VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROGRESSIVE MESHES

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JAN ZELENÝ
AUTHOR

VEDOUCÍ PRÁCE ING. STANISLAV SUMEC, PH.D.
SUPERVISOR

BRNO 2007

Abstrakt

V počítačové grafice se stále častěji setkáváme s velmi detailními modely, které kladou velké nároky na vykreslovací i paměťové zařízení. Jedním z možných zlepšení a zmírnění těchto vysokých potřeb je tzv. Progressive mesh. Jedná se o způsob reprezentace 3D objektu, díky kterému je možné měnit dynamicky úroveň detailů, optimalizovat povrch, vytvářet plynulé přechody mezi úrovněmi prokreslení a efektivně tyto objekty ukládat a přenášet.

Klíčová slova

Progresivní mesh, úroveň detailů, progresivní přenos, komprese meshe, geomorf, výběrové prokreslení,

Abstract

There are many highly detailed 3D objects in computer graphics today, which expect high-tech hardware equipment for rendering, storing or transmitting. One of many possible solutions of these problems is progressive mesh. It is a kind of representation of 3D objects, which can be real-time modified in sense of level of detail (LOD). Progressive meshes can also modify effectively their surface, and create geomorphs between two different levels of detail. Progressive transmitting and compressed storing are also possible with this feature.

Keywords

Progressive mesh, level of details, progressive transmission, mesh compression, selective refinement

Citace

Jan Zelený: Progressive mesh, bakalářská práce, Brno, FIT VUT v Brně, 2007

Progressive meshes

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ph.D. Stanislava Sumce

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Chtěl bych na tomto místě poděkovat mému vedoucímu za věcné rady a konstruktivní připomínky.

© Jan Zelený, 2007

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah ... 1
1 Úvod ... 3
2 Oblasti využití ... 4
2.1 Zjednodušení objektu ... 4
2.2 Úroveň detailů ... 4
2.3 Progresivní přenos ... 4
2.4 Komprese meshe ... 5
2.5 Geomorf ... 5
2.6 Výběrové prokreslení ... 5
3 Meshe v počítačové grafice ... 6
4 Reprezentace progresivního meshe ... 7
4.1 Celkový pohled ... 7
4.2 Zjednodušení objektu ... 9
4.3 Úroveň detailů ... 9
4.4 Progresivní přenos ... 10
4.5 Komprese a paměťová reprezentace meshe ... 10
4.6 Geomorfování ... 12
4.7 Výběrové prokreslení (selective refinement) ... 14
5 Wedges ... 15
6 Konstruování progressive meshe ... 16
6.1 Optimalizace povrchu meshe ... 17
6.1.1 Vnější smyčka ... 18
6.1.2 Vnitřní smyčka ... 18
6.2 Shrnutí optimalizační funkce ... 19
6.2.1 Zachování povrchové geometrie ... 20
6.2.2 Zachování skalárních atributů ... 21
6.2.3 Zachování křivek nespojitosti ... 22
7 Budoucí vývoj ... 23
8 Implementace ... 24
8.1 Praktické problémy při řešení ... 24
8.2 Implementovaný algoritmus ... 25
8.3 Detaily konkrétní implementace ... 26
8.3.1 Třídní dekompozice ... 26

8.3.2 Struktury a paměťová reprezentace..... 27
8.3.3 Ovládání..... 29
8.3.4 Fáze při vývoji programu..... 29
9 Závěr..... 30
Literatura..... 31
Seznam příloh..... 32
Obrazové přílohy..... 33

1 Úvod

K uspokojení zvyšujících se nároků na realismus v počítačové grafice jsou nutné velmi detailní geometrické modely. V dnešní době spolu s rozvojem modelovacích a skenovacích nástrojů vzrůstá velikost a množství detailů. 3D modely tvořené různými operacemi (jako je konstruování spline křivkami, tvarování, konstrukce plné geometrie nebo libovolné deformace) narůstají do ohromných seznamů geometrických primitiv. Pro efektivní zobrazování musejí být tyto modely obvykle rozloženy do polygonálních aproximací - meshů (tzn, že dílčí geometrické primitivy, jako jsou obecně n-úhelníky, polyhedrony, jsou rozděleny na jednotlivé trojúhelníky - polygony).

Detailní modely jsou také získávány skenováním fyzických objektů za použití vzdálenostních skenovacích systémů, které často i sochu, měřící několik metrů, naskenují v detailech 2 polygony na 1 mm^2 . Je tedy zřejmé, že výsledný model může pak mít i několik desítek miliónů polygonů.

V každém případě je výsledný komplexní objekt náročný na ukládání, přenos a vykreslování, čili motivací k vytvoření efektivní metody je mnoho praktických problémů, jako je efektivní způsob ukládání, optimalizace povrchové reprezentace pro minimální nároky při maximálním zachování celkového vzhledu a dojmu, vhodný způsob ukládání do paměti včetně efektivní metody komprese a půjdeme-li dále, také dynamická změna těchto detailů pro co nejplynulejší a zároveň nejdetailnější zobrazení 3D scén, například u počítačových her.

Jedním z možných řešení takových problémů je právě tzv Progressive meshes.

Tato práce si klade za cíl právě progresivní mesh (PM) popsat a s ním i mechanismus pro ukládání a zobrazování libovolných trojúhelníkových meshů pomocí efektivní bezeztrátové metody. Budou zde popsány: Plynulé přeměny jednotlivých stupňů detailnosti pro rychlejší a efektivnější vykreslování, progresivní přenos (například plynulé zobrazování při přicházejících datech po internetu), komprese a selektivní prokreslení (zobrazování detailů jen v těch částech modelu, které jsou kamerou skutečně vidět). Uvedeme si několik algoritmů na optimální generování takových dat a také s důrazem na výkon.

Nakonec je ještě nutné dodat, že velká část principů teoretickým postupů a jejich matematického zápisu čerpá z dokumentu *Progressive Meshes* Hoppe H [3].

2 Oblasti využití

2.1 Zjednodušení objektu

Objekty vytvořené modelovacími a skenovacími zařízeními jsou zřídkakdy optimalizovány pro efektivní vykreslování a mohou být často nahrazeny k nerozeznání podobnými aproximacemi s mnohem méně polygony. V nedávné době ještě vyžadoval takový proces jistý zákrok uživatele. Dnes již ale mnoho nástrojů pro zjednodušování objektů automatizují tuto nelehkou práci a poskytují tím i jednoduchý způsob portování pro různě výkonné platformy. Více se tím budeme zabývat v kapitole 4.2 - Zjednodušení objektu.

2.2 Úroveň detailů

Pro navýšení vykreslovacího výkonu bylo dříve běžnou praktikou definovat několik verzí modelu s různým stupněm detailů. Detailní model byl použit když je objekt blízko pozorovatele a hrubší aproximace byla naopak zobrazena při vykreslování ve velké vzdálenosti. Bohužel se k problému s o to větší velikostí v paměti ještě přidal jeden o mnoho viditelnější. Právě překlápění jednotlivých LOD (anglicky level of detail - úroveň detailů) modelů vedlo k viditelným bliknutím. Z těchto důvodů vznikla snaha o zkonstruování dynamicky generovaných úrovní detailů, které by tím pádem nebylo nutné ukládat každý zvlášť do paměti a ono „bliknutí“ by se velkou podobností dvou sousedních úrovní detailů minimalizovalo. Pro více informací se podívejte do kapitoly 4.3 - Úroveň detailů.

2.3 Progresivní přenos

Když je objekt přenášen skrze komunikační kanál, chtěli bychom zobrazovat průběžnou aproximaci modelu jak jsou data inkrementálně přijímána. Například si představme situaci, kdy sítí internet přichází velmi detailní 3D objekt a my bychom si v průběhu stahování chtěli prohlížet jeho geometrii. Nejprve by se zobrazil velmi hrubý model a postupem času a s přibývajícími informacemi by se do modelu dodávali detaily až bychom se nakonec dostali k originální úrovni detailů stahovaného modelu. Jedním z možných přístupů je přenášet hotové LOD aproximace, ale to samozřejmě vyžaduje nesrovnatelně větší přenosový čas. Další a samozřejmě mnohem efektivnější metodou je využití Progressive mesh a jeho přirozené vlastnosti, která toto umožní. Později si to vysvětlíme podrobně v kapitole 4.4 - Progresivní přenos.

2.4 Komprese meshe

Problém minimalizace prostoru pro uložení modelu se dá řešit dvěma způsoby. Prvním je zjednodušení modelu a tím snížení počtu ukládaných polygonů. Dalším je komprese modelu: minimalizování prostoru potřebného pro každý jednotlivý mesh. Díky způsobu ukládání Progressive mesh je nejen potřebné místo v paměti počítače i přes všechny úrovně detailů srovnatelné s uložením obyčejného meshe, ale i případná komprese je o něco efektivnější. Blíže se o kompresi meshe dozvíte v kapitole 4.5 - Komprese a paměťová reprezentace meshe.

2.5 Geomorf

Jak už jsme si řekli v části o úrovni detailů výše, vyvstává často problém s ostrým přechodem mezi jednotlivými úrovněmi detailů meshe. Zvláště při malém množství polygonů, kdy každý jediný úbytek či nárůst vede k citelné změně v celkovém vzhledu meshe, je často patrné „bliknutí“ při přechodu z jedné aproximace modelu do sousední. Tento problém řeší právě tzv geomorf. Jedná se o plynulý přechod mezi dvěma jakýmikoliv aproximacemi jednoho meshe, který zajistí jejich vizuální návaznost. V kapitole 4.6 - Geomorfování si tuto metodu blíže popíšeme.

2.6 Výběrové prokreslení

Každý mesh v LOD reprezentaci (Progressive mesh) uchovává model v jednotné (na zorném úhlu nezávislé) úrovni detailů. Někdy je ale potřeba adaptovat úroveň zjednodušení v jistých regionech. Například, když se kamera pohybuje nad terénem, je třeba zobrazovat terén v plných detailech pouze blízko pozorovatele a to pouze uvnitř jeho zorného pole. V ostatních částech je možné úroveň detailů buď plynule snižovat (směrem od kamery dopředu), nebo rovnou nastavit detaily na minimum v místech, které nejsou vidět vůbec. Stále ale musíme mít na paměti, že musí být terén neustále validní a nesmí ani při takové změně obsahovat díry a chyby (tzv. T cracks).

Řešením těchto problémů je konstruování, uchovávání a vykreslování 3D objektů v Progressive mesh reprezentaci.

3 Meshe v počítačové grafice

Modely v počítačové grafice jsou často reprezentovány pomocí trojúhelníkových meshů. Geometricky je trojúhelníkový mesh složení trojúhelníkových stěn spojených dohromady jejich hranami. Tím vzniká tzv. povrchová reprezentace (objekt není definován skutečným objemem materiálu, ale pouze jakousi dutou povrchovou schránkou). Geometrie meshe může být označena dvojicí (K, V), kde K je v literatuře [3] označováno jako "komplexnost jednoduchosti", určující spojení dílčích jednotek meshe (sousedství vrcholů, hran a stěn) a $V = \{v_1, v_2, \ldots, v_m\}$, což je seznam pozic vrcholů definujících tvar samotného meshe v $\mathbb{R}^3$. Jednodušeji řečeno: Zatímco V udává pozice samotných bodů v 3D prostoru, K definuje vztahy mezi nimi a tím je spojuje do hran a do polygonů.

