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

Diplomová práce

Generování stripů pro trojúhelníkové modely

Plzeň, 2002 Petr Vaněček

Abstract

Triangle Strips Generation

Triangle surface models are nowadays most often types of geometric objects description in computer graphics. Therefore, the problem of fast visualization of this type of data is often solved. One of popular approaches is stripification, i.e., a conversion of triangle surface into strips of triangles. This enables to reduce the rendering time by reducing data size and by avoiding redundant lighting and transformations computations.

Poděkování

Na tomto místě bych chtěl poděkovat všem, kteří mě při studiu podporovali a pomohli mi při překonání některých překážek. Zvláště bych chtěl poděkovat doc. Kolingerové, a to nejen za poskytnuté materiály a části kódu, ale i za její rady, poznámky a nápady.

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

V Plzni dne 1. května 2002 ....

Obsah

Seznam algoritmů

Seznam tabulek

Seznam obrázků

1 Úvod

1.1 Vizualizace dat

S počítačovou grafikou a virtuální realitou se v dnešní době můžeme setkat téměř všude – od novin a časopisů až po složité aplikace pro vývoj a výzkum nových technologií. Většina úloh počítačové grafiky se snaží vhodným způsobem převést a vizuálně interpretovat reálný svět tak, aby člověk – uživatel – dokázal této interpretaci snadno porozumět a vytěžit z ní co největší množství informací. Nejjednodušším příkladem je interpretace jednorozměrných dat zachycujících závislosti jedné veličiny v na druhé x pomocí dvourozměrných grafů (např. vývoj teploty během dne). Dvourozměrná data bývají často chápána jako body v rovině [x,y] s hodnotou v (např. nadmořská výška), u trojrozměrných dat pak navíc přibývá třetí souřadnice bodu (např. CT snímky)[1].

Protože počítačová aritmetika je diskrétní, jsou diskrétní i naměřené hodnoty, resp. body, ve kterých se měření provádí. Někdy stačí zobrazit pouze tyto diskrétní veličiny naměřené v diskrétních bodech, často se však chybějící body interpolují.

Pro vícerozměrná data se z výpočetních důvodů většinou provádí pouze po částech lineární interpolace, tedy proložení bodů nadrovinami. V případě dvourozměrných dat tedy dostáváme model složený z rovinných plošek – n-úhelníků. Protože obvykle leží v rovině pouze tři body, bývají pro zjednodušení datových struktur rozděleny všechny n-úhelníky na trojúhelníky. Takovýto model pak nazýváme plně triangulizovaný.

Protože jsou tyto trojúhelníkové sítě pro vizualizaci modelů používány nejčastěji, zabývá se tato práce metodami, jak jejich zobrazování urychlit. Je třeba poznamenat, že s rostoucím výkonem počítačů roste i složitost zobrazovaných modelů – počet trojúhelníků v modelu se dnes pohybuje řádově ve stovkách tisíců.

1.2 Základní myšlenka

Jak již bylo řečeno, trojúhelníkové sítě jsou nejčastěji užívanou reprezentací pro zobrazování povrchových modelů. Většina grafických akcelerátorů je navržena právě pro zobrazování těchto sítí. Limitujícím faktorem pro rychlost zobrazování již není rychlost výpočtů, ale šířka sběrnice – tedy rychlost, jakou jsou posílána data do akcelerátoru.

[1] Striktně vzato jsou tato data 3,5D (tři souřadnice a funkční hodnota). Podobně je tomu i ostatních uvedených příkladů.

Podobně jako u ostatních dat lze i ke snížení množství přenášených trojúhelníkových dat užít ztrátovou či neztrátovou kompresi. Protože grafické akcelerátory nepodporují klasickou datovou kompresi (LZW apod.), je třeba použít kompresi geometrickou.

Pod pojmem ztrátová komprese pak rozumíme takové techniky, které posílají do akcelerátoru nižší počet trojúhelníků za cenu snížení množství zobrazovaných detailů – redukce trojúhelníkových sítí, proměnný stupeň detailů[2], ...

Kompresí neztrátovou rozumíme takovou kompresi, která využívá toho, že v trojúhelníkových sítích existuje množství duplicitních hodnot (viz dále).

Poslední skupinou, která leží na pomezí těchto dvou, jsou takové algoritmy, které posílají do akcelerátoru pouze trojúhelníky v dané chvíli viditelné (využívají speciálních technik a struktur – BSP stromy, octrees, portálová technika, ...).

Všechny tři postupy je možné kombinovat. Tato práce se však zabývá pouze neztrátovými metodami.

1.3 Trojúhelníkové stripy

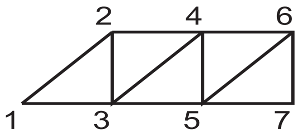
Pro vykreslování samostatných trojúhelníků potřebujeme do grafického akcelerátoru přenést tři vrcholy pro každý trojúhelník. Tento počet však můžeme minimalizovat speciálním uspořádáním trojúhelníků a to takovým, že každé dva po sobě následující trojúhelníky sdílejí jednu hranu. Použijeme-li toto uspořádání, nazývané *trojúhelníkový strip (triangle strip)*, vystačíme pouze se třemi vrcholy pro první trojúhelník a každý další trojúhelník je definován pouze jedním vrcholem.

V dnešní době jsou *stripy* podporovány většinou grafických akcelerátorů a grafických knihoven. Na obr.1.1 je příklad tzv. *sekvenčního stripu.* Každý trojúhelník je zde popsán třemi po sobě jdoucími vrcholy stripu, proto pro přenesení n trojúhelníků stačí pouze $n+2$ vrcholů místo původních $3n$.

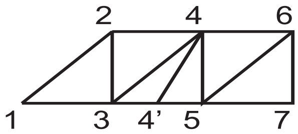
Situace však není vždy tak jednoduchá. Jak je vidět na obr. 1.2, v některých případech sousednost nedovolí použít sekvenční strip. Pro správné zakódování musíme rozdělit strip a vložit invertovanou hranu `4',4`. Tato invertující operace se nazývá *swap* a stripy, v nichž se swapy vyskytují, jsou označovány jako *obecné stripy.*

Operace *swap* je především u starších grafických knihoven (např. Irix-GL) implementována jako jednobitový atribut. Při navrhování knihovny OpenGL byla však

[2] Level-Of-Detail, LOD



Obr. 1.1: Sekvenční strip.
Pořadí vrcholů: 1,2,3,4,5,6,7



Obr. 1.2: Obecný strip.
Pořadí vrcholů: 1,2,3,4,4’,swap/4,5,6,7

tato instrukce vypuštěna z důvodu snadnější přenositelnosti. Swapy se zde nahrazují tzv. prázdnými trojúhelníky (v obrázku 1.2 je to trojúhelník `4,4’,4`). I když takto implementovaná operace zvýší počet vrcholů o jeden, je stále výhodnější než začátek nového stripu, který by zvýšil počet vrcholů o dva.

Z výše uvedených faktů je patrné, že tento způsob komprese může přinést značné urychlení. Protože je však výpočet optimální množiny stripů NP-úplný problém, je potřeba použít heuristický postup. Každá metoda stripifikace má tedy své výhody a nevýhody.

Jedním z kriterií pro výběr správného algoritmu může být i kvalita stripifikace. Kvalitou můžeme rozumět např. nízký počet stripů, nízký počet swapů, ale též rozlehlost stripu (důležité při používání LOD) atd.

V kapitole 2 je uveden přehled existujících algoritmů.

Cílem této práce je navrhnout, implementovat a porovnat stripifikační metodu, určenou pro statické, plně triangulizované sítě – trojúhelníkové sítě, u kterých je počet a rozložení trojúhelníků neměnný.

1.4 Názvosloví a definice

V této kapitole je uveden stručný přehled některých pojmů, které se v textu vyskytují. Přestože čeština je velmi bohatý jazyk a všechny pojmy by bylo možné přeložit, v některých případech tak neučiním, a to především z důvodu snažší čitelnosti textu (pro některé pojmy není český překlad ustálen).

DAG Orientovaný acyklický graf.

Fan Též trojúhelníkový fan. Speciální uspořádání množiny trojúhelníků, v němž všechny trojúhelníky sdílejí jeden vrchol a každé dva po sobě jdoucí trojúhelníky sdílejí jednu hranu. Fan je pak reprezentován množinou vrcholů. Český překlad není ustálený – v některých textech je použit překlad: trojúhelníkový vějíř.

Frame rate Též frames per second (FPS). Veličina udávající rychlost vykreslování scény ve snímcích za vteřinu. Pro zdání plynulého pohybu by tato veličina měla nabývat hodnot vyšších než 25.

Greedy (algorithm) Též hladový, hltavý, žravý (algoritmus). Zpracovává vždy maximální možný blok dat. K jednou zpracovaným datům se už nevrací.

Hladový (algoritmus) viz Greedy.

Strip Též trojúhelníkový strip. Speciální uspořádání skupiny trojúhelníků takové, že každé dva po sobě jdoucí trojúhelníky sdílejí hranu. Strip je pak reprezentován množinou vrcholů. Český překlad není bohužel ustálený – v některých textech jsou použity překlady: trojúhelníkový pás, trojúhelníkový pruh.

Stripifikace Proces, při kterém se z množiny trojúhelníků vytváří jeden nebo více stripů.

