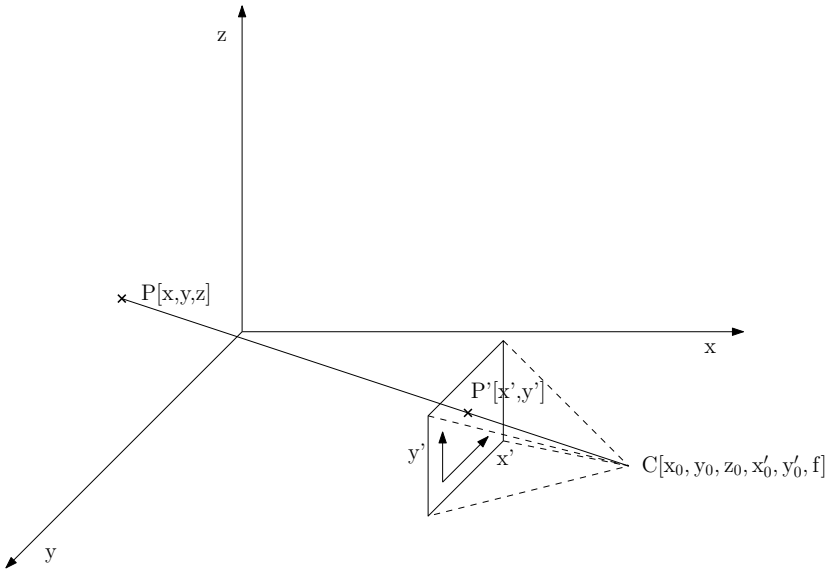
Obrázek 4.2: Výsledek skenování pomocí Artec Eva

4.3 Artec Eva

Artec Eva je příruční 3D skener od firmy Artec pro tvorbu 3D modelů středně velkých objektů. 3D skener dosahuje výjimečných výsledků především kvůli jeho hardwarovým specifikacím [3]

- 3D rozlišení: 0,5 mm
- Přesnost: 0,1 mm
- Rozlišení textur: 1,3 Mpx
- Dosah: 0,4 - 1 m
- FOV: 30 x 21°
- Frekvence snímků: 15 Hz

Přes velmi dobré rozlišení a přesnost má skener několik vad. Pro správné fungování potřebuje alespoň 12 GB paměti, procesor Intel I5 nebo I7 a grafickou kartu s pamětí 1 GB. To by nebyl až tak velký problém, ale když se k tomu přičte i astronomická pořizovací cena, která je dle výrobce €13 700, začíná být jasné, že toto není skener vhodný pro rozsáhlé komerční užití. Navíc doba, kterou musí osoba při skenování vydržet nehnutě sedět, je poměrně dlouhá a udržet stejnou polohu po celý čas je dosti obtížné, což lze vidět na Obrázku 4.2, především v oblasti okolo očí. Je zde vidět také problém s vlasy a vousy, které zařízení není schopno naskenovat a kvalita



Obrázek 4.3: Projekce prostorových bodů

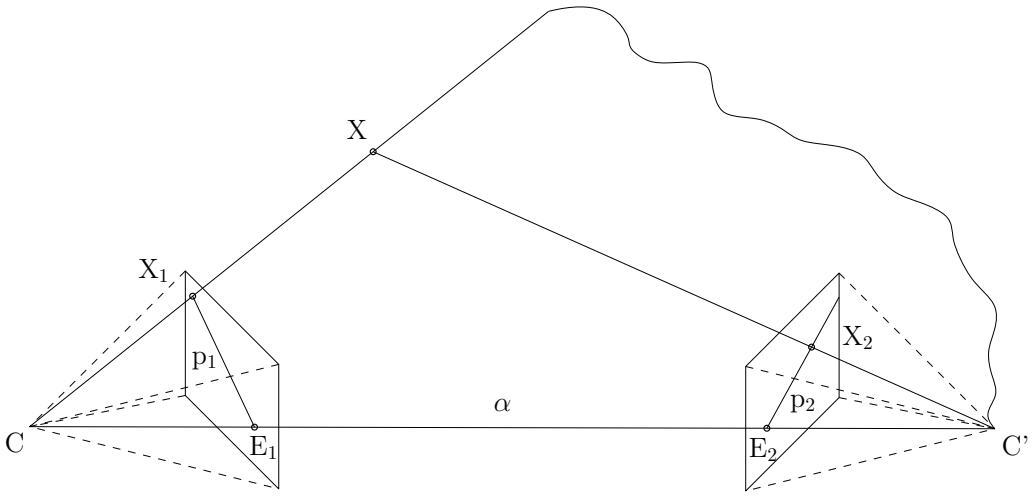
textury není nikterak ohromující, což se dalo s přihlédnutím na specifikace, především rozlišení textur, předpokládat.

4.4 Fotogrammetrie

Fotogrammetrie [6, 10, 18] je obor zabývající se zpracováním a interpretací obrazu za účelem získání tvaru a pozice objektů vyskytujících se na jedné nebo více předložených fotografií. Fotogrammetrii lze využít pro zpracování velkého rozsahu různých objektů, záleží na typu fotogrammetrie. Obvykle se dělí na pozemní (blízkou) a leteckou, přičemž letecká se využívá pro tvorbu a aktualizace map, zatímco pozemní je často užita při modelování staveb. Ač zpracovávané objekty běžně nabývají obrovských rozměrů, byla fotogrammetrie použita při modelování lidského obličeje s kvalitními výsledky.

4.4.1 Základy fotogrammetrie

Základním kamenem fotogrammetrie je získání prostorové informace z rovinných průmětů (fotografií) připravené scény. K získání rekonstrukce lze využít vícero cest. První pochází z oboru počítačového vidění a jedná se o obecnou epipolární geometrii [23, 24]. Druhý bude zmíněn specifický přístup vytvořený pro fotogrammetrii a to rovnice kolinearity [10].



Obrázek 4.4: Epipolární geometrie

Kalibrace kamery

Kalibrační techniky určují vnitřní a vnější parametry kamery. Dělí se na fotogrammetrickou a automatickou kalibraci. Automatická kalibrace [1, 9] je proces určující vnitřní a vnější parametry fotoaparátu z různých obrázků zpracovávané scény bez využití speciálního kalibračního subjektu. Fotogrammetrická kalibrace [2] naopak využívá speciální kalibrační objekt pro zjištění parametrů fotoaparátu.

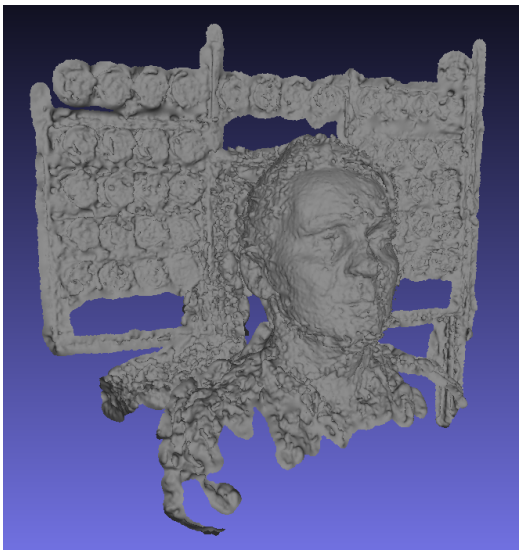
Projekční matice

Pro středové promítání $X \to X'$ v projektivním prostoru (na Obrázku 4.3) platí [2, 24]

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.1}$$

Matice $\mathbf{K}$, tzv. matice kalibrace kamery, obsahuje pět vnitřních parametrů. Tyto parametry dohromady definují ohniskovou vzdálenost, velikost snímače a kardinální body (předmětové ohnisko, obrazové ohnisko, hlavní body, uzlové body). Matice kalibrace je typu 3x4 ve tvaru

$$\mathbf{K} = \begin{bmatrix} \alpha_x & \gamma & x'_0 & 0 \\ 0 & \alpha_y & y'_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



Obrázek 4.5: První pokus fotogrammetrie

Parametry $\alpha_x = f \cdot m_x$ a $\alpha_y = f \cdot m_y$ značí ohniskovou vzdálenost v pixelech, kde m_x a m_y jsou přepočty mezi vzdáleností a počtem pixelů a f je běžná velikost ohniskové vzdálenosti v milimetrech. Zbývající parametry jsou γ, určuje koeficient sklonu os, x'_0 a y'_0 určující střed průmětu.

