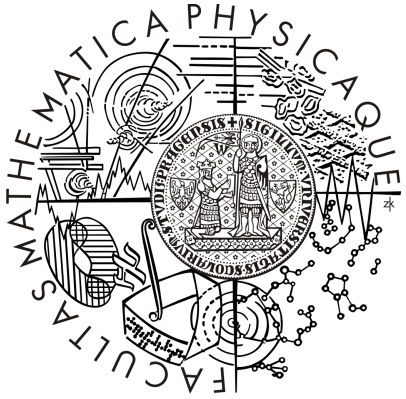
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Radek Mácha

Vyhlazování 3D modelů

Katedra aplikované matematiky

Vedoucí bakalářské práce: RNDr. Martin Pergel, Ph.D.,
Kabinet software a výuky informatiky

Studijní program: Informatika, programování

2009

Rád bych poděkoval svému vedoucímu, RNDr. Martinu Pergelovi, Ph.D., za užitečné rady a podporu při sepisování této práce.

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Radek Mácha

Obsah

1 Úvod **6**
- 1.1 Co jsou subdivizní algoritmy . . . 6
- 1.2 Motivace práce . . . 6

2 Přístup k ohodnocování **7**
- 2.1 Optimum - co bychom chtěli . . . 7
- 2.2 Omezení obecnou datovou strukturou . . . 7
- 2.3 Kvalita modelu . . . 8

3 Subdivizní algoritmy **9**
- 3.1 Vznik subdivizních algoritmů . . . 9
- 3.2 Obecné vlastnosti . . . 9
- 3.3 Terminologie . . . 10
- 3.4 Aproximační a interpolační algoritmy . . . 11
- 3.5 Formální popis použitých algoritmů . . . 12
  - 3.5.1 Catmull-Clark . . . 13
  - 3.5.2 Butterfly . . . 15
- 3.6 Některé další subdivizní algoritmy . . . 19
  - 3.6.1 Doo . . . 19
  - 3.6.2 Loop . . . 20

4 Srovnávání **21**
- 4.1 Co přesně přináší prokládání . . . 21
- 4.2 Kdy a jak prokládat, omezení . . . 21
- 4.3 Implementace měření v přiloženém software . . . 22
- 4.4 Výsledky vs. pozorování . . . 24
- 4.5 Návrh agregační funkce měření . . . 27

5 Závěr **31**
- 5.1 Shrnutí . . . 31
- 5.2 Perspektivy . . . 31
  - 5.2.1 Rozhraní pro specifikaci agregační funkce . . . 31
  - 5.2.2 Dávkové zpracování . . . 32

5.2.3 Počítání úhlových deformací 32
5.2.4 Formulace subdivizních algoritmů vzorcem 32

Literatura **33**

A Uživatelská dokumentace **35**
A.1 Funkce 35
A.2 Instalace a odinstalování 35
A.3 Ovládání 35
A.3.1 Možnosti zobrazení 36
A.4 Data 36
A.4.1 Formát `*.txt` 36
A.4.2 Formát `*.model` 38
A.5 Statistiky 38
A.6 Dostupné algoritmy 39
A.6.1 Catmull-Clark 39
A.6.2 Butterfly 40

B Programátorská dokumentace **42**
B.1 Použité technologie 42
B.2 Struktura kódu 42
B.2.1 Uspořádání souborů 42
B.2.2 Struktura `#include` direktiv 43
B.2.3 Struktura tříd 43
B.3 Důležité sekce kódu 46
B.3.1 Sestavování modelu 46
B.3.2 Přepínání renderovacího kontextu 47
B.3.3 Subdivizní algoritmy 47

Název práce: Vyhlazování 3D modelů
Autor: Radek Mácha (radek.macha@gmail.com)
Katedra (ústav): Katedra aplikované matematiky
Vedoucí bakalářské práce: RNDr. Martin Pergel, Ph.D.
Kabinet software a výuky informatiky
e-mail vedoucího: perm@kam.mff.cuni.cz

Abstrakt: Práce pojednává o subdivizních metodách vyhlazování 3D modelů. Klíčovou oblastí práce je kvalita výstupů subdivizních sekvencí a vývoj metod k ohodnocení kvality výsledku. Ke zkoumání jsou použity algoritmy Catmull-Clark a Butterfly (8-point stencil), provádí se měření na základě délky hran, obsahu stěn a úhlů mezi stěnami. Práce se dále zabývá posouzením výsledků kombinovaných sekvencí více subdivizních algoritmů a výběru sekvence vyhovující stanoveným parametrům.

Klíčová slova: subdivize, 3D vyhlazování, porovnávání

Title: 3D surface subdivision
Author: Radek Mácha (radek.macha@gmail.com)
Department: Department of Applied Mathematics
Supervisor: RNDr. Martin Pergel, Ph.D.
Department of Software and Computer Science Education
Supervisor's e-mail address: perm@kam.mff.cuni.cz

Abstract: The present work deals with 3D surface subdivision methods. The key areas are the quality of subdivision sequence outputs and the development of methods that measure it. The Catmull-Clark and Butterfly (8-point stencil) outputs are being measured for edge length, face surface and angles between adjacent faces. Furthermore, combined sequences of subdivision algorithms are being discussed, as well as the selection of one that meets custom criteria.

Keywords: subdivision, 3D surface smoothing, comparison

Kapitola 1

Úvod

1.1 Co jsou subdivizní algoritmy

Subdivizní algoritmy jsou metody vyhlazování povrchu prostorových modelů (polyedrů). Jejich hlavním principem je postupné zjemňování původního polyedru (kontrolní sítě, tzv. *control net*) přidáváním nových vrcholů, s jejichž pomocí pak rozdělují stávající polygony na menší podčásti. Výsledkem je komplexnější polyedr, který pro lidské oko představuje věrohodnější obraz simulovaného objektu. Cílový polyedr je vytvořen opakovanou subdivizí a často bývá složitý.

1.2 Motivace práce

Uplatnění subdivizních algoritmů dnes spočívá především v 3D modelování (Cinema 4D) a tvorbě animací (Pixar Studios). Většinou se jedná o aplikaci aproximačních algoritmů (Catmull-Clark, Loop) na již značně komplexní polyedry (nebo jejich části).
Možnost využití metod subdivize je ale širší: komplexita cílových polyedrů je škálovatelná pouhým rozhodnutím, zda aplikujeme další subdivizi či ne. Algoritmy jsou dostatečně rychlé, aby stíhaly dopočítávat rozumně komplexní polyedry (popř. jejich části) real-time. Proto má subdivize 3D modelů potenciál i ve vývoji 3D her se základním geometrickým primitivem polygon, specielně díky škálovatelnosti využitelné při realizaci LOD.
Nejedná se ale o příliš využívanou technologii, protože na aproximační algoritmy je počáteční polyedr herního objektu většinou příliš hrubý, interpolační algoritmy pak zase mívají problémy s neregulární topologií, kterou takovéto objekty většinou oplývají.
V této práce se pokusíme odpovědět na otázku, jestli je možné měřením určit vyhovující sekvence algoritmů, které budou fungovat i na jednoduchých polyedrech s libovolnou topologií, ale zároveň se vyvarují problémů lokálních deformací.

Kapitola 2

Přístup k ohodnocování

2.1 Optimum - co bychom chtěli

Naším primárním cílem je porovnat sekvence různých subdivizních algoritmů co do kvality výsledku. Proto je třeba zavést metody kvantifikace určitých aspektů výsledného modelu. Výsledné hodnoty těchto metod bychom pak chtěli mezi sebou volně porovnávat.
Jedna z prvních úvah, které učiníme, je, že by se taková metoda dala využít také k zohlednění aspektu cena/výkon - cena potřebná k výpočtu by se vyjádřila jako jeden z kvantifikovaných aspektů a vhodnou aritmetickou operací by byla začleněna do celkového výsledku. Tímto způsobem bychom byli schopni i poznat, zda pro nás v daném modelu má smysl další krok vůbec provádět.

2.2 Omezení obecnou datovou strukturou

Výše popsaná metoda měření cena/výkon v našem případě bohužel není rozumně proveditelná. Aby bylo možné výsledky různých algoritmů vůbec porovnávat, je třeba veškerou subdivizi provádět na obecném datovém modelu, který je schopen pojmout jakýkoliv polyedr. To však při běžném praktickém použití subdivizních algoritmů neplatí - datová struktura je v těchto případech šitá na míru jednomu konkrétnímu algoritmu. Například při opakovaném použití algoritmu Butterfly nebude stavebním kamenem modelu obecný polygon, ale trojúhelník.
Konkrétní algoritmy jsou různě postiženy faktem, že nepracují s optimálními daty. Tímto různě trpí jak výsledky algoritmů, tak jejich časová náročnost. Je ovšem pravděpodobné, že různé algoritmy, které pracují na stejné datové bázi, budou při kombinovaném použití poskytovat dobré výsledky, protože si nebudou navzájem škodit při konstrukci výsledné topologie. Tato vazba se v implementovaném prostředí projeví.
S časovou výkonností je to ovšem jinak – často by bylo možné vytvořit specifickou datovou strukturu, která by vyhovovala oběma algoritmům a poskyto-

vala jim lepší výkon, v implementovaném prostředí se ale tato vazba neprojeví, protože by bylo nutné implementovat veškeré výkonnostní modifikace ke specifickým topologiím, což by s sebou neslo implementaci všech specifických datových struktur a konverzí mezi nimi.

2.3 Kvalita modelu

Otázkou zůstává, jakým způsobem se dá změřit kvalita modelu. Od modelu požadujeme především, aby byl hladký (tedy aby úhly mezi sousedícími stěnami nebyly příliš markantní) a aby nepůsobil hrubě (tzn. neobsahoval nepoměrně velké plochy a dlouhé hrany). Na tyto míry modelu se tedy zaměříme.

Tato měření ovšem nezohledňují jednen z klíčových aspektů modelu: věrohodnost. Může se stát, že dobře vyhlazený model nepůsobí věrohodně, přestože z hlediska předcházejících kritérií by vykazoval dobré výsledky. Jedná se především o modely s ostrými hranami.
Přestože věrohodnost modelu jako takovou nelze kvantifikovat, existuje způsob, jak ohodnotit míru deformace. Pakliže zachováme hodnoty z předchozího měření a v rámci lokálního kontextu je porovnáme oproti novým hodnotám, lze vhodnou aritmetikou získat relevantní informaci o tom, do jaké míry zpracovaný model odpovídá původnímu.

Kapitola 3

Subdivizní algoritmy

3.1 Vznik subdivizních algoritmů

Povrch zjemněný subdivizí je v podstatě prostorová analogie k Bézierově křivce – jedná se o přímou kombinaci metody aproximace křivek lineární kombinací polynomiálních funkcí a konceptu fraktálů. Klíčový význam v jejich vývoji hraje algoritmus de-Casteljau (kniha [3], str. 15-18) – de-facto první subdivizní algoritmus vůbec.

Subdivizní algoritmy pro vyhlazování povrchů jsou přenosem této myšlenky do $\mathbb{R}^3$ – vychází z použití dvou os, které určují rovinu rozdělovaného polygonu. Tato rovina je tedy v daném kontextu zároveň definičním oborem funkce dvou proměnných, jejíž křivku se snažíme aproximovat. Více informací na toto téma můžeme najít v práci Edwina Catmulla ([7]) a v knize [3] (str. 28-39).

3.2 Obecné vlastnosti

Z obecného principu subdivizních algoritmů plynou následující vlastnosti:

- Časová složitost jednoho kroku algoritmu je obecně lineární vůči počtu hran (každá hrana je navštívena dvakrát) zpracovávaného modelu, tj. v nejhorším případě kvadratická vůči počtu vrcholů. Složitost je ale silně závislá na topologii zpracovávaného polyedru - pokud existuje horní odhad valence vrcholů a_V, pak každý vrchol navštívíme maximálně a_V-krát a složitost je lineární vůči počtu vrcholů.

- Počet polygonů a vrcholů zpracovávaného polyedru roste exponenciálně s počtem běhů algoritmu.