Dále uvádím několik definic, tvrzení a jejich důkazů, souvisejících s problémem stripifikace. Část definic pochází z grafové teorie – jedná se však pouze o základní definice a k porozumění textu není důležitá hlubší znalost této problematiky.

Definice 1 *Graf G (obyčejný neorientovaný) je uspořádaná dvojice (V,E), kde V je množina vrcholů a* $E \subseteq \binom{V}{2} = x, y : x, y \in V, x \neq y$.

Definice 2 *Graf G nazveme souvislý, pokud pro každé dva jeho vrcholy x,y existuje sled začínající v x a končící v y.*

Definice 3 *Sledem rozumíme posloupnost* $x_0, l_1, x_1, l_2, \ldots, l_n, x_n$, *kde* $x_i \in V$, $l_i = \{x_{i_1}, x_i\} \in E$.

Definice 4 *Podgraf* $G' = (V', E')$ *grafu* $G = (V, E) : G'$ *je graf,* $V' \subseteq V, E' \subseteq E$.

Definice 5 *Cesta (délky n) v grafu je podgraf:* $V = \{1, \ldots, n+1\}$,$E = \{\{i, i+1\}$;$i = 1, \ldots, n\}$.

Definice 6 *Kružnice (délky n) v grafu je podgraf:* $V = \{1, \ldots, n\}$,$E = \{\{i, i+1\}$;$i = 1, \ldots, n-1\} \bigcup \{1, n\}$.

Definice 7 *Stromem rozumíme souvislý graf bez kružnic.*

Definice 8 *Kostra grafu je strom, který je podgrafem se stejnou množinou vrcholů.*

Definice 9 *Hamiltonovský sled v grafu G je takový sled, který obsahuje všechny vrcholy grafu G.*

Pozn. Předchozí definice definuje i hamiltonovskou cestu a kružnici, protože obě jsou speciálním případem hamiltonovského sledu.

Definice 10 *Duálním grafem nebo také grafem sousednosti triangulace S nazveme takový graf G, jehož vrcholy jsou trojúhelníky S a jeho hrany spojují trojúhelníky, které spolu v triangulaci S sousedí.*

Definice 11 *Triangulace S je spojitá, je-li její duální graf souvislý.*

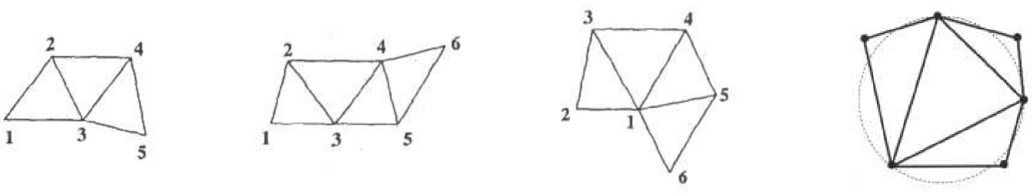
Definice 12 *Triangulace S je hamiltonovská, obsahuje-li její duální graf hamiltonovskou cestu. Hamiltonovskou triangulaci S nazveme sekvenční, jestliže žádné tři po sobě jdoucí hrany protnuté hamiltonovskou cestou nesdílejí společný vrchol.*

Definice 13 *Označme $C_s(n)$ funkci, která udává nejvyšší počet vrcholů určujících sekvenční strip o n trojúhelnících. Podobně označme $C'_s(n)$ funkci, která udává nejvyšší počet vrcholů určujících obecný strip, a konečně $C_f(n)$ funkci, která udává nejvyšší počet vrcholů určujících trojúhelníkový fan.*

Tvrzení 1 *Libovolná (spojitá) triangulace S skládající se z přesně tří trojúhelníků je sekvenční; tedy $C_s(3) = 5$.*

Důkaz. Je zřejmý z obrázku 1.3 a).

□



a) b) c) d)

Obr. 1.3: Důkazy

Tvrzení 2 *Nechť S je hamiltonovská triangulace se čtyřmi trojúhelníky. Potom S je buď sekvenční strip nebo dva stripy s maximální délkou $C_s(4) = 8$ vrcholů, nebo obecný strip s maximální délkou $C'_s(4) = 7$ vrcholů, nebo trojúhelníkový fan s maximální délkou $C_f(4) = 6$ vrcholů.*

Důkaz. Topologie S musí být stejná jako na obr.1.3 b). resp. obr.1.3 c). První případ lze reprezentovat jedním sekvenčním stripem o šesti vrcholech. Druhý případ je možné reprezentovat vějířem s šesti vrcholy, obecným stripem se sedmi vrcholy nebo dvěma sekvenčními stripy s osmi vrcholy.

□

Tvrzení 3 *Libovolnou hamiltonovskou triangulaci o n trojúhelnících lze reprezentovat pomocí sekvence stripů s celkovým počtem nejvýše $n + 2\lceil n/3 \rceil$ vrcholů. Při použití obecného stripu a fanu je možno tuto triangulaci reprezentovat pomocí nejvýše $n + 2\lceil n/4 \rceil$ vrcholů. Takovou stripifikaci je možné vytvořit v čase $O(n)$.*

Důkaz. Mějme hamiltonovskou triangulaci S o n trojúhelnících a obsahující hamiltonovskou cestu Π. Hladovou metodou lze nalézt nejdelší sekvenční část Π' cesty Π, a to se složitostí $O(\Pi')$. Z tvrzení 1 víme, že $|\Pi'| \geq 3$. S použitím hladového algoritmu lze tedy dekomponovat celou triangulaci na nejvýše $\lceil n/3 \rceil$ stripů. Výsledná stripifikace obsahuje tedy $n + 2\lceil n/3 \rceil$ vrcholů. Podobně lze tento postup použít u obecných stripů a fanů, a celou síť je možné dekomponovat na nejvýše $\lceil n/4 \rceil$ stripů.

□

Tvrzení 4 *Pro libovolnou (spojitou) triangulaci S lze se složitostí $O(n)$ vypočítat stripifikaci obsahující nejvýše $2n + 1$ vrcholů. Při použití obecných stripů a fanů dostaneme hranici $\lceil (7/4)n \rceil + 1$.*

Důkaz. Jak bylo dokázáno v [XIA99], lze libovolný binární strom rozdělit na množinu cest, ve které se všechny cesty, až na jednu, skládají z nejméně tří uzlů. S použitím tohoto algoritmu tedy můžeme rozdělit duální graf stripifikace S na jednotlivé cesty. Vezměme jednu takovou cestu Π' o délce i trojúuhelníků. Z předchozích tvrzení pak víme, že libovolnou hamiltonovskou triangulaci lze pokrýt $i + 2\lceil i/3 \rceil \leq 2i$. Součtem jednotlivých cest pak dostaneme výsledné omezení. Podobným způsobem můžeme postupovat při povolení obecných stripů a fanů.

□

Definice 14 *Triangulaci S nazveme Delaunayovou triangulací, pokud všechny trojúhelníky $t \in S$ splňují Delaunayovu podmínku (t.j. pro všechny trojúhelníky $t \in S$ platí, že uvnitř kružnice opsané tomuto trojúhelníku neleží žádný další vrchol).*

Tvrzení 5 *Delaunayova triangulace není nutně hamiltonovská.*

Důkaz. Na obrázku 1.3 d) je znázorněna triangulace, která splňuje Delaunayovu podmínku. Tuto triangulaci však není možné pokrýt jedním stripem – a tedy není hamiltonvská.

$\square$

2 Metody

V této kapitole jsou popsány některé ze stávajících stripifikačních metod. Jednotlivá porovnání jsou prováděna pouze na základě výsledků prezentovaných autory jednotlivých metod. Vzhledem k tomu, že většina autorů používá jako porovnávací standard metodu $STRIPE$, která má vyšší časovou složitost než je $O(N)$, je téměř nemožné porovnat časy jednotlivých metod (výsledky jsou u různých autorů měřeny na různých počítačích).

Stripfikační algoritmy lze dělit do dvou hlavních skupin. První skupina by mohla být nazvána „přímá“. Algoritmy z této skupiny využívají ke konstrukci stripů přímo vlastnosti trojúhelníkové sítě (trojúhelníky, vrcholy, sousednosti,...). Druhá skupina algoritmů pak využívá tzv. *duální reprezentace* trojúhelníkové sítě. Síť je nahrazená grafem, jehož vrcholy jsou duální reprezentací trojúhelníků a hrany jsou duální reprezentací sousednosti. Existuje i několik algoritmů, využívajících odlišné principy, a nelze je do těchto skupin zařadit.

2.1 Přímé metody

SGI

Jednou z prvních metod pro vytváření trojúhelníkových stripů byla metoda navržená v roce 1990 v SGI [SGI90]. Tato metoda vychází z jednoduché myšlenky – při vytváření stripu co nejméně rozdělit zbytek sítě a nevytvářet tak izolované trojúhelníky. Metoda je postavena na *hladovém algoritmu.*

Algoritmus nejprve vybere z trojúhelníkové sítě trojúhelník s nejmenším počtem sousedů (v tomto kroku je zapotřebí použít pomocnou strukturu – prioritní frontu, seznamy, tabulky,... – aby vyhledání probíhalo se složitostí $O(1)$, v opačném případě roste složitost celého algoritmu k $O(N^2)$). Tento trojúhelník je vložen na začátek nového stripu a je vyjmut z triangulace. Dále algoritmus vybírá vždy takový sousedící trojúhelník, jehož počet sousedů je nejmenší. Pokud takových sousedů existuje víc, vybere se ten, jehož soused má nejmenší počet sousedů. Tento trojúhelník je pak přidán do aktuálního stripu a vyjmut z triangulace. Pokud nelze najít žádný sousedící trojúhelník, vytvoří se stejným způsobem nový strip. Algoritmus končí v okamžiku, kdy v triangulaci už není žádný trojúhelník.