Matice $\mathbf{R}$ a vektor $\mathbf{t}$ z rovnice 4.1 určují rotaci a posun počátku soustavy souřadnic oproti pozici kamery a tím určují tzv. vnější parametry kamery.

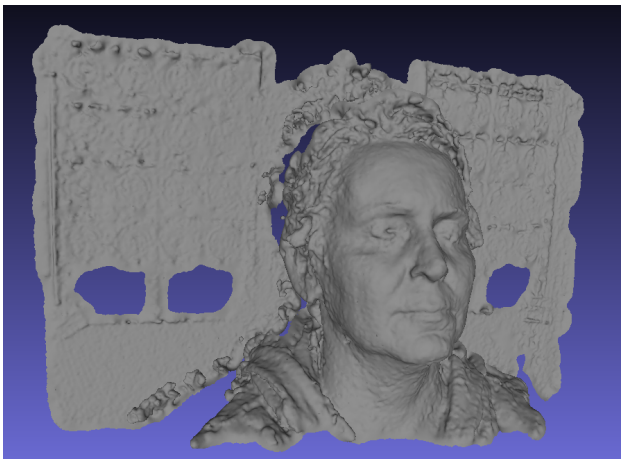
Projekční matice $\mathbf{P}$ v sobě uchovává všechny potřebné parametry kalibrace a je tedy rovna

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}$$

Epipolární geometrie

Epipolární geometrii lze definovat jako geometrii mezi dvěma průměty. Předpokladem je existence nějakého bodu X v prostoru, který je promítán do dvou pohledů ve formě X_1 a X_2, jak je ukázáno na Obrázku 4.4. Promítací paprsky bodu X vytvářejí tzv. epipolární rovinu označenou α. Průsečíky epipolární a obrazové roviny vytvářejí epipolární (sdružené) přímky označené p_1 a p_2. Tato přímka označuje možné umístění bodu X_1 na druhém průmětu.

Pro mapování bodu X_1 na epipolární přímku v příslušném pohledu se využívá fundamentální matice, kterou lze odvodit z rovnice 4.1 [24] nebo pomocí 2D homografie [23].



Obrázek 4.6: Fotogrammetrie s externím bleskem

Rovnice kolinearity

Při projekci prostorových bodů na 2D plochu (průmětnu) platí, že bod $P[x, y, z]$ na objektu, jemu odpovídající snímkový bod $P'[x', y']$ a střed promítání $C[x_0, y_0, z_0, x'_0, y'_0, f]$ musí ležet na přímce, jak říká podmínka kolinearity a ukazuje Obrázek 4.3. Podmínka kolinearity je definována pomocí dvou vztahů [19], jinak známé jako zobrazovací rovnice centrální projekce

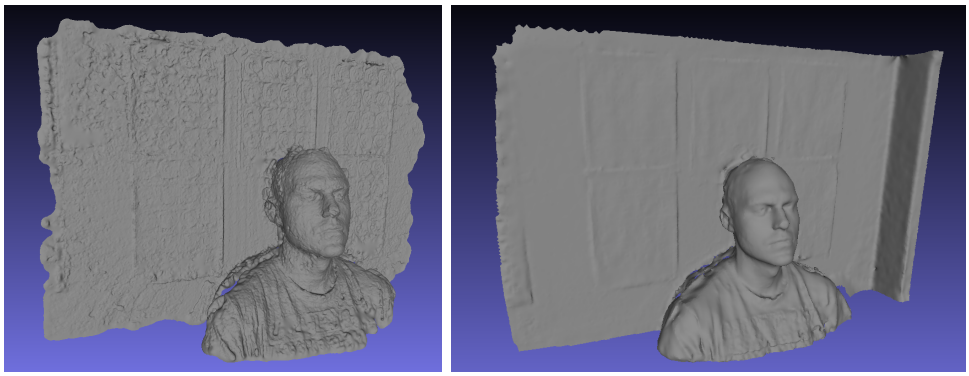
$$x' = x'_0 - f\frac{R_{11}(x - x_0) + R_{21}(y - y_0) + R_{31}(z - z_0)}{R_{13}(x - x_0) + R_{23}(y - y_0) + R_{33}(z - z_0)}$$

$$y' = y'_0 - f\frac{R_{12}(x - x_0) + R_{22}(y - y_0) + R_{32}(z - z_0)}{R_{13}(x - x_0) + R_{23}(y - y_0) + R_{33}(z - z_0)}$$

Tyto rovnice obsahují jak vnitřní parametry x'_0, y'_0 a f, tak i vnější x_0, y_0, z_0 a koeficienty matice rotace.

4.4.2 Testování fotogrammetrie

Pro otestování fotogrammetrie bylo nafoceno několik sérií různých subjektů, kdy se postupně upravovaly parametry fotoaparátu, osvětlení i okolí. Zpočátku se využíval fotoaparát bez blesku, což bylo nutné vykompenzovat nastavením parametrů fotoaparátu. To způsobilo, že produkované fotografie nevycházely v takové kvalitě, která by dostačovala potřebám procesu fotogrammetrie a výsledný 3D objekt byl svojí kvalitou na velmi nízké úrovni, jak dokládá Obrázek 4.5. Lze na něm vidět velké zašumění. Po této zkušenosti byl využit externí blesk, který výsledný model posunul na novou úroveň a odstranil velkou část šumu, ale ani to nebylo dostatečné, stále bylo vidět



(a) Photomodeler (b) Zephyr

Obrázek 4.7: Porovnání Photomodeleru a Zephyru

možné zlepšení a nevyužitý potenciál této metody (viz Obrázek 4.6). Proto byl dále vyměněn externí blesk za studiový a fotogrammetrie se prováděla ve dvou různých softwarech. Prvním byl Photomodeler[1], placený nástroj pro profesionální využití. Druhým použitým programem byl Zephyr[2], který nezachází do takových detailů jako Photomodeler, díky čemuž jsou jeho výsledky vyhlazenější. Porovnání výsledných 3D modelů z jedné a té samé série lze vidět na Obrázku 4.7).

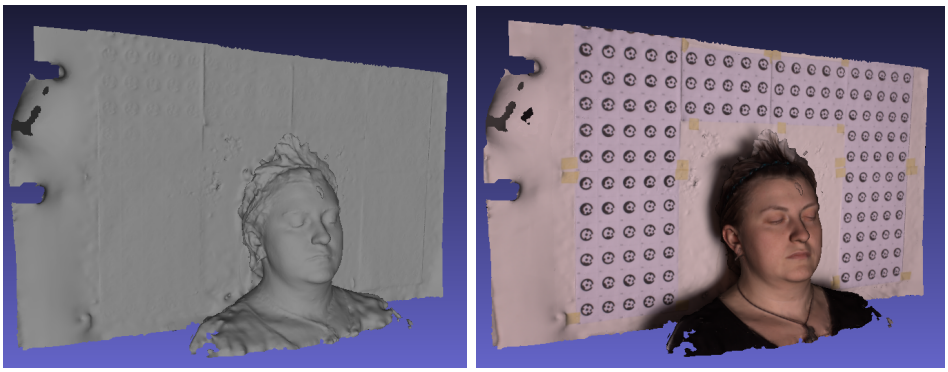
Nakonec byl použit fotoaparát Canon EOS 50D se studiovým bleskem. Pro úspěšný proces fotogrammetrie byly foceny série s minimálně 15 snímky provedené z různých úhlů. Tímto způsobem byl pořízen finální model na Obrázku 4.8).