- Umístění vrcholů závisí pouze na jejich bezprostředním okolí. Tato zdánlivě nenápadná vlastnost má důležitý význam pro použití v real-time renderování,

umožňuje totiž po přesunu několika bodů v kontrolní síti přepočítat pouze část modelu místo celku. To je důležité např. při animaci postav.

3.3 Terminologie

Následující definice budeme využívat ve formálních popisech operací na modelech. Jejich úkolem je především zpřesnit tyto popisy, ale také podtrhnout společné vlastnosti modelů s libovonlou topologií, na kterých je práce založena.

polygon Jedno z možných vyjádření polygonu je posloupnost bodů v prostoru. Protože ale v rámci polyedru chceme body sdílet mezi více polygony bez redundancí, zavedeme posloupnost vrcholů V, jejímiž hodnotami budou souřadnice vrcholů v $\mathbb{R}^3$. Polygon pak budeme interpretovat jako posloupnost indexů platných ve V. Samotný polygon tedy vyjadřuje pouze topologii; umístění jsou určena čistě hodnotami V. Pro sestavování polyedrů uvažujeme vždy pouze konvexní polygony. Pokud není řečeno jinak, jakýkoliv polygon je tedy implicitně konvexní.

hrana Neuspořádaná dvouprvková množina indexů $\{a, b\} : a, b \in [0..(|V| - 1)]$.

model Modelem rozumíme polyedr, sestávající ze sady vrcholů a sady polygonů. Formálně budeme model M zapisovat jako čtveřici:

$$M = (N_{V_M}, N_{P_M}, V_M, P_M)$$

Přitom

$$V_M : [0..(N_{V_M} - 1)] \to \mathbb{R}^3$$

je posloupnost vrcholů (viz. definice polygonu).

$$P_M : j \in [0..(N_{P_M} - 1)] \to ([0..k_j] \to [0..N_{V_M}])$$

je pouze formální zápis posloupnosti polygonů z předchozí definice, kde k_j je počet vrcholů j-tého polygonu (zavedeno pouze pro korektnost definice, dále je toto číslo běžně označované $|P_j|$).

N_{V_M} je počet vrcholů celého modelu a N_{P_M} počet polygonů celého modelu. Potom v je posloupnost souřadnic vrcholů v trojrozměrném prostoru a p je posloupnost zobrazení do množiny indexů v.

mapa sousednosti modelu je formálně zobrazení

$$A_M : (p, \{a, b\}) \to q,$$

přičemž

$$p, q \in [0..(N_{P_M} - 1)]; a, b \in [0..(N_{V_M} - 1)].$$

Mapa pro polygon p a hranu $\{a, b\}$ určí polygon q sousedící s polygonem p přes hranu $\{a, b\}$.

mapa vrcholů na hranách (tzv. *edge points*). Jedná se o zobrazení hran původního polyedru do množiny vrcholů výsledného polyedru, které má za úkol popisovat body umístěné na hrany modelu M při přechodu na M' subdivizí, což umožňuje eliminovat duplicity při vytváření takových vrcholů. Tedy pokud M' je výsledkem subdivize M, máme

$$V_{M\to M'} : \{\{a,b\} | a \neq b \wedge (\exists p \in P_M : ((a,b) \text{ nebo } (b,a) \text{ je podposloupnost } p))\}$$

$$\to [0..(N_{V_{M'}} - 1)] \cup \{\omega\}.$$

Vzor (neuspořádaná množina $\{a,b\}$) sestává z indexů krajních vrcholů hrany ve V_M, obraz $V_{M\to M'}(\{a,b\})$ je pak index vrcholu umístěného na původní hranu $\{a,b\}$. Dvojice $\{a,b\}$ je součástí definičního oboru $V_{M\to M'}$ právě tehdy, když spolu vrcholy $V_{M,a}$ a $V_{M,b}$ v modelu M sousedí, tj. jdou po sobě v nějakém polygonu.

Toto zobrazení budeme v průběhu subdivizních algoritmů postupně sestavovat. K tomu slouží literál ω, na který se hrana zobrazí, pokud vrchol ležící na této hraně ještě nebyl designován.

mapa dělení polygonů asociuje polygony původního modelu s polygony výsledného polyedru. Formálně jde o zobrazení

$$P_{M\to M'} : [0..(N_{P_M} - 1)] \to 2^{[0..(N_{P_{M'}}-1)]}.$$

Toto zobrazení mapuje každý polygon původního modelu na polygony, které jeho podrozdělením vzniknou ve výsledném modelu (viz. **obr. 4.1**).

Úhel na hraně $\{x,y\}$ označuje rozdíl úhlu svíraného polygony sousedícími přes hranu $\{x,y\}$ od přímého úhlu. Značíme jej $\angle\{x,y\}$.

3.4 Aproximační a interpolační algoritmy

Algoritmy subdivize se dělí do dvou skupin: aproximační a interpolační. Rozdílem je zacházení algoritmu s vrcholy vytvořenými předchozími kroky. Zatímco aproximační algoritmy při každé iteraci algoritmu přepočítávají souřadnice všech vrcholů (včetně vrcholů z předchozích kroků), interpolační algoritmy nechávají starší vrcholy bez úprav a pouze dopočítávají pozice nových vrcholů.

Přesněji řečeno pro interpolační algoritmy platí

$$\forall i \in [o..N_{V_M}] : V_M(i) = V_{M'}(i),$$

což neplatí pro třídu aproximačních algoritmů. Od této odlišnosti se také odvíjí zajímavé rozdíly ve vlastnostech:

- Interpolační algoritmy generují modely, které lépe odpovídají původní kontrolní síti – její vrcholy totiž zůstávají v modelu nedotčené, mění se pouze topologie. Tato vlastnost se týká nejen výsledného tvaru, ale i velikosti modelu. U aproximačních algoritmů totiž dochází k celkovému zmenšení modelu, pokud je kontrolní síť příliš hrubá.

- Interpolační algoritmy jsou méně náročné na výpočetní prostředky, nepotřebují totiž přepočítávat pozice vrcholů z předchozích kroků.

- Aproximační algoritmy mají na hrubých kontrolních sítích problémy s nepříliš šetrnými modifikacemi souřadnic původních vrcholů zpracovávaného polyedru. Výstupem jsou sice hladké polyedry, nicméně výsledky mohou být těžko předvídatelné (výsledné body jsou daleko od kontrolní sítě).

- Interpolační algoritmy oproti tomu nedeformují souřadnice odpovídající původnímu polyedru. Problémy s deformací zde nastávají v okolí vrcholů, jejichž valence se liší od ostatních (tzv. *extraordinary vertices*). Na těchto místech obvykle v modelu vznikají aplikací interpolačního algoritmu nekonvexní zóny.

Pro práci jsme vybrali algoritmy Catmull-Clark [1] a Butterfly [2], protože jsou původními algoritmy svých kategorií a dobře reprezentují jejich charakteristické rysy. Klíčovou roli hrála mimo jiné adaptabilita na libovolnou topologii, která je v našem případě nutností.

3.5 Formální popis použitých algoritmů

V přiloženém software jsou obsaženy implementace výše zmíněných algoritmů pro obecnou datovou strukturu. Algoritmus Butterfly bylo dále třeba zadaptovat na arbitrární topologie. Následuje formální zápis algoritmů odpovídajících této implementaci.

V rámci zjednodušení zápisu jsou v následujících vzorcích použity tyto zkratky:

- (o) – Střed souřadné soustavy – $o(0, 0, 0)$.

- $(\text{ø}p; p \in P)$ – Jedná se o aritmetický průměr všech vrcholů, tedy střed, polygonu p.

- (D_f, kde f je zobrazení) – Definiční obor zobrazení, měl by být patrný z předchozích definic.

- $(ext(f, D_f \cup \{x\}) : f(x) = y)$ – Jde o rozšíření zobrazení f na množinu $D_f \cup \{x\}$ takové, že $f(x) = y$.

- ($\{a, b\} \in s$, kde s je posloupnost) – Znamená, že čísla a a b leží v posloupnosti s bezprostředně za sebou. Toto značení používáme, potřebujeme-li zjistit, zda hrana {a, b} náleží polygonu reprezentovanému s.

3.5.1 Catmull-Clark

Jedná se o základní aproximační algoritmus založený na čtyřúhelnících. Odvození algoritmu a jeho detailní popis je k nalezení v článku [1]. Jednoduchý schematický popis je možno nalézt na webu [5]. Následuje jeho formalizace v rámci termínů použitých v práci. Tento popis odpovídá implementaci v přiloženém software.

- Inicializace. Je potřeba nastavit souřadnice starých vrcholů na o kvůli sčítání. Také musíme hodnoty $V_{M \to M'}$ nastavit na ω, abychom poznali, jestli jsou *edge points* již naalokované.

  $N_{P_{M'}} \leftarrow 0; P_{M'} : \emptyset \to \emptyset$
  $N_{V_{M'}} \leftarrow N_{V_M}; \forall i \in [0..(N_{V_M} - 1)] V_{M'}(i) = o$
  $\forall x \in D_{V_{M \to M'}} : V_{M \to M'}(x) = \omega$

- Budeme procházet model přes všechny polygony.

  $\forall \alpha \in [0..(N_{P_M} - 1)]$

  – $p \leftarrow P_M(\alpha)$

  – Přidáme vrchol – střed stěny.

  $i_m \leftarrow N_{V_{M'}}$
  $V_{M'} \leftarrow ext(V_{M'}, [0..N_{V_{M'}}]) : V_{M'}(N_{V_{M'}}) = \varnothing p$
  $N_{V_{M'}} \leftarrow N_{V_{M'}} + 1$

  – Následně projdeme všechny vrcholy právě procházeného polygonu.

  $\forall i \in [0..(|p| - 1)]$

  * Najdeme předchůdce a následníka i-tého vrcholu v polygonu p Pro všechny tři si uložíme jejich index do V_M.

  $i_c \leftarrow p_i$
  $i_p \leftarrow p_{(i-1) mod |p|}$;
  $i_s \leftarrow p_{(i+1) mod |p|}$;

  * Upravíme pozici současného vrcholu. Jakmile takto upravíme $V_{M'}(i_c)$ ze všech polygonů, do kterých i_c náleží, bude v $V_{M'}(i_c)$ součet všech vrcholů sousedících s i_c a středů všech polygonů obsahujících i_c.

  $V_{M'}(i_c) \leftarrow V_{M'}(i_c) + V_{M'}(i_m) + \frac{V_{M'}(i_p) + V_{M'}(i_s)}{2}$

* Najdeme *edge points* ležící na hranách $\{i_c, i_p\}$ a $\{i_c, i_s\}$. Pokud takové vrcholy ještě neexistují, budou vzápětí vytvořeny.

  $i_- \leftarrow V_{M\to M'}(\{i_c, i_p\})$
  $i_+ \leftarrow V_{M\to M'}(\{i_c, i_s\})$

* Najdeme polygon β sousedící se současným polygonem p přes hranu $\{i_c, i_p\}$.

  $\beta \leftarrow A_M(\alpha, \{i_c, i_p\})$

* Pokud *edge point* na hraně $\{i_c, i_p\}$ ještě neexistuje, vytvoříme ho.

  $IF(i_- = \omega)THEN$

  $i_- \leftarrow N_{V_{M'}}; V_{M\to M'}(\{i_c, i_p\}) \leftarrow N_{V_{M'}}$
  $V_{M'} \leftarrow ext(V_{M'}, [0..N_{V_{M'}}]) : V_{M'}(N_{V_{M'}}) = \frac{V_M(i_c)+V_M(i_p)+\text{ø}P_M(\beta)+\text{ø}p}{4}$

  $N_{V_{M'}} \leftarrow N_{V_{M'}} + 1$
  $ENDIF$

* Pokud již byl polygon β podrozdělen, projdeme všechny jeho podpolygony. Ten, který obsahuje hranu $\{i_-, i_c\}$, bude sousedem polygonu, který vytvoříme pro vrchol i_c.