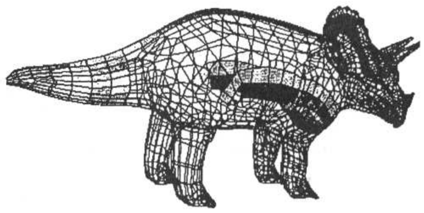
Protože tento algoritmus je poměrně jednoduchý a poskytuje dobré výsledky, je často používán jako referenční metoda. Protože je však špatně dostupný[3], provedl

[3]Za jeho sehnání bych chtěl poděkovat ing. Chlumskému z LPS. Program jsem však obdržel až po naimplementování algoritmu a k dalším měřením jsem ho zatím nepoužil.

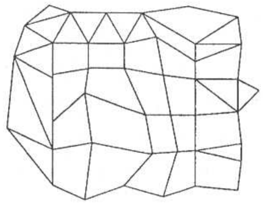
jsem jeho vlastní implementaci (viz kap. 3). Algoritmus je určen pro statické plně triangulizované modely.

STRIPE

V roce 1996 byl publikován další algoritmus částečně založený na SGI metodě – STRIPE [EVA96a]. Tento algoritmus vychází z předpokladu, že většina modelů není plně triangulizovaná[4] a obsahuje i mnoho lichoběžníků (obr. 2.1), které jsou uspořádány do větších spojitých oblastí – plátů/patchů (obr.2.2).



Obr. 2.1: Částečně triangulizovaný model (převzato z [EVA96a])



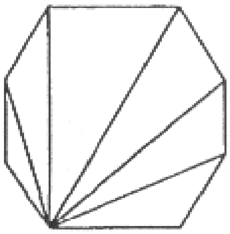
Obr. 2.2: Příklad záplaty (převzato z [EVA96a])

Algoritmus je dvoukrokový. V prvním průchodu (nazývaném *globální*) se vyhledají pláty, které mohou být stripifikovány buď po řádcích/sloupcích, nebo jako celé pláty. Při použití řádkové/sloupcové stripifikace se plát rozdělí a stripifikuje jedním *sekvenčním stripem* podle řádky či sloupce (podle toho, ve kterém směru vzniknou delší stripy). Tento způsob minimalizuje počet potřebných swapů. V případě celoplátové stripifikace se plát převede na obecný strip (strip prochází esovitě plátem, přičemž na každý obrat připadají tři swapy). Podobně jako u SGI algoritmu jsou pak stripifikované části vyřazeny ze sítě (v tomto případě se nejedná o triangulaci).

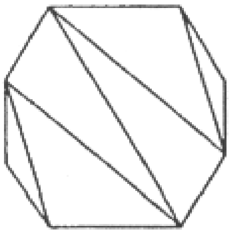
Druhý průchod (*lokální*) pak triangulizuje zbylé mnohoúhelníky a ty jsou stripifikovány algoritmem založeným na SGI metodě. Pro triangulizaci mnohoúhelníků se namísto obvyklejší a jednodušší vějířovité triangulace (obr. 2.3 a))používá sekvenční triangulace (obr. 2.3 b)), která ulehčuje stripifikaci.

Algoritmus je určen pro statické, částečně triangulizované modely. Z našich i v literatuře dostupných měření vyplývá, že metoda neumožňuje práci s většími datovými

[4] Tento předpoklad však není zcela pravdivý.



a) Vějířovitá triangulace



b) Sekvenční triangulace

Obr. 2.3: Příklady triangulace n-úhelníku (převzato z [EVA96a])

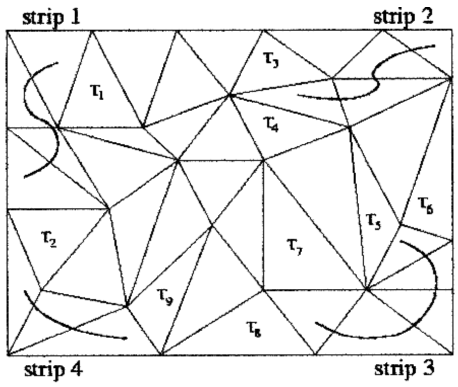
modely (max. 70 000 mnohoúhelníků). Tento algoritmus je, jako jeden z mála, volně přístupný, a je tedy i přes své nedostatky často používán jako srovnávací metoda. V porovnání s SGI metodou dává lepší výsledky co do počtu stripů, je však mnohem pomalejší (čas u $STRIP$u roste kvadraticky!).

Pseudoparalelní stripifikace

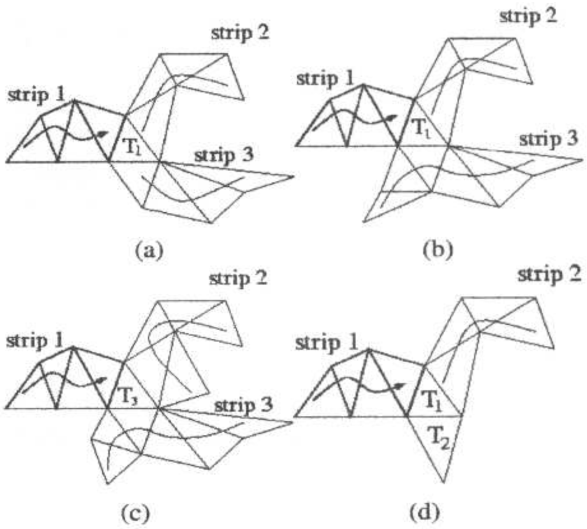
Jednou z nejnovějších metod je metoda publikovaná na WSCG'02 [SIL02]. Vychází z původní SGI metody, výběr dalšího trojúhelníku se však při shodném počtu sousedů neřídí počtem sousedů případného následovníka, ale tím, zda by vznikl swap či nikoliv.

Poměrně zajímavá je však skutečnost, že algoritmus vytváří několik stripů současně (nejedná se však o paralelní algoritmus – trojúhelníky jsou přidávány ke stripům postupně; obr. 2.4). Tím vzniká množství situací, kdy přidání dalšího trojúhelníku má za následek spojení dvou již existujících stripů. Toto spojování je pak prováděno podle následujících pravidel (obr. 2.5): a) pokud si lze vybrat mezi dvěma stripy, připojí se ten, při kterém nevznikne swap (v obr.2.5 a) tedy strip 2), b) existuje-li pouze jedna varianta, stripy se spojí (v obr.2.5 b) stripy 1 a 2), c) není-li žádná možnost, jak připojit další strip, připojí se pouze trojúhelník (v obr.2.5 c) tedy pouze trojúhelník T1), d) jestliže lze připojit buď strip nebo trojúhelník, který nemá sousedy, připojí se trojúhelník, aby nevznikl strip délky 1 (v obr.2.5 d) se připojí trojúhelník T2).

Algoritmus je stejně jako SGI určen pouze pro statické, plně triangulizované modely. Počet stripů i počet vrcholů je nižší než s použitím $STRIP$u. Časová složitost je lineární.



Obr. 2.4: Příklad vícenásobného startu (převzato z [SIL02])



Obr. 2.5: Podmínky pro spojování stripů (převzato z [SIL02])

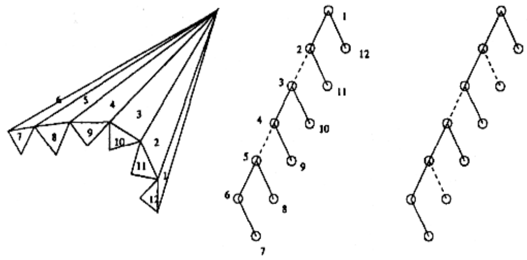
2.2 Duální metody

Xiang

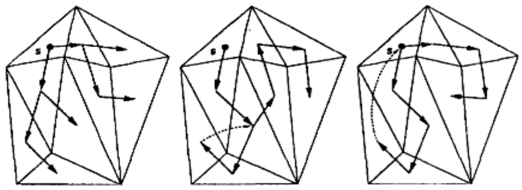
Významným zástupcem této skupiny algoritmů je algoritmus publikovaný v [XIA99]. Algoritmus nejprve (v případě potřeby) provede plnou triangulizaci sítě. Poté je nalezena kostra duálního grafu, která je rozdělena na cesty. Tyto cesty – duální reprezentace pásu trojúhelníků – jsou dále hladovým algoritmem dekomponovány na trojúhelníkové stripy, které mohou být v závěrečné fázi pospojovány. Na obrázku 2.6 je znázorněna triangulace, nalezená kostra a dva způsoby, jak tuto kostru rozdělit na cesty. Plnou čarou jsou vyznačeny jednotlivé cesty (stripy) a přerušovanou čarou hrany, které nejsou uvnitř stripu.