Naopak složitěji: konstruujeme parametrickou doménu $|K| \subset \mathbb{R}^m$ identifikováním každého vrcholu v K s kanonickým základním vektorem v $\mathbb{R}^m$ a definujeme mesh jako obraz $\phi v(|K|)$ kde $\phi v : \mathbb{R}^m \to \mathbb{R}^3$ je lineární mapování.

Často jsou další atributy (jiné než geometrie vzhledu povrchu) také asociovány s meshem. Tyto parametry mohou být rozděleny do dvou typů: diskrétní atributy a skalární.

Diskrétní atributy se často týkají jednotlivých stěn. Velmi běžný diskrétní atribut je například identifikátor materiálu, určující funkci použitou při vykreslování stěny meshe. Tím se dá ovlivnit například použitá textura, barva materiálu, lesklost, průhlednost a další atributy určené stejnorodě pro celý polygon.

Skalárních atributů je s meshem asociováno většinou více. Jsou to například atributy jak difuzní barva (r, g, b), normálový vektor (n_x, n_y, n_z) a koordináty textury (u, v). Obecněji se dá říci, že tyto atributy specifikují lokální parametry vykreslovacích funkcí definovaných pro stěny meshe. V jednodušších případech jsou tyto skalární atributy asociovány přímo s vrcholy meshe.

Nicméně k reprezentaci nespojitostí ve skalárních polích a protože sousední stěny mohou mít rozdílné vykreslovací funkce, je běžné asociovat skalární atributy ne s vrcholy, ale s rohy meshe (wedges, viz kapitola 5 - Wedges). Roh je definován jako dvojice (vrchol, stěna). Skalární atributy v rohu (v, f) specifikují vykreslovací parametry pro stěnu f ve vrcholu v. Například podle záhybu (křivka na povrchu, přes kterou není normálové pole spojité, viz kapitola 6.2.3 - Zachvání křivek nespojitosti) má každý vrchol dva odlišné normálové vektory na každé straně záhybu.

Označíme tedy mesh jako $M=(K,V,D,S)$ kde K je vztah mezi body, V specifikuje jeho geometrii, D je sada diskrétních atributů d_f asociovaných se stěnami $f=j,k,l\subset K$ a S je sada skalárních atributů $s_{(v,f)}$ asociovaných s rohy (v,f) z K.

Díky atributům D a S tedy vyniknou nespojitosti ve vizuálním vzhledu meshe. O hraně $\{v_j, v_k\}$ meshe říkáme, že je ostrá pokud platí alespoň jedna z podmínek:

a) je okrajovou (tvoří polygon jen z jedné strany)

b) její dvě přilehlé stěny f_l a f_r mají rozdílné diskrétní atributy ($d_{f_l}\neq d_{f_r}$)

c) její přilehlé rohy mají rozdílné skalární atributy ($s_{(v_j,f_l)}\neq s_{(vj,fr)}$ nebo $s_{(v_k,f_l)}\neq s_{(vk,fr)}$).

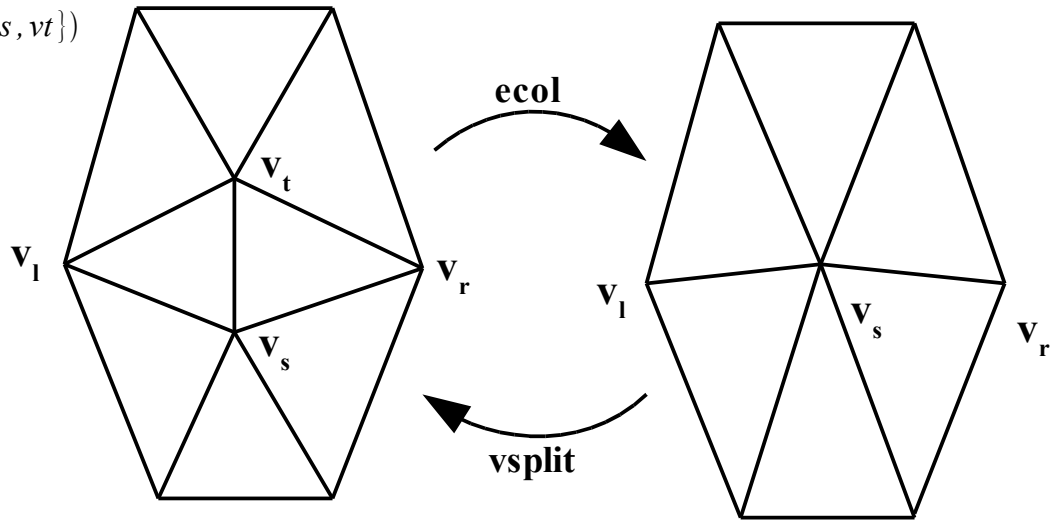
Dohromady udává sada těchto ostrých hran množinu křivek nespojitosti napříč meshem.

4 Reprezentace progresivního meshe

4.1 Celkový pohled

Hoppe H. [3] popisuje metodu, optimalizaci meshe, která může být použita k aprocimování počátečního meshe $\widehat{M}$ k mnohem jednoduššímu meshi M^0. Jejich optimalizační algoritmus, popsán v kapitole 6.1 - Optimalizace povrchu meshe, prozkoumává prostor možných meshů aplikováním sady 3 transformací: kolaps hrany, rozdělení hrany a prohození hran.

Později bylo zjištěno, že ve skutečnosti jedna z těchto transformací, kolaps hrany, naprosto dostačuje pro efektivní zjednodušení meshe. Jak je demonstrováno na obr 1, kolaps hrany $ecol(\{vs,vt\})$



Obr 1. Kolaps hrany a invertní operace rozdělení vrcholu

sjednocuje 2 sousední vrcholy v_s a v_t do jednoho vrcholu v_s. Vrchol v_t a dvě přilehlé stěny $\{v_s, v_t, v_l\}$ a $\{v_t, v_s, v_r\}$ při procesu zmizí. Pozice v_s je specifikována novým sjednoceným vrcholem.

Tudíž původní mesh $\widehat{M} = M^n$ může být zjednodušen do hrubého meshe M^0 aplikováním sekvence n úspěšných transformací kolaps hrany.

$$(\widehat{M} = M^n) \rightarrow^{ecol_{n-1}} \ldots \rightarrow^{ecol_1} M^1 \rightarrow^{ecol_0} M^0$$

Jednotlivé části posloupnosti kolapsů hran musí být voleny velmi pečlivě, neboť určují kvalitu aproximovaných meshů M^i, $i < n$. Schéma pro optimální volbu hrany ke kolapsu je prezentováno v kapitole 6 - Konstrouvání progressive meshe.

Nechť m_0 je počet vrcholů v M^0 a nechť označíme vrcholy meshe M^i jako $V^i = \{v_{1,} \ldots, v_{m_0+i}\}$, potom hrana $\{v_{s_i}, v_{m_0+i+1}\}$ je zkolabována pomocí $ecol_i$ jak je na obrázku 2a. Protože mohou mít vrcholy rozdílné pozice v rozdílných meshech, označíme pozici v_j v M^i jako v_j^i.

Klíčová věc, která dělá tuto operaci velice použitelnou a vhodnou pro rychlou počítačovou grafiku je ta, že transformace kolaps hrany je vratná. Inverzní transformace se pak nazývá vertex split (dělení vrcholu), jak je naznačeno v obr. 1. Dělení vrcholu $vsplit(s, l, r, t, A)$ přidá blízko vrcholu v_s nový vrchol v_t a dvě stěny $\{v_s, v_t, v_l\}$ a $\{v_t, v_s, v_r\}$. Pokud je hrana $\{v_s, v_t\}$ okrajová, necháme $v_r = 0$ a přidána je pouze jedna stěna. Transformace také aktualizuje atributy meshe v sousedství transformace. Tato informace o atributu, označována jako A, včetně pozice v_s a v_t od obou zúčastněných vrcholů, diskrétního atributu $d\{v_s, v_t, v_l\}$ a $d\{v_t, v_s, v_r\}$ od obou nových stěn a skalárních atributů zúčastněných rohů ($s_{(v_s,.)}$, $s_{(v_t,.)}$, $s_{(v_l,\{v_s,v_t,v_r\})}$ a $s_{(v_r,\{v_t,v_s,v_r\})}$).

Jelikož je transformace kolaps hrany vratná, můžeme proto reprezentovat libovolný trojúhelníkový mesh $\widehat{M}$ jako jednoduchý mesh M_0 spolu se sekvencí n vsplit záznamů.

$$M^0 \rightarrow^{vsplit_0} M^1 \rightarrow^{vsplit_1} \ldots \rightarrow^{vsplit_{n-1}} (M^n = \widehat{M})$$

kde je každý takový záznam parametrizován jako $vsplit_i(s_i, l_i, r_i, A_i)$. Potom $(M^{0,}\{vsplit_{0,} \ldots, vsplit_{n-1}\})$ nazýváme progresivní mesh (PM) reprezentací původního meshe M.

Tímto jsme si nastínili velmi častý způsob reprezentace progressive meshe a nyní si přiblížíme konkrétní případy jeho použití, tentokrát s jejich řešením a algoritmy.

4.2 Zjednodušení objektu

Objekty se dnes často získávají v příliš vysokém rozlišení a k jejich uchovávání a vykreslování poté potřebujeme neúměrně velký prostor. Přitom mnoho detailů v takovém objektu není potřeba, co více, nezvyšují ani celkovou kvalitu meshe. Jedná se například o větší rovnou plochu (všechny polygony leží v jedné rovině) z povrchu meshe tvořenou mnoha dílčími polygony, která by tím pádem mohla být nahrazena několika polygony většími, přičemž by se povrch celého objektu naprosto nezměnil. Často je takové zjednodušení žádoucí i s drobnou ztrátou detailnosti meshe, ale s velkým ziskem na efektivnosti uložení a zobrazení.

Dříve bylo nutné toto zjednodušení provést za velkého přispění člověka, dnes díky Progressive meshi je možné tento proces automatizovat a uživatel se pouze zvolí, na jakou míru detailů si přeje mesh upravit. Tím je mimo jiné umožněno i vytvoření různé úrovně detailů pro různě výkonné platformy. Technika hledání takové jednodušší aproximace meshe je velmi podobná konstruování progressive meshe jen s tím rozdílem, že si nemusíme ukládat informace o operacích dělení vrcholu $vsplit(s,l,r,t,A)$, které by nám pomohli při obnovení původních detailů. Generování Progressive meshe je také ukončeno jakmile je dosaženo meshe M^i s potřebným počtem polygonů/vrcholů.

4.3 Úroveň detailů

Chceme-li zvýšit rychlost vykreslování meshe v nějaké 3D aplikaci, můžeme buď předem vytvořit několik úrovní detailů (ručně, nebo zjednodušením objektu -viz výše), nebo vygenerování progressive meshe až do minimální úrovně detailů M^0 spolu se všemi vsplit informacemi, které nám pomohou k obnovení původního meshe $M^n = \widehat{M}$. V takovém případě je třeba vytvořit v aplikaci dostatečně výkonný nástroj jak pro generování celého pole vsplit informací, tak i pro jejich efektivní a rychlou aplikaci na aktuální mesh a tím docílení skutečně výhodného využití v reálném čase. Právě sledování tohoto cíle je hlavním přínosem reprezentace meshe jako progressive mesh a při kvalitním a rychlém algoritmu může být opravdovým ziskem pro aplikace zobrazující mnoho 3D objektů v reálném čase. Zde se proto použije právě taková reprezentace Progressive meshe, jakou jsme si nastínili v celkovém pohledu v úvodu této kapitoly.