  $\forall x \in P_{M\to M'}(\beta)$
  · $IF(\{i_-, i_c\} \in P_{M'}(x))THEN$

  $A_{M'}(x, \{i_-, i_c\}) \leftarrow N_{V_{M'}}$
  $A_{M'}(N_{V_{M'}}, \{i_-, i_c\}) \leftarrow x$
  $ENDIF$

* Tentýž proces opakujeme ještě pro hranu $\{i_c, i_s\}$.

  $\beta \leftarrow A_M(\alpha, \{i_c, i_s\})$

* $IF(i_+ = \omega)THEN$

  $i_+ \leftarrow N_{V_{M'}}; V_{M\to M'}(\{i_c, i_s\}) \leftarrow N_{V_{M'}}$
  $V_{M'} \leftarrow ext(V_{M'}, [0..N_{V_{M'}}]) : V_{M'}(N_{V_{M'}}) = \frac{V_M(i_c)+V_M(i_s)+\text{ø}P_M(\beta)+\text{ø}p}{4}$

  $N_{V_{M'}} \leftarrow N_{V_{M'}} + 1$
  $ENDIF$

* $\forall x \in P_{M \to M'}(\beta)$
  · $IF(\{i_+, i_c\} \in P_{M'}(x))THEN$

$$A_{M'}(x, \{i_+, i_c\}) \leftarrow N_{V_{M'}}$$
$$A_{M'}(N_{V_{M'}}, \{i_+, i_c\}) \leftarrow x$$
$$ENDIF$$

* Pokud polygon, který se chystáme vytvořit, není prvním podpolygonem p, přidáme do mapy sousednosti, že nový podpolygon sousedí s předchozím podpolygonem přes hranu $\{i_c, i_-\}$.

$IF(i \neq 0)THEN$

$$A_{M'}(N_{P_{M'}}, \{i_c, i_-\}) \leftarrow (N_{P_{M'}} - 1)$$
$$A_{M'}((N_{P_{M'}} - 1), \{i_c, i_-\}) \leftarrow N_{P_{M'}}$$
$$ENDIF$$

* Přidáme polygon tvořený vrcholy (i_m, i_+, i_c, i_-).

$$P_{M'} \leftarrow ext(P_{M'}, [0..N_{P_{M'}}]) : p_{M'}(N_{P_{M'}}) = \{i_m, i_+, i_c, i_-\}$$
$$P_{M \to M'}(\alpha) \leftarrow P_{M \to M'}(\alpha) \cup \{N_{P_{M'}}\}$$
$$N_{P_{M'}} \leftarrow N_{P_{M'}} + 1$$

- Nakonec upravíme souřadnice starých vrcholů podle koeficientů uvedených v online článku [5].

$\forall i \in [0..N_{V_M}]$

  – $val_i \leftarrow |p \in P_M | i \in p|$

  – $V_{M'}(i) \leftarrow \frac{(val_i - 2) \times V_M(i)}{val_i} + \frac{V_{M'}(i)}{(val_i)^2}$

3.5.2 Butterfly

Algoritmus Butterfly je základním prvkem třídy interpolačních subdivizních algoritmů. Obecně tento algoritmus pracuje pouze na trojúhelníkových sítích. V přiloženém software je implementováno zobecnění tohoto algoritmu na libovolnou topologii, kterému odpovídá i následující popis.

Nevýhody lokálních deformací souvisejících s vrcholy s odlišnou valencí se ovšem projevují i v této zobecněné formě. Odvození algoritmu Butterfly můžeme nalézt v článku [2], jednoduchý schematický popis pak opět na webu [5].

- Inicializace

$w \leftarrow 0$
Nastavení koeficientů (viz. obrázek **3.1**)

$a \leftarrow \frac{1}{2}; b \leftarrow \frac{1}{8} + 2 \times w; c \leftarrow -\frac{1}{16} - w$
$N_{P_{M'}} \leftarrow 0; P_{M'} \leftarrow (\emptyset \rightarrow \emptyset)$
Souřadnice původních vrcholů zde neměníme.

$N_{V_{M'}} \leftarrow N_{V_M}; V_{M'} \leftarrow V_M$
$N_s \leftarrow 0; s \leftarrow (\emptyset \rightarrow \emptyset)$
$\forall x \in D_{V_{M \rightarrow M'}} : V_{M \rightarrow M'}(x) = \omega$

- Budeme procházet model přes všechny polygony.

  $\forall \alpha \in [0..(N_{P_M} - 1)]$

  - $p \leftarrow P_M(\alpha)$

  - Přidáme poslední subpolygon p do mapy dělení polygonů. Jde o vepsaný polygon, vznikající spojením všech *edge points* p, bude naalokován na konci průchodu polygonem p.

    $P_{M \rightarrow M'}(\alpha) \leftarrow P_{M \rightarrow M'}(\alpha) \cup \{N_{P_{M'}} + |p|\}$

  - Nyní projdeme všechny vrcholy p.

    $\forall i \in [0..(|p| - 1)]$

    * Označíme vznikající polygon jako subpolygon p.

      $P_{M \rightarrow M'}(\alpha) \leftarrow P_{M \rightarrow M'}(\alpha) \cup \{N_{P_{M'}}\}$

    * Najdeme předchůdce a následníka i-tého vrcholu v polygonu p Pro všechny tři si uložíme jejich index do V_M.

      $i_c \leftarrow p_i$
      $i_p \leftarrow p_{(i-1) mod |p|}$;
      $i_s \leftarrow p_{(i+1) mod |p|}$;

    * Najdeme *edge points* ležící na hranách $\{i_c, i_p\}$ a $\{i_c, i_s\}$. Pokud takové vrcholy ještě neexistují, budou vzápětí vytvořeny.

      $i_- \leftarrow V_{M \rightarrow M'}(\{i_c, i_p\})$
      $i_+ \leftarrow V_{M \rightarrow M'}(\{i_c, i_s\})$

    * Pokud *edge point* na hraně $\{i_c, i_p\}$ ještě neexistuje, vytvoříme ho.

      $IF(i_- = \omega) THEN$

$i_- \leftarrow N_{V_{M'}}; V_{M\to M'}(\{i_c, i_p\}) \leftarrow N_{V_{M'}}$
$V_{M'} \leftarrow ext(V_{M'}, [0..N_{V_{M'}}]) : V_{M'}(N_{V_{M'}}) = o$

$N_{V_{M'}} \leftarrow N_{V_{M'}} + 1$
$ENDIF$

* Pokud *edge point* na hraně $\{i_c, i_s\}$ ještě neexistuje, vytvoříme ho.

$IF(i_+ = \omega)THEN$

$i_+ \leftarrow N_{V_{M'}}; V_{M\to M'}(\{i_c, i_s\}) \leftarrow N_{V_{M'}}$
$V_{M'} \leftarrow ext(V_{M'}, [0..N_{V_{M'}}]) : V_{M'}(N_{V_{M'}}) = o$

$N_{V_{M'}} \leftarrow N_{V_{M'}} + 1$
$ENDIF$

* Najdeme sousedy současného polygonu p přes hrany $\{i_c, i_p\}$ a $\{i_c, i_s\}$.
$\beta \leftarrow A_M(\alpha, \{i_c, i_p\})$
$\gamma \leftarrow A_M(\alpha, \{i_c, i_s\})$

* Pokud již byl polygon β podrozdělen, projdeme všechny jeho podpolygony. Ten, který obsahuje hranu $\{i_-, i_c\}$, bude sousedem polygonu, který vytvoříme pro vrchol i_c

$\forall x \in P_{M\to M'}(\beta)$

$IF(\{i_-, i_c\} \in P_{M'}(x))THEN$
$A_{M'}(x, \{i_-, i_c\}) \leftarrow N_{V_{M'}}$

$A_{M'}(N_{V_{M'}}, \{i_-, i_c\}) \leftarrow x$
$ENDIF$

* Pokud již byl polygon γ podrozdělen, projdeme všechny jeho podpolygony. Ten, který obsahuje hranu $\{i_+, i_c\}$, bude sousedem polygonu, který vytvoříme pro vrchol i_c

$\forall x \in P_{M\to M'}(\gamma)$

$IF(\{i_+, i_c\} \in P_{M'}(x))THEN$
$A_{M'}(x, \{i_+, i_c\}) \leftarrow N_{V_{M'}}$

$A_{M'}(N_{V_{M'}}, \{i_+, i_c\}) \leftarrow x$

ENDIF

* Zapíšeme sousednost mezi nově vznikajícím vnějším subpolygonem a vnitřním, vepsaným, subpolygonem.

  $A_{M'}(N_{P_{M'}}, \{i_-, i_+\}) \leftarrow max(P_{M \to M'}(\alpha))$
  $A_{M'}(max(P_{M \to M'}(\alpha)), \{i_-, i_+\}) \leftarrow N_{P_{M'}}$

* Přidáme trojúhelník sestávající z vrcholů (i_-, i_c, i_+).

  $P_{M'} \leftarrow ext(P_{M'}, [0..N_{P_{M'}}]) : P_{M'}(N_{P_{M'}}) = \{i_-, i_c, i_+\}$
  $N_{P_{M'}} \leftarrow (N_{P_{M'}} + 1)$

* Přidáme bod i_+ do posloupnosti s. Potom co projdem všechny vrcholy p, bude s obsahovat všechny vrcholy vepsaného subpolygonu.

  $s \leftarrow ext(s, [0..N_s]) : s(N_s) = i_+$
  $N_s \leftarrow (N_s + 1)$

* Najdeme pořadí vrcholů i_c a i_p v polygonu β.

  $i_{c_\beta} \leftarrow (P_M(\beta))^{-1}(i_c)$
  $i_{p_\beta} \leftarrow (P_M(\beta))^{-1}(i_p)$

* Najdeme pořadí vrcholů i_c a i_s v polygonu γ.

  $i_{c_\gamma} \leftarrow (P_M(\gamma))^{-1}(i_c)$
  $i_{s_\gamma} \leftarrow (P_M(\gamma))^{-1}(i_s)$

* Spočítáme index vrcholu ležícího v polygonu β naproti hraně $\{i_c, i_p\}$.

  $o_\beta \leftarrow P_M(((i_{p_\beta} - i_{c_\beta}) \times \lceil \frac{|P_M(\beta)|}{2} \rceil + i_{c_\beta}) mod |P_M(\beta)|)$

* Spočítáme index vrcholu ležícího v polygonu γ naproti hraně $\{i_c, i_s\}$.

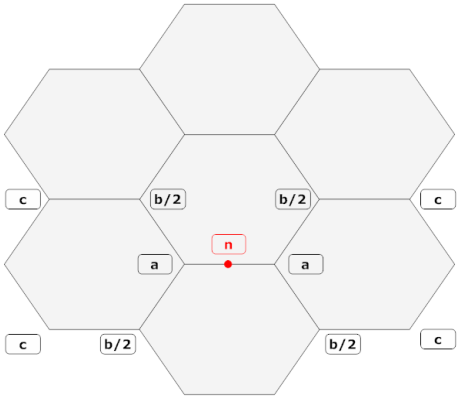
  $o_\gamma \leftarrow P_M(((i_{s_\gamma} - i_{c_\gamma}) \times \lceil \frac{|P_M(\gamma)|}{2} \rceil + i_{c_\gamma}) mod |P_M(\gamma)|)$

* Nastavíme souřadnice *edge points* i_- a i_+ podle šablony viditelné na obrázku 3.1.

  $V_M(i_-) \leftarrow V_M(i_-) + \frac{a}{2} \times V_M(i_c) + \frac{b}{2} \times V_M(i_s) + c \times V_M(o_\gamma)$
  $V_M(i_+) \leftarrow V_M(i_+) + \frac{a}{2} \times V_M(i_c) + \frac{b}{2} \times V_M(i_p) + c \times V_M(o_\beta)$

– Přidáme vnitřní polygon a vynulujeme posloupnost s.