Druhý krok – tedy nalezení kostry grafu – má na výslednou stripifikaci největší vliv. Kostru grafu lze nalézt pomocí algoritmů pro prohledávání stromu – prohledávání do šířky, prohledávání do hloubky nebo hybridní varianta, kdy hledání probíhá podobným způsobem jako prohledávání do hloubky, s tím rozdílem, že se algoritmus navrací k nejvyššímu ještě ne plně prozkoumanému uzlu (obr. 2.7). U prohledávání do šířky (levý obrázek) je jasně patrná tendence vytvářet uzly stupně tři.

Důležitým požadavkem na kostru grafu pro účely stripifikace je, aby obsahovala co nejméně uzlů stupně tři. V takovém uzlu totiž musí dojít k rozdělení stripu. Prohledávání do šířky je tedy pro tento účel poměrně nevýhodné, neboť má tendenci vytvářet vyvážený strom, kde je vysoký počet těchto uzlů. Dalším důležitým faktorem je způsob vybírání dalšího uzlu (trojúhelníku). Nejjednodušší možností je náhodný výběr. Další možností je použití heuristické metody. První metoda je shodná s metodou použitou v SGI (kritérium počtu sousedů). Druhou možností je vybírání takového trojúhelníku, který nezpůsobí swap (to je možné při uchování znalosti o vstupní hraně).



Obr. 2.6: Rozdělení kostry grafu (převzato z [XIA99])

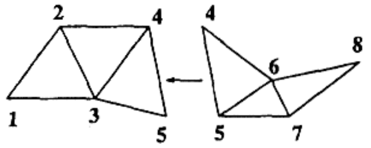


Obr. 2.7: Metody prohledávání zleva: do šířky, do hloubky, hybridně (převzato z [XIA99])

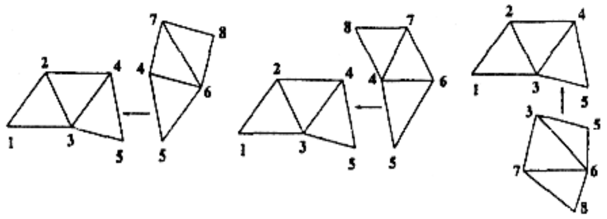
Aby při rozdělování kostry grafu na cesty vznikl co nejmenší počet cest (a tedy i stripů), je v metodě užit speciální algoritmus založený na dynamickém programování. Ten dokáže v lineárním čase vypočítat optimální dekompozici (bližší detaily je možné nalézt v [XIA99]).

V závěrečné fázi se algoritmus pokouší spojit jednotlivé stripy do delších celků. Každým takovým spojením se ušetří dva (obr. 2.8 a)), v horším případě jeden vrchol (obr. 2.8 b) – spojením těchto dvou stripů vznikne swap). Existují také další čtyři případy, při kterých je možné stripy spojit, ovšem bez ušetření vrcholu. Speciálním případem jsou pak stripy délky jedna, které je možné libovolně připojovat ke koncům stripů (je možné cyklicky permutovat pořadí vrcholů). Algoritmus však zaručuje pouze lokální optimalizaci stripifikace (pro spojení dvou stripů). Při hledání globálního optima záleží na pořadí spojování stripů a nalezení tohoto optima tedy není triviální problém.

Algoritmus je určen pro statické, částečně triangulizované modely (mohou obsahovat i nekonvexní mnohoúhelníky). Počet vrcholů je přibližně o 10-15% nižší než u $STRIP$u. Čas potřebný pro výpočet je mnohem nižší než u $STRIP$u a přibližně dvakrát až třikrát nižší než u SGI.



a) Ušetření dvou vrcholů



b) Ušetření jednoho vrcholu

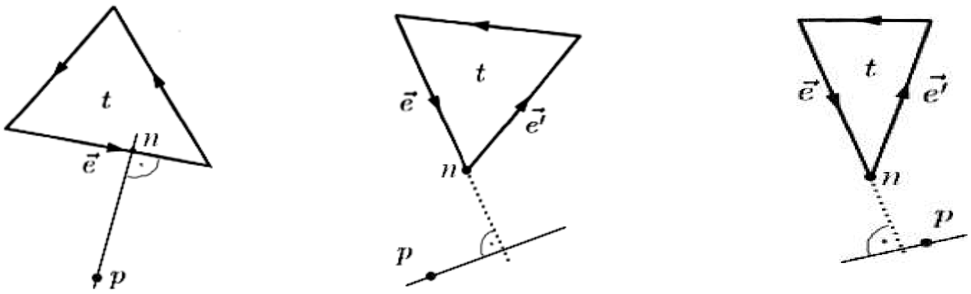
Obr. 2.8: Spojování dvou stripů (převzato z [XIA99])

Stripifikace pro terénní data

Speciálně pro terénní data, která jsou často reprezentována jako triangulizované nepravidelné sítě (TIN), byl vyvinut poměrně jednoduchý a rychlý algoritmus založený na duální reprezentaci [SPE97]. Kostra grafu, která je následně použita pro vytvoření stripifikace, je konstruována pomocí následujícího algoritmu.

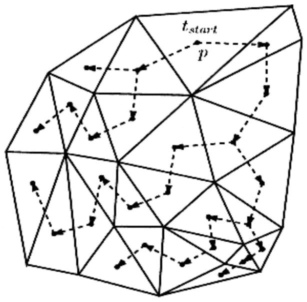
Pro každý trojúhelník triangulace T je definován *předchůdce*, a vytváří se tak orientovaný graf $G(V, E)$, kde $V = \{t|t \in T\}, E = \{(t', t)|t' \textit{ je předchůdce } t\}$. Graf G je ve skutečnosti orientovaný strom, jehož kořenem je startovací trojúhelník t_{start}. Relace *předchůdce* je definována následovně: Nejprve se určí libovolný startovací bod p a startovací trojúhelník t_{start}, který tento bod obsahuje. Pro všechny trojúhelníky $t \in T - t_{start}$ se určí bod n trojúhelníku t, jehož eukleidovská vzdálenost k bodu p je nejmenší. Jestliže bod n leží na hraně e trojúhelníku t (obr. 2.9 a)), potom předchůdce tohoto trojúhelníku je druhý trojúhelník obsahující hranu e. V opačném případě je nejbližším bodem některý z vrcholů. Je-li e hrana před a e' hrana za vrcholem (při levotočivé orientaci), pak předchůdce trojúhelníku t je buď trojúhelník obsahující hranu e – v případě, že se p nachází napravo od e (obr. 2.9 b) – nebo trojúhelník obsahující hranu e' v opačném případě (obr. 2.9 c)).

Takto vzniklý strom (2.10) má tu výhodu, že se v něm poměrně pravidelně vyskytují sekvence střídajících se levých a pravých synů. Prohledáváním do šířky se z tohoto stromu vytváří stripy, a to tak, že nový strip začne, kdykoliv je tato střídající se posloupnost porušena. Po vytvoření stripů je možné dodatečně pospojovat stripy (především stripy obsahující pouze jeden trojúhelník). Na obr.2.11) je připojení těchto stripů naznačeno tečkovanou čarou. Další redukce stripů lze dosáhnout povolením obecných stripů.

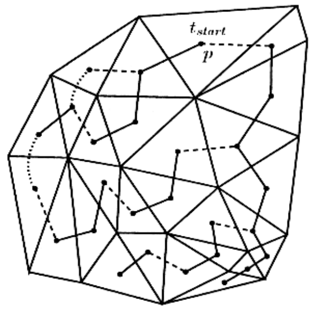


a) b) c)

Obr. 2.9: Možné polohy nejbližšího bodu (převzato z [SPE97])



Obr. 2.10: Strom duálního grafu (převzato z [SPE97])



Obr. 2.11: Spojení stripů — existující stripy … spojení stripů (převzato z [SPE97])

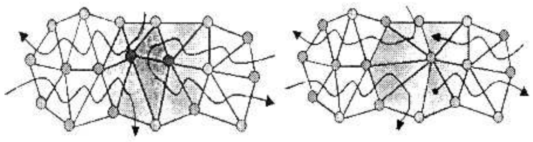
Algoritmus je určen pro plně triangulizované modely. Počet stripů je přibližně čtyřikrát vyšší než s použitím $STRIP$u, počet vrcholů je však téměř shodný (o 3-5% více)[5]. Časová složitost je lineární a mnohem nižší než u $STRIP$u.

Progresivní stripifikace

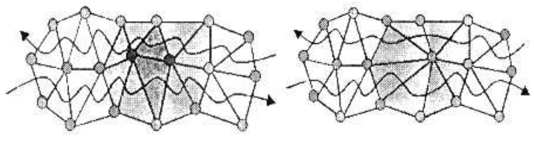
Algoritmus uveřejněný v [BEL01] je určený pro tzv. progresivní modely (tedy modely, které umožňují zobrazit nejprve hrubý model a teprve dalším upřesněním získat model původní - tato metoda je vhodná např. pro zobrazování přes webové rozhraní). Toto zjednodušování modelu se děje pomocí kontrakce nejméně významných hran modelu. To může mít u stripifikací vzniklých jinými metodami za následek přerušení stripu,

[5]To je způsobeno tím, že u STRIPu jsou použity obecné stripy, zatímco zde se jedná o stripy sekvenční – tedy bez swapů

jak je vidět na obr. 2.12. Při vypuštění hrany se původní tři stripy (vlevo) rozpadnou na čtyři (vpravo). Algoritmus pro vytváření stripů používá opět duální graf a jeho kostru. Kostra je však v tomto případě vytvářena jako maximální kostra z váženého grafu, kde váhu jednotlivých uzlů určuje důležitost hran. To zaručuje, že hrany, které budou rušeny nejdříve, nejsou vnitřní hranou stripu. Na obr. 2.13 jsou původní dva stripy (vlevo) zachovány i po redukci (vpravo).