Do procesu fotogrammetrie vstupuje velké množství faktorů, jež nakonec ovlivňují výsledný 3D model. Z průběhu testování bylo vyvozeno několik důležitých závěrů vedoucích k vysoké kvalitě výsledků procesu fotogrammetrie a je nutné je přímo specifikovat. Hlavním cílem při fotografování živých objektů by mělo být předcházení možným pohybům objektu, jelikož i drobné pohyby mohou proces fotogrammetrie narušit. Při využití procesu na lidskou hlavu se osvědčilo zapření týlu o tvrdý podklad, například zeď, což dobře stabilizuje hlavu a zabrání tak kývavým pohybům. Problémem jsou i oční víčka, kterým nejde v pohybu zcela zabránit a doporučením je mít oči raději zavřené. Nezbytným faktorem je samozřejmě spolupráce subjektu, jenž musí omezit svůj pohyb na minimální úroveň.

Než se přejde k technice a samotnému aktu sbírání fotografií je nutné také zdůraznit, že okolí zpracovávaného objektu by nemělo být jednotvárné. Tím je myšleno především jednobarevné rovné pozadí bez speciálních ar-

[1] http://www.photomodeler.com/index.html

[2] https://www.3dflow.net/3df-zephyr-pro-3d-models-from-photos/



Obrázek 4.8: Finální verze fotogrammetrie

tefaktů. V takovém případě je nutné umístit nějaké značky okolo subjektu, které pomohou softwaru se zaměřením pozice jednotlivých fotografií. Značky (tzv. Coded targets) je obvykle možné vygenerovat přímo nástroji, které fotogrammetrii vykonávají.

V průběhu pořizování fotografií je nutné používat stacionární světlo, v ideálním případě kvalitní externí studiový blesk, který scénu dokonale prosvětlí a umožní ideální seřízení fotoaparátu. Při testování se osvědčilo

- ISO-100,
- clona alespoň f/8,
- délka expozice 1/100 sec,
- žádný nebo fixovaný zoom,
- ohnisková vzdálenost 50 mm,
- maximální možné rozlišení.

Pokud objekt neobsahuje výrazné hrany, například právě zkoumaný obličej, je nutné udržovat malé úhly mezi jednotlivými fotografiemi. V horizontálním směru výborně zafungoval úhel 10° a vertikálně jsou dostačující tři série pod úhlem 45°, 90° a 135°. V případě, že zkoumaný objekt obsahuje výrazné hrany, je možné udržovat větší odstup mezi jednotlivými fotografiemi, doporučováno bývá $70-90°$. Výborně by v takovém případě zafungoval velký počet synchronizovaných fotoaparátů, jejichž použití by zcela eliminovalo jakýkoliv pohyb testovaného subjektu.

4.5 Projekt Tango

Projekt Tango je technologie od Googlu, která přivádí 3D skener a rozšířenou realitu do mobilních telefonů. Z důvodu nutnosti použití přídavného hardwaru byl tento projekt zastaven a nahrazen technologií ARCore. Vyzkoušeno bylo zařízení Lenovo Phab 2 Pro, první model využívající technologie Tango. Byl vytvořen program pro operační systém Android využívající této technologie, který přesouvá souřadnice naskenovaných bodů včetně jejich spolehlivostí do binárního či textového souboru. Tato data se poté zpracují na stolním počítači a vytvoří se z nich 3D model za pomoci Poissonovské rekonstrukce.

5 Získání animační sekvence

Základní objekt zobrazující člověka ve statické póze je již k dispozici z procesu fotogrammetrie a nyní je nutné získat a aplikovat na statický model animace získané pomocí zařízení Microsoft Kinect. Pro naskenování animace byly použity programy o nichž byla řeč již v kapitole 4.2 . Testovací subjekt různě hýbal hlavou a měnil výraz v obličeji, což snímal Kinect a v reálném čase ukládal do binárních souborů, které byly po dokončení skenování převedeny do formátu Wavefront obj. Avšak výsledky tohoto postupu nebyly ani zdaleka uspokojivé a vzhledem ke kvalitě snímacího zařízení nebylo možné výrazy v obličeji vůbec rozeznat.

Proto přišel na řadu postprocessing, který měl skeny upravit natolik, aby bylo možné změny v obličeji rozpoznat. Tento pokus ovšem také nepřinesl kýžené ovoce a bylo nutné hledat jiné alternativy, které by udržely hlavní myšlenku celé práce, především levné vytvoření animace lidského obličeje. Zde přišel zásadní zlom ve směru celé práce, jelikož se ukázalo, že Kinect jakožto skenovací zařízení pro přesnější snímání nestačí a to ani se softwarovou úpravou jeho výstupu. Nakonec se zrak upřel na osvědčenou fotogrammetrii, která zaznamenala úspěch při získání statického objektu.

Získání animace pomocí fotek se ukázalo jako složité, nicméně ne nemožné. Jak bylo řečeno dříve, je nutné aby při snímání fotek zůstal člověk v klidu a nehýbal se. To působí při snímání animace značné problémy, už z principu animace, jakožto série pohybů. Ač zdlouhavý proces, je možné pořídit sekvenci snímků tak, že testovací člověk prostě zůstane v pozici zamrzlý. To se ukázalo jako velký problém, protože udržet nějaký výraz v průběhu celého focení je poměrně náročné a vytvořit celou animační sekvenci si vyžádá svůj čas. Tento problém by bylo možné opět vyřešit použitím většího množství synchronizovaných fotoaparátů s možností vyfotit celou sekvenci snímků najednou, jenž bohužel nebyly k dispozici. S tímto vědomím byly pomocí fotogrammetrie vytvořeny pouze dva objekty s různými výrazy v obličeji, sloužící k otestování programu pro registraci 3D objektů a přehrávače animací.

Pro registraci musí být k dispozici vždy dva objekty, jeden zdrojový a jeden cílový. Hlavním cílem vytvořeného programu je registrovat bystu člověka v základní poloze na deformovanou část obličeje, na čemž bude program ilustrován. Nicméně lze registrovat objekty také opačně, tedy část na celý objekt nebo dokonce i dva celé objekty na sebe. Základní myšlenka registračního programu je podobná algoritmu Hao Li [16], ale je značně zjedno-

dušená. Program očekává nahrání objektů, aby s nimi mohl dále pracovat. Řekněme tedy, že máme nahraný zdrojový objekt s množinou vrcholů S a cílový objekt s množinou vrcholů T, a chceme deformovat zdrojový objekt tak, aby odpovídal cílovému objektu, přičemž $\#S \gg \#T$ a S je blízko T. Základem je registrovat objekty pomocí rigidních transformací a následně provést nerigidní registraci, která objekt deformuje.

5.1 Rigidní registrace

Základním pilířem rigidní registrace je iterative closest point algoritmus a jinak tomu není ani v programu. Jak již bylo popsáno v kapitole 3.1.1, uvnitř algoritmu probíhá mapování bodů mezi objekty, výpočet translace a rotace. Ukončení algoritmu je v aplikaci řešeno pomocí kontroly vzdáleností mezi mapovanými body a počtem iterací. Úryvek implementace algoritmu ICP je uveden v Listingu 5.1.

```
do
{
    Log.Debug("Number␣of␣iterations:␣" + iteration);

    // Map points.
    this.pointMapping.MapPoints(copySourcePoints,
        copyTargetPoints, out mappedSourcePoints, out
        mappedTargetPoints);

    // Compute centroid of target points and translate
        them.
    mappedTargetCentroid = new Centroid(
        mappedTargetPoints);
    Transformation3D.Translate(copyTargetPoints, -
        mappedTargetCentroid.Value);

    // Compute centroid of source points, translate them
        and add translation to a transformation matrix.
    Centroid mappedSourceCentroid = new Centroid(
        mappedSourcePoints);
    Transformation3D.Translate(copySourcePoints, -
        mappedSourceCentroid.Value);
    transformationMatrix = Transformation3D.
        CreateTranslationMatrix(-mappedSourceCentroid.
        Value) * transformationMatrix;
```

```
    // Compute rotation matrix, apply it on source points
        and add it to the transformation matrix.
    Matrix<float> rotationMatrix = this.rotation.
       CalculateRotation(mappedSourcePoints,
       mappedTargetPoints);
    Transformation3D.ApplyTransformation(copySourcePoints
       , rotationMatrix);
    transformationMatrix = rotationMatrix *
       transformationMatrix;
} while (!this.CheckDistance(mappedSourcePoints,
    mappedTargetPoints) && ++iteration < this.
    numberOfIterations);
```