4.4 Progresivní přenos

Jak jsme si řekli dříve, progresivní přenos je stručně řečeno takový přenos modelu (například po síti internet), při jehož průběhu můžeme neustále sledovat aproximaci cílového modelu, otáčet ji, prohlížet, a to vše již od rané fáze přenosu s tím, že jen postupně přibývají do modelu detaily a naše zobrazovaná aproximace se blíží přenášenému meshi $\widehat{M}$.

Reprezentace meshe jako progressive mesh je příhodně již ze své podstaty právě pro progresivní přenos ideální. Nejdříve je přenesen kompaktní mesh M^0 (za využití klasického souřadnicového systému), který je následován tokem $vsplit_i$ záznamů. Proces, který obstarává příjem takových dat pak inkrementálně rekonstruuje $\widehat{M}$ tak, jak záznamy přicházejí. Tím animuje zobrazovaný měnící se mesh. Může k tomu využít například i geomorf, aby se zabránilo viditelným skokům mezi jednotlivými úrovněmi detailů. Originální $\widehat{M}$ mesh je poté obnoven ve chvíli, kdy příjemce obdrží právě n záznamů (neboť Progressive mesh reprezentace je bezeztrátová).

Výpočetní výkon příjemce může být vyvažován mezi rekonstruováním $\widehat{M}$ meshe a interaktivním zobrazováním pro uživatele. S pomalým komunikačním spojením je jednoduchou strategií zobrazovat aktuální mesh kdykoliv zjistíme, že je vstupní buffer prázdný, zatímco máme-li tok dat o poznání rychlejší, můžeme zobrazovat takové aproximace meshe, jejichž míra detailů bude vzrůstat exponenciálně. K tomuto tématu se dá přirovnat například přenos a zobrazování progresivního jpeg formátu.

4.5 Komprese a paměťová reprezentace meshe

Uvědomíme-li si, že v jednom progressive meshi jsou uloženy všechny jednotlivé aproximace jednoho meshe, ihned nás napadne, že takový způsob uložení musí být velmi paměťově náročný. Přesto je ale progressive mesh reprezentace velice paměťově úsporná a to ze dvou důvodů: Prvním z jich je ten, že pozice vsplit operace může být zakódována velice úsporně. Namísto ukládání všech tří indexů vertexů (s_i, l_i, r_i) z operace $vsplit_i$ stačí uložit pouze v průměru 5 bitů k označení zbývajících dvou vertexů v sousedství v_{s_i} . Druhý důvod, proč je uložení progressive meshe paměťově nenáročné je ten, že ve skutečnosti má vsplit operace lokální účinek. Mnohé hned napadne, že pokud známe sousední polygony, můžeme velmi často u nových polygonů vzniklých vsplit operací očekávat podobné diskrétní a skalární atributy. Pro názornost: když je $v_{s_i}^i$ rozdělen do $v_{s_i}^{i+1}$ a $v_{m_0+i+1}^{i+1}$, můžeme předpokládat pozice $v_{s_i}^{i+1}$ a $v_{m_0+i+1}^{i+1}$ z již známé pozice $v_{s_i}^i$ a použít delta-

kódování k redukci velikosti paměti. Skalární atributy v rozích v M^{i+1} mohou být opět podobné těm odvozeným z M^i. Nakonec i identifikátory materiálu $d_{\{v_s, v_t, v_l\}}$ a $d_{\{v_t, v_s, v_r\}}$ z nových polygonů v meshi M^{i+1} mohou být často předvídány z těch, obsažených v sousedních polygonech v M^i užitím několika řídících bitů.

Ve výsledku má velikost s citem designovaného progressive meshe srovnatelné výsledky jaké obdržíme z metod pro kompresi klasického meshe.

Běžná reprezentace spojitosti K je seznam tří indexů vertexů pro každý polygon. Pokud počet vertexů je řekněme n, pak je počet polygonů zhruba $2*n$ a jejich seznam zabírá zhruba $6[\log_2(n)]n$ bitů. Použitím bufferu s 2 vertexy, zobecněná vertex strip reprezentace redukuje tento počet na zhruba $([\log_2(n)]+2\mathrm{k})n$ bitů, přičemž vertexy jsou v průměru odkazovány jen jednou a k odpovídá 2 bitům zachycujícím nahrazení. V progressive mesh reprezentaci potřebuje každý $vsplit_i$ specifikovat s_i a dva sousedy na celkovém prostoru $([\log_2(n)]+5)n$ bitů. Tyto hodnoty již se zobecněnými trojúhelníkovými stripy v normálních meshech srovnatelné jsou.

Tradiční reprezentace geometrie V meshe M vyžaduje ukládání 3n koordinátů (x, y, z), neboli 96n bitů pomocí čísel s jednoduchou přesností a plovoucí desetinnou čárkou podle IEEE [6]. Jak už uvedl Deering [7], mohou být tyto koordináty uloženy jen na 16 bitech čísel s pevnou desetinnou čárkou bez znatelné ztráty vizuální kvality, což redukuje velikost takové potřeby paměti na 48n bitů. Deering dále zkomprimoval uložená data pomocí delta-kódování (ukládá se pouze změna, nikoliv celá hodnota), čímž dostal spoustu podobných dat, která se dají výborně následně zkomprimovat pomocí Huffmanovy komprese. Na každý 16-ti bitový úsek tak uložil zhruba původních 35.8 bitů, což již zahrnuje oba typy kódování. Použitím podobného přístupu s progressive mesh reprezentací se dá zakódovat geometrie V na 31n až 50n bitů. K získání takových výsledků ale byla potřeba ještě jedna optimalizace. Její oblast působení je v rozhodování, kam s vrcholem v_n, který vznikl sloučením původních vrcholů při zhroucení hrany $\{v_s, v_t\}$. Výběr se omezil na pouze tři případy: $v_n : \{v_s, v_t, \frac{v_s+v_t}{2}\}$. V závislosti na vybrané cílové pozici zakódujeme pomocí delta-kódu $\{v_s - v_n, v_t - v_n\}$ nebo $\{\frac{v_s+v_t}{2} - v_n, \frac{v_t - v_s}{2}\}$ a použijeme oddělené tabulky Huffmanova kódu pro každou ze čtyř hodnot.

Pro další zlepšení komprese můžeme změnit algoritmus pro konstruování a nechat $v_n \in \{v_s, v_t, \frac{v_s+v_t}{2}\}$. To může degradovat kvalitu meshe, ale umožní nám to zakódovat progressive mesh reprezentaci na zhruba 30n až 37n bitů. Aritmetické kódování délek delta informace již o mnoho výsledek nezlepší, což jen odpovídá faktu, že Huffmanův strom bývá velmi dobře vybalancovaný.

4.6 Geomorfování

Krásná vlastnost transformace vsplit (a jejího opaku - kolapsu hrany) je možnost vytvoření plynulé vizuální změny (geomorfu) a to mezi dvěma meshi M^i a M^{i+1} které dělí operace $M^i \rightarrow^{vsplit_i} M^{i+1}$. Pro ilustraci na chvíli předpokládejme, že mesh neobsahuje žádnou jinou vlastnost než pozice vrcholů. S tímto předpokladem je vsplit záznam uložen jako $vsplit_i(s_i, l_i, r_i, A_i = (v_{s_i}^{i+1}, v_{m_0+i+1}^{i+1}))$. Zkonstruujme geomorf $M^G(\alpha)$ s parametrem smíšení $0 \leq \alpha \leq 1$ tak, že $M^G(0)$ vypadá jako M^i a $M^G(1)$ vypadá jako M^{i+1} (Ve skutečnosti $M^G(1) = M^{i+1}$)

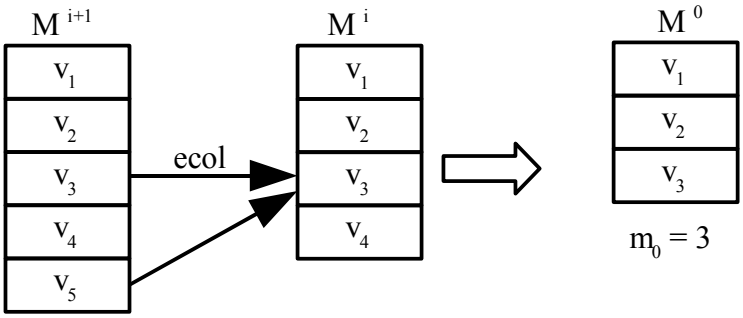
Definováním meshe $M^G(alfa) = (K^{i+1}, V^G(alfa))$ jehož spojitost je jako u M^{i+1} a jehož pozice vrcholů jsou lineární interpolací z $v_{s_i} \in M^i$ do rozdělených vrcholů $v_{s_i}, v_{m_0+i+1} \in M^{i+1}$:

$$v_j^G(\alpha) = \begin{cases} (\alpha) v_j^{i+1} + (1-\alpha) v_{s_i}^i & , j \in \{s_i, m_0+i+1\} \\ v_j^{i+1} = v_j^i & , j \notin \{s_i, m_0+i+1\} \end{cases}$$

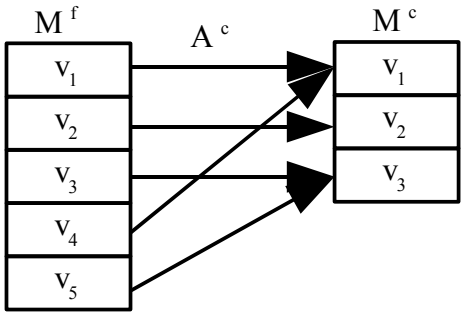
Použitím takového geomorfu mohou aplikace plynule přecházet z meshe M^i do meshů M^{i+1} nebo M^{i-1} bez nějakého viditelného skoku.

Krom toho, jakmile lze transformovat plynule samostatnou $ecol$ operaci, lze to samé udělat i se složením takových operací. Geomorf může být proto zkonstruován mezi jakýmikoliv dvěma úrovněmi detailů. Samozřejmě tedy i mezi nejjednodušším meshem M^0 a originálním meshem M^n .

Je to možné díky tomu, že mezi jejich vertexy existuje jisté spojení: Každý vertex meshe M^n má svého jedinečného předka v meshi M^0 . Toto surjektivní zobrazení A je možné díky tomu, že z jednoho vertexu vznikají vsplit operací dva nové. Žádný tedy nevzniká jen tak někde v prostoru objektu a tudíž se dá pomocí pole informací o vsplit operacích vysledovat předek $v_c \in M^c$ jakéhokoliv bodu $v_f \in M^f$. Lépe to ilustruje obrázek 2.



Obrázek 2 a): Sekvence kolabsů hran, která vytváří hierarchii vertexů.



Obrázek 2 b): Výsledná korespondence vertexů (každý má jedinečného předka)

Můžeme to vyjádřit opět pomocí matematiky: vertex v_j z meshe M^f koresponduje s vertexem $v_{A^c(j)}$ z meshe M^c :

$$A^c(j)=\begin{cases} j & , j \leq m_0+c \\ A^c(S_{j-m_0-1}) & , j > m_0+c \end{cases}$$

Právě díky těmto informacím je možné vytvořit geomorf $M^G(\alpha)$ jako je $M^G(0)$, který vypadá jako M^0 a $M^G(1)$ odpovídá zase M^n .

Jednoduše definujeme $M^G(\alpha)=(K^f, V^G(\alpha))$ mající spojitost jako M^f a pozice vertexů $v_j^G(\alpha)=(\alpha)v_j^f+(1-\alpha)v_{A^c(j)}^c$.