$P_{M'} \leftarrow ext(P_{M'}, [0..N_{P_{M'}}]) : P_{M'}(N_{P_{M'}}) = s$
$N_{P_{M'}} \leftarrow (N_{P_{M'}} + 1)$
$s \leftarrow (\emptyset \rightarrow \emptyset)$
$N_s \leftarrow 0$



Obrázek 3.1: Šablona upraveného Butterfly algoritmu

3.6 Některé další subdivizní algoritmy

Algoritmy Catmull-Clark a Butterfly pochopitelně nejsou jedinými algoritmy zabývajícími se 3D subdivizí – kolem roku 1978 vyšlo dost publikací na toto téma – spíše než vznik spousty algoritmů ale můžeme pozorovat postupné vylepšování několika subdivizních metod. Za všechny zmiňme algoritmy Daniela Doo (občas také nazývaný Doo-Sabin) a Charlese Loopa.

3.6.1 Doo

V roce 1978 vytvořil Daniel Doo algoritmus pro podrozdělování, založený na vkládání menší kopie sebe sama do každého polygonu. Jeho detailní popis můžeme najít v článku [8]. Tento algoritmus není v praxi příliš používaný, mezi subdivizními algoritmy je však unikátní tím, že odstraňuje staré hrany a nahrazuje je novými. Je také schopný se vypořádat s libovolnou topologií a navíc sám vytváří polygony s různým počtem vrcholů.

Přestože tento algoritmus lze formálně zařadit mezi aproximační algoritmy, tvoří spíše třídu sám pro sebe. Nereprezentuje totiž charakteristické znaky aproximačních, ani interpolačních algoritmů, a proto je nepoužitelný pro zobecnění výsledků.

3.6.2 Loop

Algoritmus který vytvořil Charles Loop je oproti ostatním subdivizním algoritmům relativně nový – byl prezentován v roce 2002. Jde o aproximační algoritmus pro trojúhelníkovou topologii, který používá tzv. *ternární subdivizi* (viz. článek [6]).

Tento algoritmus dobře reprezentuje třídu aproximačních algoritmů. Díky trojúhelníkové topologii navíc vypadá slibně pro spolupráci s algoritmem Butterfly, proto by byl přínosným přírůstkem mezi implementované algoritmy. Jeho adaptace na libovolnou topologii je ale přinejmenším složitá.

Kapitola 4

Srovnávání

4.1 Co přesně přináší prokládání

Primárním přínosem prokládání různých algoritmů je eliminace problémů specifických pro jeden algoritmus. Zvažme libovolné dva různé subdivizní algoritmy A a B. Použitím algoritmu A na model podrozdělený algoritmem B obvykle snižujeme efekt deformace specifický pro algoritmus B. Toto platí speciálně pro prokládání interpolačních algoritmů s aproximačními: jako příklad můžeme uvést celkový povrch modelu - zatímco algoritmus Catmull-Clark [1] celkový povrch modelu zmenšuje (a také model reálně zmenšuje), algoritmus Butterfly [2] činí pravý opak, protože nepřemisťuje staré vrcholy blíže k novým. S použitím prokládání se dá vliv na celkovou velikost modelu regulovat.

4.2 Kdy a jak prokládat, omezení

Přínos prokládání může být diskutabilní pro specifické modely, např. modely s regulární trojúhelníkovou topologií, které je lepší zpracovat algoritmem vytvořeným pro tu kterou topologii (zde např. Butterfly).

Prokládání dále nemá valný dopad pokud již na začátku pracujeme s komplexní kontrolní sítí (řádově stovky vrcholů, tzv. high-end rendering). V tomto případě se vyplatí vybrat si jeden aproximační algoritmus a ten používat – deformace modelu se na takto složitém modelu projeví minimálně.

Prokládání různých algoritmů má tedy význam, chceme-li pracovat na hrubém modelu s libovolnou topologií a vytvořit z něj v rámci možností věrohodnou imitaci objektu.

Pozorováním lze určit, že nejdřív je vhodné použít interpolační algoritmus, abychom zachovali tvar a rozměry modelu a vytvořili další vrcholy – většinou postačí jedna až tři iterace, v závislosti na oblosti cílového objektu, který se snažíme modelovat (méně interpolací tvoří oblejší cílový model), a komplexitě kontrolní sítě. Pokud námi zvolený interpolační algoritmus podporuje nastavení

koeficientu napětí (*tension coefficient*, např. algoritmus Butterfly), je dobré tuto hodnotu nastavit relativně malou – omezí to výraznost možných nekonvexit vytvořených interpolací.

Následují aplikace zvoleného aproximačního algoritmu, jejichž cílem je vyhladit zbývající nerovnosti a dodat modelu realistický vzhled, většinou postačí tři až šest iterací, podle úrovně jemnosti, které chceme dosáhnout. Pokud má model charakteristické nerovnosti na povrchu a přejeme si je zvýraznit, vložíme mezi iterace aproximačního algoritmu ještě jeden až dva běhy interpolačního algoritmu.

4.3 Implementace měření v přiloženém software

V implementovaném prostředí nalezneme tři druhy měření odpovídající předchozímu odstavci:

Obsah polygonů (s) měříme, abychom dostali celkovou představu o hrubosti výsledného modelu. Pro renderování animací se používá poučka, že subdivizi je vhodné aplikovat, dokud velikost jednoho polygonu není zhruba stejná, jako velikost jednoho pixelu. Tyto míry jsou sice pro herní animaci nepoužitelné, pravdou ale zůstává, že větší rovné plochy jsou to, co obvykle na modelu působí uměle.

Délka hran (e) je míra úzce spjatá s obsahem polygonů, protože příliš dlouhé hrany stejně často dávají modelu umělý vzhled. Toto měření lze také použít pro penalizaci vzniku polygonů s nízkým obsahem v poměru k délce.

Úhel mezi sousedícími polygony (a) je míra, která vyjadřuje, jak moc hranatě model působí. Přesněji řečeno, měříme odchylku úhlu mezi polygony od přímého úhlu, tedy úhel mezi normálovými vektory. Čím více se totiž přibližujeme přímému úhlu, tím hladčeji a plynuleji model působí.

U každého z poskytnutých měření jsou dostupné tyto hodnoty:

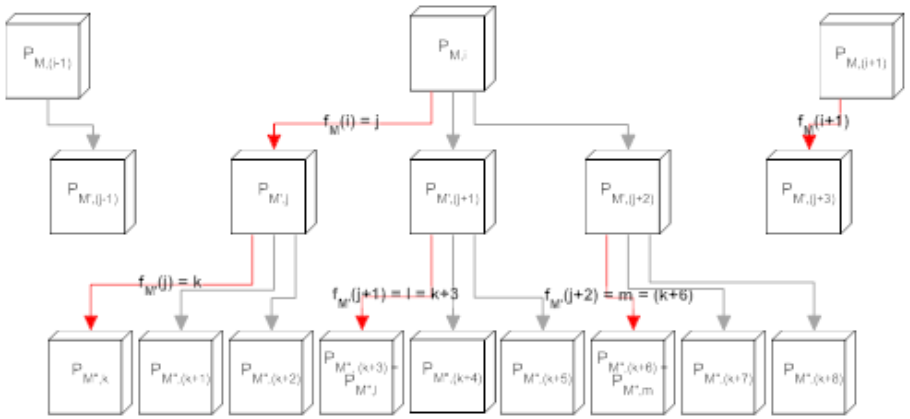
Aritmetický průměr (avg) je v našem případě nejzákladnější míra, kterou kvantifikujeme základní kvality modelu.

Variance (var) nám poskytuje představu o heterogenitách na modelu.

Maximum a minimum (max, min) slouží jako doplňkové míry, např. pokud chceme znát nejhorší případ.

Průměr poměru k předchozí hodnotě (d) je hodnota, která nám slouží z větší části k penalizaci deformací modelu v procesu zjemňování - pro každou měřenou veličinu změříme poměr nové hodnoty vůči staré tam, kde to má smysl, a výsledky zprůměrujeme.

Obrázek 4.1:



Důležitou roli pří porovnávání nové hodnoty s předchozí hraje určení kontextu porovnávání. Topologický vývoj modelu aplikací subdivizních algoritmů můžeme znázornit s použitím stromu, tak jak ukazuje schéma 4.1 - každé patro stromu odpovídá jednomu stádiu modelu.

Vazba mezi indexy jednotlivých stádií polygonů (funkce f z obrázku 4.1) je zachována pomocí překladového pole – kontext porovnání polygonu $P_{S(M),i}$ je právě polygon $P_{M,j}$, kde

$$j = f^{-1}(sup(x \in f_M[0..N_{P_M}|x <= i])).$$

To umožňuje měřit výše popsané odchylky:

odchylka d_e délky hrany Pro $\{a, c\} \in P_{S(M),i}$ počítáme odchylku

$$d_e(\{a, c\}) = \frac{l(\{a, c\})}{\frac{1}{2} \times l(\{a, b\})},$$

kde

$$\{a, b\} = \{P_{M,j}(x), P_{M,j}((x + 1)\ mod\ k_{M,j})\} :$$
$$x \in [0..k_{M,j}], P_{M,j}(x) = a \vee P_{M,j}(x + 1) = a,$$

pokud takové $\{a, b\}$ existuje, tzn. $a \in [0..N_{V_M}]$.

Pokud takové $\{a, b\}$ neexistuje, poměr není směrodatný, tudíž se nezapočítává do průměru. Pozn.: pokud

$$a \notin [0..N_{V_M}], \text{ ale } c \in [0..N_{V_M}],$$

jedná se o stejný případ, pouze s opačným pojmenováním.

odchylka d_s obsahu polygonu Jde o porovnání obsahu nového polygonu s rovným dílem obsahu starého polygonu:

$$d_s(P_{S(M),i}) = \frac{S(P_{S(M),i})}{\frac{1}{k_{M,j}} \times S(P_{M,j})},$$

kde j je výše popsaný index polygonu z předchozí generace.

odchylka d_a úhlu mezi polygony Zvažujeme hranu $\{a, b\}$ nalezenou stejně jako v případě d_e. Obdobně jako u d_e, počítáme pouze na hranách vzniklých vložením bodu na již existující hranu.

$$d_a(\{a, c\}) = \frac{\angle\{a, b\}}{\angle\{a, c\}}$$

4.4 Výsledky vs. pozorování

Empiricky můžeme ověřit, jak dobře odpovídají zvolené míry kritériím uvedeným v sekci **2.3**. Zajímat nás bude relevance měření využívajících porovnání s předchozím stádiem modelu – účel a funkčnost ostatních měření jsou zřejmé.

Poměry délky hran (d_e) a plochy stěn (d_s) mají za úkol sledovat především vývoj celkového objemu modelu. Optimum pro poměr plochy lze určit relativně jednoduše: $d_s = 1$ odpovídá zároveň rovnoměrnému rozdělení ploch i zachování celkové plochy, proto je to pro nás optimální hodnota.

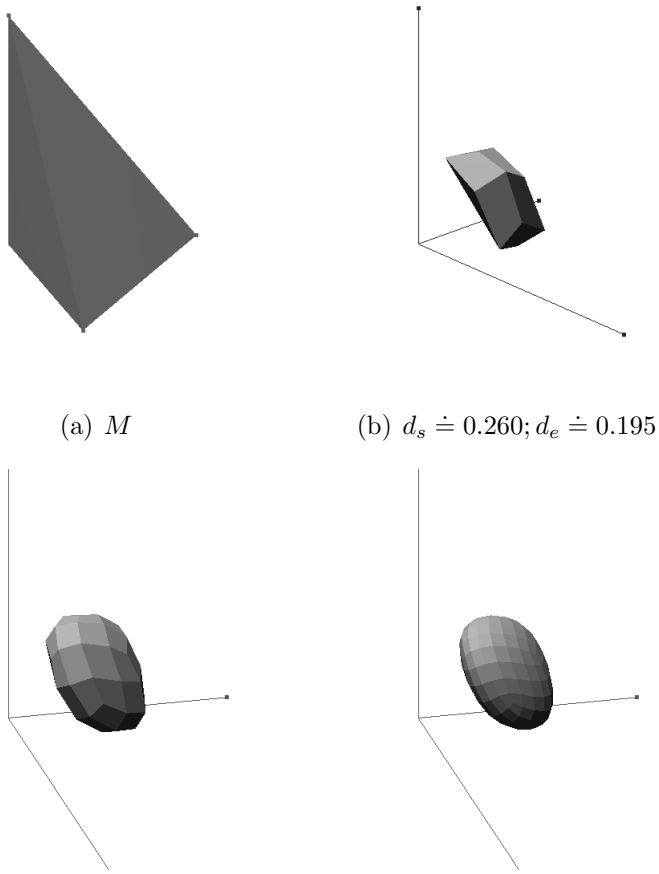
Co se týče odchylky délky hran, závisí optimální hodnota na počtu hran, na které se při každém kroku každá hrana původního modelu rozdělí. Oba implementované algoritmy rozdělují hranu na dvě části, dále tedy budeme považovat $d_e = \frac{1}{2}$ za optimální hodnotu. Pokud by se podobné měření vztahovalo i na algoritmus Loop (nebo další algoritmy využívající jiné než binární subdivize), bylo by třeba tento speciální případ ošetřit.