Listing 5.1: Rigidní registrace

Program nabízí výběr mezi dvěma mapovacími algoritmy. Prvním je algoritmus hrubé síly, který pro všechny body vyhledá ten s nejmenší vzdáleností, který k němu přísluší. Složitost tohoto řešení je $O(n^2)$, kde n je rovno počtu bodů. Druhým algoritmem je vyhledání nejbližšího souseda pomocí struktury Kd-strom. Toto řešení ukládá vrcholy do stromu, podle velikosti souřadnice, přičemž souřadnice, jež se bere v úvahu se mění s aktuální úrovní ve stromu. Vyhledání nejbližšího souseda funguje tak, že se hodnota souřadnice bodu, pro který se soused hledá, porovná se všemi potomky uzlu ve stromu a dále se pokračuje na nejbližší. Složitost mapování všech bodů s Kd-stromem je $O(n \log n)$. Předpokladem mapování je téměř stoprocentní překryv jednotlivých objektů a ten s menším počtem bodů je mapován na větší. Pokud by tomu bylo naopak mohlo by dojít k smrsknutí většího objektu na ten menší. Mapovací rutina vrací nejen namapované cílové body, nýbrž také mapované zdrojové body a dále se pracuje pouze s mapovanými, shodně velkými, překrývajícími se částmi.

Z mapovaných bodů je spočítáno těžiště, které je použito k translaci celého objektu. Navíc je v případě zdrojového objektu translační matice přidáno k celkové transformační matici. Po přesunu objektů do počátku soustavy souřadnic lze spočítat matici rotace mezi seznamy mapovaných bodů. Rotace je vypočtena podle Kabschova algoritmu probraného v kapitole 3.1.2. Všechny transformační matice jsou řádu čtyři, aby je bylo možné skládat. Stejně tak body obsahují homogenní složku.

Proces rigidní registrace lze samozřejmě dále vylepšit, například zvětšením počtu složek ve vektoru reprezentujícího bod na meshi. Vektor by se dal zvětšit o normálu v daném bodě či křivost. Vylepšit lze i proces mapování o určení přesnosti nebo spolehlivosti mapované dvojice, dle které by bylo možné odstranit body, které jsou od sebe příliš vzdálené nebo přesně nesedí.

To by navíc odstranilo nutnost mapování menšího objektu na větší a vyřešilo by i částečné pokrytí.

5.2 Nerigidní registrace

Prvním krokem nerigidní registrace je napozicovat na sebe jednotlivé objekty. Na což je využita rigidní registrace, jejíž implementace byla popsána v předchozí kapitole. Tímto krokem se zdrojová a cílová mesh přesunou na sebe, ale není známo, které části objektů se překrývají. Z toho důvodu jsou namapovány registrované, zdrojové body na cílové body, s nimiž se dále pracuje. Algoritmus pro nerigidní registraci vytváří síť řídících bodů v mapované oblasti, také známou jako deformační graf. Řídící body jsou vybírány náhodně a jejich počet je roven α. Každý řídící bod reprezentuje oblast o poloměru β a nemůže být vytvořen uvnitř oblasti, která přísluší jinému bodu. Deformace objektu následně probíhá pouze ve vytyčených oblastech kolem řídících bodů. Toto řešení samotné by ovšem způsobilo značně nerealistické přechody mezi deformovanými a nedeformovanými částmi obličeje, proto je nutné deformace interpolovat pro všechny body na zdrojovém objektu. Pro interpolaci byla vybrána RBF s bázovou funkcí $\phi(r) = e^{-\epsilon r^2}$, využívající konstantu ϵ. Interpolovány jsou translační korekční vektory řídících bodů, reprezentující jejich deformaci.

Pro získání translačního korekčního vektoru pro řídící bod $c_i, i \in \langle 0, \alpha)$ je nezbytné nalézt všechny body v jeho okolí. Tyto body se následně registrují na cílovou mesh pomocí rigidní registrace a odečtením původní hodnoty řídícího bodu od jeho přesunuté verze lze získat korekční vektor.

```
/*
 * Find mapped pairs between registered source points
    and target points (they overlap) because one can be
    smaller than the other and control points should
 * be generated only on the mapped part. For example
    imagine a head as source points and a face as target
     points. Control points should be generated
 * only on the face of source points.
 */
Log.Info("Mapping␣points.");
treeMapping.MapPoints(registeredSourcePoints,
   targetPoints, out mappedSourcePoints, out
   mappedTargetPoints);

// Selects random points (control points) from the
```

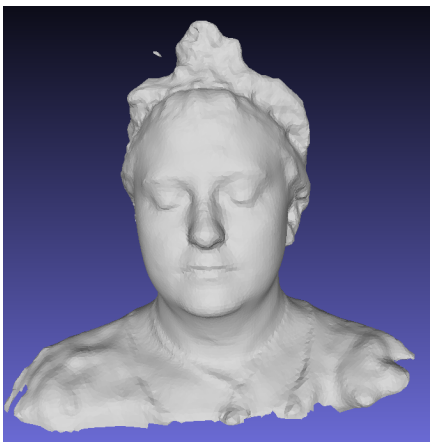
```
   mapped source points. Control points are used for
   mapping of deformed parts of a mesh.
Log.Info("Generating control points.");
ControlPointsGenerator pointsGenerator = new
   ControlPointsGenerator(mappedSourcePoints);
List<Vector<float>> controlPoints = pointsGenerator.
   GetRandomPoints();

/*
 * Compute correction vectors for each control point. It
     is done by searching points in an area around the
    control point. Afterwards these searched points are
 * passed to the rigid registration which returns a
    transformation matrix for related to mapping between
     the area and the target points. This matrix is then
     used
 * to transform the control point and the correction
    vector is computed as difference between transformed
     control point and its original value.
 */
Log.Info("Computing correction vectors.");
List<Vector<float>> correctionVectors = new List<Vector<
   float>>();
for (int i = 0; i < controlPoints.Count; i++)
{
   List<Vector<float>> closePoints = pointsGenerator.
      FindClosePoints(controlPoints[i]);
   Matrix<float> transformMatrix = this.
      rigidRegistration.ComputeRegistrationMatrix(
      closePoints, targetPoints);
   Vector<float> copyPoint = controlPoints[i].Clone();
   copyPoint = transformMatrix * copyPoint;
   correctionVectors.Add(copyPoint - controlPoints[i]);
}

// Interpolate correction vectors for a whole registered
    source point cloud. Each point in the point cloud
   will get own correction vector.
Log.Info("Interpolating correction vectors.");
Rbf rbf = new Rbf();
List<Vector<float>> interpolatedCorrectionVectors = rbf.
   Interpolate(registeredSourcePoints, controlPoints,
```

```
    correctionVectors);
for (int i = 0; i < registeredSourcePoints.Count; i++)
{
    registeredSourcePoints[i] = registeredSourcePoints[i]
        + interpolatedCorrectionVectors[i];
}
```

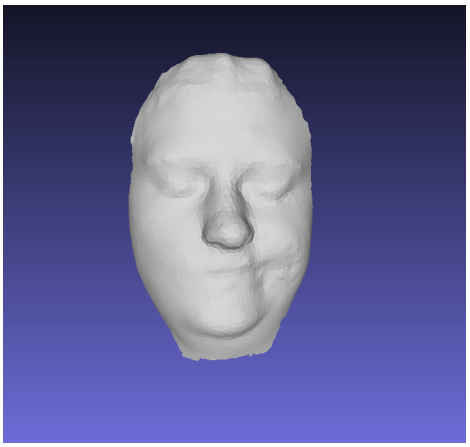
Listing 5.2: Nerigidní registrace

Určit konstanty α, β a ϵ lze pouze testováním různých vstupů. Důležité je zmínit, že konstanty jsou na sobě závislé, je třeba korigovat velikost oblasti úměrně k počtu bodům, jinak by mohlo dojít k tomu, že by program nedokázal všechny řídící body nalézt. Navíc volba velkého počtu bodů značně ovlivňuje rychlost celého procesu. Jako vhodné se ukázaly hodnoty $\alpha = 10\%$ celkového počtu bodů ve zdrojové meshi, $\beta = 1\%$ maximální vzdálenosti mezi body a $\epsilon =$ průměr nejmenších vzdáleností mezi řídícími body $* 1800$.