Nyní, když jsme si již ukázali jak konstruovat geomorf mezi meshi obsahujícími pouze pozice vertexů, můžeme samozřejmě vytvořil tento plynulý přechod i mezi meshi obsahujícími jak diskrétní, tak skalární atributy.

Diskrétní atributy sice pro svoji přirozenou podstatu nemohou být plynule interpolovány, ale naštěstí jsou svázány s polygony meshe, které mohou být do meshe „plynule“ vloženy. Právě díky tomu, že pozice vertexů je v geomorfu $M^G(0)$ stejná, jako v jednodušším z obou meshů (v našem případě M^0), jsou polygony, které by normálně M^0 neobsahoval, zhrouceny do nulové plochy

(alespoň dva ze tří vrcholů polygonu leží v jednom místě). Díky tomu není polygon vidět a v průběhu zvyšování $\alpha \to 1$ se polygon „rozevírá“ a tím je jeho diskrétní atribut stále více patrný.

Skalární atributy definované v rozích mohou být mnohem snáze interpolovány. V podstatě jako pozice vertexů. Je zde jen drobná komplikace v tom, že roh (v, f) není asociován přímo s nějakým svým „předkem“. Jako jedno z možných řešení rohů, které v hrubším meshi nejsou, se jeví nalézt takový mesh M^i, ve kterém byl tento roh (s jeho skalárními informacemi) poprvé vytvořen a nalézt jeho sousední roh $(v, f') \in M^i$, jehož hodnoty atributů jsou stejné. Stejně tak se musíme rekurzivně vracet s tímto novým rohem (v, f') a opět hledat kde byl vytvořen a z jakého rohu. Tak dojdeme až k hrubšímu z obou meshů v geomorfu a tím i k hodnotě, ze které budeme interpolovat až do původního rohu (v, f). Pokud v průběhu našeho rekurzivního návratu zjistíme, že neexistuje takový roh (v, f') se stejnými atributy, pak našemu rohu jeho atributy ponecháme konstantní po celou dobu geomorfu.

Interpolační funkce při geomorfu nemusí být nutně lineární. Například normálové vektory se lépe mění pomocí jednotkové koule (nejkratší vzdáleností, na jejím povrchu, mezi oběma body), nebo barva může být interpolována v jiném než právě RGB prostoru.

4.7 Výběrové prokreslení (selective refinement)

Progressive mesh také podporuje výběrové prokreslení pouze ve vybraným oblastech z modelu. Pro názornost si definujme funkci $PROKRESLIT(v)$, která vrací ano/ne indikující, zda by okolí vertexu v mělo být více prokresleno. Výchozí mesh M^0 bude stejně jako jindy prokreslen polem operací $\{vsplit_0, \ldots, vsplit_{n-1}\}$, ale samotná operace $vsplit_i(s_i, l_i, r_i, A_i)$ je provedena pokud platí, že

a) všechny tři vertexy $\{v_{s_i}, v_{l_i}, v_{r_i}\}$ jsou v meshi M^i přítomny a zároveň

b) $PROKRESLIT(v_{s_i})$ vrací Ano

(Vertex v_j v Meshi M^i chybí pokud operace, kterou měl být do meshe přidán ($vsplit_{j-m_0-1}$) nebyla provedena pro nesplnění podmínek.

Například pro prokreslení modelu pouze v zorném poli kamery definujeme funkci $PROKRESLIT(v)$ tak, aby vracela Ano pouze pokud v je uvnitř zorného pole. Taková funkce ale přesto plně nedostačuje, protože například okolí vertexu v_{s_i} uvnitř zorného pole nemusí být prokresleno, z důvodu chybějícího vertexu v_{l_i}, který sousedí s vertexem v_{s_i}, ale je jednoduše mimo zorné pole a nebyl tedy vytvořen. Problém se napraví tím, že trochu pozměníme první

podmínku **a)**. Definujeme si takzvaného nejbližšího žijícího souseda vertexu v_j s následujícím indexem:

$$A'(j)=\begin{cases} j & \text{, pokud } v_j \text{ existuje v meshi} \\ A'(S_{j-m_0-1}) & \text{, v ostatních případech} \end{cases}$$

Nová podmínka tedy bude následující: **a')** vertex v_{s_i} je přítomný v meshi (neboli $A'(s_i)=s_i$) a vertex $v_{A'(l_i)}$ a $v_{A'(r_i)}$ jsou oba sousedy v_{s_i} .

Stejně jako v geomorfu může být i informace A' efektivně ukládána jak postupujeme vsplit operacemi. Pokud jsou podmínky **a')** a **b)** splněny, $vsplit(s_i, A'(l_i), A'(r_i), A_i)$ operace je aplikována do meshe. I s těmito novými vylepšeními je samozřejmě stále možné vytvářet mezi jakýmikoliv dvěma úrovněmi geomorf.

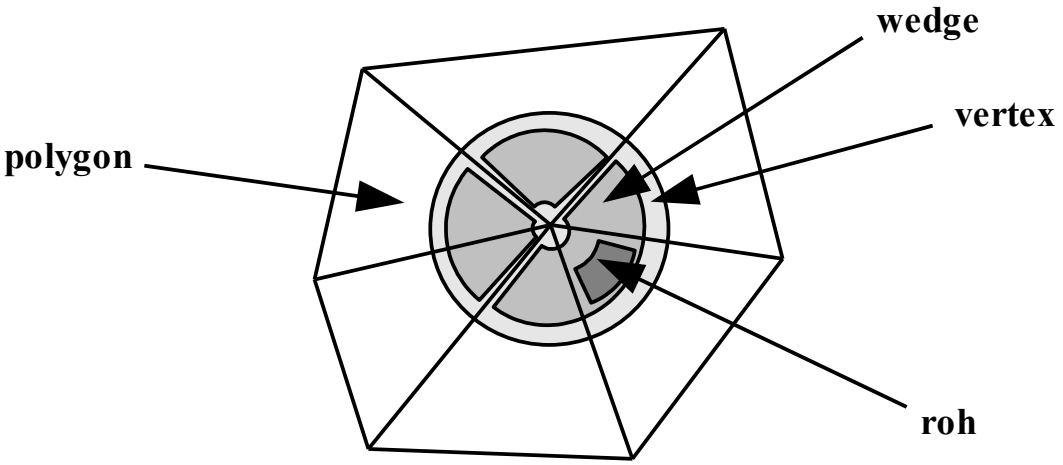
Zajímavou aplikací by v tomto případě mohlo být zobrazování výběrově prokresleného modelu načítaného přes pomalý komunikační kanál. Přicházeli by jen ty vsplit operace, pro které $PROKRESLIT(V)$ vrací Ano a které dosud staženy nebyly.

5 Wedges

Meshe v počítačové grafice mají často celou řadu atributů, které společně ovlivňují výsledný vzhled a dojem z 3D objektu. Dají se rozdělit do dvou základních skupin. Jednou jsou diskrétní atributy, asociované převážně s polygony. Druhou skupinu tvoří atributy skalární, které se asociují přímo s vrcholy meshe. Jak už bylo dříve řečeno, diskrétní atributy udávají použitou funkci pro vykreslení polygonu, skalární atributy udávají její lokální atributy. Zde se ale objevuje jeden významný problém. Lokální atributy patří jedné funkci, ale vertexy nepatří jednomu polygonu. Z toho vyplývá, že asociujeme-li k vertexu lokální atribut funkce pro vykreslení polygonu, který tento vertex obsahuje, bude nejspíše existovat jiný polygon, který tento vertex také používá, ale funkci má jinou a chce v daném vertexu mít i jiný atribut.

Řešením tohoto svízelného problému může být jistá abstrakce těchto skalárních atributů nad samotnými vertexy. Vytvoříme datovou strukturu, které bude mít svůj atribut a ukazatel na vertex, ke kterému tento atribut patří. Těmto členům budeme říkat anglicky wedge. Díky této abstrakci tak bude moci mít jeden vertex více atributů, které jsou s ním svázány. Polygony poté nebudou používat přímo vertexy, ale wedge, které je na daný vertex odkáží.

Dalším pojmem je „roh“ (anglicky corner), který se definuje pro spojení jednoho polygonu a jednoho vertexu. Lépe tyto všechny pojmy ilustruje následující obrázek:



Obrázek 3: Rozdělení jednoho vertexu na wedge a rohy.

6 Konstruování progressive meshe

V progressive meshi se nestačí pouze umět správně pohybovat skrze sekvenci vsplit operací, musíme je samozřejmě také správně vygenerovat. Tato kapitola se právě touto fází práce s progressive mesh zabývá. Rozdíl oproti generování nějaké aproximace mezi již vypočtenými M^0 a M^n je především v možnosti věnovat generování vsplit operací více strojového času. Konstruování progressive meshe totiž probíhá předem, buď při spuštění aplikace, nebo již při vytvoření modelu (pak jsou vsplit operace uloženy spolu s hrubým meshem M^0 díky speciálnímu formátu přímo v souboru). Konstruování probíhá v následujícím pořadí: Nejprve obdržíme nejdetailnější mesh M^n, vybereme pomocí algoritmů, které si ukážeme později, nejvhodnější hranu ke kolapsu, vygenerujeme vsplit záznam o této operaci, poté podle něho zhroutíme hranu (ecol) a tím získáme nový mesh M^{n-1}. Takto postupujeme dále, dokud již nelze korektně zhroutit ani jednu hranu a nám tak zůstane nejhrubší mesh M^0. Právě algoritmus, který pro výběr hrany zvolíme a jaké přiřadíme zjednodušenému meshi skalární a diskrétní atributy, nejvíce ovlivní výslednou kvalitu jednotlivých aproximací M^i, $i<n$.

Existuje celá řada různých algoritmů, které se liší jak kvalitou, tak i rychlostí. Extrémním případem je např. volit hrany kompletně náhodně (samozřejmě musejí být přesto některé lokální podmínky splněny, jinak by nešlo o validní transformaci). Sofistikovanější metody často využívají

heuristiku ke zlepšení strategie výběru hrany. Například může hrát roli vzdálenost od roviny, jak ji uvádí Hoppe [3]. Druhý extrém je prohledání všech možných meshů a nalezení opravdu optimálního řešení s respektem na nějaké vzhledové vlastnosti, jako je E_{dist} metrika od Hoppe [3]. Jakmile je progressive mesh vygenerován, můžeme se již pohybovat všemi aproximacemi velmi rychle pouhým aplikováním vsplit/ecol transformací. V následujících kapitolách se podíváme detailněji právě na algoritmy, které konstruování obstarávají.

