Fakulta Aplikovaných věd
Západočeská Univerzita v Plzni
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Morfování geometrických objektů v povrchové reprezentaci

Plzeň, 2006 Martina Málková

***Tady bude originál zadání práce (s razítkem)***

**Abstract**

Morphing is usually defined as a process of continuously transforming one geometric into another. This work concentrates on a multimorphing problem, which has not been clearly defined yet. Here it will be considered as problem of finding final object as combination of more than two input objects by using their weights. Two sub-problems occur: first, how to define the weights and second, how to use them to obtain the resulting object. When defining weights, because of the possibility of having sum of weights equal to one, the use of barycentric coordinates is described.

The problem of using the weights to obtain resulting object is mainly the aim of this work. First, it is examined how contemporary programs as 3dmax deal with it. The main problem here is how to make objects with the same number of vertices. Solving of this problem is then analyzed. Two solutions of the problem are considered: using method of remeshing with same parameters for all input objects, or using topology merging technique, which enables us remake two objects to the same number of vertices. An algorithm for using these programs for more than objects is shown.

**Poděkování**

Na prvním místě bych chtěla poděkovat Doc. Dr. Ing. Ivaně Kolingerové nejen za odbornou pomoc při zpracování práce, ale také za ochotu pomoci se všemi problémy, nejen těmi týkajícími se vlastní práce. Stejné díky patří i Jindřichu Parusovi, který mi poskytl obrovské zázemí ohledně této práce.

Dále bych ráda poděkovala svým rodičům, kteří mi poskytli jak fyzické zajištění během studií, tak psychické zázemí.

Velké díky patří také mým přátelům, bez kterých bych se nikdy nedostala tam, kde teď jsem, a kteří byli vždy ochotni mi pomoci s mými problémy a strastmi.

Poslední, ale ne nejmenší dík patří mému příteli Michalu Rejdovi, který práci vytiskl a vyřídil potřebné formality, zatímco jsem byla na studiích v Anglii, čímž mi umožnil absolvovat bakalářskou zkoušku v správném termínu.

# Obsah

Prohlašuji, že jsem bakalářskou/diplomovou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne ……. ……..

# 1. Úvod

Jindřich Parus vypracoval v roce 2003 diplomovou práci zabývající se morphingem 3D geometrických objektů. V této práci rozebírá, jakým způsobem lze morphovat mezi dvěmi objekty. Tématem multimorphingu se zde zabývá pouze okrajově. Mým úkolem bylo se na tento problém zaměřit a více ho prozkoumat.

Morphing bývá často definován jako proces spojité přeměny z jednoho geometrického objektu do druhého. Tato práce se zaměřuje multimorphing, neboli morphing více než dvou objektů. Multimorphing už není přesně definovaný pojem, ze zadání mé bakalářské práce vyplynul tento přístup: Multimorphing budeme chápat tak, že objekt, který vytváříme, se skládá určitým způsobem z různých objektů: např. 30% z krychle, 20% z koule a 50% z válce. Z důvodů snadného získávání modelů, ukládání a zobrazování se práce zaměřuje na multimorphing trojúhelníkových sítí. Tudíž i veškerá teorie a praxe popsaná v textu se týká trojúhelníkových sítí.

Morphing všeobecně má mnoho využití, Jindřich Parus ve své práci [1] zmiňuje využití v oblasti animací, pro tvorbu speciálních efektů, v oblasti designu nebo herního průmyslu. Tato práce se zaměřuje na využití morphingu pro vizualizaci vícerozměrných dat, kompresi obrazu a výhru v šachách.

## 1.1 Rozvržení práce

V teoretické části se zabývám těmito problémy:

- Jak při morphingu, tak multimorphingu se setkáváme s potřebou stejného počtu vrcholů morphovaných objektů, což je velmi omezující. Je zde popsána technika, jak tento problém řešit.
- Při morphingu dvou těles určujeme, jakým dílem bude ve výsledném objektu zastoupeno první těleso a jakým druhé. Tento postup si můžeme představit jako úsečku s jedním tělesem na jedné straně a druhým na druhé, jak přesouváme bod po úsečce, mění se výsledný objekt v závislosti na vzdálenostech od objektů (krajů úsečky). Při multimorphingu ale potřebujeme zadávat zastoupení více než dvou objektů. Je zde uveden způsob, jak rozšířit toto zadávání pro více objektů.
- Jak lze v praxi multimorphing využít popisuji v poslední podkapitole.

V praktické části jsem se soustředila na:

- Průzkum, jak se s problémem multimorphingu vypořádává 3ds max.
- Algoritmus, jak výstupy J.Paruse (vytvářející ze dvou těles s různým počtem vrcholů dvě tělesa se stejným počtem vrcholů) použít jako vstup mého programu pro víc objektů.
- Program pro jiný způsob získávání více těles se stejným počtem vrcholů.
- Vlastní program pro multimorphing.

# 2. Problém stejného počtu vrcholů

Jak je napsáno v [1] (pro dva objekty, ale lze zobecnit i pro více), výpočet morphingu je často rozdělován do tří základních kroků:

- nalezení korespondence mezi objekty (nejvhodnější je zde korespondence vrcholů)
- vygenerování objektu, který "sdílí" topologii všech zdrojových těles
- interpolace poloh vrcholů výsledného objektu (lze různými způsoby, tímto problémem se zabývám v 8.2 – Interpretace vah)

Nalezení korespondence vrcholů mezi objekty není snadná záležitost, tento problém je více rozebrán v [1] - kapitola 3.3.

Existují dva základní přístupy jak řešit tento problém – slučování topologie a remeshing. Rozbor těchto dvou metod lze nalézt v [3]. Já se ve svém zpracování zaměřila pouze na metodu remeshingu.

Naší snahou je převzorkovat objekty, mezi kterými chceme morphovat – tedy nalézt pokud možno pravidelně rozmístěné body na povrchu objektů, ty pak pospojovat do trojúhelníkové sítě.

Vstupem naší metody je trojrozměrné těleso. Nejprve hledáme jeho vzor v parametrické doméně – pro 3d uzavřená tělesa je jí jednotková koule. Pro každý vrchol tělesa tedy máme jeho obraz na povrchu jednotkové koule.

Jednotkovou kouli můžeme pokrýt pravidelnou mřížkou. Pak, protože zobrazení objektu na povrch jednotkové koule je bijektivní (každý vrchol má právě jeden obraz), můžeme bijektivně zobrazit každý bod tělesa.

Postup je takový, že pro každý bod b, navzorkovaný na parametrické doméně, zjistím, do jakého trojúhelníku T na parametrické doméně padl. Spočítám jeho barycentrické souřadnice v rámci tohoto trojúhelníku (označím je $w_0$, $w_1$, $w_2$). Zjistím obraz trojúhelníku T (na tělese, označím ho T'). Nový bod spočítám pomocí barycentrických souřadnic $w_0$, $w_1$, $w_2$ ale vzhledem k T'.

Pokud postup aplikujeme na všechny body, získáme remeshované těleso skládající se z takto vypočítaných bodů.

Pokud takto remeshujeme více těles podle stejné mřížky, budeme mít všechna tělesa o stejném počtu vrcholů, navíc budeme znát korespondenci jednotlivých vrcholů. Pokud jsme použili pravidelnou mřížku, bude i síť výsledných těles pravidelná.

Remeshing je pouze aproximací původního tělesa, kvalita této aproximace závisí na tom, jak blízko budou body supermeshe vrcholům původního tělesa (Platí zde ale, že pro větší počet bodů získáme lepší aproximaci).

Metoda není dobrá pro objekty s ostrými hranami, u kterých je velmi nepravděpodobné, že by se vrcholy přiblížily původním natolik, aby zůstaly ostré hrany zachovány.

Výhodou metody je její jednoduchost implementace.

Ukázky výstupů vlastního programu na remeshing lze nalézt v kapitole 7.1 – Výsledky.

# 3. Definice vah

Existují dva hlavní způsoby, jak zadávat váhy:

První a nejjednodušší je manuálně – každému tělesu přiřadíme takovou váhu, jakou bychom si představovali, že se bude projevovat ve výsledném objektu. Toto zadávání vah má jeden nedostatek – součet vah se v různých případech liší, může přesáhnout i 100%. To má většinou za následky jiný vzhled objektu, než bychom si představovali.

Tento způsob je ale vhodný pro tělesa, mezi kterými jsou jen lokální změny (např. pro případ ruky s různě ohnutými prsty, hlavy s různými výrazy tváře – viz ukázky v kapitole 8.4). Pro případ ruky s různě ohnutými prsty je pro výsledný objekt se všemi prsty ohnutými potřeba mít 100% všech vstupních objektů.

Alternativou je zaručit, že součet vah bude vždy 100%. Pro morphing dvou objektů znamená, že použijeme pro zadávání vah jednotlivých těles úsečku – procento zastoupení tělesa je určeno vzdáleností bodu od konce úsečky reprezentující těleso.

Jak tomu ale bude při více objektech? Zde se nabízí rozšíření úsečky na n-úhelník a k zajištění jedničkového součtu použití barycentrických souřadnic.

Ve článku [4] definují (po překladu) barycentrické souřadnice takto:

Nechť $q_1, q_2, ..., q_n$ je $n$ vrcholů konvexního polygonu $Q$ v $R^2$, kde $n \geq 3$. Dále nechť $p$ je libovolný bod uvnitř $Q$. Barycentrickými souřadnicemi bodu $p$ vzhledem k $\{q_j\}_{j=1..n}$ nazýváme každou n-tici $(\alpha_1, \alpha_2, ..., \alpha_n)$ závisející na $Q$ a $p$ takovou, že platí:

- $p = \sum_{j \in [1..n]} \alpha_j q_j$, kde $\sum_{j \in [1..n]} \alpha_j = 1$ **(3.1)**
- $\{\alpha_j\}_{j=1..n}$ musí být nekonečně diferencovatelné vzhledem k $p$ a vrcholům $Q$. To zajistí hladkost změny koeficientů $\alpha_j$ při změně jakéhokoli vrcholu $q_j$.
- $q_j \geq 0$ pro $\forall j \in [1..n]$. To zajišťuje, že $\alpha_j \in \langle 0,1 \rangle \ \forall j \in [1..n]$

Barycentrické souřadnice pro trojúhelník, běžně používané v různých oblastech počítačové grafiky, lze spočítat podle následujících rovnic (převzatých z [4]):

$$\alpha_1 = \frac{A(p, q_2, q_3)}{A(q_1, q_2, q_3)}$$
$$\alpha_2 = \frac{A(p, q_3, q_1)}{A(q_1, q_2, q_3)} \quad \textbf{(3.2)}$$
$$\alpha_3 = \frac{A(p, q_1, q_2)}{A(q_1, q_2, q_3)}$$

kde $A(p, q, r)$ označuje plochu trojúhelníku *(p,q,r)*.

Lze dokázat, že tyto souřadnice splňují podmínky (3.1).

Pro n-úhelník, kde $n > 3$, už ale tyto jednoduché rovnice nelze použít. Ve článku [4] však popisují jinou – o něco složitější – metodu, jak pro n-úhelník barycentrické souřadnice spočítat – pomocí následujících dvou rovnic:

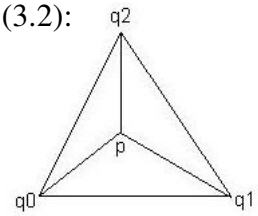
Nejprve spočítáme váhy:

$$w_j = A(q_{j-1}, q_j, q_{j+1}) * \prod_{k \notin \{j, j+1\}} A(q_{k-1}, q_k, p) \qquad \textbf{(3.3)}$$

Pomocí nich pak spočítáme barycentrické souřadnice:

$$\alpha_j = \frac{w_j}{\sum_k w_k} \qquad \textbf{(3.4)}$$

Lze ukázat, že pro n = 3 se tyto rovnice stanou rovnicemi (3.2):




$$w_0 = A(q_2, q_0, q_1) * A(q_1, q_2, p)$$
$$w_1 = A(q_0, q_1, q_2) * A(q_2, q_0, p)$$
$$w_2 = A(q_1, q_2, q_0) * A(q_0, q_1, p)$$

$$\alpha_0 = \frac{A(q_2, q_0, q_1) * A(q_1, q_2, p)}{A(q_2, q_0, q_1) * A(q_1, q_2, p) + A(q_0, q_1, q_2) * A(q_2, q_0, p) + A(q_1, q_2, q_0) * A(q_0, q_1, p)}$$

Protože $A(q_2, q_0, q_1) = A(q_0, q_1, q_2) = A(q_1, q_2, q_0)$, lze tento vztah přepsat na:

$$\alpha_0 = \frac{A(q_2, q_0, q_1) * A(q_1, q_2, p)}{A(q_2, q_0, q_1) * [A(q_1, q_2, p) + A(q_2, q_0, p) + A(q_0, q_1, p)]}$$

Pokud $A(q_2, q_0, q_1) \neq 0$:

$$\alpha_0 = \frac{A(q_1, q_2, p)}{A(q_1, q_2, p) + A(q_2, q_0, p) + A(q_0, q_1, p)}$$

Protože (z obr.) $A(q_0, q_1, q_2) = A(q_1, q_2, p) + A(q_2, q_0, p) + A(q_0, q_1, p)$

$$\alpha_0 = \frac{A(q_1, q_2, p)}{A(q_0, q_1, q_2)}$$

Stejným způsobem lze vyjádřit i $\alpha_1, \alpha_2$.