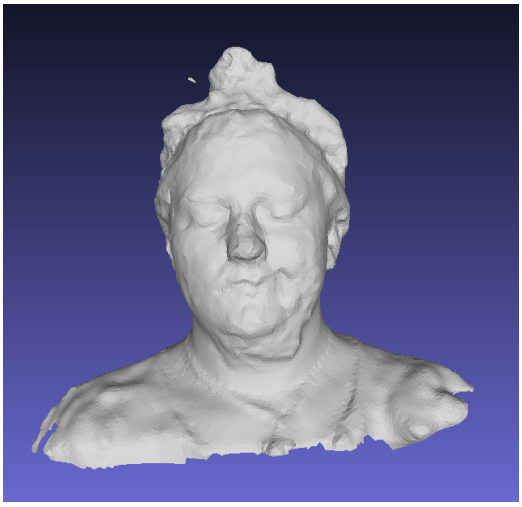
Průběh nerigidní registrace je zobrazen na Obrázku 5.1. Je zde jasně viditelné, že deformace funguje, nicméně nevytváří příliš přirozený profil obličeje. To lze přisoudit jednoduchosti navrženého algoritmu, který má oproti dříve zmíněnému Hao Li značné rezervy. Z toho důvodu bylo na výsledný objekt aplikováno Laplaceovo vyhlazování a následně Poissonova rekonstrukce, která vyplnila vzniklé díry. Výsledek postprocessingu již vypadá velmi dobře a po aplikování textury by se vytratila i lehká ztráta detailů.



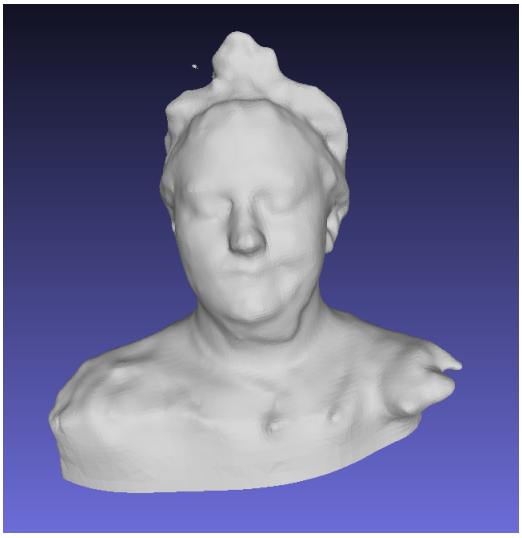
(a) Zdrojový objekt



(b) Cílový objekt



(c) Výsledný objekt po registraci



(d) Výsledný objekt po postprocessingu

Obrázek 5.1: Průběh nerigidní registrace

6 Zobrazení animací

K ukázání vizuální stránky animací a celkové funkčnosti animačního procesu je nutné animace předvést a zobrazit v nějakém vhodném prohlížeči. Pro tento proces byla vyvinuta vlastní javascriptová knihovna umožňující zobrazit animaci ve webovém prohlížeči. Jelikož ale 3D animace obecně zabírají poměrně hodně místa, rozumějte stovky megabytů rozmístěných do podobného počtu souborů, což není pro použití na webu zcela vhodné, bylo nutné před uložením animací na server provést jejich kompresi.

Pro tu byla vytvořena aplikace v programovacím jazyce C# s využitím knihovny pro tvorbu grafického uživatelského rozhraní Windows Forms. Program očekává od uživatele vstup v podobě seznamu souborů ve formátu Wavefront .obj pomocí dialogového okna. Po potvrzení výběru aplikace spustí kompresní algoritmus v `BackgroundWorkeru`, který zajistí funkčnost uživatelského rozhraní v průběhu probíhajícího výpočtu a informuje uživatele o postupu kompresního algoritmu. Kompresní algoritmus je zjednodušenou verzí práce Connectivity Driven Dynamic Mesh Compression [22].

Prvním krokem algoritmu je načtení dat ze všech poskytnutých souborů, prozatím pouze ve formátu .obj, nicméně rozhraní pro další možné formáty je již připraveno. Čtení probíhá podle abecedního pořadí názvu souborů, přičemž z prvního souboru se nahrají body, trojúhelníky a texturovací souřadnice, zatímco z ostatních souborů se čtou pouze body, jelikož se trojúhelníky ani texturovací souřadnice v průběhu animace nemění.

Komprese souřadnic bodů probíhá pomocí výpočtu trajektorií mezi jednotlivými snímky animace. Pro tento výpočet je nejprve nutné převést body do matice označené **D**

$$\mathbf{D} = \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,v-1} \\ y_{0,0} & y_{0,1} & \cdots & y_{0,v-1} \\ z_{0,0} & z_{0,1} & \cdots & z_{0,v-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,v-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,v-1} \\ z_{1,0} & z_{1,1} & \cdots & z_{1,v-1} \\ \cdots & \cdots & \cdots & \cdots \\ x_{f-1,0} & x_{f-1,1} & \cdots & x_{f-1,v-1} \\ y_{f-1,0} & y_{f-1,1} & \cdots & y_{f-1,v-1} \\ z_{f-1,0} & z_{f-1,1} & \cdots & z_{f-1,v-1} \end{pmatrix}$$

kde v označuje počet vrcholů a f počet snímků v animaci, což ukazuje že

matice $\mathbf{D}$ je řádu $3f \times v$. Z matice $\mathbf{D}$ se následně spočte průměrná trajektorie $\mathbf{m}$, což je vektor obsahující průměr všech sloupců matice $\mathbf{D}$.

$$\mathbf{m} = \begin{pmatrix} \bar{x}_0 \\ \bar{y}_0 \\ \bar{z}_0 \\ \bar{x}_1 \\ \bar{y}_1 \\ \bar{z}_1 \\ \dots \\ \bar{x}_{f-1} \\ \bar{y}_{f-1} \\ \bar{z}_{f-1} \end{pmatrix}$$

Od každého sloupce matice $\mathbf{D}$ se nyní odečte průměrná trajektorie, čímž vznikne matice $\mathbf{S}$ a vypočítá se její autokorelační matice

$$\mathbf{A} = \mathbf{S} \cdot \mathbf{S^T}$$

Nad autokorelační maticí se provede výpočet vlastních čísel a vlastních vektorů, z jejichž matice $\mathbf{EV}$ se vybere určitý počet vlastních vektorů, jimž přísluší největší vlastní čísla. Počet vybraných vlastních vektorů určuje uživatel při zapnutí komprese a ovlivňuje tak její ztrátovost. Pokud vezmeme v úvahu, že vlastní vektory jsou v matici $\mathbf{EV}$ uloženy po sloupcích, potom finální krok celého algoritmu bude

$$\mathbf{EV_n^T} \cdot \mathbf{S} = \mathbf{C}$$

pro něž platí, že $\mathbf{EV_n}$ je podmatice matice $\mathbf{EV}$, kde n je rovno počtu vybraných vlastních vektorů, a $\mathbf{C}$ je matice kontrolních trajektorií typu $n \times v$. Implementace algoritmu pro kompresi bodů 3D objektu, za pomoci knihovny MathNet pro maticové výpočty, je uvedena v Listing 6.1.