6.1 Optimalizace povrchu meshe

Pokud progressive mesh použijeme na jednorázové zjednodušení meshe, například při optimalizaci pro uložení na disk, přenos a nebo zobrazení, je vhodné po zjednodušení na požadovanou úroveň detailů zoptimalizovat výslednou aproximaci pro co nejpřesnější vyjádření s ohledem na původní mesh. Cílem optimalizace meshe je tedy nalezení takového meshe $M=(K,V)$, který zároveň co nejpřesněji odpovídá množině X bodů $x_i \in \mathbb{R}^3$, které jsou pokud možno rovnoměrně rozprostřeny na každém vrcholu a na jednotlivých polygonech meshe, a má malý počet vrcholů. Tyto body si vysvětlíme za nedlouho, ale nyní si k něm jen řekněme, že slouží pro přesnější měření kvality výsledné aproximace. Problém optimalizace je označován jako minimalizace energetické funkce $E(M)=E_{dist}(M)+E_{rep}(M)+E_{spring}(M)$. První dva členy korespondují s dvěma cíly přesnosti a konzistence aproximace: výraz "energie vzdálenosti" $E_{dist}(M)=\sum_i d^2(x_i, \phi v(|K|))$ označuje totální vzdálenost (čtverečnou - tedy umocněnou dvěmi) bodů od povrchu originálního meshe a tzv. "reprezentační energie" $E_{rep}(M)=c_{rep}m$, která penalizuje počet m vertexů meshe M. Třetí složka vztahu je označována jako "pružinová energie" $E_{spring}(M)$ upravuje optimalizační problém tak, aby se povrch nového meshe zbytečně nezvětšoval. Je toho dosaženo tak, že do každé jedné hrany meshe se umístí imaginární pružina s klidovou délkou nula a tahem κ: $E_{spring}(M)=\sum_{\{j,k\}\in K} \kappa \|v_j - v_k\|^2$ Výsledná energetická funce je minimalizována za použití časově náročné metody, která sestává ze dvou cyklů. Ten vnější optimalizuje povrch meshe pomocí transformací nad ním, zatímco ten vnitřní pro každou možnou operaci vypočítává hodnotu, jak moc se daná změna projeví do celkové přesnosti meshe. Vnější smyčka pak podle této hodnoty rozhodne o tom, zde se daná operace provede nebo se bude hledat jiná.

6.1.1 Vnější smyčka

Algoritmus pracuje se spojitostí K meshe M a to tak, že náhodně aplikuje jednu ze tří možných transformací: První je nám již známý kolabs hrany, dalším je prohození hrany (kdy v rámci jednoho čtyřúhelníku, neboli dvou sousedním polygonů, prohodíme jednu úhlopříčku s druhou) a poslední transformací je rozdělení hrany. Tato sada tří transformací je kompletní v tom smyslu, že jakákoliv spojitost K stejného typu topologie jako má $\widehat{K}$ může být dosažena z původního meshe aplikací řady transformací právě těchto tří typů. Algoritmus funguje tak, že pro jakéhokoliv kandidáta na transformaci $K \rightarrow K'$ vypočítá $E_{k'}$. Minimum z E pak vede k nové spojitosti K'. Pokud se zjistí že $\Delta E = E_{K'} - E_K$ je shledána jako negativní, pak je daná transformace aplikována a pokračuje se s novou spojitostí K'. Tato metoda se dá srovnat s metodou simulovaného žíhání.

6.1.2 Vnitřní smyčka

Pro každého kandidáta na transformaci meshe spočítá algoritmus potencionální energii takové operace $E_{K'} = min_v E_{dist}(V) + E_{spring}(V)$ optimalizací pozic vertexu V. Pro efektivitu optimalizuje algoritmus pouze jednu pozici v_s a vyřeší pouze podmnožinu takových bodů X, která se promítne do sousedství ovlivněného použitou transformací. Je třeba se ještě vyvarovat průniků polygonů a tak se daná transformace zakáže, pokud maximální úhel dvou stěn (v anglické literatuře označovaný jako dihedral) přesáhne nějaké předem určené maximum. H. Hoppe [3] uvedl, že pružinvá energie $E_{spring}(M)$ je nejvíce důležitá právě v počátečních krocích optimalizace meshe a vůbec nejlepších výsledků dosáhl opakovaným prováděním časově náročné optimalizační metody, u které ale ještě měnili pružinovou konstantu κ v závislosti na tom, kolik je v okolí operace polygonů versus bodů X.

Nyní si tyto body trochu popíšeme. Optimalizace meshe je jistě účinný nástroj na zjednodušení 3D objektu, ale je třeba mít nějakou metriku, podle které poznáme, jak kvalitní aproximaci meshe vlastně máme. K tomu slouží tyto body z množiny X. Na původní mesh $\widehat{M}$ je na každý vrchol umístěn jeden bod x a zároveň nebo na přání uživatele i rovnoměrně (nebo náhodně) na jeho polygonech. Optimalizační algoritmus poté s tímto meshem $\widehat{M}$ začne. Pokud měníme reprezentační konstantu c_{rep} (připomeňme si, že tato konstanta penalizuje počet vertexů v meshi), dostáváme různé aproximace s rozdílným poměrem přesnosti a velikosti.

6.2 Shrnutí optimalizační funkce

I zde definujeme energetickou funkci $E(M)$ pro lepší kontrolu kvality daných aproximací meshe $M=(K,V,D,S)$ s respektem na původní mesh $\widehat{M}$ a stejně budeme i zjednodušovat tento mesh minimalizováním $E(M)$. Tentokrát ale energetická funkce obsahuje tyto složky: $E(M)=E_{dist}(M)+E_{spring}(M)+E_{scalar}(M)+E_{disc}(M)$

První dva prvky (E_{dist} a E_{spring}) jsou stejné jako v předchozím případě, další dva jsou přidány pro zachování diskrétních a skalárních atributů asociovaných s meshem. E_{scalar} zohledňuje přesnost jeho skalárních atributů a E_{disc} zase diskrétních atributů. Podle H. Hoppe [3] k tomuto atributu je jeho ústřední význam zachování především geometrické přesnosti tzv. křivek nespojitosti. Ty si vysvětlíme za chvilku, ale předem si jen řekněme, že právě rozdílné diskrétní atributy u sousedních polygonů jsou jednou z příčin vzniku právě těchto křivek. Ještě musíme doplnit, že pro dosažení nezávislosti na velikosti, či chcete-li měřítku meshe je třeba ho před výpočtem zmenšit/zvětšit do rozměru jednotkové krychle.

Tato metoda pro zjednodušení ale nepotřebuje všechny tři operace zmíněné v minulé kapitole. Na zjednodušení meshe ve skutečnosti stačí transformace kolabs hrany sama o sobě. Ostatní dvě operace, potřebné především při optimalizaci meshe, která i motivovala předešlý algoritmus, již nejsou tolik potřebné. Sice při dané aproximaci nedostaneme nejlepší řešení, ale díky tomuto zjednodušení hledáme vždy jen nejlepší hranu ke kolabsu a každou takovou operací nám klesne počet vrcholů o jeden, počet polygonů o jeden až dva a získáme tím vždy souvislou řadu jednotlivých aproximací od původního meshe $\widehat{M}$ do nejhrubšího meshe M^0. Tím dosáhneme nejen znatelného navýšení výkonu, ale i přes to, že tato jediná operace nemůže projít celý prostor možných aproximací meshe, je výsledná kvalita minimálně stejně dobrá, v mnoha případech i lepší. Důvodem je další vylepšení tohoto algoritmu o tzv. prioritní frontu. Každá přípustná operace se totiž zařadí do fronty, kde je jí vypočítána hodnota energie, kterou by tato operace stála ΔE při jejím použití. Z fronty se vybere nejvyšší operace, tedy s nejnižší energií ΔE, a ta se provede. Změny, které touto operací nastanou, se promítnou do jejího okolí a příslušné hodnoty se ve frontě aktualizují. Právě díky tomu, že každou operací klesne počet vertexů a polygonů, nemusíme již započítávat c_{rep} stejně jako $E_{rep}(M)$.

Pro každý kolabs hrany $K \rightarrow K'$ tedy spočítáme jeho energetickou cenu $\Delta E = E_{K'} - E_K$ vypočtením vztahu: $E_{K'} = \min_{V,S} E_{dist}(V)+E_{spring}(V)+E_{scalar}(V,S)+E_{disc}(V)$ z pozic vertexů a skalárních atributů S meshe o spojitosti K'. V následujících kapitolách si tento výpočet trochu přiblížíme.

6.2.1 Zachování povrchové geometrie

(E_{dist} + E_{spring}).

Jak už jsme si řekli dříve, promítneme nejprve na mesh řadu bodů $x_i \in X$. Musíme umístit body nejméně tak, aby byl jeden v každém vertexu. Samozřejmě je lepší, když další body rozprostřeme i po polygonech. Prvním z obou členů vztahu je E_{dist} , která zohledňuje vzdálenost povrchu konkrétní aproximace od povrchu původního objektu. Její výpočet je následující:

$$d^2(X_i, \phi_v(|K|)) = \min_{b_i \in |K|} \|x_i - \phi_v(b_i)\|^2$$

Zde se objevuje neznámá b_i , kterou H. Hoppe[3] definuje jako parametr projekce bodů x_i na mesh. Nelineární minimalizace $E_{dist}(V) + E_{spring}(V)$ je vyřešena pomocí iterativní procedury využívající dva kroky:

1) pokud máme fixní pozice vertexů V , spočítáme optimální parametrizaci $B = \{b_1, b_2, \ldots, b_{|x|}\}$ projekcí bodů X do meshe. Nebo za

2) pro fixní parametr B , spočítáme optimální pozice vertexů V vypočítáním pomocí řešení problému minimálních čtverců (suma umocněných vzdáleností).

Dalším prvkem výpočtu je zvážení pozice v_s^i při kolabsu hrany $ecol(\{v_s, v_t\})$. Vypočítáme ji se třemi výchozími pozicemi a to $v_s^i = (1-\alpha) v_s^{i+1} + (\alpha) v_t^{i+1}$ a to pro $\alpha = \{0, \frac{1}{2}, 1\}$ a použijeme nejlepší z nich.

Dalším rozdílem tohoto algoritmu je fakt, že nepoužijeme konstantní parametr κ pro E_{spring} , ale postupem zjednodušování meshe jeho hodnotu změníme. Jak už jsme si řekli (v kapitole 6.1.2 - Vnitřní smyčka), jeho význam je největší právě v počátečních fázích optimalizace, tudíž tehdy, kdy je pouze pár bodů x_i promítnuto do okolí transformace. Proto po každém provedení transformace κ přepočítáme a to tak, že κ bude funkcí poměru bodů a počtu polygonů v okolí transformace. Hoppe H. [3] zavedl výpočet (pokud použijeme C notaci) jako: $\kappa = r < 4 ? 10^{-2} : r < 8 ? 10^{-4} : 10^{-8}$, kde r je onen poměr bodů a polygonů v okolí kolapsu hrany. Tímto schématem tedy bude důležitost E_{spring} exponenciálně klesat spolu se zjednodušením meshe.

6.2.2 Zachování skalárních atributů

V kapitole 3 - Meshe v počítačové grafice jsme si uvedli spojité pole skalárních atributů S v rozích meshe. Nyní si popíšeme způsoby, jak tyto atributy co možná nejvíce zachovat při zjednodušování meshe. Nejprve si přiblížíme zjednodušení s takovými spojitými skalárními atributy, kde všechny rohy v jednom vertexu obsahují stejné skalární hodnoty. Zobecnění pro případy, kdy se i jednotlivé rohy ve vrcholech liší si přiblížíme hned poté.

Optimalizování skalárních atributů ve vrcholech: Přidáme-li do originálního meshe $\widehat{M}$ ke každému vertexu mimo jeho pozici v_j i vektor jeho skalárních atributů $\underline{v_j} \in \mathbb{R}^d$. Abychom měli opět lepší představu o kvalitě dané aproximace, rozmístíme znovu body x_i, tentokrát ale po skalárním poli atributů, tedy umístíme $\underline{x_i} \in \mathbb{R}^d$. Nyní stačí pozměnit člena energetické funkce E_{dist} tak, aby vypočítával nejen vzdálenost povrchu od originálu, ale také jeho skalárních aributů. Pomocí vzdálenosti množiny bodů $\underline{X}$ od těch v meshi M. Jedním způsobem jak tento výpočet E_{dist} upravit je předefinování metrik vzdálenosti tak, aby namísto $\mathbb{R}^3$ zvládla $\mathbb{R}^{3+d}$:

$$d^2((x_i, \underline{x_i}), M(K, V, \underline{V})) = \min_{b_i \in |K|} \|(x_i \;\; \underline{x_i}) - (\phi_V(b_i) \;\; \phi_{\underline{V}}(b_i))\|^2$$

Použití této funkce navenek zůstává stejné. Jediným problémem je to, že výpočet parametru b_i může být díky nutnosti projekce do $\mathbb{R}^{3+d}$ velmi náročný a neintuitivní, neboť skalární atributy nejsou založeny na geometrické pozici a není ani pro člověka lehké si projekci do více než tří-rozměrného sytému představit.