# 4. Využití morphingu

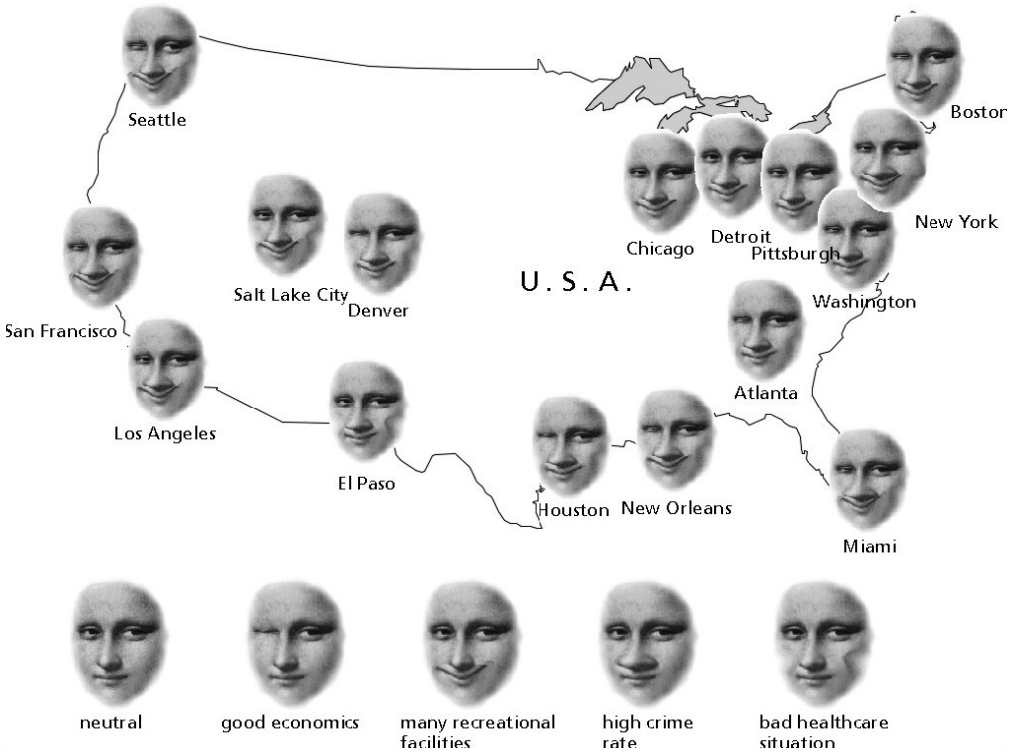
Využití morphingu lze nalézt v nejrůznějších oblastech: od nejčastějších – filmy, design – po oblasti, kde bychom morphing běžně nehledali. Na tyto oblasti bych se chtěla zaměřit v následujících kapitolách.

## 4.1 Pro vizualizaci vícerozměrných dat

V článku [5] popisují Müller a Alexa způsob, jak použít multimorphing pro vizualizaci informace:

Pro vizualizaci informací potřebují jeden neutrální objekt, a pak pro každý měnící se atribut objekt představující maximální hodnotu atributu. Váhy těchto objektů jsou pak určeny aktuálními hodnotami atributů.

Na obr. 4.1 ukazují příklad vizualizace informací o městech v U.S.A. Ve spodní části obr. jsou použité objekty, na mapě už zmorphované objekty podle daných hodnot atributů. Neutrální tvář Mony Lisy představuje minimální hodnoty atributů (špatná ekonomika, málo rekreačních zařízení, vysoká zločinnost, špatný stav zdravotnictví). Ostatní tváře představují jednotlivě maximální hodnoty atributů.




**Obr. 4.1 Tvář Mony Lisy jako reprezentace hodnocení měst v USA (obr. převzat z [5])**

Stejným způsobem by se mohl multimorphing použít na jakýkoliv způsob vizualizace dat.

## 4.2 Pro kompresi obrazu

Ve své práci [6] popisuje L.K.Siew jak použít morphing pro kompresi obrazu. Při přenosu videa v reálném čase je potřeba velká šířka pásma linky, proto se objevuje snaha o kompresi přenášeného obrazu. Při běžném přenosu se posílá okolo 15–25 rámců za sekundu. Siew popisuje, jak k této kompresi využít morphing – pošlou se jen „klíčové“ rámce. Rámce, které jsou vynechány, nahradí cílová stanice pomocí morphingu. Dále píše, že tímto způsobem lze dosáhnout přibližně 80% komprese. Ve zbylé části textu popisuje metody, jak hledat body na obraze klíčové pro morphing, a jak vytvořit chybějící rámce.

## 4.3 Pro výhru v šachách

Toto je hodně oddychové téma, ale přesto bych ho zde chtěla zmínit. Na internetových stránkách [7] se můžete dočíst techniku, jak použít morphing pro nenápadné výměny figurek při hře šachů, aniž by si toho soupeř všiml. Nejprve zjistí, kam je zaměřena pozornost soupeře. Jsou definovány dvojice figurek, mezi kterými jde morphovat, aniž by se to dalo bez přímé pozornosti zpozorovat (např. střelec a pěšec, všeobecně dvojice, které jsou si podobné). Potom už stačí jen vybrat, kde by bylo nejvhodnější morphing provést, a dostatečně pomalu (přesto také dost rychle, aby se mezitím soupeř nerozhodl) provést morphing.

Na uvedených stránkách lze také stáhnout video s ukázkami.

# 5. Pokusy v 3ds max

V této kapitole bych chtěla prezentovat své pokusy při práci s morphingem v 3ds max (verze 7). Pro morphing se zde používá tzv. „morpher modifier“. Jak ho použít, popíši v první podkapitole. Dále už se budu zabývat svými vlastními výsledky, ukázkami a problémy.

## 5.1 Morpher modifier a postup jeho použití

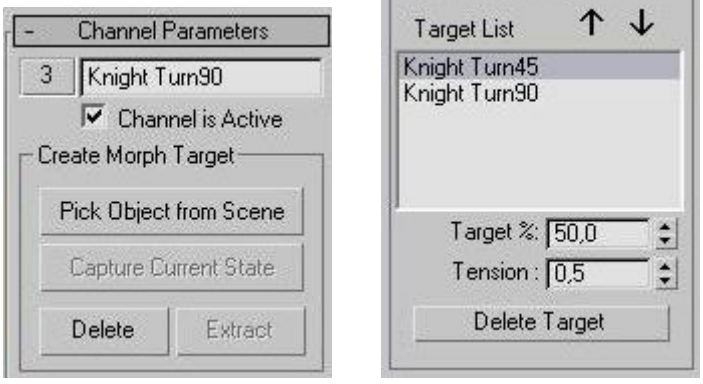
Nejdříve musíme mít vytvořený určitý počet objektů, mezi kterými chceme morphovat. Tyto objekty musí mít stejný počet vrcholů a stejnou topologii, což je velké omezení, možností jeho odstranění se budu zabývat v kapitole 5.4 (problémy).

Jeden objekt vybereme jako bázový a v **Modifier List** mu přidáme „morpher modifier“. Objeví se okno s volbou cílových objektů (= objektů, do kterých chceme morphovat, tedy tzv. „morph targets“). Všechny zbylé objekty přidáme do cílových objektů pomocí **Load Multiple Targets**.

Pak stiskneme na dolním panelu volbu „set key“, a pomocí časového posuvníku a procent v cílových objektech nastavujeme, kolik bude v jakém čase procent kterého objektu. Následnou interpolaci vah už zajišťuje 3ds max sám.

Pozn.: 3ds max nabízí ještě volbu tzv. „progressive morph“:

Pokud chceme morphovat z jednoho objektu do druhého přes několik jiných objektů, podle předchozího postupu bychom objekty přidali do jednotlivých cílových objektů. Progressive morph nám ale nabízí přidat do cílových objektů jen některé z mezilehlých objektů, a ty pak ovlivňovat ostatními. Například pro tři objekty: Do cílových objektů přidáme pouze první a třetí, a pak zadáme, že třetí bude ovlivňován druhým.




**Obr. 5.1, 5.2: Progressive morph**

To nám 3ds max umožňuje pomocí **channel parametres,** kde pomocí **pick object from the scene** vybereme, kterým objektem daný cílový ovlivníme. Ve výpisu objektů zvaném **target list** pak budou objekty dva – ten, který chceme ovlivnit, a ten, který jsme přidali. Ovlivňovaný objekt přesuneme pomocí šipek dospod výběru.

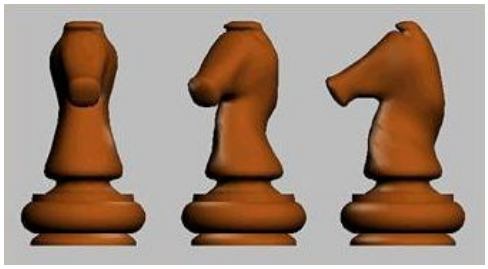
Bez použití **progressive morph** je trajektorie vrcholů lineární (tedy v naší animaci se bod pohybuje mezi jednotlivými body po úsečkách). Pokud ho ale použijeme, výběrem dalšího objektu do **target list** způsobíme přidání dalšího interpolačního bodu k trajektorii – vznikne křivka vyššího stupně.

Tuto volbu jde ale dobře využít, pouze pokud jsou si dané dva objekty hodně podobné (např. ruka s nataženými prsty a ruka s jedním prstem lehce pokrčeným).

V následující kapitole je příklad na progressive morph, převzaný z tutoriálu 3ds max 7. Tento příklad lze také najít na CD v ukazky\3dsmax\progressive.max.

## Příklad – progressive morph

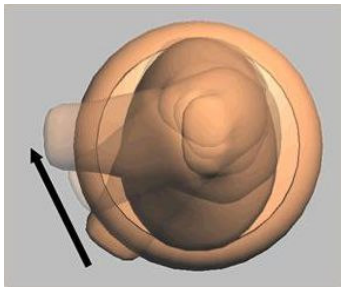
Tento příklad je převzat [8]. Jde o morphing mezi 3 objekty šachové figurky koně (obr. 5.1)



**Obr. 5.3 – Vstupy morphingu (převzat z [8])**

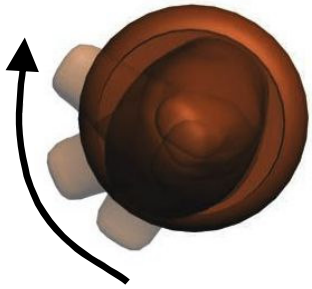
Při morphingu se snažíme docílit výsledku, že nejprve bude hlava koně v poloze 1 (na obr. 5.1 vlevo), pak se bude otáčet a skončí v poloze otočená o devadesát stupňů.

Bez použití progressive morphu se bude hlava otáčet po lineární trajektorii, takže dojde k deformaci hlavy koně – jak je vidět na obrázku 5.2.



**Obr. 5.4 – Deformace hlavy (převzat z [8])**

Ale pokud použijeme progressive morph, výsledek dopadne jak je na obrázku 5.3.



**Obr. 5.5 – Po použití progressive morph**

## 5.2 Výsledky

Bohužel jsem kvůli nedostupnosti zdrojových kódů 3ds max nedokázala vyzkoumat, jak počítá výsledné těleso. Při práci na svém programu jsem ale zjistila, že stejné výsledky jako v 3ds max dosáhnu, když použiji následující rovnici:

$$M = B + \sum_{i=1}^{n} (T_i - B) * w_i \qquad \textbf{(5.2.1)}$$

Kde:

- $M$ … výsledný objekt (produkt morfování)
- $B$ … bázový objekt
- $T$ … množina objektů, mezi kterými morphujeme
- $w$ … váhy objektů
- $n$ … počet objektů

*(Rovnice je převzata z článku [2])*

Vizuální výsledky multimorphingu v 3ds max nebyly příliš uspokojivé, rozumné výsledky vycházely pouze při vhodné volbě vah (procent zastoupení jednotlivých těles) – stačilo, aby součet všech vah byl menší nebo roven jedné. U čtyř různých těles už ani vhodná volba vah výsledky nezlepšovala.

Za důvod špatných výsledků považuji to, že v 3ds max je morpher navržen pro animace, kde se mění pouze určitá oblast tělesa, ostatní části zůstávají stejné (viz výše zmiňovaná ruka ohýbající prst).

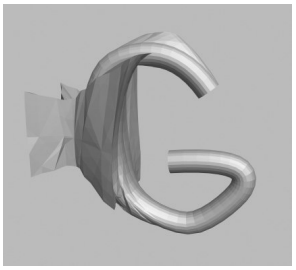
## 5.3 Ukázky výsledků

### Tři objekty

**Objekty pro morphing:** krychle, G, válec

**Výsledek morphingu při vahách:**

- válec 25%
- krychle 25%
- G 50%

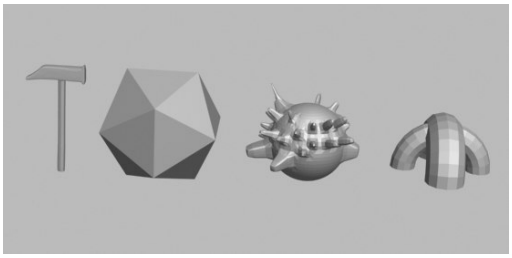


**Už ne tak pěkný výsledek při vahách:**

- válec 100%
- krychle 100%
- G 100%

### Čtyři objekty

**Objekty pro morphing:** kladivo, desetistěn, beruška, dvouválec



**Výsledek při vahách:**

- kladivo 100%
- desetistěn 20%
- beruška 10%
- dvouválec 10%

## 5.4 Problémy

Velkým problémem se stala podmínka **stejného počtu vrcholů objektů** a **stejné topologie** (vrcholy jsou stejně pospojovány hranami). Různé objekty o stejném počtu vrcholů a stejné topologii je těžké vytvořit, možné je snad pouze vytvořit je upravováním jednoho objektu. To by ale nebylo úplně to pravé, bylo potřeba tento problém vyřešit.