Na obrázcích **4.2** a **4.3** vidíme postupné zpracování tetraedru algoritmy Catmull-Clark a Butterfly spolu s naměřenými hodnotami d_s a d_e.

Je viditelné, že algoritmus Catmull-Clark nadmíru zmenšil model, což se projevilo i na hodnotách měření. Naopak algoritmus Butterfly zachoval původní kontrolní síť a polyedr lehce nabyl na objemu, což je opět znát i na hodnotách d_s a d_e.

Poslední implementovanou mírou porovnávající aktuální model s předchozím stádiem je d_a – ta měří poměr odchylek od přímého úhlu na stejné hraně. Optimální hodnota se zde určuje obtížně, proto zvolme interval, ve kterém by se hodnota měla pohybovat. Připomeňme, že

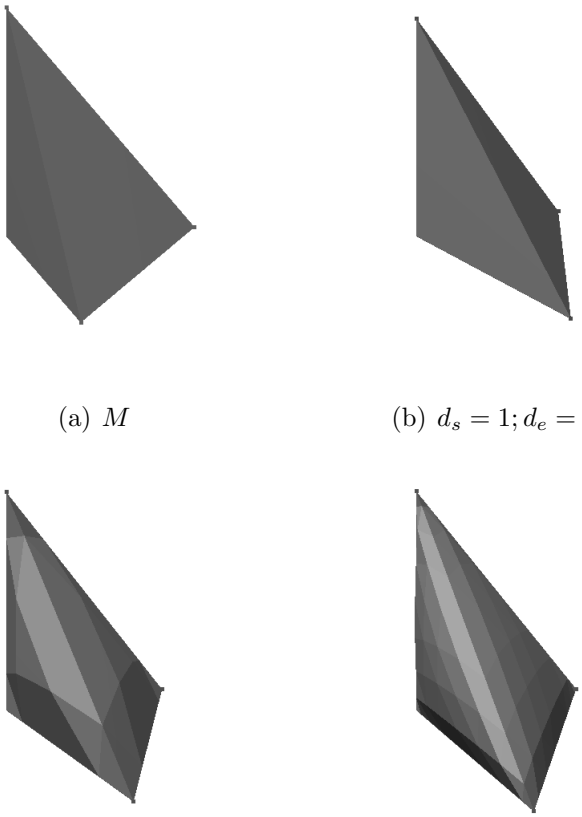
$$d_a(\{a, c\}) = \frac{\angle\{a, b\}}{\angle\{a, c\}},$$



(a) M (b) $d_s \doteq 0.260; d_e \doteq 0.195$

(c) $d_s \doteq 0.708; d_e \doteq 0.390$ (d) $d_s \doteq 0.922; d_e \doteq 0.470$

Obrázek 4.2: Zpracování tetraedru algoritmem Catmull-Clark



(a) M (b) $d_s = 1; d_e = 0.5$

(c) $d_s \doteq 1.030; d_e \doteq 0.510$ (d) $d_s \doteq 1.032; d_e \doteq 0.507$

Obrázek 4.3: Zpracování tetraedru algoritmem Butterfly

(kde $\{a, b\}$ je odpovídající hrana na předchozím stádiu polyedru, viz. sekce 3.3, str. 10). Očekávali bychom, že $\angle\{a, c\}$ bude i v případě silnějšího vyhlazení nejméně $\frac{1}{2} \times \angle\{a, b\}$. Nižší hodnota (odchylka od přímého úhlu) znamená buď příliš horlivé zaoblování, nebo vznik nekonvexit na neměřených hranách modelu, v horším případě obojí. Zároveň nechceme, aby subdivize hrany zostřovala, $\angle\{a, c\}$ by tedy měla dosahovat nejvýše hodnoty $\angle\{a, b\}$. To znamená, že hodnota $d_a(\{a, c\})$ by se měla pohybovat po intervalu $\langle 1; 2\rangle$.

Na **4.4(a)** vidíme rozdíl mezi modelem živočicha upraveným algoritmem Catmull-Clark a algoritmem Butterfly. Použití algoritmu Butterfly je v tomto případě nevhodné, protože v černých oblastech došlo k vytvoření nekonvexit s následkem překřížení stěn. V rámci měření se tento jev promítne především do úhlových deformací: s použitím alg. Catmull-Clark $d_a \doteq 1.772$ (v očekávaném intervalu), zatímco při použití Butterfly $d_a \doteq 4.039$, což je hodnota výrazně mimo povolený rozsah.

4.5 Návrh agregační funkce měření

K rozšíření možností měření je vhodné použít agregační funkci, která zohledňuje námi vybraná kritéria a seskupuje je do jedné míry. Nemá valný smysl používat všechna implementovaná měření, nejen protože některá z nich jdou částečně proti sobě – například penalizace deformace úhlu a průměr úhlových odchylek – ale také proto, že žádný z testovaných algoritmů pravděpodobně nebude vynikat ve všech ohledech. Je tedy potřeba předem zvolit, na co budeme při ohodnocování algoritmů klást důraz a podle toho sestavit agregační funkci.

Agregační funkce by měla být

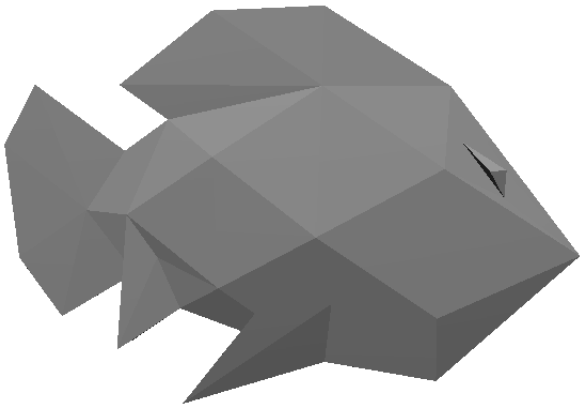
1. modulární (měli bychom být schopni přidávat a ubírat jednotlivé komponenty, aniž by se tyto změny výrazně projevily na formátu výsledku)

2. omezená (pevný interval hodnot).

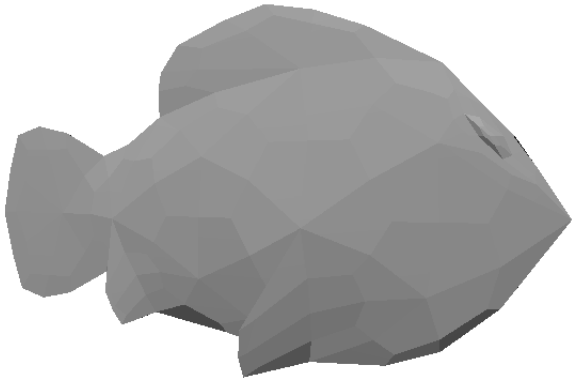
V tabulce **4.1** je prezentovaný návrh převodu měření na komponenty takové funkce. Všechna měření jsou zde převedena na výstupy s rozsahem hodnot v intervalu $\langle 0; 1\rangle$. Samotnou agregační funkci pak můžeme sestavit např. jako vážený průměr hodnot:

$$A = \begin{pmatrix} a_1 & \dots & a_9 \end{pmatrix} \times \begin{pmatrix} b_1 \\ \vdots \\ b_9 \end{pmatrix},$$

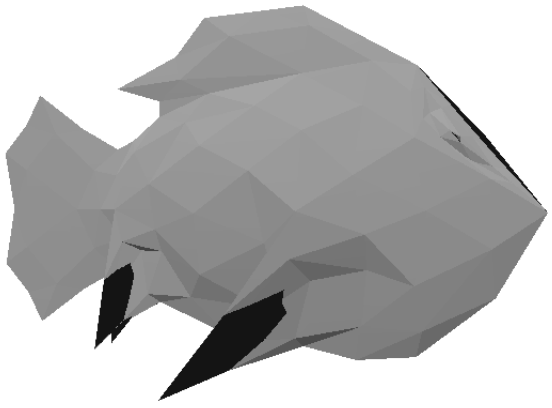
kde $\begin{pmatrix} b_1 \\ \vdots \\ b_9 \end{pmatrix}$ jsou zvolené váhy, které splňují $\sum_{i=1}^{i\leq 9} b_i = 1$.



(a) Původní model



(b) Po kroku Catmull-Clark



(c) Po kroku Butterfly

Pokud nás zajímají deformace modelu v průběhu sekvence, je vhodné zaznamenávat výstupy po každém kroku subdivize, protože měření umožňují pouze výpočet deformace oproti předchozímu stádiu modelu. Výstupy agregační funkce $A^1 \dots A^k$ (k je počet subdivizí) pak můžeme spojit součinem

$$\left(A = \prod_{i=1}^{i \leq k} A^i \right),$$

nebo opět váženým průměrem

$$\left(\sum_{i=1}^{i \leq k} c_k A^i \right), \text{ kde } \left(\sum_{i=1}^{i \leq k} c_i = 1 \right).$$

ozn.	**Míra**	**Charakteristika**	**Obor hodnot**	**Optimum**
a_1	$\frac{1}{1+avg_s}$, ev. $\frac{1}{1+max_s}$	Ohodnocení jemnosti ploch.	$(0;1)$	1
a_2	$\frac{1}{1+avg_e}$, ev. $\frac{1}{1+max_e}$	Ohodnocení jemnosti hran.	$(0;1)$	1
a_3	$\frac{1}{1+avg_a}$, ev. $\frac{1}{1+max_a}$	Ohodnocení hladkosti povrchu.	$\langle\frac{1}{181};1)$	1
a_4	$1-\lvert 1-\sqrt{d_s}\rvert$	Penalizace odchylky obsahu nového polygonu od rovného dílu obsahu starého.	$(0;1\rangle$	1
a_5	$1-\lvert\frac{1}{2}-d_e\rvert$	Penalizace odchylky délky nové hrany od poloviny délky staré hrany.	$(0;1\rangle$	1
a_6	$\frac{1}{1+max(\lvert\frac{3}{2}-d_a\rvert-\frac{1}{2},0)}$	Penalizace příliš masivní modifikace úhlu. Změna úhlu subdivizí nemá optimální hodnotu, proto budeme penalizovat pouze hodnoty mimo určený interval – v našem případě $\langle 1;2\rangle$ – pomocí vzdálenosti d_a od tohoto intervalu.	$(0;1\rangle$	1
a_7	$\frac{avg_s}{avg_s+\sqrt{var_s}}$	Penalizace heterogenity ploch.	$(0;1\rangle$	1
a_8	$\frac{avg_e}{avg_e+\sqrt{var_e}}$	Penalizace heterogenity délky hran.	$(0;1\rangle$	1
a_9	$\frac{avg_a}{avg_a+\sqrt{var_a}}$	Penalizace heterogenity úhlu.	$(0;1\rangle$	1

Tabulka 4.1: komponenty agregační funkce

Kapitola 5

Závěr

5.1 Shrnutí

Souvislost po částech první derivace křivky povrchu modelu je dodnes pilířem subdivizních algoritmů. Je to zároveň jedna z jejich nejsilnějších vlastností a jedno z jejich nejradikálnějších omezení – algoritmy obvykle vycházející z *patches* a bikubických B-spline nejsou schopné rozlišit, kdy se spojitost první derivace nehodí.