Druhou možností je vytvoření specializované funkce E_{scalar}, která bude výpočet energie rozdílných skalárních atributů obstarávat. Její výpočet je následující:

$$E_{scalar}(\underline{V}) = (c_{scalar})^2 \sum_i \|\underline{x_i} - \phi_{\underline{V}}(b_i)\|^2$$

Všimněte si nové konstanty c_{scalar}, která zde určuje poměr mezi váhou chyby ve skalárních atributech (E_{scalar}) a v geometrické pozici (E_{dist}).

Nakonec tedy výpočet minimalizování $E(V, \underline{V}) = E_{dist}(V) + E_{spring}(V) + E_{scalar}(\underline{V})$ proběhne tak, že nejprve algoritmus nalezne nejvhodnější pozici vertexu v_s, minimalizací $E_{dist}(V) + E_{spring}(V)$ projektováním bodů $x \in X$ do meshe a řešením lineárního problému minimálních čtverců. Poté za použití stejné parametrizace b_i, jakou jsme v minulém kroku obdrželi, nalezne atribut vertexu $\underline{v_s}$ minimalizováním E_{scalar} pomocí jednoho vyřešení minimálních čtverců. Proto také uvedením E_{scalar} do optimalizace způsobí zanedbatelné snížení výkonu.

Optimalizování skalárních atributů v rozích

Toto schéma pro optimalizování skalárních atributů S v rozích je jen zobecnění výpočtu, který byl právě nastíněn. Pro zrychlení nebudeme počítat v každém rohu ΔE zvlášť, ale rozdělíme si rohy kolem vertexu v_s do takových shluků, kde mají všechny rohy stejné atributy. Poté pro každou takovou řadu vyřešíme její nejlepší hodnoty skalárních atributů zvlášť.

Omezení rozsahu

Některé skalární atributy mají omezený rozsah svých hodnot. Například barva (r, g, b) má všechny své složky mezi hodnotami 0 a 1. Výpočet minimálních čtverců může vést k nastavení této hodnoty mimo její rozsah a proto je nutné tyto hodnoty saturovat (či chcete-li oříznout) aby pasovaly do svého rozsahu.

Normálové vektory

Normály povrchu (n_x, n_y, n_z) jsou dalším drobným problémem, neboť jejich délka musí být rovna 1. Tudíž jejich souřadnicový systém není Karteziánský. Řešením je buď normálně prolnout mezi oběma normálovými vektory v bodech $v_{s_i}^{i+1}$ a $v_{m_0+i+1}^{i+1}$ a výsledný normálový vektor normalizovat na délku rovnající se 1 (to přináší několik komplikací, když například oba původní vektory leží v jedné přímce, ale jsou opačně orientovány), druhým způsobem, který uvádí H. Hoppe [3] je získat normálový vektor ve vrcholu $v_{s_i}^{i}$ opět prolnutím dvou výchozích bodů a to za použití α hodnoty získané při nalezení nejlepší pozice vrcholu $v_{s_i}^{i}$ v kapitole 6.2.1 - Zachování povrchové geometrie. Naštěstí nejsou konkrétní hodnoty normálových vektorů až tolik vizuálně důležité, jako je jejich nespojitost. Jejímu řešení se zabývá následující část.

6.2.3 Zachování křivek nespojitosti

Velký význam na celkovém dojmu z vykresleného meshe mají křivky na jeho povrchu, kde se nespojitě mění povrchové vlastnosti objektu. Jedná se nejen o rozdílné diskrétní atributy asociované s polygony, ale samozřejmě i rozdílné skalární atributy asociované s rohy polygonů. Vzhledem k jejich viditelnosti je dobré optimalizovat tyto křivky nejen jejich parametry (řekněme topologicky), ale i jejich umístěním (geometricky).

Existují způsoby jak detekovat, kdy zamýšlená transformace změní právě tyto křivky a tím i celý vzhled meshe. Stačí použít několik jednoduchých testů na přítomnost ostrých hran v sousedství transformace. Označíme si funkci, která bude ostrost hrany detekovat, třeba $OSTROST(v_j, v_k)$. Pokud vrátí Ano , znamená to, že hrana $\{v_j, v_k\}$ je ostrá a funkce $POCET(v_j)$ bude funkce

vracející počet ostrých hran obsahující vertex v_j . Potom transformace $ecol(\{v_s, v_t\})$ modifikuje topologii křivek nespojitosti pokud platí alespoň jedna z následujících možností:

a) $OSTROST(v_s, v_l)$ a zároveň $OSTROST(v_t, v_l)$

b) $OSTROST(v_s, v_r)$ a zároveň $OSTROST(v_t, v_r)$

c) $POCET(v_s) \geq 1$ a zároveň $POCET(v_t) \geq 1$ a neplatí $OSTROST(v_s, v_t)$

d) $POCET(v_s) \geq 3$ a zároveň $POCET(v_t) \geq 3$ a $OSTROST(v_s, v_t)$

e) $OSTROST(v_s, v_t)$ a zároveň $POCET(v_s) = 1$ a zároveň $POCET(v_t) \neq 2$

f) $OSTROST(v_s, v_t)$ a zároveň $POCET(v_t) = 1$ a zároveň $POCET(v_s) \neq 2$

Pokud tedy má daná transformace za následek změnu křivek nespojitosti, buď ji zcela zakážeme, nebo ji alespoň penalizujeme nějakou hodnotou, přičtenou k výsledné energii $E(M)$.

Hoppe [3] opět uvádí jako nejvhodnější řešení přidání dalších bodů X_{disc} do ostrých hran meshe $\widehat{M}$ a definování další energie E_{disc} , která je v podstatě ekvivalentní k již definované E_{dist} , jen s tímto rozdílem, že namísto bodů rozprostřených po vrcholech a náhodně po polygonech, bude zkoumat odchýlení bodů rozprostřených po křivkách nespojitosti.

7 Budoucí vývoj

Jak již bylo zmíněno v kapitole 6.2.2 - Zachování skalárních atributů, není zatím zcela vyřešena přesná metrika pro interpolaci normálových vektorů, stejně jako barevné složky a dalších skalárních atributů asociovaných s meshem.

Další možností budoucího vývoje je optimalizace progressive mesh reprezentace pro animované objekty a to jak geometricky, měnící svoji topologii, tak i s proměnlivými skalárními atributy.

Stejně je třeba i výkonnější metody pro chybovou metriku a pro optimalizaci výběrového prokreslení.

Všechny tyto a další požadavky dávají dostatek cílů pro budoucí vývoj v této oblasti a jistě bude zajímavé tento vývoj sledovat.

8 Implementace

Výsledná implementace je k nahlédnutí v Příloze 3. - přiložený kompaktní disk. Konečnou podobu implementovaného ukázkového algoritmu a práce s progressive meshem ovlivnilo několik podstatných okolností. Jednou z nich byla i samotná dostupnost materiálů a především implementačních detailů a různých nástrah při vývoji. Mnoho z těchto problémů bylo nutné řešit spíše neinženýrsky - a to znovu řešení vymyslet - namísto navázání či využití poznatků již existujících.

Proto pro případný další vývoj implementace progressive mesh reprezentace bylo několik nejčastějších problémů spolu s řešeními sepsáno a spolu se samotnými výsledky implementování algoritmu se s nimi můžete seznámit v následujících kapitole. V dalších si pak přiblížíme konkrétní implementaci.

8.1 Praktické problémy při řešení

V průběhu studia tohoto problému a především při implementaci některých řešení se vyskytlo několik nesrovnalostí a problémů, kterým se literatura příliš nevěnovala. Uvádím je tedy ve výčtu spolu s jejich řešením.

Nejprve je třeba vytvořit generování hrubého meshe M^0 spolu s celým polem vratných informací vsplit. Uvědomíme-li si, že aplikace jedné vratné operace vsplit přidá jeden až dva polygony a jeden vertex „na konec“ celého seznamu polygonů potažmo vrcholů, je zřejmé, že i před aplikací transformace kolaps hrany (ecol) musí být právě odstraňované polygony a vrchol uloženy až na samém konci. To ještě před samotnou možností provedení jednoho aproximačního kroku vyžaduje vyřešení problémů s vyměněním polygonu s jiným (posledním), s jeho natočením, dále s výměnou vrcholu a s ním souvisejících wedgů (viz kapitola 5 - Wedge) za poslední a to vše stále s důrazem na zachování validity meshe a správné návaznosti sousedním polygonů (každý takový si totiž nese informace o indexech sousedním polygonů, což umožňuje snadný pohyb po povrchu meshe při generování operací) a také všech již uložených vsplit operací, které s těmito polygony souvisejí. Samo prohození je již otázkou implementace, ale musíme mít na paměti, že takové operace se musí vyřešit ještě před tím, než provedeme první aproximační krok.

Dalším problémem jsou zmíněné wedge. Při zjednodušení meshe se totiž musí klást důraz i na zachování spojitosti povrchu a to i v případě, že dva sousední polygony nemají společné diskrétní ani skalární vlastnosti. Je tedy nutné nějakým způsobem abstrahovat pozici vrcholu a uvedené vlastnosti ukládat zvlášť včetně vhodného navázání na příslušný vrchol. Zároveň se takové informace

musí zákonitě promítnout i do uložených vsplit operací, viz kapitola 4.5 - Komprese a paměťová reprezentace meshe. Je proto třeba již od počátku vývoje mít na paměti nutnou abstrakci vertexů pomocí wedgů, díky které je samotná vícečetnost skalárních atributů v jednom vertexu možná.

Již při samotném vykreslování nastává další problém, kterým je (díky zmiňovanému ukládání wedges, potažmo tedy více skalárních informací k jednomu vrcholu) zobrazování dané aproximace s informace v těchto vrcholech opět co možná nejspojitějšími. Při generování a pohybu v progressive mesh reprezentaci je dobré jednotlivé rozdíly v atributech znát, ale při vykreslování je opět pole skalárních atributů třeba mít spojité. Tím musíme před zobrazením tyto informace v každém vrcholu sloučit do jedné a s tou mesh vykreslit. Většinou postačí jejich zprůměrování, ale dají se použít daleko sofistikovanější metody, jako je vážený průměr podle velikosti polygonu, nebo v případě normálových vektorů i průměrná pozice na povrchu jednotkové koule. Přiloženém program zobrazuje i nezprůměrované hodnoty ve wedge, viz dokumentace.

8.2 Implementovaný algoritmus

Pro generování, zobrazení a pohyb mezi jednotlivými aproximacemi meshe (či chcete-li úrovněmi detailů) bylo potřeba implementovat samotné operace vsplit a ecol, stejně jako efektivní ukládání do paměti a mnoho další rutin, které jsou pro progressive mesh reprezentaci potřebné (jako jsou pro výše zmíněné prohození polygonů, vrcholů, wedgů, tak i pro testy validyty dané operace, zda nevede ke zhroucení do roviny a k intersekci polygonů).