K jeho řešení mi pomohly dva programy Jindřicha Paruse: MorphingProject a Embedding. Embedding umožňuje parametrizaci objektu ve formátu .asc, .ase, .tri. Objekty a tuto parametrizaci (pro každý objekt) potřebuje pak program MorphingProject, který spočítá supermesh vybraných dvou těles *(přesný popis viz [1])*. Ten je pak možné exportovat do formátu wrml, podporovaném v 3ds max.

Tyto programy mají několik omezení, které není těžké překonat. S těmito programy lze vytvořit supermesh pouze pro dvě tělesa. To pro mé pokusy nestačilo, vytvořila jsem tedy algoritmus, jak pomocí těchto programů získat pro N těles s různými počty vrcholů N těles se stejným počtem vrcholů.

# 6. Získání objektů se stejným počtem vrcholů

## 6.1 Myšlenka

### Obecně

Máme N výchozích objektů a proceduru pro tvorbu supermeshe ze 2 vstupních těles. Potřebujeme získat N objektů se stejným počtem vrcholů.

Myšlenkou algoritmu je použít proceduru nejprve na dvojice z N objektů (pokud dvojice nevyjdou, lichý objekt zůstane stejný do dalšího kroku). Po použití procedury máme N objektů, kde každé dva jsou stejné topologie (i stejný počet vrcholů), každý z nich jen v jiné konečné poloze morfování) – například pro krychli a válec by měly výsledné dva objekty stejný počet vrcholů, ale první by po zobrazení vypadal jako krychle a druhý jako válec.

V dalším kroku už pracujeme s N objekty z prvního kroku. Představme si, že každé dvě dvojice vytvoří čtveřici. V rámci této čtveřice si dvojice vymění své partnery (Nové dvě dvojice vzniknou jako první z první dvojice s prvním z druhé dvojice, druhý s druhým). Na takto prohozené dvojice opět použijeme proceduru. Tím nám vznikne N objektů, kde každé čtyři mají stejný počet vrcholů.

Ve třetím průchodu uděláme totéž, ale pro osmice (nové dvojice budou jako první z jedné čtveřice s prvním z druhé čtveřice, druhý s druhým, třetí s třetím, čtvrtý s čtvrtým). Tímto způsobem pokračujeme. Kroků (průchodů) je $\log_2 n$. Po posledním průchodu máme N objektů se stejným počet vrcholů, každý v poloze, kde po zobrazení vypadá stejně jako dřív (krychle vypadá opět jako krychle atp.).

### Složitost

Za elementární operaci budu považovat použití procedury na tvorbu supermeshe ze 2 těles.

Algoritmus má $\log_2 n$ průchodů. V každém průchodu použije proceduru (n/2)-krát (pro každou dvojici objektů). Tedy celkově $T(n) = \frac{1}{2} n \cdot \log_2 n$. Algoritmus je tedy složitosti $\Theta(n \cdot \log_2 n)$.

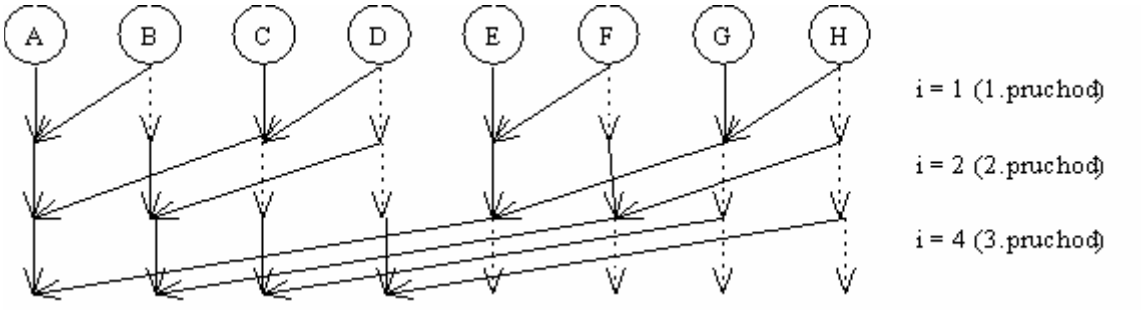
## Pro osm těles

Dále ukáži příklad algoritmu pro osm těles.

Na obr. 6.1 použitím procedury na A a B vznikne supermesh, morphováním do tvaru A získám první objekt pro 2. průchod, morphováním do tvaru B druhý objekt.

pozn.: objekty A-H budou uloženy v poli A[]

pozn. 2: čárkovaná šipka znázorňuje objekt, který vznikl v páru s druhým (pouze morphováním do příslušného tvaru)




**Obr. 6.1 – Postup vytváření 8 objektů o stejném počtu vrcholů**

Na obr. 6.2 vidíme vzniklé dvojice objektů, na které se použije procedura (vypsány jen indexy dvou morphovaných objektů):

⊕ … použití procedury pro vytvoření supermeshe ze dvou objektů

| i = 1 | i = 2 | i = 4 |
|---|---|---|
| 0 ⊕ 1 | 0 ⊕ 2 | 0 ⊕ 4 |
| 2 ⊕ 3 | 1 ⊕ 3 | 1 ⊕ 5 |
| 4 ⊕ 5 | 4 ⊕ 6 | 2 ⊕ 6 |
| 6 ⊕ 7 | 5 ⊕ 7 | 3 ⊕ 7 |

**Obr. 6.2 – Kroky algoritmu**

## 6.2 Algoritmus

```
int i; // proměnná určující počet průchodů (kroků) algoritmu
int j,k; // pro rozdělení průchodů na logické celky
int l; // určení indexu ve výstupním poli

/* A .. pole se vstupními objekty, B .. pole s výstupními objekty
* po každém cyklu se prohodí (B přesunu do A, takže bude volné pro další
zapisování */
Object [] B;A;

// n je počet objektů, mezi kterými budu morfovat
for ( i = 1; i < log2n; i = 2*i )
{
   l = 0; // na začátku tohoto cyklu začíná i kariéra pole B
   for ( j = 0; j < n/(2*i); j++ )
   {
      for( k = 0; k < i; k++ )
      {     /* průchody pro 2^k prvků - pokud není prvků přesně, může být
pole tohoto indexu null - v takovém případě tam pouze druhý prvek
překopíruji a nemorfuji nic */
          if ( A[2*i+k+j] == null )
          {
              B[l++] = A[2*i*j];
              B[l++] = A[2*i*j];
           }
      /* to samé pro druhý prvek - oba dva prvky neošetřuji, v takovém
případě v poli zůstane null */
           else if( A[2*i*j] == null)
           {
               B[l++] = A[2*i+k+j];
               B[l++] = A[2*i+k+j];
            }
            else
            { /* a vlastní výpočet indexů mezi kterými morfuji, ⊕ označuje
výpočet supermeshe */
                B[l++] = A[2*i*j] ⊕ A[2*i+k+j];
                B[l++] = A[2*i+k+j] ⊕ A[2*i*j];
             }
       }
   A ← B; // přesunu B do A
 }
```

# 7. Program pro Remeshing

V dalších kapitolách popisuji své vlastní programové práce na daném tématu. Zde se soustředím na implementaci problémů popsaných v teoretické části. Nejprve byla implementována metoda remeshingu popsaná v kapitole 2, čímž vznikl program Remeshing. Následně byl vytvořen hlavní program pro Multimorphing, řešící problémy definicí vah a interpretací vah a umožňující uživateli vytvářet animace z dosažených výsledků.

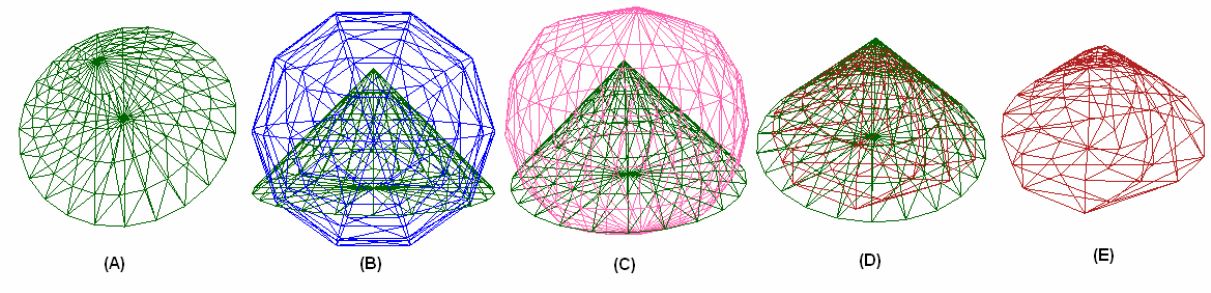
Metodu remeshingu popsanou v kapitole 2 jsem implementovala v jazyce C#, v prostředí .NET Framework 2.0 (pomocí DirectX 9.0 August).

Ve svém programu vygeneruji nejprve mřížku – tedy kouli – podle zadaných parametrů (rozlišení ve směru poledníků a rovnoběžek). Tyto parametry ovlivňují počet vrcholů mřížky, a tedy i kvalitu výsledné aproximace.

V uživatelském rozhraní umožňuji načíst více objektů zároveň, ale metodu aplikuji na objekty jeden po druhém. Načítané objekty musí být ve formátu .tri, každý se stejnojmenným souborem .par obsahujícím parametrizaci daného objektu na jednotkovou kouli (tuto parametrizaci lze získat z programu Embedding3D od Jindry Paruse)

Pro každý bod mřížky:

- zjistím, v jakém trojúhelníku (označím T) v parametrizaci tělesa leží
- vypočítám barycentrické souřadnice bodu vzhledem k T
- zjistím obraz trojúhelníku T (označím ho T')
- nový bod spočítám pomocí barycentrických souřadnic $w_0$, $w_1$, $w_2$, ale vzhledem k T'




**Obr. 6.2 – Ukázka práce programu**

**(A) původní těleso (B) původní těleso a mřížka (koule) (C) původní těleso a jeho parametrizace (D) původní těleso a remeshované těleso (E) remeshované těleso**

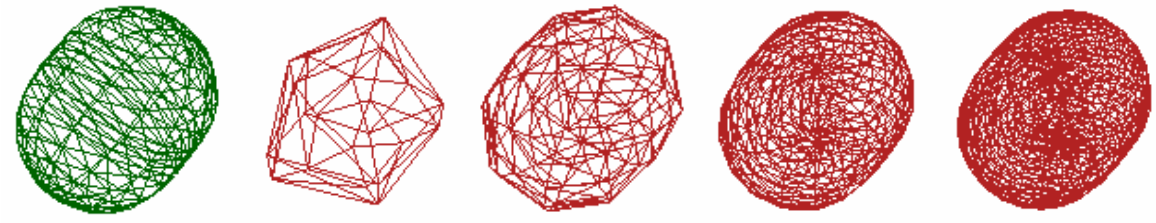
Program vykazuje dobré výsledky pro objekty, které jsou přibližně stejně široké jako dlouhé, pro podlouhlé objekty není schopen vypočítat rozumně použitelnou aproximaci. Prozatím nevím, zda je to chybou metody, programu, nebo špatné parametrizace.

## 7.1 Výsledky

Počty vrcholů jsou psány ve formátu *n x m,* kde n je rozlišení v jednom směru na kouli (poledníky), m druhý směr (rovnoběžky). Výsledný objekt má stejný počet vrcholů jako mřížka (koule).

Na ukázkách si lze povšimnout, že pro některé objekty (většinou hladké) jsou výsledky remeshingu uspokojivé. Pokud ale objekt obsahuje ostré hrany, pomocí remeshingu už ostré hrany nelze u výsledného tělesa docílit. Také se program špatně vypořádává s podlouhlými objekty, nejspíše kvůli příliš malým trojúhelníkům v jejich parametrizacích.

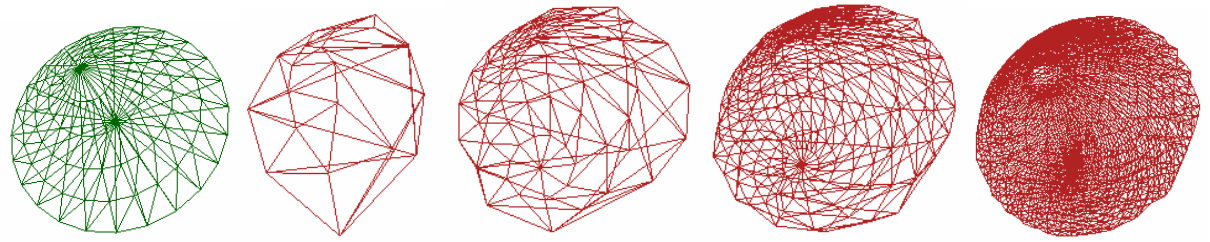
**kapsule**



originál 5x5 10x10 20x20 30x30

V případu kapsule lze dosáhnout uspokojivé aproximace již při malém počtu vrcholů koule.