Metody měření prezentované v této práci sice napomáhají penalizovat sekvence algoritmů za to, že např. některé hrany zaoblují příliš radikálně, nebo že vehementně snižují celkovou velikost modelu a přesouvají důležité body, ale ještě nemění samotné chování algoritmů.

Částečnou odpověď na otázku co s tím můžeme najít u Pixar Studios, kde se neustále vyvíjí algoritmus Catmull-Clark přidáváním pravidel pro speciální případy. Jedním z vylepšení je právě přidání pravidel pro speciálně označené *ostré* hrany a vrcholy, jiné se zase týká interpolace označených vrcholů, aby se zamezilo nežádoucí redukci objemu polyedru.

Otázkou zůstává, zda by taková pravidla nebylo možné zavést, aniž bychom museli ručně značit hrany a vrcholy pro rozeznání speciálních případů.

5.2 Perspektivy

5.2.1 Rozhraní pro specifikaci agregační funkce

V rámci rychlejšího zpracování výsledků měření by bylo dobré aplikaci rozšířit o rozhraní, které by přijímalo vzorec pro agregační funkci a spolu se samotným měřením by počítalo její hodnotu.

5.2.2 Dávkové zpracování

Ruční aplikace subdivizí jedna po druhé je zdlouhavé a pozorování výsledků často ještě zdlouhavější. Proto bych v budoucnu rád do aplikace přidal dávkové zpracování zvolených subdivizí na vybranou sérii testovacích dat a žurnálování výsledků do formy vhodné pro strojové zpracování.

5.2.3 Počítání úhlových deformací

V neposlední řadě by bylo možné vylepšit měření o počítání deformací úhlů modelu, tzn.

přechodů úhlu mezi polygony z ostrého na tupý Toto měření by mohlo pomoci pro penalizaci nemístného zaoblování ostrých hran.

prolomení úhlu vzniku nekonvexní oblasti následkem lokální deformace

Obě tyto míry by odpovídaly počtu deformací (popř. procentu vůči celkovému počtu zkoumaných úhlů) a mohly by sloužit coby thresholdový filtr nebo penalizační míra při vybírání sekvencí z dávkového zpracovaní.

5.2.4 Formulace subdivizních algoritmů vzorcem

Pravidla subdivizních algoritmů mají sice složité základy, ale sama o sobě jsou celkem jednoduchá (což se občas projeví na výsledcích). Proto by je bylo možné kodifikovat do vzorců a tímto způsobem umožnit snadné testování nových schémat (porovnáním efektivity s již existujícími). Tato metoda by zároveň umožnila specifikovat algoritmy, které nejsou formálně založeny na principu aproximace křivek. Taková formulace by pak měla větší vyjadřovací potenciál než klasické odvozování subdivizních schémat.

Literatura

[1] CATMULL, E.; CLARK, J. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-aided design.* Vol. 10, no. 6 (November 1978), s. 350-355. Digitalizovaný článek je ve formě PDF dostupný na: <http://www.idi.ntnu.no/~fredrior/files/Catmull-Clark%201978%20Recursively%20generated%20surfaces.pdf>. ISSN 0010-4485.

[2] DYN, Nira; LEVIN, David; GREGORY, John A. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Transactions on Graphics.* Vol. 9, no. 2 (April 1990), s. 160-169. Digitalizovaný článek je ve formě PDF dostupný na: <http://www.math.tau.ac.il/~niradyn/papers/butterfly.pdf>. ISSN 0730-0301.

[3] WARREN, Joe; WEIMER, Henrik. *Subdivision Methods for Geometric Design : A Constructive Approach.* San Francisco : Morgan Kaufmann Publishers, c2002. xvi, 299 s. ISBN 1-55860-446-4.

[4] SHREINER, Dave, et al. *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Version 1.4.* 4th ed. Boston : Addison-Wesley, c2004. xliii, 759 s. ISBN 0-321-17348-1.

[5] SHARP, Brian. *Subdivision Surface Theory* [online]. April 11, 2000 [cit. 2009-07-21]. Dostupné na: <http://www.gamasutra.com/view/feature/3177/subdivision_surface_theory.php>

[6] LOOP, Charles. Smooth Ternary Subdivision of Triangle Meshes. In *Curve and Surface Fitting: Saint-Malo 2002.* Nashville (TN) : Nashboro Press, c2002, pp. 1–3. Digitalizovaný článek je formě PDF dostupný na: <http://research.microsoft.com/en-us/um/people/cloop/LoopCS2002.pdf>

[7] CATMULL, Edwin Earl. *A subdivision algorithm for computer display of curved surfaces : dissertation* [online]. University of Utah, Department of Computer Science, December 1974 [cit. 2009-07-21]. vi, 77 s. Dostupné ve formě PDF na: <http://thepixartouch.typepad.com/files/catmull_thesis.pdf>

[8] DOO, D. W. H. *A subdivision algorithm for smoothing down irregularly shaped polyhedrons* In Proceedings on Interactive Techniques in Computer Aided Design. 1978, pp. 157 – 165. Digitalizovaný článek je formě PDF dostupný na: <http://www.idi.ntnu.no/~fredrior/files/Doo%201978%20Subdivision%20algorithm.pdf>

Příloha A

Uživatelská dokumentace

A.1 Funkce

Funkcí aplikace je prohlížení výsledků různých algoritmů 3D subdivize za účelem jejich porovnávání.

Aplikace je testovaná pro funkčnost na systémech Windows XP a Vista, platformy 32bit a 64bit.

A.2 Instalace a odinstalování

Ke spuštění aplikace není potřeba instalace, stačí zkopírovat spustitelný soubor.

Vzhledem k tomu, že software využívá knihovnu OpenGL k vykreslování 3D objektů, je třeba, aby cílový stroj touto knihovnou disponoval. Binární distribuce této knihovny by měla být součástí ovladačů ke grafické kartě.

K odinstalování aplikace stačí smazat spustitelný soubor.

A.3 Ovládání

Při otevření, zavření či uložení dokumentu s modelem postupujte jako u standardních aplikací operačního systému Windows(R), t.j. pomocí známých klávesových zkratek nebo pomocí **File→Open**, **File→Close** a **File→Save**.

Jakmile otevřete nějaký soubor obsahující model, obsah dokumentu se zobrazí v novém okně. Model otočíte držením levého tlačítka a posouváním myši. Můžete jej také přiblížit či oddálit držením pravého tlačítka a posouváním myši nahoru nebo dolů.

Na model můžete aplikovat jednotlivé subdivizní algoritmy tak, že vyberete příslušný algoritmus z nabídky **Subdivision** v menu hlavního okna. Subdivize a veškeré další úpravy se aplikují na model v posledním aktivním okně.

A.3.1 Možnosti zobrazení

Zobrazení statistik je možné zapnout/vypnout pomocí **View→Stats**

Drátěný model často umožní lépe prozkoumat topologii modelu. Aktivujeme ho pomocí **View→Wireframe**

Normálové vektory určují kolmice k polygonům. Jejich zobrazení zapneme zaškrtnutím **View→Normals**

Je možné, že při prohlížení modelu při stínovaném zobrazení (t.j. s nezaškrtnutým **View→Wireframe**) narazíte na polyedr, který nebude odrážet světlo. To proto, že soubory s modely neobsahují informace o vnitřní nebo vnější stěně polyedru. Proto lze pomocí **Edit→Invert normals** obrátit kolmice určující odraz světla (zobrazitelné pomocí **View→Normals**), následkem čehož by se polyedr měl zobrazit normálně.

A.4 Data

Aplikace uchovává data o modelech v jednotlivých souborech, t.j. co model, to soubor. Podporovány jsou dva různé formáty datových souborů pro uchovávání informací o modelech: formát `*.txt` a formát `*.model`.

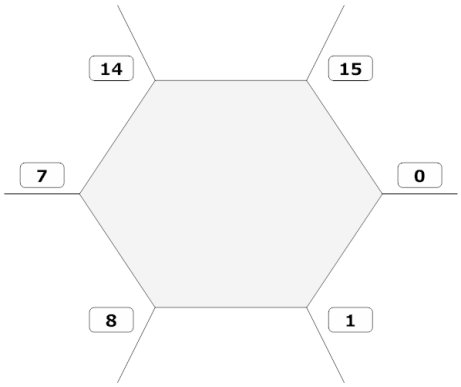
A.4.1 Formát `*.txt`

Jedná se o textový formát uchovávání informací o modelech. Je určen především k přenosu a ruční úpravě dat - dá se použít libovolný textový editor, například poznámkový blok operačního systému Windows (R). Soubor je rozdělen na následující dvě části:

Část s vrcholy

V této části jsou zapsány souřadnice všech vrcholů modelu ve formátu (x, y, z), kde x, y a z jsou jednotlivé složky v trojrozměrném kartézském souřadnicovém systému. Pořadí je v této části důležité, protože vrcholy budou v následující části identifikovány právě podle něj. Zápis se řídí následujícími pravidly:

- Každý řádek začíná znakem `(`. První řádek nesplňující tuto podmínku je považován za začátek následující sekce.
- Za znakem `(` následují souřadnice na třech osách prostoru, oddělené čárkou, ukončené znakem `)`
- Text za znakem `)` je ignorován, znovu se interpretuje až další řádek. Tuto vlastnost lze využít ke psaní komentářů.



Obrázek A.1: Platný zápis polygonu

Část se stěnami

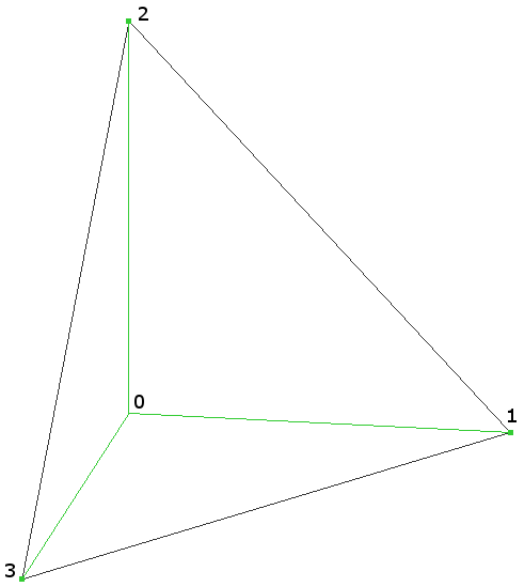
V této části je zaznamenána topologie modelu. Stěny jsou zde zaneseny jako sekvence indexů vrcholů. Zápis této části má následující pravidla:

- Indexy vrcholů musí být z platného rozsahu - ten je dán počtem vrcholů v předchozí části souboru, kde je index dán pořadím vrcholu v souboru. Vrcholy jsou číslovány od nuly.
- Jednotlivé indexy jsou odděleny bílými znaky.
- Každá sekvence (t.j. každá stěna) musí být ukončena středníkem.
- Jakýkoliv text následující za středníkem na tomtéž řádku je ignorován (tuto vlastnost lze využít pro psaní komentářů).
- Indexy musí být v pořadí, kterým lze po hranách opsat obvod stěny. Pro polygon na obrázku **A.1** je tedy sekvence 14 15 0 1 8 7; platný zápis zobrazeného polygonu, zatímco 0 1 7 8 14 15; není.
- Jakýkoliv polygon, který nemá alespoň 3 vrcholy, je ignorován.

Příklad

Jako příklad poslouží model tetraedru přiložený v souboru `Samples/tetrahedron.txt`. Jeho obsah je následující:

```
01|  (0.0,0.0,0.0)
02|  (1.0,0.0,0.0)
03|  (0.0,1.0,0.0)
04|  (0.0,0.0,1.0)
05|  0 1 2;
```



Obrázek A.2: Model tetraedru uvedený v příkladu po otevření v aplikaci.

```
06|  0 2 3;
07|  0 3 1;
08|  1 2 3;
```

Řádky 1-4 obsahují definice vrcholů (s indexy 0-3), řádky 5-8 pak určují topologii. Výsledný model můžeme shlédnout na obrázku **A.2**.