Jako nevhodnější reprezentace progressive mesh se zdála být verze popsaná v kapitole 6.2 - Shrnutí optimalizační funkce, která využívá pouze jednu operaci kolaps hrany a k ní opačnou operaci rozdělení vrcholu. Tím je generována souvislá řada jednotlivých aproximací meshe od nejkomplexnějšího $M^n = \widehat{M}$ až do nejhrubšího M^0. Reprezentace byla tedy implementována, včetně práce s wedges, materiály, texturami a jeho možnost nasazení je univerzální, ať už na objekty konvexní, konkávní, či vůbec neuzavřené.

Na počátku směřoval vývoj ke zdárnému konci, ale jako největší a téměř nepřekonatelný problém se ukázala implementace chybové metriky, která slouží k měření přesnosti jednotlivých verzí meshe. Jak již bylo řečeno, taková metrika prochází celý povrch meshe a to pro každou přípustnou operaci, aby dokázala, která z nich je nejefektivnější, tudíž s nejmenší ztrátou energie (přesnosti meshe). I přes velké optimalizace, jako je promítnutí a přepočítání změn pouze v okolí operace se výsledná implementace nezdařila vytvořit dostatečně výkonná (a to ani při třetím pokusu o její implementaci) a byla nakonec zvolena jednodušší chybová metrika, která ale bohužel nedokáže dostatečně konkurovat metrice zde popsané.

I přesto byla do programu nakonec přidána metrika Microsoftu skrze Progressive mesh reprezentaci implementovanou v rozhraní DirectX9 a je tedy možno obě získané implementace přesto porovnat.

Implementovaný algoritmus obsahuje i efektivní způsob ukládání meshe, díky vhodné volbě struktur pro ukládání, využívajících každý jednotlivý bit pro nějakou informaci. Tento vhodný a efektivní způsob paměťové reprezentace byl definován v práci Deering M [7].

Navíc obsahuje výsledný program i geomorf, který demonstruje plynulý přechod mezi dvěma velmi vzdálenými úrovněmi detailů. Bohužel progressive mesh reprezentace od Microsoft DirectX9 tento přechod neobsahuje a není proto možné jej porovnat.

Výsledná progressive mesh reprezentace byla vsazena do jednoduchého terénu a díky dynamickému generování úrovní detailů je možné ji porovnat v reálném nasazení například do her.

8.3 Detaily konkrétní implementace

Pro implementaci jsem si vybral jazyk C++ a výsledný progressive mesh je zobrazován pomocí knihovny DirectX9, což má za následek menší multiplatformnost a program je tedy koncipován pro operační systém Windows. Přesto byl psán z velkým důrazem na samostatnost, a proto je jediným problémem třída DxWindow (viz 8.3.1 - Třídní dekompozice), která se stará o zobrazení scény.

V implementovaném programu mi nešlo jen o demonstraci algoritmu a práce s progressive mesh reprezentací, ale také o ukázku reálného nasazení Progressive mesh v 3D virtuální scéně. Program má proto dva režimy. Prvním z nich je zobrazená mapa s rozesetými objekty, ve které se může uživatel libovolně pohybovat. Navíc má možnost si výběrem nějakého 3D objektu zobrazit jeho konkrétní statistiky (přepnout se do druhého režimu). Zde je už je samotný objekt, spolu s ekvivalentní reprezentací Progressive mesh od Microsoftu generovaným přímo DirectX9. Navíc má zde uživatel další možnosti, jako je ruční změna úrovně detailů, otáčení a přibližování modelu, nebo přepnutí se na další model, aby nebyl nucen onoho pohybu po mapě a mohl okamžitě srovnávat jednotlivé objekty.

Výsledný program s implementovaným algoritmem je k nahlédnutí na přiloženém CD nosiči, viz příloha 3.

8.3.1 Třídní dekompozice

Důležitou částí návrhu byla dekompozice řešení do jednotlivých objektů/tříd. Výsledný program má tyto tři:

ProgressiveMesh

Hlavní třída starající se o ukládání, generování, transformování a optimalizování meshe. Tvoří největší část zdrojového textu a je naprosto nezávislá na použité platformě, či vykreslovacím zařízení. Sama, ale vykreslovat neumí a pouze vrazí struktury obsahující data pro vykreslení.

DxWindow

Třída starající se o práci s DirectX9. Jejím úkolem je na začátku vytvořit okno pomocí WinAPI a do něho inicializovat DirectX, včetně práce s ním při změnách rozměrů okna, minimalizaci, ztrátě vykreslovacího kontextu, atd.

DxKamera

Třída obstarávající interakci s uživatelem. Reaguje na vstupy z klávesnice a z myši, mění matice pohledu a zobrazení scény.

DxApp

Pro maximální objektovost je i zbytek programu zapouzdřen do objektu. Je to jakýsi hlavní uzel obstarávající koordinaci a kooperaci jednotlivých dílčích objektů v programu.

Třídy spolu komunikují pomocí callbacků - tj. funkcí, které se mezi třídami vyměňují jako ukazatele na ně a každý objekt si poté volá funkci z jiného zcela sám. To bylo nejvíce využito ve třídě DxWindow, která obsahuje celou řadu obecných rutin pro například inicializaci a pak několik funkcí, které jsou pro každý program jedinečné a proto jsou implementovány přímo v DxApp a třída DxWindow si je volá kdy uzná za vhodné (např. vykreslení scény, odalokace, atd.)

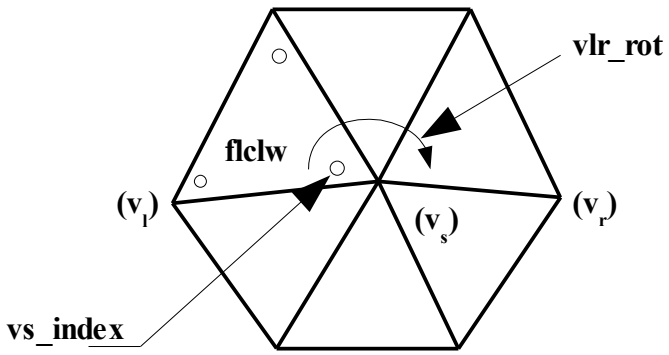
8.3.2 Struktury a paměťová reprezentace

Nejlepším řešením pro implementaci paměťové reprezentace se zdála být verze od Hoppe H.[2].
Například paměťová reprezentace vsplit informace byla navržena takto:

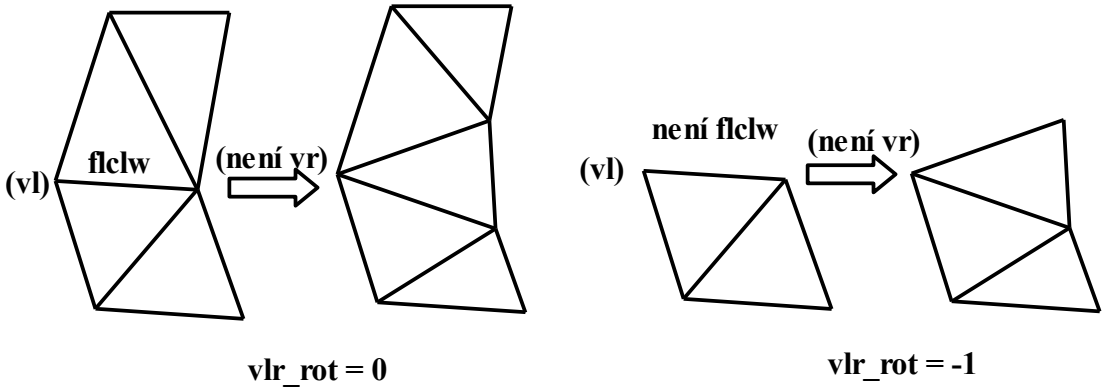
```
struct Vsplit {
     int flclw;           // polygon v sousednství operace
     short vlr rot;      // zakódování vertexu vr
     struct {
          short vs index : 2;           // index (0..2) vs uvnitř flclw
          short corners : 10;           // návaznost hran
          short ii : 2;                 // geometrický predikát
          short matid predict : 2;      // zda jsou fl matid, fr matid potřeba?
     } code;                            // sada 4 bitových-polí (16-bit celkem)
     short fl matid;                    // matid polygonu fl pokud není predikát
     short fr matid;                    // matid polygonu fr pokud není predikát
     VertexAttribD vad l, vad s;
     Array<WedgeAttribD> wads;
};
```

Obrázek 4: paměťová reprezentace vsplit informace

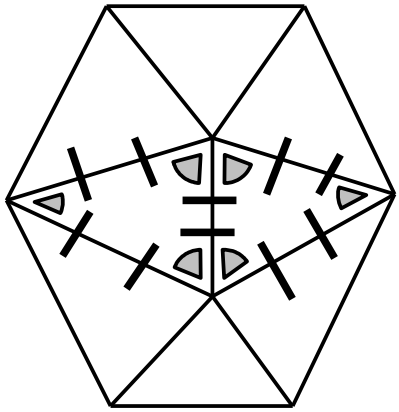
Což odpovídá těmto schématům zakódování vsplit informací:



Obrázek 5: Vysvětlení vsplit parametrů.



Obrázek 6: Speciální případy zakódované do vsplit informace



Obrázek 7: 6 Nových rohů přidaných vsplit operací a 10 bitů informací (corners) o návaznosti rohů na sousední, již existující.

8.3.3 Ovládání

Program byl navržen pro maximální intuitivnost, která vyžaduje vhodnou kombinaci ovládání klávesnicí a myší. Po spuštění programu (a vygenerování progressive mesh reprezentací) se uživateli zobrazí 3D scéna s rozmístěnými objekty. V tuto chvíli je ovládání srovnatelné s obvyklými 3D počítačovými hrami (tedy pohyb klávesnicí pomocí šipek a otáčení kamery pomocí myši).

Jediným rozdílem je, že pro otáčení kamery musí být současně stisknuto pravé tlačítko u myši. Je to dáno proto, že samotný pohyb kurzoru myši po oknu aplikace je také potřebný. Tím totiž uživatel vybírá objekty, jejichž detaily chce zobrazit. To dá programu najevo stiskem levého tlačítka myši, čímž se program přepne do režimu zobrazování detailů.

V tomto režimu se ovládání částečně mění. Pohybem myši se stisknutým pravým tlačítkem již uživatel neotáčí kamerou, ale objektem před sebou, aby si ho mohl snáze prohlédnout ze všech stran. Má zde možnost měnit úroveň detailů ('Q' přidá do modelu detaily, 'A' je opět ubere), dále může přepnout na další objekt ('S') A může také objekt libovolně vzdalovat a přibližovat (kolečko myši, popř. 'X','Y').

V obou režimech pak může uživatel zapínat/vypínat geomorf ('G'), textury ('T'), materiály ('M') nebo zapínat/vypínat vyhlazení wedgů v jednotlivých vertexech ('W').

8.3.4 Fáze při vývoji programu

Nejdříve bylo nutné vytvořit vhodné struktury pro paměťovou reprezentaci (viz kapitola 8.3.2 - Struktury a paměťová reprezentace). To obnášelo i implementaci jejich operátorů, konstruktorů, destruktorů. Poté bylo nezbytně nutné tuto reprezentaci vizualizovat a vytvořit nástroj na výpis 3D modelu do souboru. Spousta informací totiž není vizuálně patrná a pro takové věci jako je například správná sousednost (rozumějme správný index na sousední polygon a u něho zase správný index na polygon aktuální) bylo nutné několik týdnů až měsíců kontrolovat správnost vývoje pouze pomocí vypsaného souboru (dump 3D modelu).