Obr. 2.12: Špatná stripifikace



Obr. 2.13: Dobrá stripifikace ([BEL01])

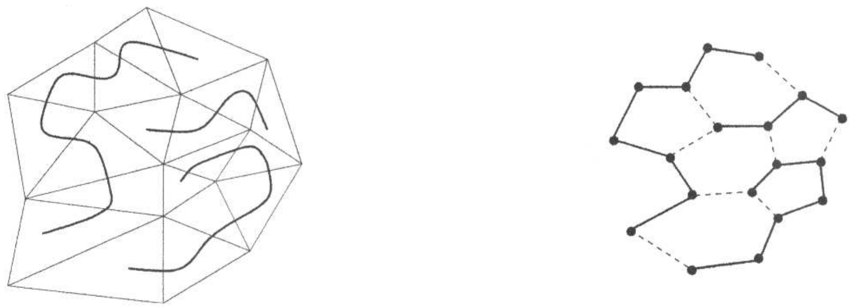
Pro nalezení kostry grafu je v metodě použit převzatý algoritmus, který pracuje s váženým grafem a zaručuje, že součet vah v nalezeném stromě je minimální. Váhy jednotlivých hran jsou určeny podle důležitosti jednotlivých hran – hrany, jejichž kontrakce způsobí minimální, chybu mají nejvyšší ohodnocení a opačně.

Algoritmus je určen pro plně triangulizované modely. Výsledky jsou horší než u metody *STRIPE*, ovšem při zjednodušování modelu zůstává velká část stripů zachována. Přestože časy nejsou autory přímo uváděny, je pravděpodobné, že časová složitost bude vzhledem ke způsobu konstrukce lineární.

Tunelování

Algoritmus tunelování [STE01] definuje a používá tzv. *tunelovací operátor*, který spojuje dva konce stripů, a tím redukuje jejich počet o jeden. Tunelovací operátor lze chápat jako jedno z pravidel gramatiky pro transformaci grafu a bylo by zřejmě možné přidat další pravidla, která by byla schopná podobným způsobem vytvářet stripy s nějakou konkrétní vlastností. Stávající tunelovací operátor vytváří pouze obecné stripy.

V duální reprezentaci stripifikovaného modelu existují dva typy hran: **stripifikované**, které odpovídají hraně spojující dva sousedící trojúhelníky ve stripu, a **nestripifikované**. Na obr. 2.14 a) je znázorněn příklad stripifikace a na obr.2.14 b)



a) Model b) Duální reprezentace

Obr. 2.14: Příklad stripifikace a druhy hran v duálním grafu (převzato z [STE01])

je znázorněna její duální reprezentace. Stripifikované hrany jsou znázorněny plnou čarou a hrany nestripifikované přerušovanou čarou.

Tunel je posloupnost procházející střídavě stripifikovanou a nestripifikovanou hranou, začínající a končící nestripifikovanou hranou a spojující dva koncové uzly stripů. Jestliže takový tunel v duální reprezentaci existuje, pak celkový počet stripů může být snížen o jeden výměnou stripifikovaných hran za nestripifikované a opačně (obr.2.15 a) a 2.15 b)).

a) Před aplikací operátoru b) Po aplikaci operátoru

Obr. 2.15: Příklad tunelu (převzato z [STE01])

Tunelovací algoritmus je opakován (pokaždé z jiného startovacího bodu), dokud lze v triangulaci najít tunel. Nelze-li již další tunel najít, bylo dosaženo *lokálního* minima. Vzhledem k tomu, že nalezení optimální stripifikace je NP-úplný problém, není jisté, že nalezené minimum je zároveň *globální* (stripifikace navíc závisí i na pořadí startovacích bodů).

Algoritmus je určen pro statické, LOD i progresivní trojúhelníkové sítě. Výsledná stripifikace je mnohem lepší než u ostatních metod, tomu ovšem odpovídá i několikanásobně delší doba výpočtu. Tu lze částečně snížit omezením maximální délky tunelu. Toto omezení má samozřejmě negativní vliv na výslednou stripifikaci, ale ukazuje se, že při vhodném nastavení tohoto parametru dostaneme jen nepatrně horší stripifikaci, ale několikanásobně rychleji.

2.3 Jiné přístupy

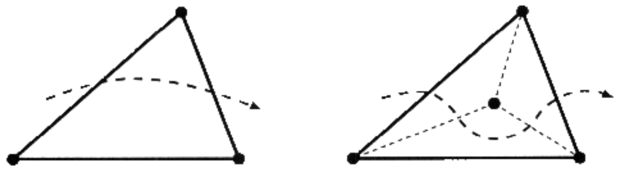
Hamiltonovská triangulace

Zcela odlišné metody publikoval v roce 1994 E.M.Arkin [ARK94, ARK96]. Problematika stripifikace je zde převedena na duální problém, tj. hledání hamiltonovské cesty v grafu. Metody však nehledají stripifikaci, ale vytvářejí z množiny vrcholů hamiltonovskou triangulaci (triangulaci obsahující hamiltonovskou cestu), tedy triangulaci, kterou je možné reprezentovat jediným stripem.

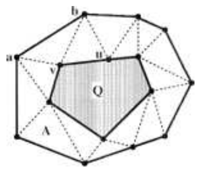
Pro obecný strip jsou navrženy hned dva způsoby triangulace. Při metodě **vkládání** se začne tím, že se do již existující konvexní obálky vloží bod a vytvoří se taková triangulace, v níž každý trojúhelník obsahuje tento bod. V takto vzniklé triangulaci se vytvoří strip a u každého trojúhelníku se uloží vstupní a výstupní hrana. Dále se postupně vkládají zbývající vrcholy. Každý nově vložený vrchol rozdělí trojúhelník, ve kterém leží, na tři nové trojúhelníky, přičemž lze vždy zachovat nepřerušený strip (obr. 2.16).

Tzv. **cibulová** metoda vypočítá nejprve z množiny bodů konvexní obálku a dále konvexní obálku ze zbylých bodů. Následně se provede triangulace takto vzniklého prstence (obr. 2.17). Z množiny bodů se pak odečtou body ležící na vnější konvexní obálce a celý proces se opakuje, dokud není vytvořena kompletní trojúhelníková síť. Výsledný strip pak prochází spirálovitě jednotlivými vrstvami této triangulace. Touto metodou vznikají pravidelnější trojúhelníky než u metody vkládání (obr. 2.18 a) a 2.18 b)).

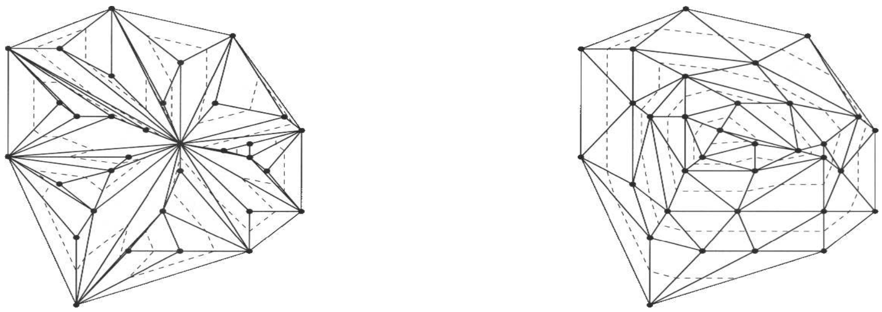
Metody jsou určeny pro netriangulizované množiny bodů a pouze pro rovinné případy. Výsledná stripifikace obsahuje jediný strip, triangulace však nemusí splňovat žádnou geometrickou podmínku zaručující její kvalitu. Časová složitost triangulací je $O(NlogN)$ (strip se vytváří při konstrukci sítě).



Obr. 2.16: Obnovení hamiltonovské triangulace po vložení vrcholu (převzato z [ARK96])



Obr. 2.17: Průchod vrstvy cibulovou metodou (převzato z [ARK96])



a) Metoda vkládání

b) Cibulová metoda

Obr. 2.18: Příklady hamiltonovské triangulace (převzato z [ARK96])

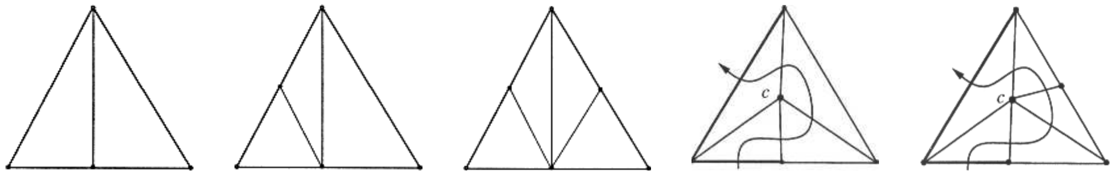
Hierarchické stripy

Poslední zde uvedenou metodou je metoda navržená v [VEL99]. Podobně jako [BEL01] je i tato metoda určena pro sítě s proměnným stupněm detailů. Zde však nedochází k redukci sítě, ale naopak k jejímu postupnému zjemňování[6].