A.4.2 Formát `*.model`

Přestože formát `*.model` funguje obdobně jako formát `*.txt`, není určen k ručnímu upravování modelů. Jeho výhodou je rychlejší ukládání a načítání, pro člověka je ovšem nečitelný. Pokud potřebujete pozměnit nějaký model uložený ve formátu `*.model`, uložte si ho nejprve ve formátu `*.txt` a pak do něj zaneste potřebné změny. Potom si ho můžete, pokud chcete, přeuložit opět do formátu `*.model`.

A.5 Statistiky

Hlavním cílem aplikace je poskytnout uživateli měření na modelech zpracovaných subdivizními algoritmy, měření jsou celkem 4 kategorie, v aplikaci jsou zobrazeny pod následujícími jmény:

General udávají počet stěn a vrcholů.

Surface měří veškeré údaje související s povrchem, tedy:

- (avg_s) – průměrná plocha polygonu.
- (min_s) – minimální naměřená plocha.
- (max_s) – maximální naměřená plocha.
- (var_s) – variance plochy polygonu.
- (d_s) – průměrná odchylka od rovného dílu plochy odpovídajícího polygonu z předchozího stádia.

Edge shrnují měření délky hran:

- (avg_e) – průměrná délka hrany.
- (min_e) – minimální naměřená délka.
- (max_e) – maximální naměřená délka.
- (var_e) – variance délky hrany.
- (d_e) – průměrný poměr délky hrany vůči délce odpovídající hrany z předchozího stádia.

Angle sjednocují měření velikosti úhlů:

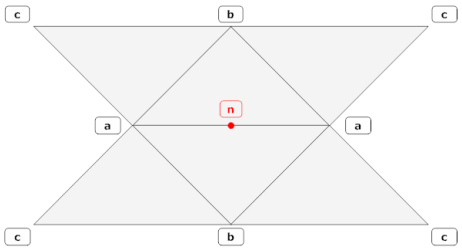
- (avg_a) – průměrná odchylka úhlu mezi sousedícími polygony od přímého úhlu.
- (min_a) – minimální takový naměřený úhel.
- (max_a) – maximální takový naměřený úhel.
- (var_a) – variance této odchylky.
- (d_a) – průměrný poměr této odchylky vůči odchylce naměřené v předchozím stádiu.

A.6 Dostupné algoritmy

A.6.1 Catmull-Clark

Algoritmus Catmull-Clark je univerzální aproximační algoritmus funkční pro všechny modely. Postupuje následujícím způsobem:

- Pro každou stěnu přidáme bod - její střed.
- Každou hranu rozdělíme bodem. S ním podnikneme následující:
  - Spojíme ho s původními konci hrany a se středy obou přilehlých stěn.



Obrázek A.3: Šablona původního Butterfly algoritmu

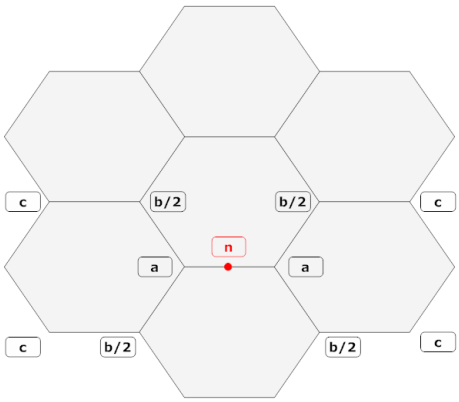
  - Jeho souřadnice nastavíme na aritmetický průměr všech (i nově) přilehlých bodů, tj.:
    * obou původních bodů dělené hrany
    * obou nově připojených středů přilehlých stěn

- Souřadnice každého původního bodu p o n sousedních stěnách (a i hranách) nastavíme na $\frac{F(p)+R(p)+(n-2)p}{n}$, kde R je průměr všech n bodů přidaných na hrany v tomto kroku algoritmu a F je průměr středů všech n stěn přilehlých k p.

A.6.2 Butterfly

Algoritmus Butterfly je interpolační algoritmus původně určený pro modely sestávající z trojúhelníků. Jeho postup je následující

- Nastavíme koeficienty a, b, c: Buď w koeficient tažnosti. Potom:
  - $a = \frac{1}{2}$
  - $b = \frac{1}{8} + 2w$
  - $c = -(\frac{1}{16} + w)$
- Pro každou hranu:
  - Rozdělíme hranu bodem.
  - Ten spojíme s původními konci hrany.
  - Jeho souřadnice nastavíme na vážený průměr koeficientovaný tak, jak ukazuje obrázek **A.3** (jeho souřadnice budou součtem souřadnic daných bodů s příslušnými koeficienty). Koeficienty jsou nastavené tak, aby vždy $2a + 2b - 4c = 1$, takže nové body nebudou dál, nežli původní.
- Doprostřed každé stěny vepíšeme trojúhelník sestávající ze tří nově vytvořených vrcholů



Obrázek A.4: Šablona upraveného Butterfly algoritmu

V aplikaci naleznete upravenou verzi Butterfly algoritmu, schopnou upravovat i jiné než pouze trojúhelníkové modely. Váhovou šablonu upraveného algoritmu (v našem případě aplikovanou na šestiúhelníky) ukazuje obrázek **A.4**. Můžete si všimnout, že šablona je pro trojúhelníkový model identická s předchozí - body s vahou c jsou vybírány jako předchozí a následující vrchol dvojice s vahami a a b v polygonu sousedícím právě přes tyto dva vrcholy.

Příloha B

Programátorská dokumentace

B.1 Použité technologie

Aplikace je psaná v C++ s použitím předdefinovaných tříd STL. Pro vizualizaci modelů byla použita technologie OpenGL, přesněji řečeno knihovny <gl.h>, <glu.h> a <glaux.h>. Grafický engine je sestaven z transformací GL a GLU.

Pro vytvoření uživatelského rozhraní a snímání vstupů byla použita technologie MFC, projekt byl vytvořen pomocí MFC šablony s podporou MDI a Document/View architektury ve vývojovém prostředí Microsoft Visual Studio 2005.

MFC a OpenGL jsou společně nakonfigurované tak, že z renderovacího kontextu OpenGL se obraz načítá vždy do kontextu zařízení aktivního okna aplikace. Do renderovacího kontextu OpenGL se zase vždy vykreslují data dokumentu příslušejícímu aktivnímu zobrazení (oknu).

B.2 Struktura kódu

B.2.1 Uspořádání souborů

Kód MFC aplikace jako takové je ponechán ve víceméně původní struktuře a je možné jej nalézt v kořenovém adresáři projektu. Kromě kódu MFC aplikace obsahuje kořenová složka projektu ještě 5 podsložek obsahující zdrojový kód:

_Doc obsahuje zdrojový kód související s dokumentem v rámci MFC paradigmatu Document/View, tzn. datové struktury modelu a veškeré třídy jím používané. Také zde najdeme uložené informace o vývoji modelu (stádia, přechody) a samotnou třídu odvozenou od `CDocument`.

_OpenGLView shromažďuje zdrojový kód obsluhující renderovací kontext OpenGL a jeho zobrazování do správného okna. Lze zde také nalézt handlery ke vstupům z myši.

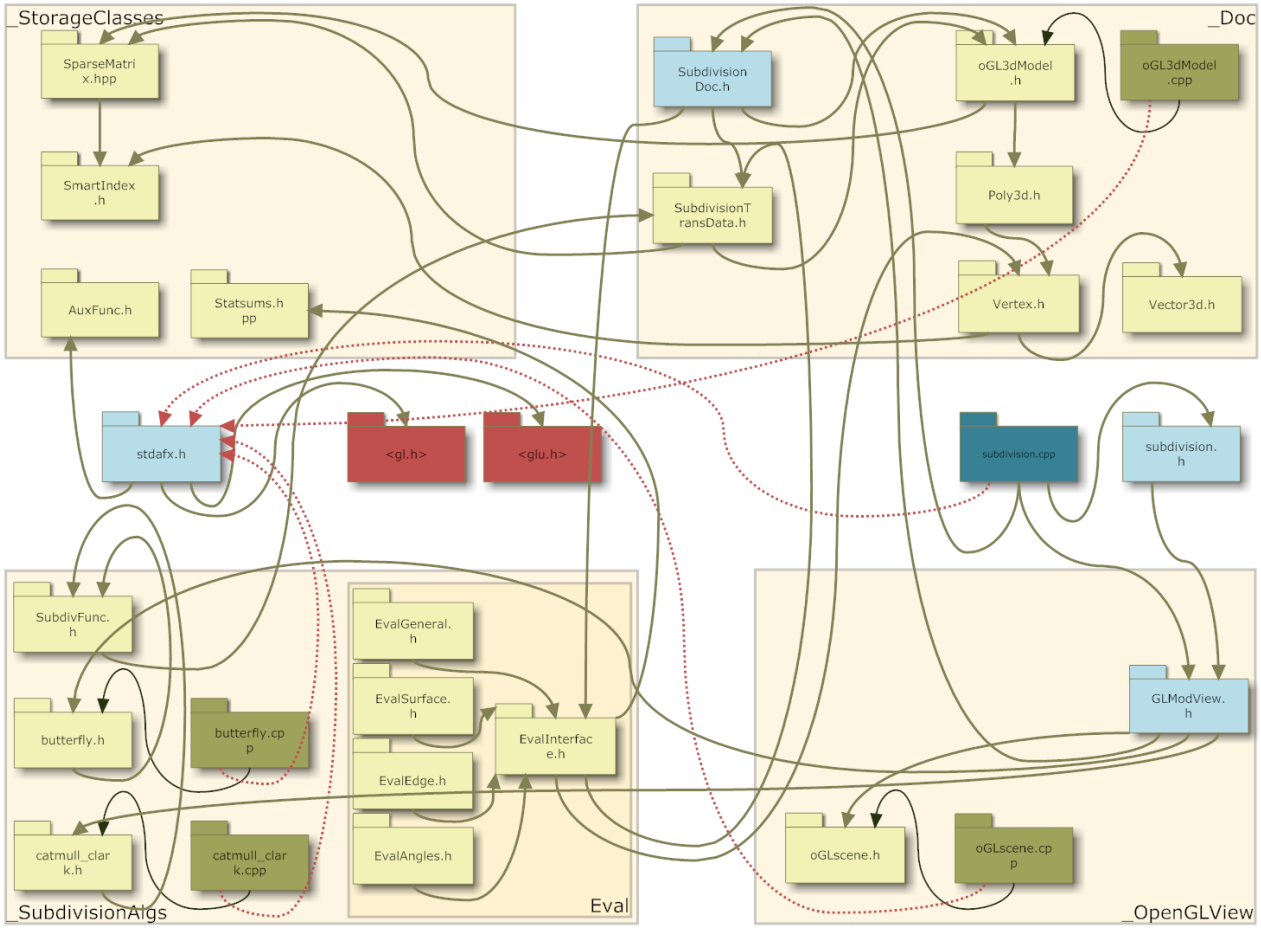
_**Resources** dle očekávání obsahuje resource soubory s vytvořenými šablonami oken.

_**StorageClasses** zahrnují samostatné datové struktury a pomocné funkce, jejichž použitelnost není omezena na tuto aplikaci.

_**SubdivisionAlgs** zahrnuje všechny subdivizní algoritmy. V podsložce **Eval** najdeme kód měřících tříd.