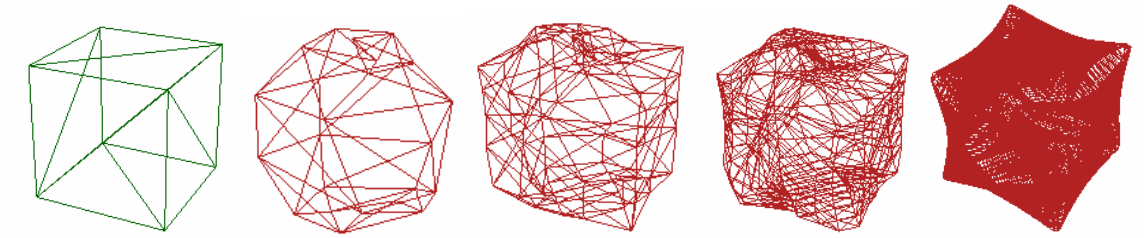
**kužel**



originál 5x5 10x10 20x20 50x50

Zde už je problém s ostrou hranou mezi pláštěm a podstavou kužele. Zvyšování počtu vrcholů mřížky sice aproximaci zlepší, výsledná hrana už ale nikdy nebude ostrá.

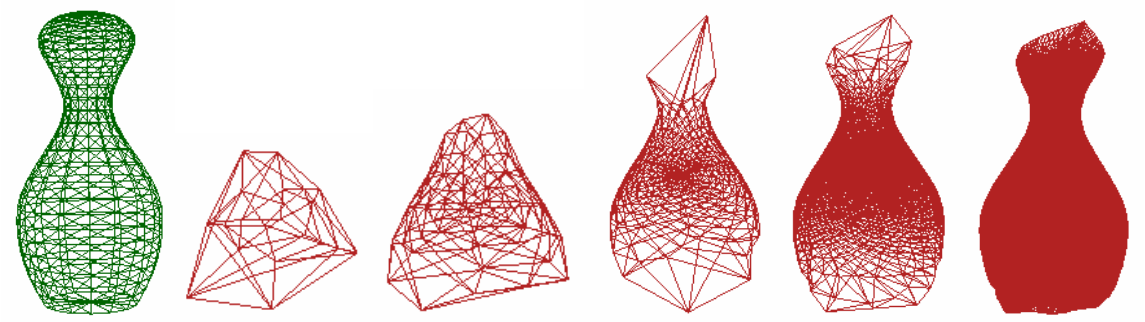
**krychle**



originál 5x5 10x10 20x20 100x100

Zde je vidět, jak u objektu s více ostrými hranami nemá remeshing příliš smysl, výsledný objekt nikdy nebude dobrou aproximací vstupního.

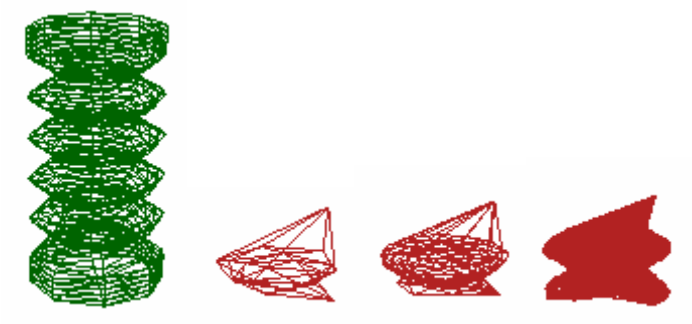
**kuželka**



originál 5x5 10x10 20x20 60x60 60x100

Zde je vidět, jak má program problémy s podlouhlými tělesy. To lze dát za příčinu nejspíš špatné parametrizaci, kde na výsledné kouli vzniknou velmi malé trojúhelníky. Pak pravděpodobnost, že se vrcholy mřížky „strefí“ do těchto trojúhelníčků, je prakticky nulová.

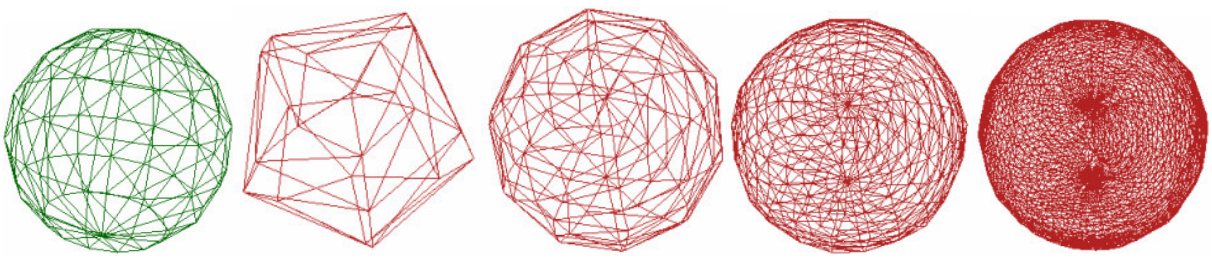
**hose**



originál 5x5 10x10 200x50

Zde narážíme na stejný problém jako u kuželky, pouze je více zřejmý.

**koule**



originál 5x5 10x10 20x20 50x50

Koule je zde pro ukázku, co vznikne, když vezmeme libovolnou kouli, a remeshujeme jí podle mřížky, která je také koulí.

# 8. Program pro Multimorphing

Pro pokusy s multimorphingem jsem vytvořila vlastní program (v jazyce C#, .NET Framework 2.0, pomocí DirectX 9.0 August).

V něm se snažím prozkoumat dva pohledy na multimorphing:

- jak jsou výsledky ovlivněny definicí vah (zde zkouším dva způsoby – buď jsou váhy definovány tak, aby jejich součet dal jedna, nebo jsou definovány libovolně)
- jakými způsoby váhy interpretovat pro získání výsledného objektu (to je např. výše zmíněná rovnice)

## 8.1 Definice vah

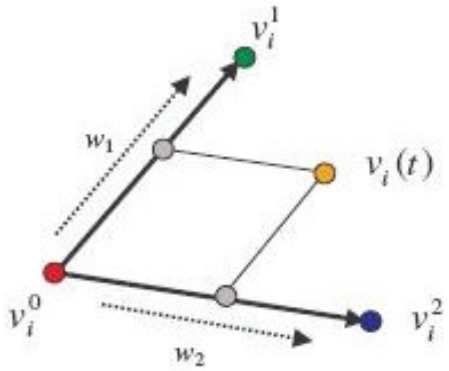
Teoretické pozadí popisuji v kapitole 3. Zde rozeberu, jakým stylem se s definicí vah v programu vypořádávám, dále pak výhody a nevýhody z hlediska uživatele.

**První způsob – pomocí barycentrických souřadnic**: Při použití barycentrických souřadnic vytvořím n-úhelník reprezentující tzv. „morphing space“ – každý vrchol n-úhelníku je pomyslně jeden objekt. Uživatel pak pouze kliknutím vybere bod z n-úhelníku, v případě, že bodem bude jeden z vrcholů, výsledným objektem se stane objekt přiřazený tomuto vrcholu, čím dál bude bod od vrcholu, tím menší bude zastoupení tohoto objektu ve výsledném. Obecně řečeno, pro bod se spočítají barycentrické souřadnice v n-úhelníku a podle nich výsledné váhy objektů (jak je v kapitole 3).

**Druhý způsob:** Při použití vah, které jsou navzájem nezávislé, lze zvolit každou z vah v intervalu < 0, 100 >. Tento způsob je výhodný pro měnění vah objektů, které se liší pouze lokálně – zde totiž celkový součet vah 1 plně nevyhovuje, pokud chceme například u ruky (kde vstupními objekty jsou ruce s různými ohnutými prsty) mít ohnuté všechny prsty, musí mít všechny objekty 100%.

## 8.2 Interpretace vah

Váhy interpretuji dvěma způsoby. Prvním je rovnice **(5.2.1)**. Tato rovnice používá *(n-1)* z daných *n* vah, protože první objekt je považován za bázový, a u něj se váha nemění (vždy je 100%). Pro tři vrcholy si můžeme rovnici představit podle obr. 8.1.




**Obr. 8.1 – Interpretace vah pomocí sčítání vektorů (obr. přejat z [1])**

Vrchol s horním indexem 0 je vrchol bázového objektu, druhé dva jsou vrcholy cílových objektů; w1, w2 jsou zvolené váhy.

Z pokusů na mém programu vyplývá, že tato interpretace vah přináší uspokojivé výsledky převážně tehdy, když součet zadaných vah nepřesáhne 1 (tedy 100%). Jen v případě, že se vstupní objekty liší jen lokálně, lze dosáhnout příjemných výsledků i při vyšším součtu.

Druhým způsobem, jak interpretuji váhy, je rovnice **(5.2.1)** upravená tak, aby nebyl vyžadován žádný bázový objekt. V závěru jde o vážený průměr. Výsledná rovnice vypadá takto:

$$M = \frac{\sum T_i * w_i}{\sum w_i} \qquad \textbf{(8.2.1)}$$

Tato interpretace bohužel jako vedlejší efekt způsobuje, že pokud mají oba objekty váhu 100%, výsledek vypadá stejně, jako kdyby měl každý z nich váhu 50%. Je to ale jen jedna z interpretací tohoto stavu. To, jak by měl výsledný objekt vypadat, si totiž nejde dost dobře ani představit. 100% krychle a zároveň 100% koule by čistě teoreticky ani neměla existovat. Ale v případě, že se objekty mezi sebou liší jen lokálně, už si 100% obou představit dokážeme. Například u ruky ohýbající prsty: pokud jeden objekt má ohnutý malíček, druhý prsteníček a třetí prostředníček, výsledkem 100% všech tří objektů bude ruka s ohnutými třemi prsty.

## 8.3 Animace

Dále můj program umožňuje při zadání klíčových bodů animace mezi těmito body. Uživatel zadá váhy v časech t1, t2, ... , tn. Mezilehlé váhy program počítá pomocí lineární interpolace.

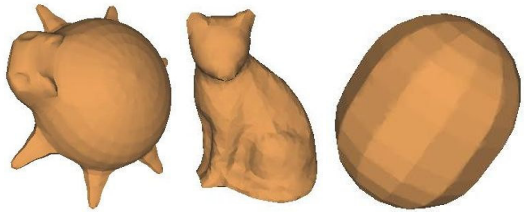
Nastavení animace lze uložit do xml souboru, a pak použít pro libovolné objekty (musí jich být stejný počet jako v nastavení, ale to je minimální omezení).

Program také umožňuje export animace do videa, video je ale ve výsledku dost nekvalitní, protože tato volba vyžaduje výkonný počítač, každý snímek zobrazený na plátně totiž uloží do paměti, na konci animace pak snímky poskládá do videa uživatelem vybrané komprese.

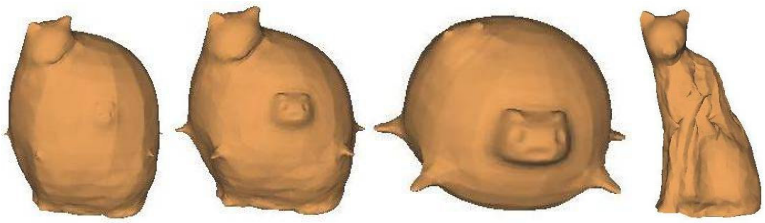
Všechny ukázky výsledků jsou vytvořeny použitím rovnice (1) v odstavci 5.2. U této metody „nezávisí“ na váze bázového objektu – má vždy 100%. To ale neznamená, že by se ve výsledku jako stoprocentní projevil. Pouze doplňuje váhy ostatních objektů do 100% (pokud jsou všechny ostatní váhy nulové, sám je stoprocentní, pokud je některá 70%, je 30% atp.).

## 8.4 Ukázky výsledků

### Beruška, kočka a kapsule



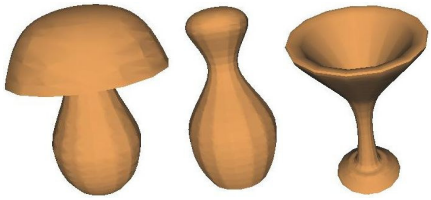
vstupní objekty, bázový objekt: beruška



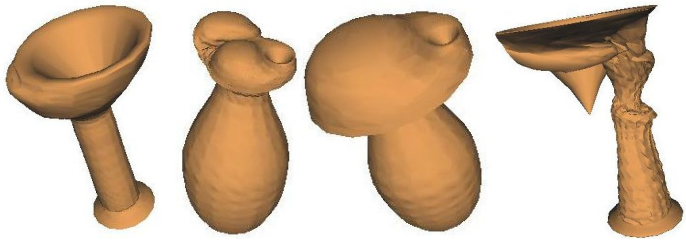
| | | | | |
|---|---|---|---|---|
| beruška: | 11% | 27% | 51% | 92% |
| kapsule: | 45% | 24% | 28% | 83% |
| kočka: | 43% | 49% | 21% | 83% |

**Závěr:** Nebylo použito barycentrických souřadnic, přesto je výsledek příjemný oku. Pokud jsou všechna tělesa zastoupena skoro 100%, už ale výsledek dobře nevyjde.