Je zde navrženo celkem pět různých šablon, podle kterých lze rozdělit trojúhelník tak, aby byla zachována vstupní a výstupní hrana stripu. Těchto pět šablon (obr.2.19) a jejich různá vnitřní uspořádání by měly zajistit zachování stripů při libovolném zjemňování modelu (při zjemňování se nové body nacházejí na hranách stávajících trojúhelníků).

Protože se jedná spíše o teoretickou práci, nejsou v ní uvedeny žádné konkrétní výsledky, takže jakékoliv srovnání s jinými metodami je nemožné.

[6]Z anglického názvu Mesh Refinement



a) b) c) d) e)

Obr. 2.19: Šablony pro dělení trojúhelníku (převzato z [VEL99])

2.4 Shrnutí

Z výše uvedených metod jsem k dalšímu prozkoumání vybral původní SGI metodu, a to z několika důvodů. Prvním důvodem je fakt, že tato metoda je spolu se *STRIPE* nejčastěji používanou metodou pro srovnání. Dalším důvodem je poměrně jednoduchá implementace této metody a s ní spojená snadná modifikovatelnost (úprava heuristické funkce pro výběr dalšího trojúhelníku). Navíc lze tuto metodu použít i pro jiná než terénní data.

Na podnět vedoucí DP jsem se též zabýval algoritmem vytvářejícím stripifikaci při tvorbě Delaunayovy triangulace. Pro výpočet Delaunayovy triangulace jsem použil program doc. Kolingerové, který jsem dále modifikoval. Poměrně jednoduchou úpravou v programu bylo možné vytvořit i program pro vytvoření hamiltonovské triangulace (vkládání).

Porovnání většiny výše uvedených a vlastních metod je uvedeno v tab.4.14 a 4.15.

3 Implementace

Většina kódu byla implementována ve vývojovém prostředí Borland Delphi. Důvodem volby je moje dobrá znalost programovacího jazyku Pascal. Výjimku tvořily pouze úpravy programu STRIPE, napsaného v ANSI C, nutné pro získávání dat ze souborů v TRI formátu [ZCU99][7].

3.1 Datové struktury

Dříve než se budu věnovat jednotlivým implementovaným metodám, rád bych také poukázal na důležitost vhodných datových struktur. Všechny dále implementované programy využívají pro zobrazování výstupních dat knihovnu OpenGL, která dnes patří mezi nejčastěji používané. Pro zobrazování trojúhelníkových sítí lze v OpenGL využít dvou odlišných přístupů. U prvního (klasického) přístupu se do grafického akcelerátoru posílají jednotlivé trojúhelníky jako množina tří vrcholů o třech souřadnicích (alg.3.1). Pro n trojúhelníků je tedy počet přenesených dat roven $n{*}3{*}3 = 9{*}n$ (počet trojúhelníků * počet vrcholů * počet souřadnic).

```
glBegin(GL_TRIANGLES);
  for i:= 0 to pocet_trojuhelniku do
      for j:=0 to 2 do
          glVertex3f(trojuhelnik[i].vrchol[j].x,
                     trojuhelnik[i].vrchol[j].y,
                     trojuhelnik[i].vrchol[j].z);
glEnd;
```

Alg. 3.1: Příklad klasického zobrazování trojúhelníků

Touto metodou se však u obvyklých trojúhelníkových sítí posílá většina vrcholů do grafického akcelerátoru vícekrát. První cesta ke snížení množství přenášených dat je tedy najít způsob, jak těmto duplicitám zabránit. Ve většině sítí, a to jak rovinných tak prostorových, platí, že stupeň vrcholu je přibližně šest. S tím souvisí i fakt, že počet vrcholů je přibližně dvakrát nižší než počet trojúhelníků. Od OpenGL verze 1.2 je možné pomocí tzv. vertex array – tedy pole vrcholů – poslat do akcelerátoru nejprve souřadnice všech vrcholů a u jednotlivých trojúhelníků pak místo souřadnic vrcholů posílat pouze index do již existující tabulky vrcholů. Pro n trojúhelníků (a tedy přibližně $n/2$ vrcholů) je množství přenášených dat přibližně $3 * n/2 + n * 3 = 9/2 * n$

[7] Tuto možnost považuji za jednodušší než vytváření programu pro konverzi dat z formátu TRI do nativního formátu STRIPU (OBJ) či opačně.

(počet vrcholů * počet souřadnic + počet trojúhelníků * počet vrcholů – index do tabulky). Množství dat je tedy proti ve srovnání s původní metodou poloviční, musíme však dodržet určité podmínky. Hlavní podmínka je, že vrcholy musí být uloženy v souvislém bloku paměti, a to ve tvaru $[x1y1z1(w1)][x2y2z2(w2)]$.... To může přinést problémy při přidávání nových vrcholů – realokace souvislého bloku paměti. Aby se tento způsob zobrazování maximálně využil, je výhodné mít uloženy v poli i definice jednotlivých trojúhelníků. Celá datová struktura je naznačena v alg.3.2.

```
TVertex = array[0..2] of glDouble;     // vrchol
AVertex = array of TVertex;            // pole vrcholu

TTriangle = array[0..2] of glInt;      // trojuhelnik
ATriangle = array of TTriangle;        // pole trojuhelniku

TStrip    = record                     // strip
              len    : integer;        // delka stripu
              col    : TColor;         // barva stripu
              vertex : array of glInt; // vrcholy
            end;
AStrip    = array of TStrip;           // pole stripu
```

Alg. 3.2: Datová struktura

Vlastní vykreslování se pak provede pouhým předáním ukazatelů na jednotlivá pole (alg.3.3). Normálové vektory a souřadnice pro mapování textur se předávají podobným způsobem.

```
glEnableClientState(GL_VERTEX_ARRAY);             // povoleni vertex arrays
glVertexPointer(3, GL_DOUBLE, 0, @vrchol);        // naplneni pole vrcholu
glDrawElements(GL_TRIANGLES, 3*(pocet_trojuhelniku),
               GL_UNSIGNED_INT, @trojuhelnik[0]); // naplneni pole trojuhelniku
glDisableClientState(GL_VERTEX_ARRAY);            // zakazani vertex arrays
```

Alg. 3.3: Příklad zobrazování trojúhelníků pomocí pole vrcholů.

Toto rozšíření lze kombinovat i s trojúhelníkovými stripy. Celkové urychlení oproti klasické metodě je pak, zvláště u větších modelů, značné (tab.4.2).

3.2 Implementace SGI

SGI algoritmus je založen na velmi jednoduché hladové metodě, která se snaží zahrnovat do stripu ty trojúhelníky, které mají nízký počet sousedů. Tím se snaží zamezit tomu, aby vznikaly izolované trojúhelníky – stripy délky jedna.

Původní SGI algoritmus byl navržen pro Iris GL, kde je příznak swapu indikován jedním bitem. Proto tento algoritmus generuje obecné stripy a nesnaží se omezit počet swapů. Jak jsem již zmínil v první kapitole, u OpenGL je každý swap emulován prázdným trojúhelníkem, a tedy jedním vrcholem navíc. Celý postup je možné popsat algoritmem 3.4.

1. Jestliže triangulace obsahuje alespoň jeden trojúhelník, pokračuj.
2. Nalezni trojúhelník t s nejmenším početem sousedů (v případě shody vyber libovolný).
3. Začni nový strip.
4. Vlož trojúhelník t do stripu a vyjmi jej z triangulace.
5. Jestliže neexistuje žádný soused trojúhelníku t, pokračuj krokem 1. V opačném případě vyber trojúhelník t' sousedící s t, a to takový, který má nejmenší počet sousedů. V případě shody vyber ten, jehož soused má nejmenší počet sousedů. Je-li stále shoda, vyber libovolně.
6. $t \leftarrow t'$. Pokračuj krokem 4.

Alg. 3.4: Základní SGI algoritmus

Z hlediska efektivity algoritmu je nejkritičtější druhý krok. Nejnižší možná složitost algoritmu je $O(n)$, kde n je počet trojúhelníků. Jestliže je druhý krok implementován jako sekvenční procházení, vzroste celková složitost na $O(n \cdot s)$, kde s je počet stripů. Aby se tomuto nárůstu zamezilo, musí se použít pomocná struktura, která vyhledání startovacího trojúhelníku urychlí.

V originální verzi je podle dostupných informací použita prioritní fronta. V mé implementaci je použita kombinace pole a obousměrně zřetězeného seznamu (alg.3.5). Dynamické pole *sousede* obsahuje jednotlivé prvky seznamu – pořadí prvků je shodné s pořadím trojúhelníků: první prvek odpovídá prvnímu trojúhelníku, druhý druhému, atd. Proměnné *vyrazeny* a *sousedu* představují hlavičky pěti různých front. Ve frontě *vyrazeny* je seznam trojúhelníků, které již byly vyřazeny z triangulace. Fronty $sousedu[0]..sousedu[3]$ obsahují trojúhelníky, které mají daný počet sousedů. Na začátku algoritmu jsou jednotlivé trojúhelníky zařazeny podle počtu sousedů do odpovídajících front. Vyhledání trojúhelníku s nejmenším počtem sousedů je pak provedeno se složitostí $O(1)$ (vybere se první trojúhelník z první nenulové fronty).