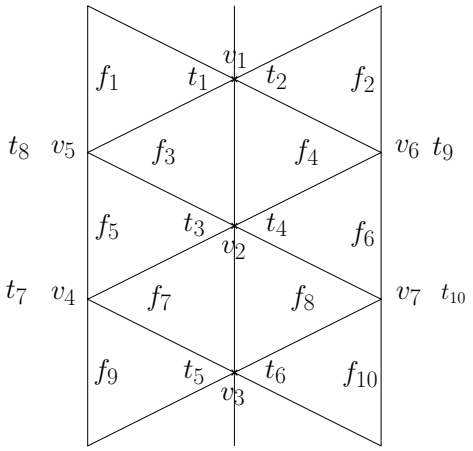
```
this.averageTrajectory = matrix.RowSums().Divide(matrix.
    ColumnCount);

foreach (var column in matrix.EnumerateColumnsIndexed())
{
    var subtColumn = column.Item2 - this.
        averageTrajectory;
    matrix.SetColumn(column.Item1, subtColumn);
}
```

```
var autocorelation = matrix * matrix.Transpose();
var evd = autocorelation.Evd(Symmetricity.Symmetric);
var eigenValues = evd.EigenValues;
var eigenVectors = evd.EigenVectors;

if (eigenVectors.ColumnCount - controlTrajectoriesCount
   < 0)
{
   this.subEigenVectors = eigenVectors.SubMatrix(0,
      eigenVectors.RowCount, 0, eigenVectors.ColumnCount
      );
} else
{
   this.subEigenVectors = eigenVectors.SubMatrix(0,
      eigenVectors.RowCount, eigenVectors.ColumnCount -
      controlTrajectoriesCount, controlTrajectoriesCount
      );
}

this.controlTrajectories = this.subEigenVectors.
   Transpose() * matrix;
```

Listing 6.1: Kompresní algoritmus

Nakonec se zkomprimovaná data uloží do binárního souboru v novém formátu jenž obsahuje:

- velikost vektoru průměrné trajektorie (int),
- vektor průměrné trajektorie (float),
- počet řádků podmatice vlastních vektorů (int),
- počet sloupců podmatice vlastních vektorů (int),
- podmatice vlastních vektorů (float),
- počet řádků matice řídících trajektorií (int),
- počet sloupců matice řídících trajektorií (int),
- matice řídících trajektorií (float),
- počet trojúhelníků (int),
- seznam trojúhelníků {index vrcholu (int), index bodu textury (int)},



Obrázek 6.1: Spojnice textury

- počet souřadnic textury (int),
- seznam souřadnic textury (float).

Webový prohlížeč přijímá tento binární soubor, provádí jeho dekompresi a zobrazuje ho pomocí Javascriptového API WebGL, vytvořeného pro využití 3D grafiky na webu. Dekomprese probíhá opačným směrem než byl představen výše a to podle následujících dvou vzorců.

$$\mathbf{EV_n} \cdot \mathbf{C} = \mathbf{S}$$

$$\mathbf{S} + \mathbf{m} = \mathbf{D}$$

kde $\mathbf{EV_n}$ je podmatice matice $\mathbf{EV}$, u níž je n rovno počtu vybraných vlastních vektorů, $\mathbf{C}$ je matice kontrolních trajektorií typu $n \times v$ a součet matice $\mathbf{S}$ s $\mathbf{m}$ znamená přičtení vektoru $\mathbf{m}$ ke každému sloupci matice $\mathbf{S}$. Potom výsledná matice $\mathbf{D}$ obsahuje původní body, byť ovlivněné drobnou chybou způsobenou ztrátovostí komprese. Tímto krokem a přečtením zbytku souboru získá prohlížeč data o všech vrcholech a texturovací souřadnice. Pro správné vykreslení 3D objektu a tím pádem i celé animace je zapotřebí ještě získat normály a přiřadit texturovací souřadnice k jednotlivým vrcholům.

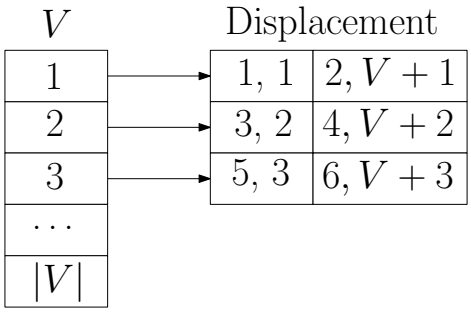
Pro výpočet normály vrcholu je nutné spočítat normály příslušné všem trojúhelníkům, v nichž se vrchol nachází a spočítat jejich průměr. Toho lze docílit i sečtením jednotlivých normál a normalizací výsledku, která je nutná pro výpočet stínování v GPU. Problémem je nalezení trojúhelníků příslušných k určitému vrcholu. Způsobů jak tento problém řešit existuje více, od v praxi nepoužitelných, až po relativně dobré přístupy o složitosti $O(N)$ kde N představuje počet trojúhelníků.

Do skupiny nepoužitelných algoritmů se v tomto případě řadí procházení hrubou silou, kdy pro každý vrchol $v_i \in V$ procházíme celou množinu trojúhelníků F a hledáme, kde se vrchol nachází. Toto řešení má složitost $O(N^2)$. Další metoda využívá třídění klíčů, které se skládají ze souřadnic x, y, z a spolu s identifikátorem trojúhelníka a pořadím vrcholu v trojúhelníku se setřídí a vytvoří struktury pro vytvoření trojúhelníkové sítě. Tento způsob je náročný na implementaci a jeho složitost $O(N \log N)$ je stále příliš velká k nějakému rozumnému využití. Další možností by bylo využít geometrický hashing, jenž je založen na vytvoření otisku (hashe) ze souřadnic x, y, z. Při vhodné volbě hashovací funkce lze dosáhnout složitosti až $O(N)$, nicméně její implementace je příliš náročná. Naštěstí lze využít postupu, který je snadný naimplementovat a jeho složitost je také rovna $O(N)$. Tento algoritmus prochází množinu trojúhelníků F a pro každý trojúhelník f_i spočte normálu, kterou následně připočte do pole na index shodný s indexem vrcholu v trojúhelníku f_i. V pseudokódu lze tuto metodu výpočtu normál najít v algoritmu 6.2.

```
Normal3D[] n = new Normal3D[];
for (i = 0; i < |F|; i++)
{
   ni = calculate normal
   n[F[i].v1] += ni;
   n[F[i].v2] += ni;
   n[F[i].v3] += ni;
}

for (i = 0; i < n.length; i++)
{
   n[i] = normalize(n[i]);
}
```

Listing 6.2: Výpočet normál

Normály je nutné určit pro všechny snímky animace zvlášť, a tak je algoritmus volán pro každý frame. U přerovnání texturovacích souřadnic je nezbytné, aby každý vrchol měl přidělený právě jeden texturovací bod, který by bylo možné zpracovat pomocí shaderů na grafické kartě. Problém ovšem nastává v místě, kde se setkávají různé konce textury. Tato situace je vyobrazena na Obrázku 6.1, kde body v_1, v_2 a v_3 leží právě na spojnici dvou konců textury. Jak lze vypozorovat, tyto vrcholy budou mít určeny dva různé texturovací body. Příkladem je například vrchol v_2, který v trojúhelnících f_2, f_3 a f_4 bude mít přidělen texturovací bod t_2, zatímco v trojúhelnících f_1, f_6 a



Obrázek 6.2: Datová struktura pro mapování textur

f_5 texturovací bod t_1. Pro odstranění nejednoznačnosti přidělených textur je nutné každý vrchol rozdělit na tolik vrcholů, kolik je třeba texturovacích bodů. Takto nově vytvořený vrchol bude mít stejné souřadnice, stejnou normálu, ale odlišné texturovací souřadnice. Pro vyřešení této problematiky vznikl algoritmus využívající speciální datové struktury vyobrazené pro trojúhelníkovou síť z Obrázku 6.1 na Obrázku 6.2. Jedná se tedy o dynamické dvojrozměrné pole, jehož délka je rovna počtu vrcholů. Každý vrchol je reprezentován indexem v poli, na němž se vyskytuje pole obsahující instance třídy `Displacements` dvěma atributy, indexem textury a pozicí, na kterou byl bod s danou texturou přesunut. Algoritmus prochází přes všechny vrcholy a příslušné texturovací body v trojúhelnících a postupně pro každý i-tý vrchol prochází pole `Displacementů`, kde při nalezení shodného indexu textury zkontroluje index, na který měl být vrchol přesunut a pokud se tento přesun nerovná původnímu vrcholu, změní hodnotu v trojúhelníku. Když je velikost pole `Displacementů` nulová, uloží element s původním vrcholem i texturou a přesune texturovací souřadnice na index vrcholu. Když pole shodný index textury neobsahuje a jeho délka není nulová, vytvoří nový vrchol s původními texturovacími souřadnicemi a normálou.