## Hříbeček, kuželka a sklenička



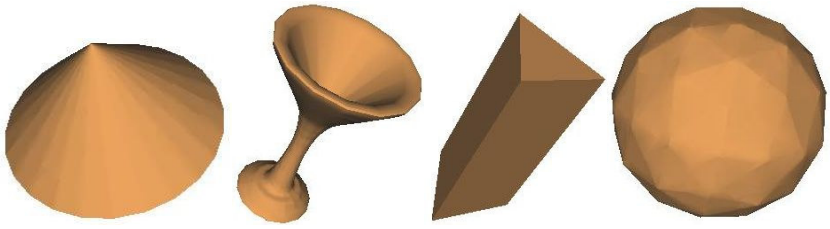
vstupní objekty, bázový objekt: hříbeček



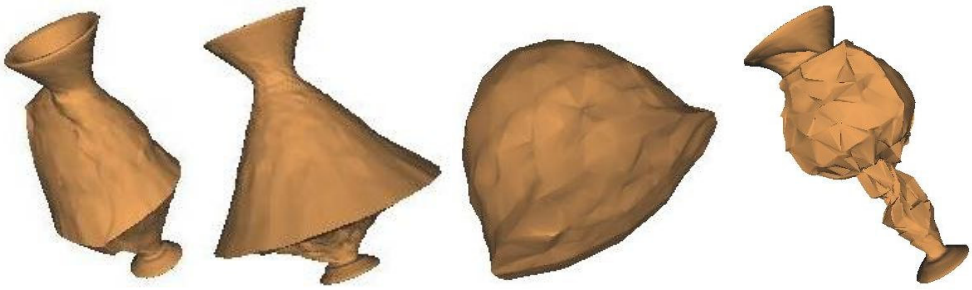
| | | | | |
|---|---|---|---|---|
| hříbeček: | 16% | 24% | 58% | 100% |
| kuželka: | 9% | 59% | 23% | 65% |
| sklenička: | 76% | 17% | 18% | 87% |

**Závěr:** V případě, že je použito barycentrických souřadnic (první až třetí objekt), výsledný objekt je příjemný oku. V případě, že barycentrické souřadnice nepoužijeme a součet vah není roven 100%, může výsledek dopadnout dobře, ale ve většině případů dopadne podobně jako čtvrtý objekt.

## Kužel, sklenička, hranol a zeměkoule



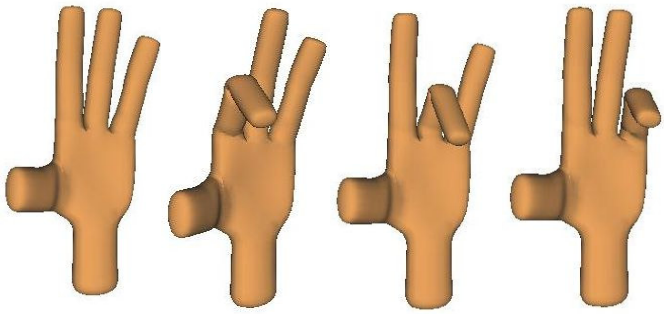
vstupní objekty, bázový objekt: kužel



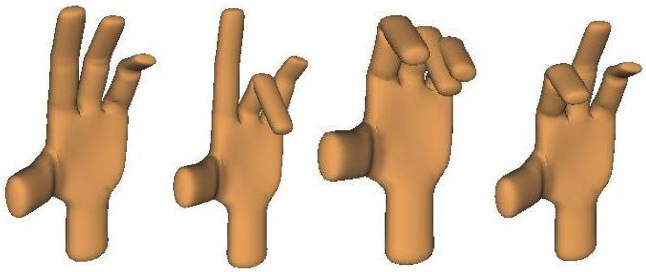
| | | | | |
|---|---|---|---|---|
| kužel: | 16% | 40% | 40% | 100% |
| sklenička: | 44% | 59% | 53% | 79% |
| hranol: | 19% | 0% | 0% | 42% |
| zeměkoule: | 11% | 0% | 0% | 79% |

**Závěr:** Zde nebylo použito barycentrických souřadnic, ale na prvních třech objektech je vidět, že pokud použijeme váhy, jejichž součet je menší jak 100%, výsledek je přijatelný. Druhý a třetí objekt je složením pouze dvou, což je ukázka, jak program použít pro morphing, a ne multimorphing – program přináší tu výhodu, že můžeme morphovat z jednoho tělesa do druhého a z druhého hned do třetího, z třetího do čtvrtého,… A to bez nutnosti mezitím provádět výpočty. Dále opět jako v předchozím případě při překročení 100% vznikne podivnost.

## Ruka ohýbající prsty



vstupní objekty, bázový objekt ruka s prsty nataženými

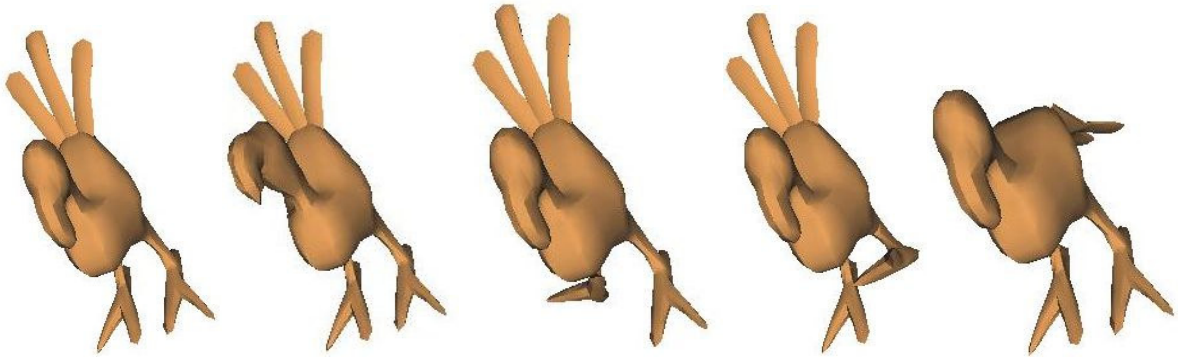


| | | | | |
|---|---|---|---|---|
| základ: | 26% | 100% | 100% | 2% |
| ukazováček: | 12% | 0% | 100% | 44% |
| prostředníček: | 19% | 100% | 100% | 17% |
| malíček: | 43% | 30% | 100% | 82% |

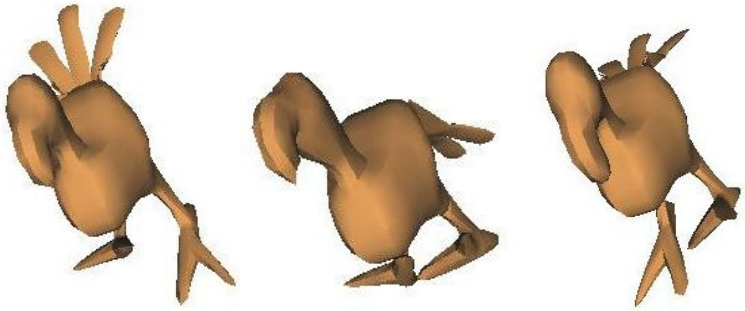
**Závěr:** V tomto případě nebylo potřeba použití barycentrických souřadnic, protože se objekty lišily pouze lokálně. Naopak, pokud bylo potřeba ohnout oba prsty, musely mít dva objekty

stoprocentní váhu. Zde se nám nepodaří dosáhnout nezdařeného výsledku, ať jsou objekty zastoupeny jakkoli.

## Dodo



Zleva: dodo základ, s hlavou nahoru, prava noha nahoru, leva noha nahoru, ocas dolů.



| | | | |
|---|---|---|---|
| základ: | 100% | 100% | 100% |
| hlava nahoru: | 26% | 100% | 0% |
| pravá noha: | 81% | 100% | 0% |
| levá noha: | 0% | 100% | 72% |
| ocas nahoru: | 25% | 100% | 54% |

**Závěr:** Stejný případ jako u ruky, pokud chceme získat např. skákajícího doda (prostřední obr.), musíme všechny váhy nastavit na 100%.

# 9. Závěr

Na začátku práce bylo prozkoumáno, jak se s problémem multimorphingu vypořádává 3ds max 7. Bylo zjištěno, že v 3ds max je multimorphing určen převážně pro animace, nejprve se morphuje z prvního objektu do druhého, z druhého pak do třetího a takto postupně přes všechny objekty. V tomto směru podává 3ds max skvělé výsledky. Pokud ale chceme výsledné těleso jako produkt nenulových vah více než dvou těles, začíná mít problémy s příliš odlišnými tělesy. Pokud se ale tělesa liší jen lokálně (ruka ohýbající prsty, pes vrtící ocáskem a zvedající nohy), vypořádává se s problémem výborně. Kvůli komerčnosti 3dsmaxu se bohužel nepodařilo zjistit, podle jakých rovnic počítá výsledné těleso.

Po zkoumání 3ds max začaly práce na vlastním programu pro multimorphing. Během práce vyvstalo mnoho problémů. Základním problémem byla potřeba stejného počtu vrcholů všech těles. Ten byl vyřešen dvěma způsoby, z nichž každý přinášel výhody i nevýhody. Prvním bylo využití programů Jindřicha Paruse pracujícími s dvěmi objekty – vymyslet algoritmus, který pomocí těchto programů, tvořících dva objekty o stejném počtu vrcholů, vytvoří n objektů o stejném počtu vrcholů. To se úspěšně podařilo, ale pro příliš kombinované objekty nebyly programy dostatečně stabilní. V případě, že procedura proběhla úspěšně, přinášely zdařilé výsledky – n objektů, sice s mnohonásobně více vrcholy, zato splňující danou podmínku.

Dalším způsobem bylo vytvořit program na remeshing, používající pouze program Jindřicha Paruse pro parametrizaci objektu. Tento program remeshoval všechny zadané objekty podle koule daných parametrů, tím vznikly objekty o stejném počtu vrcholů, jako měla daná koule. Bohužel remeshing je pouze aproximace daného objektu, ne vždy přináší uspokojivé výsledky, velmi také závisí na dobré parametrizaci. Pro většinu objektů ale lze metodu úspěšně využít.

Dalším problémem bylo uživatelské rozhraní – jak zadávat váhy jednotlivých objektů. V 3ds max je nutno zadávat váhy jednotlivě pro každý objekt, myšlenkou programu ale bylo vyzkoušet, zda je rozdíl, když je součtem vah všech objektů 1, nebo zda na součtu vah nezáleží. Pro součet vah 1 bylo použito barycentrických souřadnic – jednotlivé objekty ležely pomyslně na vrcholu n-úhelníku, uživatel zvolil bod v n-úhelníku, barycentrické souřadnice bodu pak určovaly váhy jednotlivých objektů – tedy čím blíže byl bod nějakému vrcholu, tím větší mělo těleso ležící pomyslně na tomto vrcholu.

Problémem bylo spočítat barycentrické souřadnice n-úhelníku, protože pro $n > 3$ už není jejich výpočet jednoznačný. Pro tento případ byl použit článek [4], výsledné váhy už nebyly tak přesné jako pro trojúhelník, počítání mnohonásobně složitější, ale požadovaného výsledku bylo úspěšně dosaženo.

V závěru program přináší přibližně stejné výsledky jako 3ds max, ale umožňuje také vytváření průměrů těles, zjednodušené zadávání vah jednotlivých objektů, vytváření animací, ukládání animací do videa a ukládání nastavení animací do xml souboru (pak lze použít stejné nastavení animace pro různé objekty, což je velká výhoda oproti 3ds max, kde byly animace velmi pracné).

Do budoucna je ještě mnoho co zlepšit, například rozšíření programu pro barevné objekty, experimentování s jinými způsoby výpočtu výsledného tělesa, nebo jiné způsoby počítání parametrizace, které by přinesly lepší výsledky programu pro remeshing. Celkově je toto téma velmi široké a zajímavé, práce na něm bude ještě v budoucnu pokračovat.

# Použitá literatura

[1] PARUS, J., 2003. Morphing 3D geometrických objektů. Diplomová práce. Západočeská Univerzita v Plzni.

[2] ISO/IEC 14496-16:2004/FDAM 1:2005. Coding of audio-visual objects. Amendment.

[3] PARUS, J., 2005. Morphing of Meshes. Technical report, No. DCSE/TR-2005-02. Západočeská Univerzita v Plzni.

[4] MEYER, M., LEE, H., BARR, A., DESBRUN, M.. 2002. Generalized Barycentric Coordinates on Irregular Polygons. Journal of Graphics Tools, 7, pp. 1086—7651.

[5] MÜLLER, W., ALEXA, M., 1998. Using Morphing for Information Visualization. Darmstadt University of Technology.

[6] SIEW, L.K., 1995. Video Compression via Morphing.