Poté bylo již možné vyvíjet Progressive mesh utility. Nejprve to byla vsplit operace (zatím s konstantními daty) a následně i ecol operace. Ihned poté vznikla potřeba správného přeskupení polygonů, vertexů a především informací pro vsplit operace, které záviseli na polygonu, jenž byl přesunut. Následně bylo vytvořeno generování vsplit operace ze zadaného polygonu a indexu jeho hrany. Nyní již bylo možné, byť náhodným přístupem, objekt skutečně zjednodušovat.

Dalším krokem bylo přidání několika testů, především test na zhroucení do roviny a test na zhroucení středu plochy do bodu, což by znemožnilo validitu následujících transformací. S tím souviselo i navrácení meshe do přesně stejné podoby, pokud se prováděná transformace zakázala. Poté se, do této doby statické, modely začaly načítat ze souboru. Zvolen byl formát *.x, neboť nejen

přirozeně souvisí s DirectX a umožňuje velmi mnoho přídavných informací, ale navíc může být ukládán v textové podobě a jeho editace je díky tomu naprosto intuitivní. Spolu s načítáním ze souboru přišla potřeba načítat i uložené normálové vektory a to vedlo nakonec k velkým změnám v celém zdrojovém textu a k nastolení práce s wedges namísto přímo vertexů. Také bylo nutné zakódovat informace o rozích (viz Obrázek 7).

Dále se upravil program aby rozpoznal kdy už nejde vygenerovat další operaci a použít ji na zjednodušení meshe a aby generování ukončil. Tím začal vznikat také celý řetěz vsplit informací. To vedlo k implementaci fronty s předpočítáním E(M) a vybírání hrany ke kolapsu podle heuristiky. Ta byla zvolena na základě délky hrany a představuje tak chybovou metriku sice nejrychlejší, ale se špatnými výsledky.Poté byl implementován geomorf a stal se součástí detailního výpisu. Pro složitost ovládání jednotlivých mezí byl zvolen rozsah od nejjednoduššího meshe po nejsložitější a tento geomorf je proto určen od počátku spíše pro demonstraci, než ukázku reálného nasazení. Do implementovaného programu poté ještě přibyla spousta dalších věcí, jako je práce s texturami, materiály, lepší ovládání kamery a pohyb po mapě, jednotlivé vymodelované objekty, zahlazení hran wedgů, uživatelské rozhraní, a mnoho dalšího.

9 Závěr

Závěrem nutno říci, že jsem byl se zvoleným tématem v průběhu vývoje i přes značné problémy spokojený a byl mi velkým přínosem. Mohl jsem využít dlouholeté zkušenosti z 3D počítačové grafiky a pokusit se vytvořit netriviální nástroj na zefektivnění aplikací využívajících 3D grafiku.

Téma této bakalářské práce jsem si vybral s cílem vytvořit vlastní implementaci uváděného algoritmu a díky možné rozšiřitelnosti časem dosáhnout možností (byť ne výkonu), které progressive mesh reprezentaci implementovanou firmou Microsoft v DirectX9 překonává. Především ve výběrovém prokreslení a geomorfu (viz kapitoly 4.7 a 4.6) , které tato implementace nezahrnuje. I přes částečný neúspěch při implementaci vhodné chybové metriky pro tento progressive mesh se tedy jistě s výsledkem nespokojím a budu na projektu i nadále pokračovat.

Nabízejí se i témata, která před nedávnou dobou Hugues Hoppe publikoval a která byť na práci na progressive mesh přímo nenavazují, přesto její výsledky pro svou implementaci přímo vyžadují. Mám na mysli například Geometrické obrázky (v originále Geometry images), které umožňují ukládání 3D objektů do 2D obrázků (zjednodušeně řečeno převodem (x, y, z) do (r, g, b)) a mají jistě velkou budoucnost v optimalizaci datové komunikace na sběrnici grafické karty. A zajímavých témat využívajících progressive mesh je celá řada.

I pro tyto důvody je v mém zájmu dále tento projekt rozvíjet.

Literatura

[1] Garland M., Heckbert P. S., *Surface Simplification Using Quadric Error Metrics*, Carnegie Mellon University, 1997

[2] Hoppe H., *Efficient Implementation of Progressive Meshes,* Technical Report MSR-TR-98-02 (Microsoft Research), 1998

[3] Hoppe H., *Progressive Meshes,* Computer graphics (SIGGRAPH '96), 1996

[4] Hoppe H., *View-Dependent Refinment of Progressive Meshes,* (SIGGRAPH '97) 1997

[5] Shaffer E., Garland M., *A Multiresolution Representation for Massive Meshes*, Computer graphics, 2005

[6] IEEE - *The world's leading professional association for the advancement of* technology (cit. 2007-05-04) Dostupný z www: <http://www.ieee.org>

[7] Deering M., Geometry compression, Computer graphics (SIGGRAPH '95), 1995

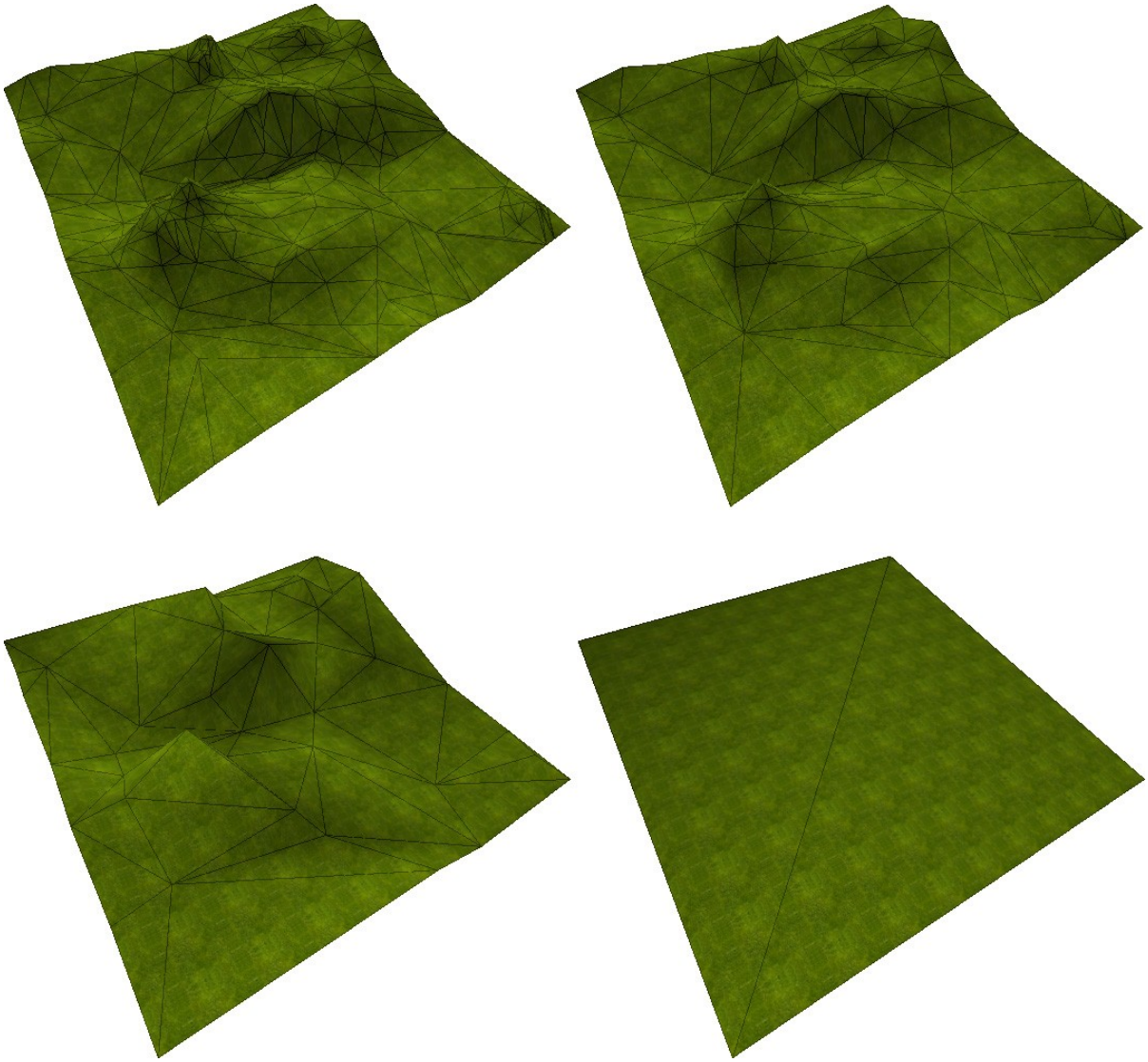
Seznam příloh

Příloha 1. Obrazové přílohy

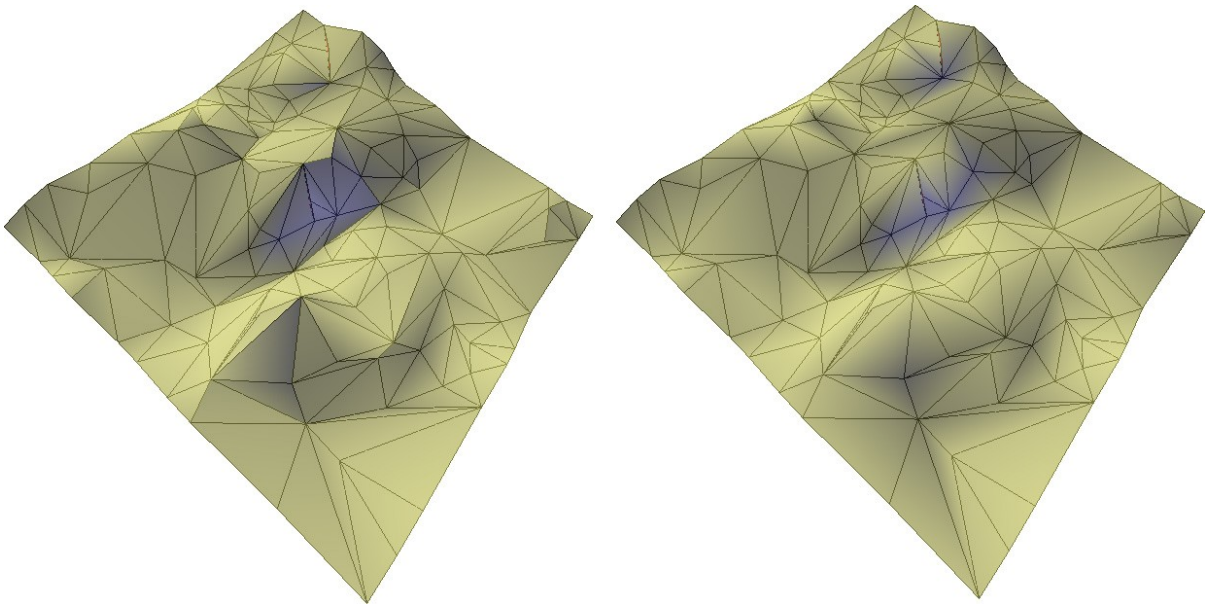
Příloha 2. Plakát

Příloha 3. CD s implementovaným algoritmem a manuálem

Obrazové přílohy



Obrázek 8. Ukázka Progressive mesh reprezentace na terénu. a) 425 polygonů, b) 215 polygonů, c) 70 polygonů a d) 2 polygony



Obrázek 9: Ukázka vypnutí/zapnutí zahlazení wedges v jednotlivých vertexech.

Obrázek 10: Ukázka geomorfu a) α=1.0, b) α=0.75, c) α=0.50, d) α=0.25 e) α=0.0