Udržování tohoto seznamu probíhá rovněž se složitostí $O(1)$: Je-li trojúhelník s indexem i vyřazen z triangulace, vyjme se prvek $sousede[i]$ z fronty (je jedno, ve které se nachází) a vloží se za hlavičku fronty *vyrazene*. Dále se všichni sousedé trojúhelníku i přesunou podobným způsobem do odpovídajících front (podle aktuálního počtu sousedů – nyní mají o jednoho souseda méně).

```
type
  PMyList = ^TMyList;                // ukazatel na obousmerny seznam
  TMyList = record                   // obousmerny seznam
              Prev : PMyList;        // predchozi
              Next : PMylist;        // nasledujici
            end;
  AMyList = array of TMyList;        // pole ukazatelu

var
  sousede  : AMyList;                // pole prvku seznamu
  vyrazeny : TMyList;                // hlava seznamu vyrazenych trojuhelniku
  sousedu  : array[0..3] of TMyList; // hlavy seznamu trojuhelniku
                                     // podle poctu sousedu
```

Alg. 3.5: Struktura pro vyhledávání podle počtu sousedů

Silný vliv na dobu trvání algoritmu má také způsob alokace a dealokace paměti. Ačkoliv se tento problém zdá být nepodstatný, je možné nesprávným zacházením s pamětí velmi zpomalit algoritmus anebo dokonce zvednout jeho složitost o řád (potřebujeme-li např. spojitý blok paměti a alokujeme jej po malých částech, musí při dalším požadavku na paměť systém vždy naalokovat větší spojitý kus a již existující data zkopírovat; proto se vyplatí naalokovat více paměti a v případě potřeby ji uvolnit). V příkladu kódu (alg.3.6) lze nastavováním proměnné *predpokladana_delka* zvýšit, popřípadě snížit dobu běhu programu. Optimální je naalokovat potřebné místo přesně – alokace většího množství paměti trvá déle, při naalokování menšího množství musíme alokaci zopakovat a v případě potřeby překopírovat obsah paměti.

```
while podminka do begin
 ...
 inc(delka_stripu[j]);                    // alokace nove pameti
 if delka_stripu[j] mod predpokladana_delka = 0 then
    setlength(strip[j],delka_stripu[j]+predpokladana_delka);
 ...
end;
```

Alg. 3.6: Příklad alokace paměti

O kvalitě výsledné stripifikace se rozhoduje hlavně v pátém kroku algoritmu – heuristické funkci pro výběr dalšího trojúhelníku. Tuto funkci jsem se pokusil nahradit několika modifikacemi, a zjistit tak chování různých typů stripifikací. Následující výčet ukazuje všechny testované možnosti.

1. Náhodný výběr dalšího trojúhelníku (dále RN).

2. Výběr trojúhelníku s nejmenším počtem sousedů, v případě shody se vybere náhodně (dále LNLR).

3. Výběr trojúhelníku, který netvoří swap (dále LS).

4. Výběr trojúhelníku s nejmenším počtem sousedů, v případě shody se vybere trojúhelník, jehož soused má nejmenší počet sousedů (původní heuristika - dále LNLN).

5. Výběr trojúhelníku s nejmenším počtem sousedů, v případě shody se vybere trojúhelník, jehož zařazením nevznikne swap (dále LNLS).

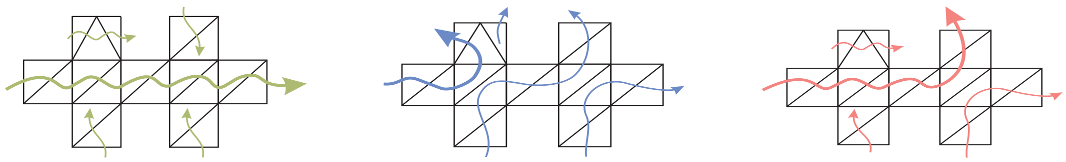
První modifikace vybírá další trojúhelník zcela náhodně (vybere prvního možného souseda – záleží tedy na pořadí, v jakém byly zadávány vrcholy, resp. sousedé). Měla by přinést určité urychlení, protože zde odpadá složité testování. Výsledná stripifikace by pak měla být horší než ostatní případy.

Druhá modifikace by už měla vytvářet kvalitnější stripifikaci, protože výběr je zde cílený. Další trojúhelník je podobně jako u původní SGI verze vybírán podle počtu sousedů, ovšem v případě shody se vybere náhodně (první možný).

Třetí modifikace je založena na zcela jiné podmínce. Jako další trojúhelník se vybere buď jediný existující nebo ten, jehož zařazení do stripu nezpůsobí swap. Tato metoda by měla poskytovat lepší výsledky pro zobrazování v OpenGL. Počet stripů by měl být vyšší, ovšem počet vrcholů by se měl snížit. Na obr.3.1 a) je vidět první strip, který prochází sítí bez vytvoření swapu. Na pokrytí celé sítě je za potřebí 5 stripů a 29 vrcholů.

Čtvrtá modifikace je původní funkce. Při vytváření stripu nerozlišuje mezi obecným a sekvenčním stripem. Na obr.3.1 b) vytvořil první strip swap už při první možné příležitosti – protože počet sousedů byl shodný, rozhodl nejnižší počet následujících sousedů. Počet stripů je sice nižší než v předchozím případě (pouze 4 stripy), počet vrcholů je však 35.

Poslední modifikace se snaží spojit obě varianty - tedy co nejméně dělit stávající stripifikaci a vybírat takové trojúhelníky, které nebudou vytvářet swap. Další trojúhelník tedy vybírá podle počtu sousedů a v případě shody vezme ten, který nevytváří swap. Na obr.3.1 c) je vidět, že u prvního stripu se při rovnosti počtu sousedů uplatnila podmínka nevytvářet strip a ve druhém případě rozhodl nižší počet sousedů. Počet stripů je stejný jako u předchozí metody (4 stripy), ale počet vrcholů je o něco nižší – 31.



a) LS modifikace b) LNLN funkce c) LNLS modifikace

Obr. 3.1: Příklad stripifikace pomocí různých modifikací základního algoritmu (první vytvořený strip je vyznačen silnější čarou)

Poslední problém, který je třeba vyřešit, je dekompozice seznamu trojúhelníků na strip. Začátek stripu tvoří tři vrcholy prvního trojúhelníku, uspořádané tak, aby při připojení následujícího trojúhelníku nevznikl swap (tedy aby druhý a třetí vrchol tvořil společnou hranu prvního a druhého trojúhelníku). Pak postupně zpracováváme jednotlivé trojúhelníky. Najdeme vrchol aktuálního trojúhelníku, který ještě není ve stripu (tedy vrchol, který netvoří společnou hranu aktuálního a posledního již zpracovaného trojúhelníku). Pokud nalezený vrchol tvoří se dvěma posledními vrcholy stripu aktuální trojúhelník, je vše v pořádku a můžeme tento vrchol zařadit do stripu a pokračovat ve zpracování dalšího trojúhelníku. V opačném případě je třeba vytvořit swap a teprve pak vložit tento vrchol. Celý postup je naznačen v alg.3.7. Funkce $OpacnyVrchol$ vrací vrchol, který je vrcholem aktuálního trojúhelníku a netvoří společnou hranu aktuálního a předchozího trojúhelníku. Funkce $JeTrojuhelnik$ vrací hodnotu $PRAVDA$ v případě, že poslední dva vrcholy stripu a nově vkládaný vrchol tvoří aktuální trojúhelník.

```
novy_vrchol=OpacnyVrchol(trojuhelnik[aktualni-1],
                         trojuhelnik[aktualni]);
if not JeTrojuhelnik(trojuhelnik[aktualni],    // vrcholy netvori trojuhelnik
   novy_vrchol, strip[delka-1], strip[delka-2])then begin
   strip[delka]=strip[delka-1];           // proved swap
   strip[delka-1]=strip[delka-3];
   inc(delka);
end;
strip[delka]=novy_vrchol;                 // vloz novy vrchol
inc(delka);
inc(aktualni);
```

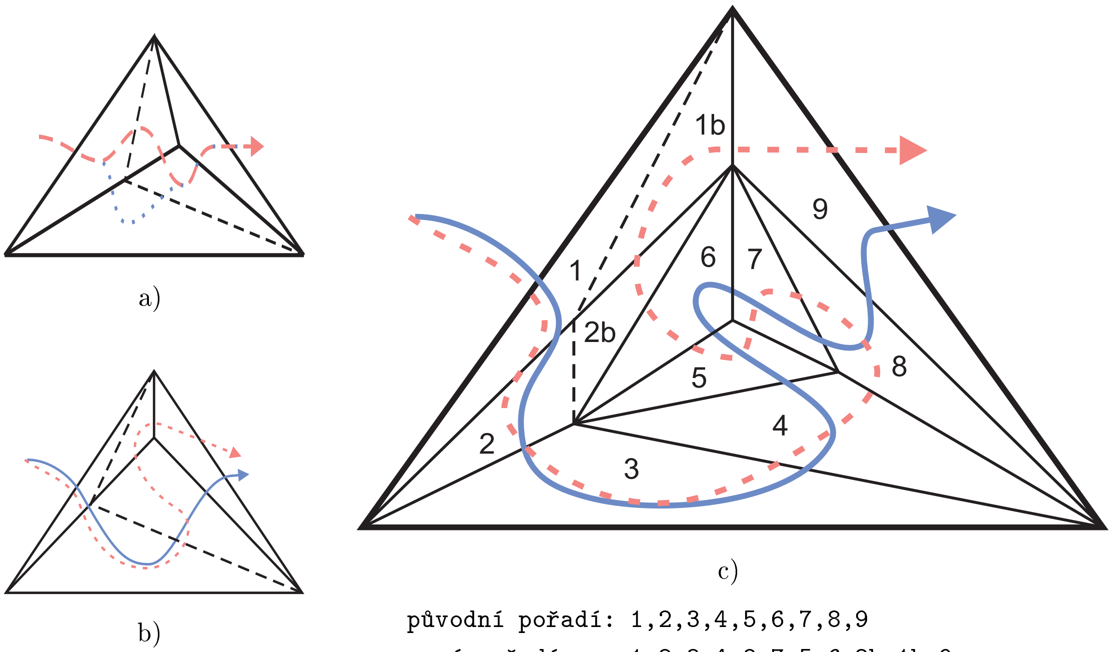
Alg. 3.7: Dekompozice seznamu trojúhelníků na strip