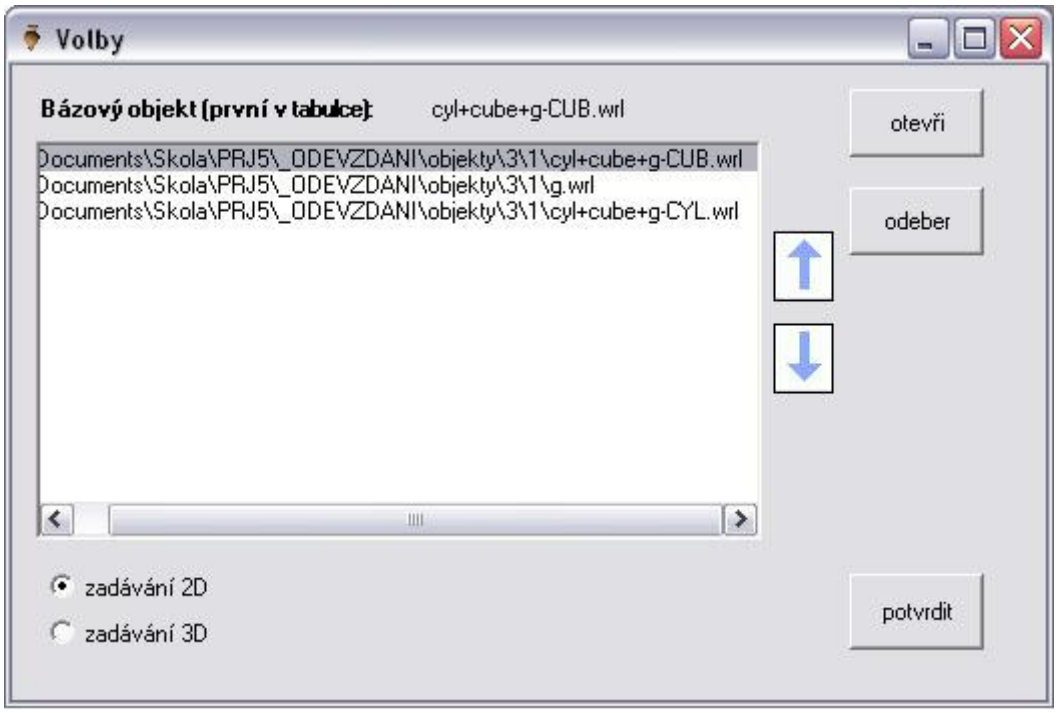
[7] http://3d-morphing.com [Poslední přístup 14.5.2006]

[8] 3ds max 7 Tutorials: Morphing a Knight

[9] http://herakles.zcu.cz/education/zpg/cviceni.php [Poslední přístup 14.5.2006]

# Příloha č. 1: Multimorphing - uživatelská dokumentace

Po spuštění programu se objeví okno voleb (obr. p1.1)




**Obr. p1.1 – Úvodní okno**

Zde si můžeme otevřít objekty a pomocí šipek vybrat, jak budou seřazeny za sebou. První v seznamu bude brán jako bázový objekt (v případě, že zvolíme metodu počítající s bázovým objektem). Ostatní budou v daném pořadí seřazeny vedle sebe (pomyslně) na vrcholech n-úhelníku (pro zadávání barycentrických souřadnic).

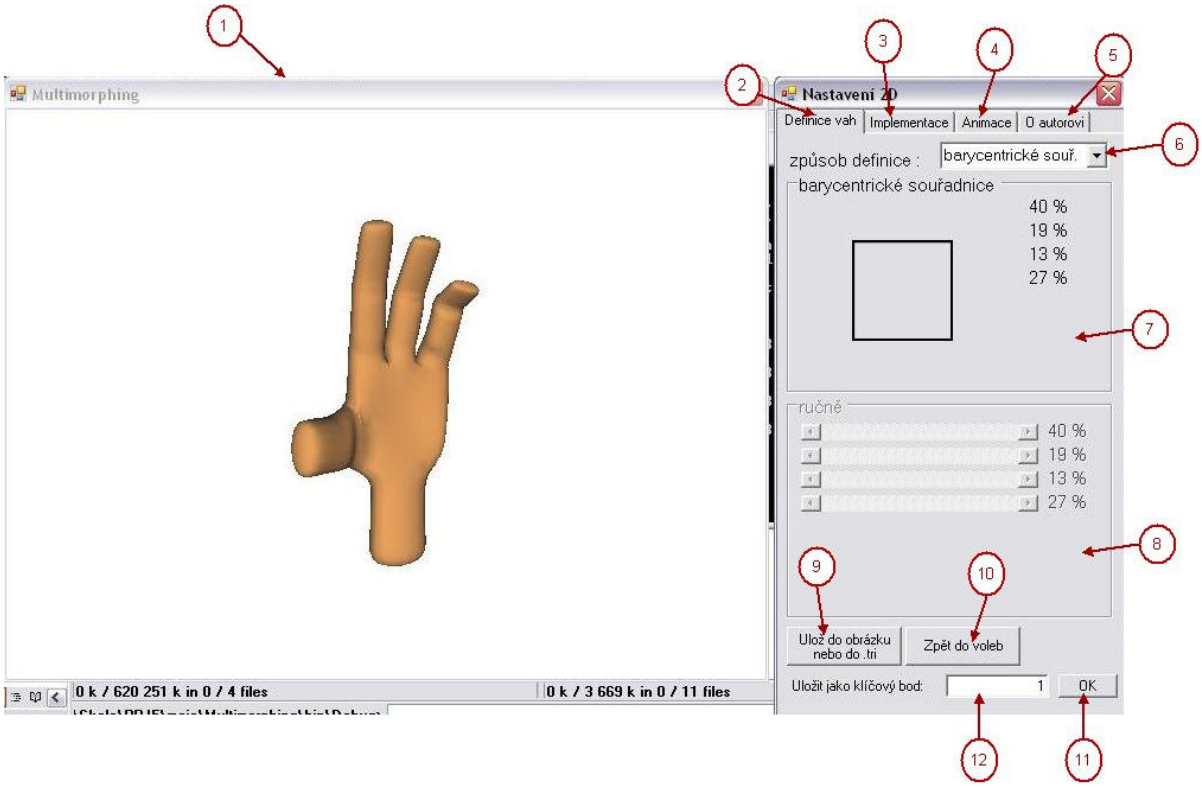
V případě, že chceme nějaký objekt odebrat ze seznamu, stačí použít tlačítko „odeber“.

V levém dolním rohu okna pak můžeme zvolit, zda budeme chtít zadávat váhy (pomocí barycentrických souřadnic) ve 2D (n-úhelník), nebo ve 3D (n-stěn, implementován pouze čtyřstěn). Tento výběr omezuje počet načtených objektů, tedy při výběru 2D bude program potřebovat minimálně 3 objekty, 3D je omezeno pouze na čtyři objekty.

Tlačítkem potvrdit spustíme hlavní část programu. Program načte zadané objekty a podle výběru (2D / 3D) zobrazí příslušná okna (liší se pouze v obrázku pro zadávání barycentrických souřadnic).

Z těchto oken se lze kdykoli zpátky vrátit do okna voleb (na první záložce zvolím „zpět do voleb“).

## Definice vah




1. Plátno, na které se vykresluje výsledný objekt. Na začátku je zde vykreslen bázový objekt (tedy má implicitně 100%). Objektem můžete otáčet (kliknutím a táhnutím myší), přibližovat/oddalovat (kolečko myši nebo +/-), posouvat (šipky). Stisknutím F11 lze zobrazit nápovědu, F12 zobrazí drátěný model objektu.
2. Záložka definice vah slouží k zadávání vah jednotlivých objektů, přidávání klíčových bodů do animace, ukládání výsledku do obrázku.
3. Záložka implementace slouží k výběru, jakým způsobem bude z vah počítán výsledný objekt.
4. Záložka animace slouží k odebírání bodů, spuštění animace, uložení animace do videa.
5. Záložka o autorovi nám zobrazí informace o autorovi a programu.
6. Zde můžeme vybrat, jakým způsobem budeme zadávat váhy – buď pomocí barycentrických souřadnic (budeme zadávat bod v n-úhelníku), nebo ručně – to umožňuje zadat i váhy, jejichž součtem není 1.
7. Při volbě zadávání vah pomocí barycentrických souřadnic se aktivuje tato skupina. N-úhelník představuje tzv. „morphing space“ – jednotlivé objekty si můžeme představit na vrcholech n-úhelníku, kliknutím pak určíme váhy objektů – čím blíže bude námi vybraný bod vrcholu, tím větší váhu bude mít daný objekt.
    - Pozn.: To, že součet procent není přesně 1, způsobují zaokrouhlovací chyby, které ovšem vznikají pouze při výpisu vah na obrazovku, pracuji s vahami jako s reálnými čísly, takže tam už tak velké zaokrouhlovací chyby nejsou.

- Pozn.: Implicitně je zadána ve druhé záložce implementace, ve které má bázový objekt vždy 100%. Zde tedy nehraje roli, jakou váhu zadáme prvnímu (= bázovému) objektu. V případě druhé možné implementace to ale roli hraje.

**8.** Při ručním zadávání nás omezuje pouze to, že u jednotlivého objektu nemůžeme přesáhnout váhu 100%. Opět zde platí, že pro první volbu v implementaci má bázový objekt stále stoprocentní váhu, proto je první posuvník deaktivován.

**9.** Volba uloží objekt zobrazený na plátně buď do obrázku, nebo do *.tri (umožní případnou další práci s objektem).

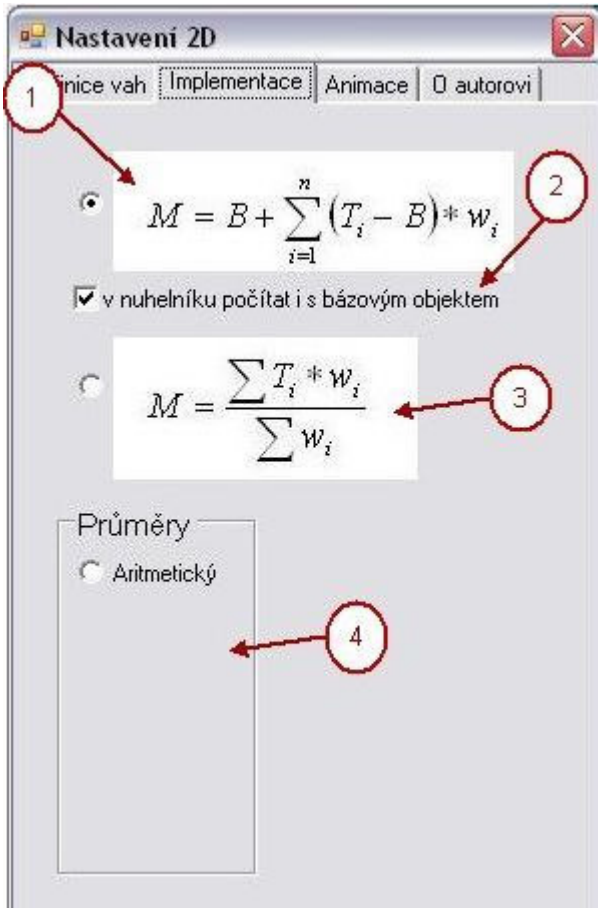
**10.** Návrat do prvního okna (Volby) s možností změny vstupních objektů – pozor, všechna nastavení zde na záložkách budou ztracena.

**11.** Potvrdí přidání bodu do klíčových (s časem napsaném vlevo, aktuálními váhami)

**12.** Zde lze zadat čas. Pokud je čas stejný jako čas nějakého jiného klíčového bodu, program nám nedovolí takový čas zadat.

Úvodní okna zadávání 3D jsou stejná, pouze zobrazovaný objekt pro zadávání barycentrických souřadnic není n-úhelník, ale čtyřstěn. Ten umožňuje zadávání souřadnic ve „3D“. Bohužel pouze pro 4 objekty. V čtyřstěnu určíme posuvníkem výšku, v této výšce se udělá řez – vznikne trojuhelník, kde už lze vybrat určitý bod.

## Implementace vah



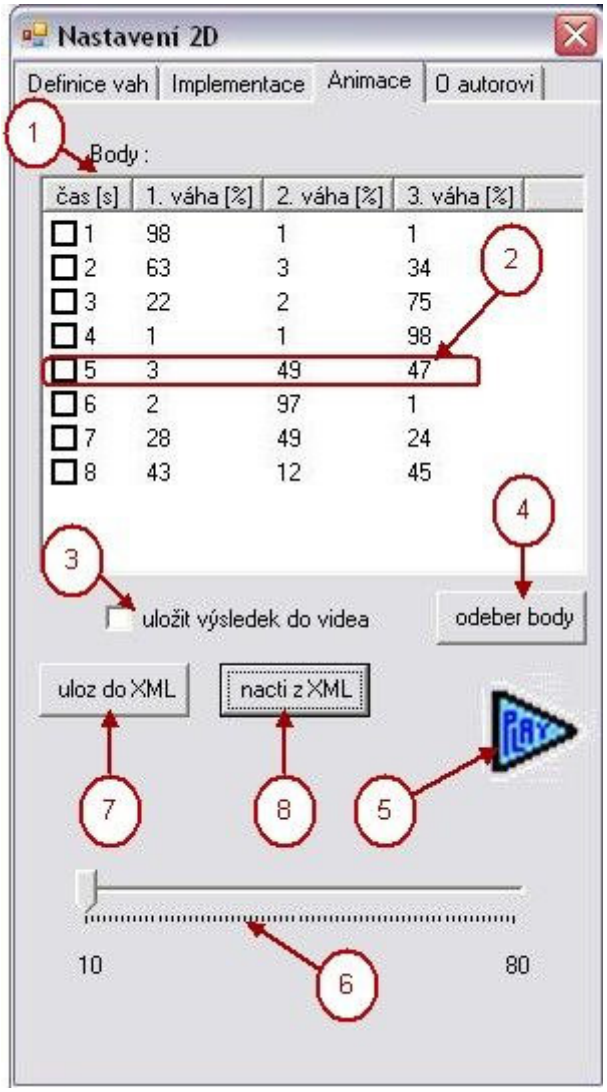

**1.** Tato rovnice je implicitně vybrána. Její popis lze nalézt v 8.2.

**2.** Zaškrtávátko platí pro první rovnici – pokud rovnice bere bázový objekt jako stále stoprocentní (jak je vidět z rovnice), mohlo by se zdát, že není potřeba zadávat jeho váhu. Váha bázového objektu ale hraje roli při zadávání souřadnic, protože ve chvíli, kdy chceme mít zobrazen pouze bázový objekt, musí mít všechny ostatní váhu 0%, a toho bychom jinak v n-úhelníku nedocílili. Přesto je tu na vyzkoušení tato možnost.