Dekomprese, výpočet normál i přerovnání textur jsou výpočetně poměrně náročné úkony, které vytěžují webovou stránku, což působí její výpadek a ignorování akcí od uživatele. Aby k takovému chování nedocházelo, jsou dekomprese i ostatní výpočty vykonávány asynchronně pomocí `WebWorkeru`. Jakmile má prohlížeč k dispozici veškerá potřebná data, vytvoří shadery, přiřadí jim vertexy, texturovací souřadnice, normály a obraz textury, jenž se bude aplikovat na meshe v animaci. Shadery se odešlou na GPU, kde se zpracují a vykreslí výsledný obraz. Animace probíhá nekonečným voláním kreslící funkce, v níž dochází ke střídání vrcholů a normál, příslušných různým snímkům, jenž se odesílají na grafický čip.

7 Závěr

Hlavním tématem práce bylo najít levnější alternativu pro tvorbu animací, díky zařízení Microsoft Kinect For Windows 2.0. Bohužel práce ukázala, že levné zařízení, jakým je Kinect for Windows 2.0, nezvládá srovnat kvalitu svého výstupu s ostatními metodami a to ani za použití různých přístupů při skenování nebo postprocessingu. Jedinou spásou Kinectu by se stal hardwarový upgrade. V tomto směru je ale příliš pozdě, vzhledem k tomu, že se výrobce rozhodl ukončit produkci a tak do budoucna žádná hardwarová vylepšení Kinect nečekají. Na druhou stranu, při úvaze ceny za kterou byl Kinect prodáván, nejsou výsledné skeny tak špatné, pro realistickou vypadající animaci jsou nicméně nepoužitelné.

Práce nezůstala pouze u Kinectu a ukázala i další možnosti skenování 3D objektů. Byla čest moci si vyzkoušet zařízení jakým je Artec Eva, jejíž kvality skoro dosahují výše její pořizovací ceny. Překvapením v porovnání se zmíněným zařízením se stal proces fotogrammetrie, který dokázal po zjištění správného know-how a vhodných nástrojů, dosáhnout podobných kvalit jako Artec Eva za desetinu jejích nákladů a byl zvolen pro získání základního 3D modelu. Do budoucna by bylo možné prozkoumat trh s 3D skenery do větších detailů a vyzkoušet větší množství zařízení, kterých bezpochyby existuje nepřeberné množství. Jejich pořizovací cena ovšem často tuto možnost hatí. S tím se ale musí u takovéto techniky bohužel počítat.

I když se nepodařilo využít Kinect pro vytvoření animace, práce se nezastavila. Rozhodli jsme se využít úspěchů fotogrammetrie v oblasti tvorby 3D modelů a jednotlivé snímky animace získali touto metodou. Pro každý snímek bylo nutné nafotit sérii fotografií. Doba focení se v porovnání s normálním skenerem značně protáhla a pro člověka, který musí zůstat nehnutě v jedné pozici, se mohlo stát focení značně nekomfortní. Řešením by se stalo využití několika synchronizovaných fotoaparátů se stejnými parametry, snímající osobu z různých úhlů. Toto řešení v práci bohužel nebylo vyzkoušeno, protože nebylo k dispozici více stejných fotoaparátů. Proto bylo sáhnuto ke zkrácení animace a bylo vytvořeno jen pár snímků pro demonstraci celého procesu a prokázání jeho funkčnosti.

Finální proces se skládal ze získání základního modelu a snímků animace pomocí fotogrammetrie, deformace základního modelu pomocí každého snímku a zobrazení výsledné animace ve webovém prohlížeči, umožňující prohlížení kompletní 3D animace a pohyb v prostoru. Pro deformaci byla vytvořena aplikace, která využívá rigidní a nerigidní registrace a umožňuje

zobrazit výsledek. V práci bylo popsáno několik konkurenčních způsobů, řešící registrace dvou ploch, z důvodu ukázání různých postupů a vizuálního porovnání výsledků. Program nedosahuje takových kvalit jako Hao Li [16], ale po postprocessingu nad daty se mu alespoň blíží. Směrů, kterými by šlo řešení vylepšit je několik, bylo by možné vylepšit mapování volbou vícesložkového vektoru, reprezentujícího bod v prostoru, což by zlepšilo mapování objektů a tím i urychlilo průběh celého algoritmu. Dále by mohly být navrženy optimalizační kritéria, které by vylepšily kvalitu počítaných transformací nebo v RBF interpolaci využít místo translačních korekčních vektorů kvaterniony, které fungují obecně lépe.

Nakonec byl vytvořen pomocný program, který komprimuje sérii .obj souborů, jenž tvoří animaci a ukládá je tak, aby je bylo možné uložit na server v jednom souboru. Ten pak zpracovává vytvořený prohlížeč, provádí dekompresi a zobrazuje výsledky hlavního jádra celé práce.

Literatura

[1] *Camera auto-calibration*, 2018. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/w/index.php?title=Camera_auto-calibration&oldid=820142336.

[2] *Camera resectioning*, 2018. Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/w/index.php?title=Camera_resectioning&oldid=820894730.

[3] *Artec Eva* [online]. Artec3D, 2017. [cit. 2017/12/17]. Dostupné z: https://www.artec3d.com/3d-scanner/artec-eva#specifications.

[4] Audette, M. A. – Ferrie, F. P. – Peters, T. M. An algorithmic overview of surface registration techniques for medical imaging. *Medical Image Analysis*. 2000, 4, 3, s. 201–217. ISSN 1361-8415. doi: https://doi.org/10.1016/S1361-8415(00)00014-1. Dostupné z: http://www.sciencedirect.com/science/article/pii/S1361841500000141.

[5] Bendels, G. H. – Schnabel, R. – Klein, R. Detecting holes in point set surfaces. *Journal of WSCG*. 2006, 14, s. 89 – 96.

[6] Böhm, J. Fotogrammetrie. *Vysoká škola báňská-technická univerzita Ostrava, Univerzita Ostrava, Hornicko - geologická fakulta, Institut geodézie a důlního měřičství, přednáškové texty*. 2002. Dostupné z: http://igdm.vsb.cz/igdm/materialy/Fotogrammetrie.pdf.

[7] Duchon, J. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In Schempp, W. – Zeller, K. (Ed.) *Constructive Theory of Functions of Several Variables*, s. 85–100, Berlin, Heidelberg, 1977. Springer Berlin Heidelberg. ISBN 978-3-540-37496-1.

[8] Feldmar, J. – Ayache, N. Rigid, affine and locally affine registration of free-form surfaces. *International Journal of Computer Vision*. May 1996, 18, 2, s. 99–119. ISSN 1573-1405. doi: 10.1007/BF00054998. Dostupné z: https://doi.org/10.1007/BF00054998.

[9] Fraser, C. S. Automatic camera calibration in close range photogrammetry. *Photogrammetric Engineering & Remote Sensing*. 2013, 79, 4, s. 381–388. Dostupné z: https://www.researchgate.net/publication/275579723_Automatic_Camera_Calibration_in_Close_Range_Photogrammetry.

[10] Hanzl, V. Fotogrammetrie 1 - Teoretické základy fotogrammetrie. *Vysoké učení technické v Brně, Fakulta stavební*. 2006. Dostupné z: http://fast.darmy.net/opory%20-%20III%20Bc/GE15-Fotogrammetrie_I--M01-Teoreticke_zaklady_fotogrammetrie.pdf.

[11] Ježek, F. Diferenciální geometrie: Pomocný učební text – díl II. 2005. Dostupné z: https://www.fd.cvut.cz/personal/voracsar/GeometriePG/PGR020/DG_Jezek02.pdf.

[12] Ju, T. Fixing Geometric Errors on Polygonal Models: A Survey. *Journal of Computer Science and Technology*. Jan 2009, 24, 1, s. 19–29. ISSN 1860-4749. doi: 10.1007/s11390-009-9206-7. Dostupné z: https://doi.org/10.1007/s11390-009-9206-7.

[13] Kabsch, W. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*. Sep 1976, 32, 5, s. 922–923. doi: 10.1107/S0567739476001873. Dostupné z: https://doi.org/10.1107/S0567739476001873.