B.2.2 Struktura `#include` direktiv

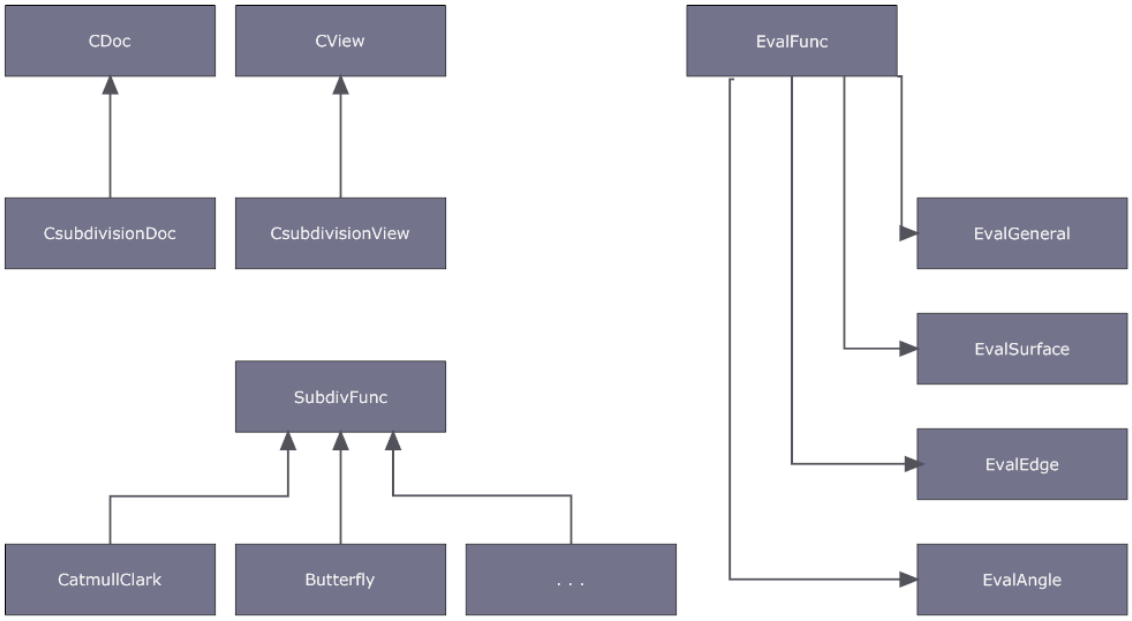
Struktura include příkazů je nejlépe vidět na schématu **B.1**.



Obrázek B.1: Include struktura

B.2.3 Struktura tříd

Dědičnost není v aplikaci příliš důležitá, za zmínku ale stojí následující čtyři případy znázorněné na obrázku **B.2**.



Obrázek B.2: Dědičnost tříd

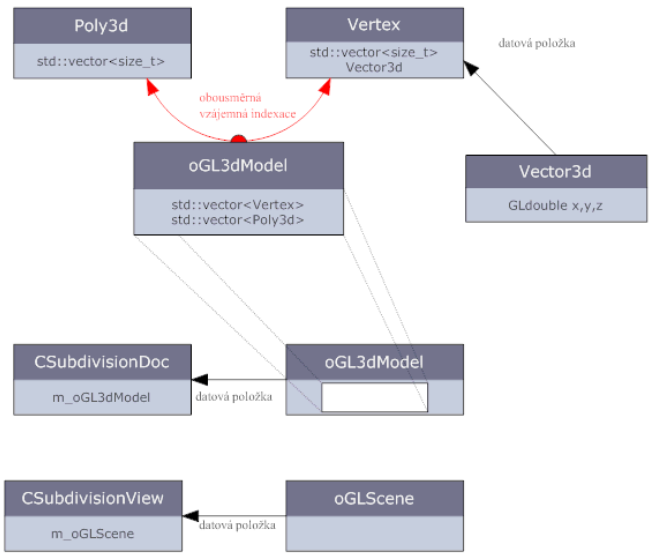
Dědičnost

`CDoc` → `CsubdivisionDoc` – obvyklá dědičnost, kterou se zajišťuje správná funkce třídy dokument pro danou aplikaci. Umožňuje přístup k overridu metody `CsubdivisionDoc::OnSaveDocument(LPCTSTR)`, kde se rozhoduje mezi textovým otevíráním souboru (pomocí `std::ifstream`) pro textový mód čtení a vestavěným otevíráním souboru pomocí rozhraní třídy CArchive pomocí `CsubdivisionDoc::Serialize(CArchive &)` pro binární soubory.

`CView` → `CGLModView` – taktéž obvyklá dědičnost umožňující třídě `CGLModView` především override rutin `OnPaint()`, `OnSize()` a handlerů na události myši, zároveň ale také přidání položek do menu a všechna vlastnosti okna.

`SubdivFunc` → `...` – dědičnost od společného rozhraní pro subdivizní algoritmy (funktory).

`EvalFunc` → `...` – opět společné rozhraní, tentokrát pro měřící funkce. Ty můžeme díky polymorfismu registrovat do instance třídy `EvalInterface` a měření i výpisy dále provádět pomocí metod této instance.



Obrázek B.3: Schéma složení tříd oGL3dModel, CGLModView a CsubdivisionDoc

Kompozice tříd

Zvolená reprezentace 3D modelu stojí na těchto třech třídách:

`Vector3d` je jednoduchá třída obsahující trojici souřadnic typu GLdouble. Třída podporuje běžné aritmetické operace a streamové operátory pro načítání a zápis do dvou typů vstupu/výstupu: `std::istream`/`std::ostream` a `CArchive`.

`Poly3d` je třída reprezentující polygon v paměti. Její hlavní komponentou je seznam indexů do pole vrcholů.

`Vertex` je obalová třída pro třídu `Vector3d`. K datovému členu typu `Vector3d` přidává ještě seznam indexů do pole polygonů.

Jak je možné vidět na schématu **B.3**, třída `oGL3dModel` obsahuje dva objekty std::vector - jeden pro body (`Vertex`) a jeden pro stěny (`Poly3d`). Každý objekt typu Vertex dále obsahuje std::vector indexů na polygony, které obsahují bod jím reprezentovaný.

Každý z těchto datových typů má uloženy indexy do objektů std::vector opačného typu v příslušné instanci třídy `oGL3dModel`. Stěny takto indexují vrcholy, které je tvoří, a vrcholy zase naopak indexují stěny, do kterých náleží. Tato obousměrná indexace umožňuje různým algoritmům prozkoumávat model, aniž by v nich bylo třeba implementovat složité vyhledávání.

Dále je z obrázku **B.3** patrné, že třída CsubdivisionDoc obsahuje položku třídy `oGL3dModel` - to umožňuje načítat a ukládat data do třídy oGL3dModel pomocí standardních procedur vlastních třídě `CDoc`.

Z obdobných důvodů obsahuje třída `CGLModView` položku třídy `oGLscene` - `oGLscene` tak získává přístup ke vstupům z myši a kontextům zařízení.

Mimo jiné `oGL3dModel` obsahuje ještě mapu sousednosti, která umožňuje pro určený polygon p a hranu $\{a, b\}$ najít polygon q sousedící s polygonem p přes hranu {a,b}. Informaci tohoto typu bychom sice mohli dostat i z výše popsané obousměrné indexace, pro účely měření by to ovšem bylo příliš pomalé.

Zatímco třída oGL3dModel obsahuje veškeré informace o modelu a je tedy vcelku objemná, třída oGLscene sestává pouze z několika datových položek uchovávajících informace relevantní pro OpenGL kameru a vykreslování:

`m_hDC` - ID (handle) kontextu zařízení svého okna

`m_hRC` - ID (handle) render kontextu OpenGL

`xangle` **a** `yangle` - úhly rotace modelu okolo os `x` a `y` ve stupních, slouží k zařízení otočné kamery pomocí funkce `glRotated()`

`distance` - vzdálenost kamery od počátku souřadné soustavy modelu, slouží k zoomování přes funkci `gluLookAt()`

B.3 Důležité sekce kódu

B.3.1 Sestavování modelu

V první fázi je třída `oGL3dModel` inicializována prázdná, plní se pomocí streamových operátorů deklarovaných jako `friend` třídy. Ty jsou definované v souboru `oGL3dModel.cpp`. Oba typy streamových operátorů (jeden pro řadu objektů `std::istream`, druhý pro `CArchive`) fungují rámcově stejně.

Operátor pro CArchive je jen lehce odlišný tím, že před samotným načítáním dat načte ze souboru velikost následujících dat, proto není nutné používat oddělovače a řetězce a načítání je rychlejší.

Cyklus přes vrcholy načítá souřadnice vrcholů ve formátu uvedeném v uživatelské příručce. Zastaví se na prvním řádku nezačínajícím znakem `(`.

Cyklus přes stěny dále načítá až na konec souboru. Třída `Poly3d` ale sama nevlastní streamový operátor, proto se načítání provádí přes pomocný `std::vector<size_t>`. Ten se po vyplnění předá přímo metodě `push_back` pole `std::vector<Poly3d>`, kam je díky konverznímu konstruktoru `Poly3d::Poly3d(const std::vector<size_t> &)` vložen objekt správného typu.

Při sestavování pomocného pole jsou právě načtené indexy vrcholů ještě používány k tomu, aby se ke každému z nich poznamenalo, že patří do konstruovaného polygonu. K tomu slouží metoda `Vertex::AddPolyIndex( index_t)`.

Po průchodu souborem je ještě sestavena matice sousednosti modelu.

Streamové operátory pro zápis fungují obdobně jako ty načítací, opět pomocí streamových operátorů třídy `Vector3d` pro vrcholy a přímo z dat (dostupných pomocí `Poly3d::GetVertexData()`) pro stěny.

B.3.2 Přepínání renderovacího kontextu

Aplikace je tvořená pomocí MDI MFC šablony a podporuje tedy více oken otevřených najednou. Proto je potřeba mezi ně distribuovat renderovací schopnosti OpenGL. V aplikaci není nutné řešit zvenku řízenou animaci, proto zde nejsou použity timery. Renderování je řízeno mainframem aplikace, dochází k němu vždy, když je na řadě zpráva typu `WM_PAINT`.

O výstup generovaný OpenGL si vždy zažádá příslušné okno aplikace. V handleru zprávy `WM_PAINT` - metodě `CGLModView::OnPaint()` se zažádá o renderovací kontext OpenGL pomocí `wglMakeCurrent()` s argumenty uloženými v datovém členu typu oGLscene - `m_hRC`, inicializovaným v metodě `CGLModView::SetupPixelFormat`, volanou v `CGLModView::OnCreate`, a `m_hDC`, inicializovaným přímo v `CGLModView::OnCreate`. Poté se vyčistí aktuálně disponibilní buffer, aplikují se maticové transformace a předají se souřadnice vrcholů. Poté se buffery prohodí a obraz se vykreslí do kontextu zařízení daného okna.

B.3.3 Subdivizní algoritmy

Jak bylo již řečeno výše, třída `SubdivFunc` je obecným vzorkem pro funktor, který provádí subdivizní algoritmus na zvolené datové struktuře.

Samotné vykonání algoritmu neprobíhá po vytvoření jeho funktoru, ale po spuštění jeho členské metody `Execute(const oGL3dModel &, oGL3dModelModifier &)` - v její definici leží vždy celé tělo algoritmu.

Oba níže popsané algoritmy také používají seznam středů hran - tato struktura umožňuje určit, jestli hrana už byla dělená, nebo ne. Pokud ano, použije se již vytvořený bod. Toto opatření činí výpočet algoritmů trochu složitější, ale výsledné modely jsou naopak jednodušší a kompaktnější. V aplikaci je k tomuto seznamu přistupováno jako k řídké matici, implementace je definována v souboru `_StorageClasses/Sparsematrix.hpp` jako třída s volným datovým typem a využívá kontejner `std::map` s indexem typu `std::pair< size_t, size_t>`. V případě subdivizních algoritmů je datovým typem index, tedy typ `index_t` (typ size_t s příznakem platnosti typu boolean).

Za zmínku také stojí funkce `Poly3d::OpposedVertexIndex`. Její úloha je pro daný index polygonu `p` a dva indexy vrcholů `x` a `y` nalézt vrchol, který leží v polygonu sousedícím s `p` přes hranu `x-y` naproti hraně `x-y`. Tuto funkci využívá algoritmus Butterfly.

Po provedení libovolného subdivizního algoritmu se v rámce zpracovávaného modelu volá funkce `CsubdivisionDoc::PostSubdivisionStep( )`, která obstará výpočet a případné zobrazení výstupu měřících funktorů (potomků `EvalFunc`) registrovaných v třídě `EvalInterface` (datové položce třídy CsubdivisionDoc).