**3.** Tato rovnice představuje vážený průměr objektů, není zařazena v průměrech, neboť skupina průměry je pro počítání průměru bez zastoupení vah.

**4.** Pouze výpočet průměru – bez jakéhokoli ovlivnění váhami. Po volbě možnosti z této skupiny se znemožní většina voleb z definice vah a všechny volby z animací.

# Animace




1. Tabulka s body. Kliknutím na položku čas jednotlivého bodu zobrazíme výsledný objekt, zaškrtnutím bodu (více bodů) a stisknutím tlačítka odeber lze body odebrat.
2. Jedna položka z tabulky = jeden klíčový bod. První položkou je čas, další pak váhy jednotlivých objektů v daném čase.
3. Pokud zaškrtneme tuto volbu, bude animace výrazně pomalejší, neboť se v průběhu zobrazovaný objekt ukládá do obrázků, ze kterých se na konci animace vytvoří video.
4. Tlačítko odebere zaškrtlé body z tabulky.
5. Play – spustí animaci. Pro spuštění animace musíte mít alespoň dva body v tabulce, dobré je také mít u bodů různé váhy (jinak probíhající animace nelze rozeznat).
6. Posuvník – aktivní pouze pokud zrovna neprobíhá animace. Zde lze ručně zkoušet jak vypadá výsledný objekt v jednotlivých časech.
7. Toto tlačítko uloží nastavené body s váhami do xml. Pro stejný počet objektů lze pak v budoucnu pomocí tlačítka 8 nastavení opětovně načíst.
8. Umožňuje načtení nastavení animace z xml. Nastavení musí být pro stejný počet objektů.

# Příloha č. 2: Multimorphing – programová dokumentace

## Hlavní třída

Hlavní třída má název Hlavni a obstarává práci s okny – úvodní okno (Volby), následně okno, kam se vykreslují objekty (Platno), okno nastavení (Nastaveni) a také s třídou Zobraz, která vykresluje objekty na plátno.

## Třída Voleb

Tato třída obstarává okno, které se zobrazí uživateli po startu programu. Umožňuje výběr souborů (sama je nenačítá, pouze předá jména souborů hlavní třídě), dále výběr, zda uživatel bude chtít zadávat barycentrické souřadnice v 2D nebo 3D (tedy vybírat bod z n-úhelníku nebo n-stěnu), pro dané nastavení dále kontroluje, zda je zadán dostatečný počet souborů.

## Třída Nastavení

Tato třída obstarává veškeré nastavování programu, tedy uživatelské rozhraní. Protože možnosti nastavení se liší podle toho, zda chce uživatel zadávat barycentrické souřadnice ve 2D nebo 3D, od této třídy dědí dvě třídy – Nastaveni2D a Nastaveni3D. V třídě Nastaveni jsou pak metody, které mají tyto dvě třídy společné. Je zde mnoho práce s uživatelským rozhraním, animacemi, exportem do videa a xml, ale žádný zajímavý algoritmus, který by stál za zmínku.

## Nastaveni2D

Zde jsou navíc pouze záležitosti n-úhelníku, tedy tvorba a obsluha. Vlastní výpočty barycentrických souřadnic obstarává třída n-úhelník. Výpočet probíhá přesně podle rovnic uvedených v kapitole 3.

## Nastaveni3D

Zde jsou navíc pouze záležitosti čtyřstěnu. Zobrazení čtyřstěnu je vyřešeno tak, že je zobrazen statický obr. čtyřstěnu, vedle kterého je posuvník. Posuvníkem uživatel určí řez čtyřstěnem, podle kterého je zobrazen příslušně velký trojúhelník. V trojúhelníku pak lze vybrat bod, podle kterého se již vypočítají příslušné souřadnice.

Třída používá třídu Ctyrsten – pro zobrazení čtyřstěnu a výpočet souřadnic, dále pak třídu Trojúhelník, která sice umí počítat souřadnice v trojúhelníku, ale její funkce pro tento výpočet se momentálně nevolají, jsou zde zakonzervovány pro případ budoucího použití, třída je tedy použita pouze na vykreslení trojúhelníků určité velikosti.

## Zobrazování výsledku

Zobrazování výsledku obstarává třída Zobraz, která pro vykreslení používá třídu Plátno (zde jsou nastaveny všechny běžné věci jako kamera, otáčení a světlo). Pro výpočet výsledného objektu pak třída Zobraz používá třídu Morphing.

Hlavní metody třídy jsou:

- SetVahy(float [] vahy): Pomocí této metody se přepočítá (použitím třídy Morphing) výsledné těleso a vykreslí na plátno.
- SetVyber(int vyber): Slouží k nastavení, kterou metodou třídy Morphing se bude počítat výsledný objekt.

## Morphing

Celý morphing obstarává stejnojmenná třída. Při vytváření instance této třídy je vyžadován seznam jmen souborů, které se mají načíst. Třída si pak pomocí třídy NactiZWRML přečte ze souborů vrcholy a indexy objektů, také zjistí jejich počet. První ze seznamu následně považuje za bázový objekt.

Ještě v konstruktoru se zavolá metoda na výpočet normál. Ta nejprve pro každý vrchol vyhledá trojúhelníky, kterým je společný, a to brutální silou:

- vytvoří pole o stejné velikosti, jako je počet vrcholů. Každá položka pole je typu List (seznam odkazů do seznamu trojúhelníku)
- postupně prochází indexy (neboli pole, kde každá položka obsahuje tři čísla vrcholů => trojúhelník), každý trojúhelník „zapíše“ na tři místa do pole
- na konci má tedy pole, kde na každém místě je seznam odkazů na trojúhelníky vrcholu stejného indexu jako je index pole

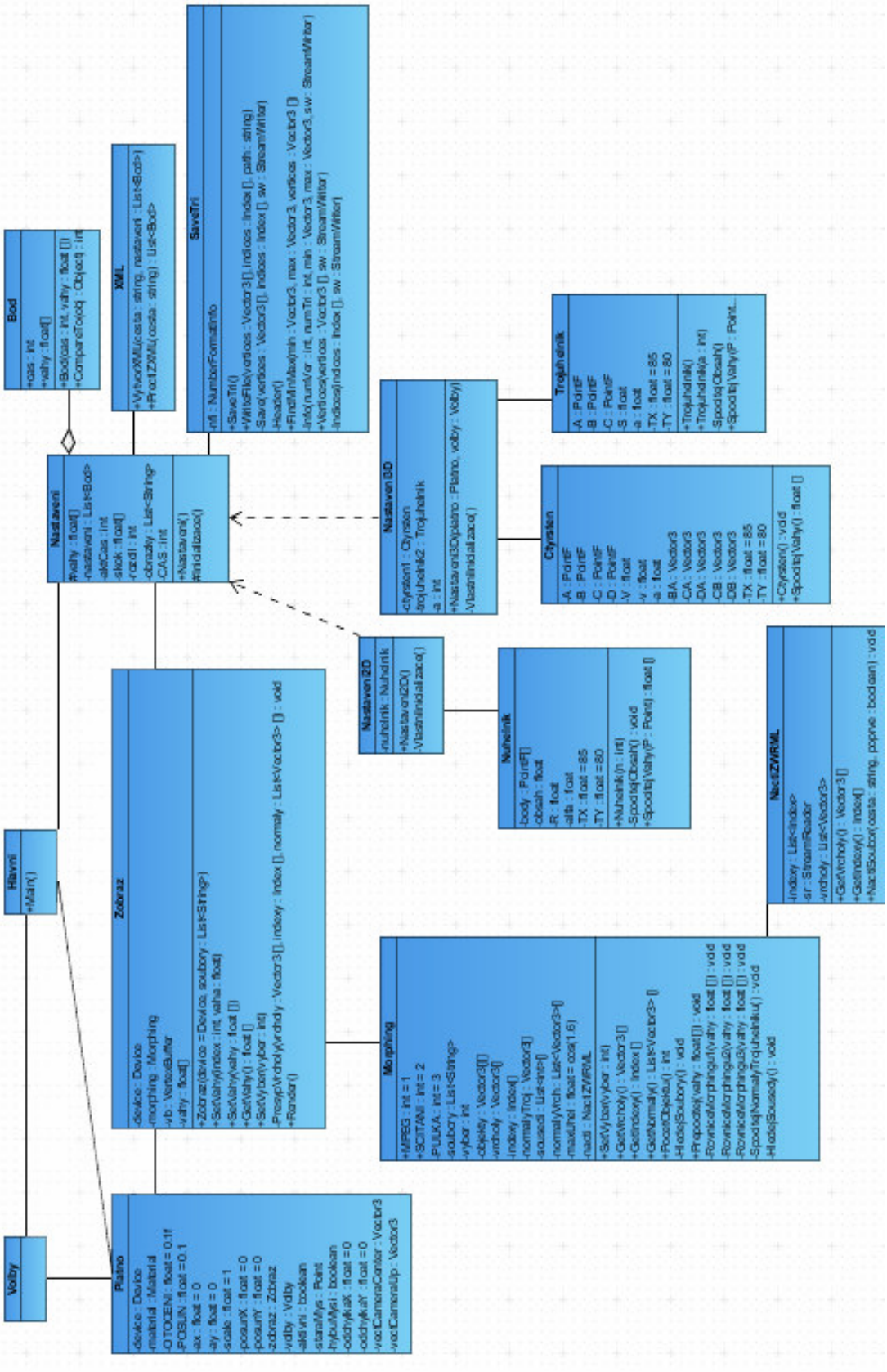
Pak se spočítají normály všech trojúhelníků.

Potom se pro každý vrchol spočítá stejný počet normál, jako je trojúhelníků společných tomuto vrcholu. Ty se spočítají tak, že se pro každý trojúhelník vrcholu považuje jeho normála za referenční. Z normál zbylých trojúhelníků společných vrcholu se vyberou ty, které s referenční normálou svírají úhel menší než tzv. „crease angle“. Z těchto normál a referenční normály se vypočítá průměr, který se uloží jako výsledek.

Tímto způsobem vznikne pro každý vrchol tolik normál, v kolika trojúhelníkách je obsažen. Pro případ, že trojúhelník není na ostré hraně, je ukládání tolika normál neefektivní, ale pro jednoduchost byl tento způsob výpočtu a ukládání normál zvolen.

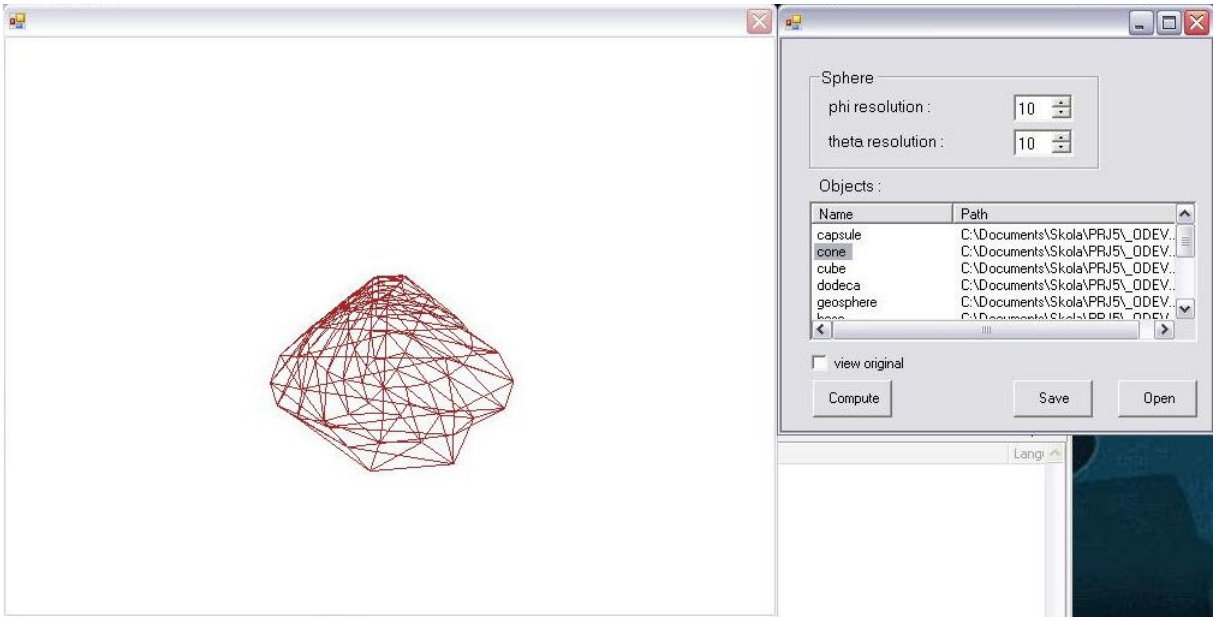
V případě, že už jsou načteny soubory a vypočítány normály, může být zavolána metoda třídy Compute(). Ta zavolá podle nastavení (int vyber) příslušnou metodu na výpočet výsledného objektu. Metody jsou popsány v textu práce.