[14] Kazhdan, M. – Hoppe, H. Screened Poisson Surface Reconstruction. *ACM Trans. Graph.* July 2013, 32, 3, s. 29:1–29:13. ISSN 0730-0301. doi: 10.1145/2487228.2487237. Dostupné z: http://doi.acm.org/10.1145/2487228.2487237.

[15] Kazhdan, M. – Bolitho, M. – Hoppe, H. Poisson Surface Reconstruction. 2006, s. 61–70. Dostupné z: http://dl.acm.org/citation.cfm?id=1281957.1281965.

[16] Li, H. – Sumner, R. W. – Pauly, M. Global Correspondence Optimization for Non-Rigid Registration of Depth Scans. *Computer Graphics Forum*. 27, 5, s. 1421–1430. doi: 10.1111/j.1467-8659.2008.01282.x. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01282.x.

[17] Li, H. et al. Temporally Coherent Completion of Dynamic Shapes. *ACM Trans. Graph.* February 2012, 31, 1, s. 2:1–2:11. ISSN 0730-0301. doi: 10.1145/2077341.2077343. Dostupné z: http://doi.acm.org/10.1145/2077341.2077343.

[18] Luhmann, T. *Close Range Photogrammetry: Principles, Techniques and Applications*. Whittles, 2006. ISBN 9781870325509.

[19] Luhmann, T. – Fraser, C. – Maas, H.-G. Sensor modelling and camera calibration for close-range photogrammetry. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2016, 115, s. 37 – 46. ISSN 0924-2716. doi: https://doi.org/10.1016/j.isprsjprs.2015.10.006. Dostupné z:

http:
//www.sciencedirect.com/science/article/pii/S0924271615002361.

[20] *Kinect hardware* [online]. Microsoft, 2017. [cit. 2017/12/17]. Dostupné z: https://developer.microsoft.com/en-us/windows/kinect/hardware.

[21] Penny, W. et al. *Statistical Parametric Mapping: The Analysis of Functional Brain Images*. Elsevier Science, 2011. ISBN 9780080466507.

[22] Vása, L. – Skala, V. CODDYAC: Connectivity Driven Dynamic Mesh Compression. *2007 3DTV Conference*. 2007, s. 1–4.

[23] Říha, K. – Hujka, P. *Epipolární geometrie* [online]. [cit. 2018-01-29]. Dostupné z: http://www.elektrorevue.cz/clanky/05017/index.html.

[24] Šárka, V. Aplikace epipolární geometrie. *Sborník*. 2004, 24, s. 236–242. Dostupné z:
https://www.fd.cvut.cz/personal/voracsar/epipolar.pdf.

Uživatelská dokumentace Static scanning

Aplikace slouží ke snímání nehybných 3D objektů zařízením Microsoft Kinect. Aplikace funguje na principu interpolování většího množství dat, jenž jsou nasbírány za delší časový interval. Aplikace je na přiloženém CD uložena v adresáři Static_scanning, který obsahuje tři podadresáře, bin, src a test. Src obsahuje všechny zdrojové soubory psané v jazyce C# a ve složce test se vyskytují jednotkové testy kontrolující správnost zdrojového kódu. Bin obsahuje všechny potřebné soubory pro spuštění aplikace, hlavně pak soubor 3DScanning.exe, kterým se aplikace zapíná.

Static scanning je okenní aplikace, jejíž dominantou je panel se záložkami Point Cloud a Depth Frame. Point Cloud ukazuje 3D náhled skenované scény, který lze zobrazit stisknutím tlačítka Náhled. Na druhé záložce si lze prohlédnout hloubkovou mapu skenované scény generovanou v reálném čase. Ovládání programu je poměrně jednoduché. Je možné nastavit minimální a maximální hloubku snímání, počet interpolací (počet nasbíraných framů), typ meshe a další. Snímání meshe se zapíná tlačítkem Generovat mesh. Aplikace ukládá výstupní soubor do vybrané složky. Hloubková mapa na záložce Depth Frame funguje jako automatická kontrola dostupnosti zařízení Kinect. Pokud nezobrazuje žádný výstup je jasné, že Kinect není připojen. Když uživatel v tomto stavu spustí generování meshe, dojde k čekání na příchozí data z Kinectu. Z toho důvodu aplikace nabízí možnost zastavit proces generování.

Uživatelská dokumentace Dynamic scanning

Dynamic scanning je skupina utilit zabývající se skenováním pohyblivých cílů. Obsahuje skenovací programy Scanning a GScanning, přičemž první jmenovaný je spustitelný pouze z příkazové řádky pomocí příkazu

» Scanning.exe <numberOfFrames>

a druhý obsahuje grafické uživatelské rozhraní. Oba programy fungují v principu stejně. Zvolí se počet hloubkových map, které se následně uloží do binárního souboru. Soubory zpracovává poslední aplikace Generating, která je převádí na obj soubory. Soubory se ukládají do adresáře, ze kterého byl program spuštěn.

Uživatelská dokumentace Tango

Tango je mobilní aplikace sloužící pro skenování okolí za použití technologie Projekt Tango, jenž byla vyvíjena společností Google. Aplikace byla navržena speciálně pro zařízení Lenovo Phab 2 Pro, které technologií Projekt Tango disponuje. Aplikace by neměla jít nainstalovat na zařízení bez zmíněné technologie z důvodu absence potřebných knihoven. Aplikace je funkční od Androidu 19 až po 26. Ovládání aplikace je velmi jednoduché. Funguje jako běžný fotoaparát. Zobrazí se náhled skenované scény a tlačítko pro odstartování snímání. Jakmile se začne snímat objeví se počet aktuálně naskenovaných objektů a stejným tlačítkem lze proces ukončit. Následně je uživatel přesunut na obrazovku s výstupem, která ukazuje umístění souborů. Obsahem souborů jsou body s hodnotami x, y, z a spolehlivostí daného bodu. Apk soubor pro instalaci je uložen v Tango/bin a zdrojové soubory v adresáři Tango/src.

Uživatelská dokumentace Registration

Registration je program sloužící k registrování dvou 3D ploch. V aplikaci je nutné nejprve importovat modely, s kterými se následně pracuje. Lze tak učinit v menu pomocí Soubor->Nahrát modely. Panel s nahranými modely lze zobrazit pomocí položky v menu Zobrazit->3D modely. Registrace se provádí v záložkách Nástroje. Při zvolení rigidní nebo nerigidní registrace se objeví formulář pro zadání parametrů algoritmů a zvolení zdrojového a cílového modelu. Doporučuji 200 iterací pro obě registrace. Po dokončení registrace lze modely uložit pomocí Soubor->Uložit model nebo kliknutím pravého tlačítka myši v seznamu modelů. Ukládá se model, který je vybrán v panelu se seznamem modelů. Zaškrtávací políčko u modelů v seznamu slouží pouze k zobrazí 3D objektu ve vieweru, nemá nic společného s ukládáním. Ve složce Registration/data jsou uložena testovací data pro rigidní a nerigidní registrace.

Uživatelská dokumentace Compression

Compression slouží pro kompresi 3D animace do takového formátu, který umožní nahrání a uložení animace na webový server. V aplikaci stačí zvolit seznam obj souborů a kliknout na tlačítko Compress. Výsledný soubor se uloží do adresáře, ze kterého byl program spuštěn. Důležité je zmínit několik parametrů, které musí zvolené objekty splňovat:

1. Musí mít stejný počet bodů.
2. Pokud obsahují texturovací souřadnice berou se v úvahu pouze u prvního zvoleného objektu.

Uživatelská dokumentace Viewer

Viewer je javascriptová knihovna, která slouží pro zobrazení 3D animací. Zpracovává soubory ve formátu .3ba, vytvořené pomocí programu Compression. Ve složce Viewer jsou uloženy všechny potřebné soubory. Index.html je zde pouze pro zobrazení přehrávače a nemá žádnou jinou funkci. Přehrávač je nutné spouštět na serveru, například apache, jelikož přehrání animace je přesunuto do jiného vlákna, což lze provést pouze na serveru. Díky tomu není blokován zbytek webové stránky. Adresář Viewer/data obsahuje několik testovacích animací, které lze zobrazit nastavením cesty a textury v souboru index.html. Pokud animace texturu nemá, cesta k souboru musí být prázdná.