# UML diagram


Bod
+cas : int
+vahy : float[]
+Bod(cas : int, vahy : float [])
+CompareTo(obj : Object) : int
XML
+VytvorXML(cesta : string, nastaveni : List<Bod>)
+NactiZXML(cesta : string) : List<Bod>
SaveTri
-inf : NumberFormatInfo
+SaveTri()
+WriteFile(vertices : Vector3[], indices : index [], path : string)
-Save(vertices : Vector3[], indices : index [], sw : StreamWriter)
-Header()
+FindMinMax(min : Vector3, max : Vector3, vertices : Vector3 [])
-Info(numVer : int, numTri : int, min : Vector3, max : Vector3, sw : StreamWriter)
+Vertices(vertices : Vector3 [], sw : StreamWriter)
-Indices(indices : index [], sw : StreamWriter)
Nastaveni
#vahy : float[]
-nastaveni : List<Bod>
-aktCas : int
-skok : float[]
-razil : int
-obrazky : List<String>
-CAS : int
+Nastaveni()
#Inicializace()
Nastaveni3D
-Cyklon1 : Cyklon
-trojUhelnik2 : TrojUhelnik
-a : int
+Nastaveni3D(pUhlno : Platno, volby : Volby)
-VlastniInicializace()
TrojUhelnik
-A : PointF
-B : PointF
-C : PointF
-S : float
-a : float
-TX : float = 85
-TY : float = 80
+TrojUhelnik()
+TrojUhelnik(a : int)
-SpocitejObsah()
+SpocitejVahy(P : Point
Cyklon
-A : PointF
-B : PointF
-C : PointF
-D : PointF
-V : float
-v : float
-a : float
-BA : Vector3
-CA : Vector3
-DA : Vector3
-CB : Vector3
-DB : Vector3
-TX : float = 85
-TY : float = 80
+Cyklon() : void
+SpocitejVahy() : float []
Nastaveni2D
-nuhelnik : Nuhelnik
+Nastaveni2D()
-VlastniInicializace()
Nuhelnik
-body : PointF[]
-obsah : float
-R : float
-alfa : float
-TX : float = 85
-TY : float = 80
+Nuhelnik(n : int)
-SpocitejObsah() : void
+SpocitejVahyP : Point) : float []
Hlavni
+Main()
Zobraz
-device : Device
-morphing : Morphing
-vb : VertexBuffer
-vahy : float[]
+Zobraz(device = Device, soubory : List<String>)
+SetVahy(index : int, vaha : float)
+SetVahy(vahy : float [])
+GetVahy() : float []
+SetVyber(vybor : int)
-PresunVrcholy(vrcholy : Vector3 [], indexy : index [], normaly : List<Vector3> [] : void
+Render()
Morphing
+MPEG : int = 1
+SOUTAVI : int = 2
-PUUKA : int = 3
-soubory : List<String>
-vybor : int
-objekty : Vector3[][]
-vrcholy : Vector3[]
-indexy : index[]
-normalyTrq : Vector3[]
-sousedi : List<int>[]
-normalyVrch : List<Vector3>[]
-maxUhel : float = cos(1.6)
-nadr : NactiZVRML
+SetVyber(vybor : int)
+GetVrcholy() : Vector3[]
+GetIndexy() : index []
+GetNormaly() : List<Vector3> []
+PocetObjektu() : int
-HledejSoubory() : void
+PrpocitejVahy : float[]) : void
-RovnicaMorphingu1(vahy : float []) : void
-RovnicaMorphingu2(vahy : float []) : void
-RovnicaMorphingu3(vahy : float []) : void
-SpocitejNormalyTrojuhelniku() : void
-HledejSousedy() : void
NactiZVRML
-indexy : List<index>
-sr : StreamReader
-vrcholy : List<Vector3>
+GetVrcholy() : Vector3 []
+GetIndexy() : index[]
+NactiSoubor(cesta : string, popmve : boolean) : void
Volby
Platno
-device : Device
-material : Material
-OTOCENI : float = 0.1f
-POSUN : float = 0.1
-ax : float = 0
-ay : float = 0
-scale : float = 1
-posunX : float = 0
-posunY : float = 0
-zdroj : Zdroj
-volby : Volby
-aktivni : boolean
-staraMys : Point
-hybuMys : boolean
-oddyAlaX : float = 0
-oddyAlaY : float = 0
-vecCameraCenter : Vector3
-vecCameraUp : Vector3

# Příloha č. 3: Remeshing – uživatelská dokumentace

Program poskytuje jen jednoduché ovládání.




Pro načtení souborů zvolíme „Open“. Lze otevřít více souborů naráz. Každý soubor musí být ve formátu *.tri a mít příslušný (stejnojmenný) soubor parametrizací ve formátu *.par. Při načítání program přímo provede remeshing objektů (podle zadaných nastavení v rámečku Sphere). Na objekty v tabulce pak lze kliknout pro zjištění, jak úspěšně remeshing proběhl. Při zaškrtnutí „view original“ se ukáže originální objekt v jiné barvě.

Přesnost remeshingu můžeme měnit přidáváním či ubíráním bodů koule, podle které se remeshing provádí. Toho lze docílit v rámečku „Sphere“, pro přepočítání pak je třeba stisknout tlačítko „Compute“.

Při stisknutí tlačítka „Save“ se všechny remeshované objekty uloží do adresáře, odkud byly načteny, a to ve formátu *.wrl.

# Příloha č. 4: Remeshing – programová dokumentace

## Načítání a ukládání souborů

Načítání i ukládání souborů probíhá přes rozhraní Save a Load, což umožňuje budoucí rozšíření programu o další typy souborů. Prozatím umí program načítat pouze ze souborů .tri, .par a ukládat do souboru .wrl.

## Uživatelské rozhraní

Uživatelské rozhraní obstarává jak plátno (RenderForm), na které se vykreslují objekty pomocí třídy ViewObject – cokoli je potřeba vykreslit, vytvoří se objekt ViewObject, při vytváření objektu je požadován seznam vrcholů, indexů a barva objektu.

Dále třída Properties poskytuje ovládání programu – nastavování parametrů koule, otvírání a ukládání souborů, vybírání co zobrazit. Je vytvořena pomocí automatického generátoru v Microsoft Visual Studio 2005.

## Remeshing

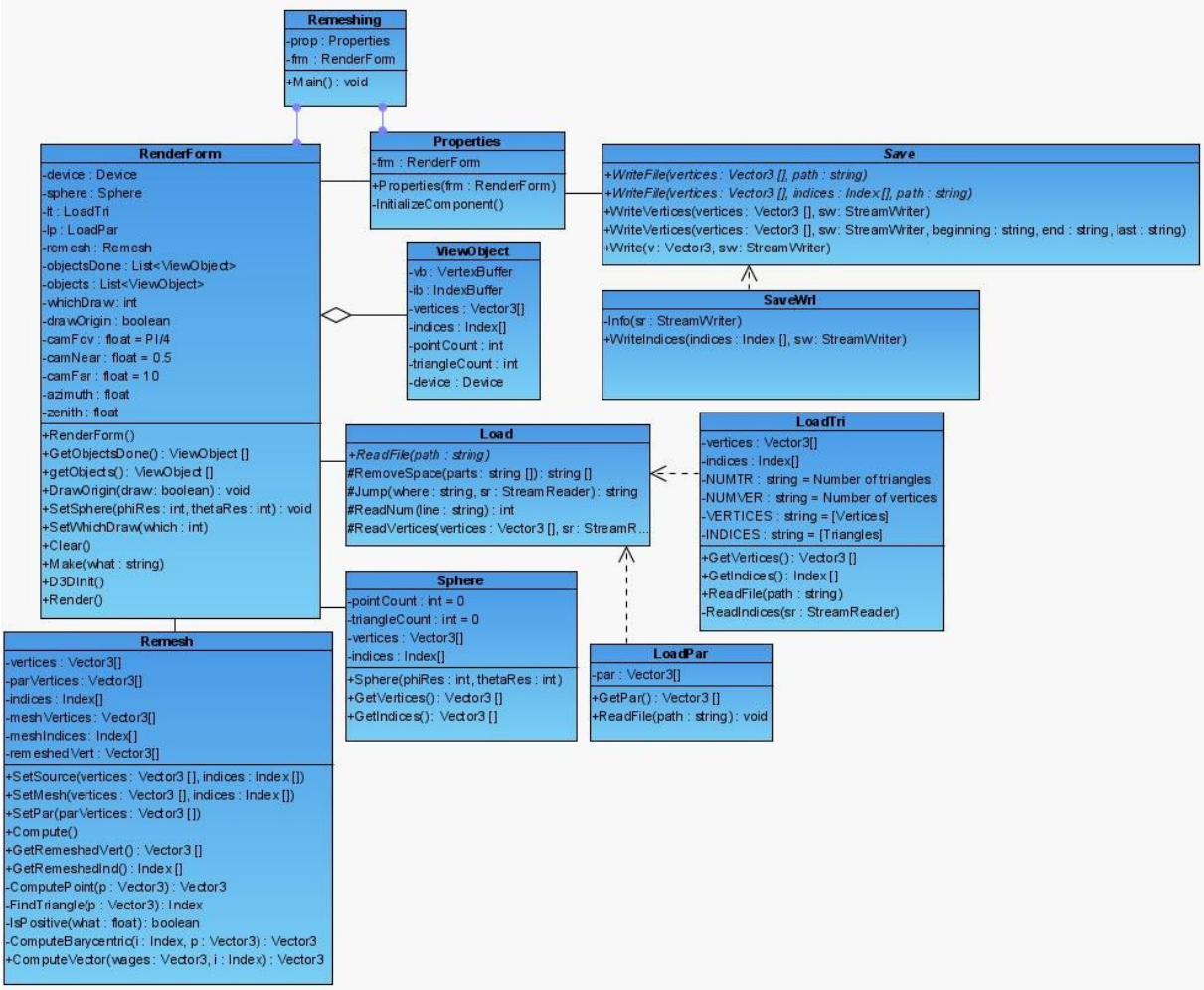
Vlastní remeshing obstarávají třídy Remesh a Sphere.

Třída Sphere vygeneruje kouli dle předaných parametrů. Je převzata z [9] a lehce upravena pro potřeby remeshingu (zjednodušena).

Třída Remesh vyžaduje nastavení objektu podle kterého remeshovat (meshVertices, meshIndices), objektu který remeshovat (remeshedVert) a parametrizace (parVertices). Pak už lze zavolat metodu Compute(), která vypočítá výsledný remeshovaný objekt dle postupu popsaného v kapitole 7.

Výsledek výpočtu lze získat pomocí běžných metod Get().

# UML diagram


Remeshing
-prop : Properties
-frm : RenderForm
+Main() : void
RenderForm
-device : Device
-sphere : Sphere
-lt : LoadTri
-lp : LoadPar
-remesh : Remesh
-objectsDone : List<ViewObject>
-objects : List<ViewObject>
-whichDraw: int
-drawOrigin : boolean
-camFov : float = PI/4
-camNear : float = 0.5
-camFar : float = 10
-azimuth : float
-zenith : float
+RenderForm()
+GetObjectsDone() : ViewObject []
+getObjects(): ViewObject []
+DrawOrigin(draw: boolean) : void
+SetSphere(phiRes: int, thetaRes: int) : void
+SetWhichDraw(which : int)
+Clear()
+Make(what : string)
+D3DInit()
+Render()
Properties
-frm : RenderForm
+Properties(frm : RenderForm)
-InitializeComponent()
Save
+WriteFile(vertices : Vector3 [], path : string)
+WriteFile(vertices : Vector3 [], indices : Index[], path : string)
+WriteVertices(vertices : Vector3 [], sw: StreamWriter)
+WriteVertices(vertices : Vector3 [], sw: StreamWriter, beginning : string, end : string, last : string)
+Write(v : Vector3, sw: StreamWriter)
SaveWrl
-Info(sr : StreamWriter)
+WriteIndices(indices : Index [], sw: StreamWriter)
ViewObject
-vb : VertexBuffer
-ib : IndexBuffer
-vertices : Vector3[]
-indices : Index[]
-pointCount : int
-triangleCount : int
-device : Device
Load
+ReadFile(path : string)
#RemoveSpace(parts: string []): string []
#Jump(where : string, sr : StreamReader): string
#ReadNum(line : string) : int
#ReadVertices(vertices : Vector3 [], sr : StreamR...
LoadTri
-vertices : Vector3[]
-indices : Index[]
-NUMTR : string = Number of triangles
-NUMVER : string = Number of vertices
-VERTICES : string = [Vertices]
-INDICES : string = [Triangles]
+GetVertices(): Vector3 []
+GetIndices(): Index []
+ReadFile(path : string)
-ReadIndices(sr : StreamReader)
Sphere
-pointCount : int = 0
-triangleCount : int = 0
-vertices : Vector3[]
-indices : Index[]
+Sphere(phiRes : int, thetaRes : int)
+GetVertices(): Vector3 []
+GetIndices(): Vector3 []
LoadPar
-par : Vector3[]
+GetPar() : Vector3 []
+ReadFile(path : string) : void
Remesh
-vertices : Vector3[]
-parVertices : Vector3[]
-indices : Index[]
-meshVertices : Vector3[]
-meshIndices : Index[]
-remeshedVert : Vector3[]
+SetSource(vertices : Vector3 [], indices : Index [])
+SetMesh(vertices : Vector3 [], indices : Index [])
+SetPar(parVertices : Vector3 [])
+Compute()
+GetRemeshedVert() : Vector3 []
+GetRemeshedInd() : Index []
-ComputePoint(p : Vector3) : Vector3
-FindTriangle(p : Vector3): Index
-IsPositive(what : float) : boolean
-ComputeBarycentric(i : Index, p : Vector3) : Vector3
+ComputeVector(wages: Vector3, i : Index) : Vector3