3.3 Hamiltonovská triangulace

Protože nový stripifikační algoritmus popsaný v kap.3.4 je založen na Delauneyově triangulaci vytvářené vkládací metodou, rozhodl jsem se nejprve implementovat triangulaci hamiltonovskou. Ta se totiž při metodě vkládání liší od delaunayovské „pouze" tím, že nekontroluje zda nově vkládané hrany splňují nějakou podmínku. Vlastní implementace této metody vychází z původní Delaunayovy triangulace implementované doc. Kolingerovou ([ZCU02]). Tato implementace je založena na inkrementálním vkládání vrcholů a pro rychlé vyhledávání používá strukturu nazvanou DAG (orientovaný acyklický graf).

Základní myšlenka hamiltonovské triangulace je popsána v kapitole 2.3. Co však v této kapitole ani v původním článku [ARK96] není, je způsob, jak se vypořádat s případem, kdy nově vkládaný vrchol leží na již existující hraně trojúhelníku. Řešení jsou dvě. První „pštrosí" řešení je zacházet s tímto vrcholem jako s kterýmkoliv jiným. Tím však vzniká trojúhelník s nulovým obsahem (což by samo o sobě nemuselo vadit), u něhož může dojít k problému s orientací. To může při dalších operacích se sítí již nějaký problém způsobit. Ve druhém řešení je daná hrana bodem rozdělena a vznikají tak z původních dvou trojúhelníků trojúhelníky čtyři.

U druhého řešení pak mohou nastat dva případy. V prvním případě (obr.3.2), kdy vstupní hrana prvního trojúhelníku a výstupní hrana druhého trojúhelníku nemá společný bod, vzniknou vložením nového vrcholu dva nové trojúhelníky, a to takové, že jeden z nich není možné začlenit do stripu a je třeba založit strip nový (na obrázku jsou přerušovanou a tečkovanou čarou naznačeny dvě možnosti jak vytvořit strip). U tohoto případu se nabízí ještě jedno řešení – do stripu nejprve zapojí všechny čtyři nově vzniklé trojúhelníky a pak se pokračuje zbylým trojúhelníkem (obr.3.2 b); plnou čarou je vyznačen původní strip před vložením vrcholu a přerušovanou čarou nově vzniklý strip). U komplikovanější trojúhelníkové sítě to však s sebou přináší značné změny ve stripifikaci – a to i u trojúhelníků, které s nově vzniklými trojúhelníky nesousedí (obr.3.2 c)). To je způsobeno tím, že změna vstupní a výstupní hrany vyvolá změny v celé vnořené hierarchii. V obr.3.2 c) existuje ještě jedna možná cesta, která by, narozdíl od předchozí, nevyvolala žádné další změny. Tato cesta by nejprve prošla všechny čtyři nové trojúhelníky, a poté by pokračovala zbytkem stripu (pořadí: `1,1b,2,2b,3,...`). Tento postup však nelze aplikovat na situaci na obrázku 3.2 b). Může také nastat situace, Detekce těchto situací je tedy – vzhledem k množství možností, které mohou nastat – poměrně náročná, a proto se jí pravděpodobně nevyplatí zabývat.

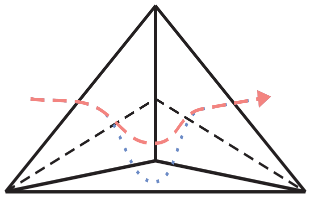


`původní pořadí: 1,2,3,4,5,6,7,8,9`
`nové pořadí:    1,2,3,4,8,7,5,6,2b,1b,9`

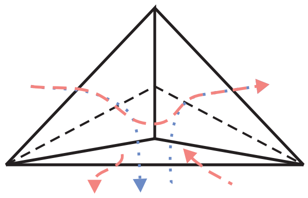
Obr. 3.2: První případ při vkládání vrcholu

Ve druhém případě se může stát, že lze vzniklé trojúhelníky jednoduchým způsobem pospojovat. Často však dochází k situaci, která je naznačena na obrázku 3.3 a). Zde můžeme postupovat dvěma způsoby. Buď do stripu zapojíme všechny čtyři trojúhelníky a starý strip tím přerušíme (přerušovaná čára). Nebo se na vzniklou situaci díváme odděleně a každý trojúhelník vložíme do stávajícího stripu (tečkovaná čára).

Situace navíc může být zkomplikována tím, že strip vstupující do spodního trojúhelníku nemusí být shodný se stripem, který z něj vystupuje (obr.3.3 b)).



a)



b)

Obr. 3.3: Druhý případ při vkládání vrcholu

On-line metoda

První a složitější implementace vytváří trojúhelníkový strip přímo při vytváření triangulace. U tohoto přístupu vzniká problém, pokud se povolí nově vkládaným vrcholům, které leží na některé z hran, aby ji rozdělily. Z předchozí sekce víme, že tímto vložením může vzniknout nový strip, ale vzhledem k lokálnímu přístupu je tento strip složen pouze z izolovaného trojúhelníku a není možnost jej spojit s jiným trojúhelníkem (strip se prodlouží pouze v případě, že je do tohoto trojúhelníku vložen další bod). Z tohoto důvodu je v mé implementaci takovýto způsob vkládání vrcholu zakázán, a vrchol se vkládá stejným postupem jako kterýkoliv jiný.

Do původního programu byly dodány datové typy TTriStrip a TTriStripElem (alg.3.8). V TTriStrip jsou uloženy informace o jednotlivých stripech (jednotlivé trojúhelníky, délka, začátek,...). Jednotlivé prvky stripu (trojúhelníky) pak obsahují především informace o vstupní a výstupní hraně a ukazatele na předchůdce a následovníka. Rozsáhlejší modifikace programu jsou pouze ve funkci *Subdivide_triangle* (rozdělení trojúhelníku po vložení vrcholu), kde se do již existujícího stripu vloží nové elementy (podobně jako v obrázku 2.16). Dekompozice seznamu trojúhelníku na vrcholy stripu je provedena až po dokončení triangulizace (alg.3.7).

```
PTriStripElem = ^TTriStripElem;
TTriStripElem = record                // defininice struktury pro tristrip
  ine,oute    : NVector2;             // vstupni a vystupni hrana trojuhelniku
  pred,succ   : PTriStripElem;        // predchudce a naslednik
  DAG         : PTDAG;                // odkaz zpet do DAGu
end;

TTriStrip = record                    // struktura pro stripy
  strip   : array of TTriStripElem;   // pole vsech trojuhelniku
  start   : array of TIndex;          // indexy na startovaci trojuhelniky
  len     : array of integer;         // delky stripu
  no,last : integer;                  // pocet a posledni pozity strip
end;
```

Alg. 3.8: Přidané datové struktury.

DAG

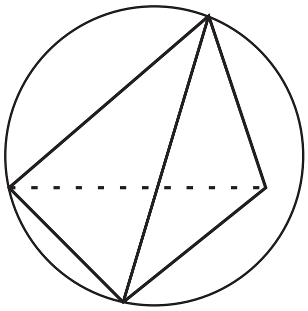
Druhý, elegantnější způsob, provede stripifikaci až po dokončení triangulace. Využívá při tom orientovaný acyklický graf (DAG), který během triangulace vznikl a který obsahuje ve svých listech jednotlivé trojúhelníky. Při vytváření stripifikace pak stačí postupně projít všechny listy a vytvářet průběžně stripy, které lze v případě hamiltonovské triangulace spojit v jeden. V tomto případě je modifikována především procedura *Extract_triangulation*, ve které se z DAG získává i samotná triangulace. Případné problémy se spojitostí stripu, vzniklé vložením vrcholu na hranu, jsou zde vyřešeny zcela automaticky – pokud je to možné, stripy se samy spojí.

V programu jsou zavedeny podobné struktury jako v případě On-line metody. Poněkud problematickou věcí je zde spojování jednotlivých stripů. Toto spojení může být v podstatě dvojího typu. První případ nastává, když se má spojit konec jednoho stripu se začátkem jiného stripu. V tomto případě musíme u jednotlivých prvků připojovaného stripu změnit ukazatele na nový počátek stripu. Druhý případ je o něco složitější – jedná se buď o spojení dvou konců nebo dvou začátků stripů. Pro tuto operaci by bylo nejvhodnější zavést nový datový typ, který by měl vlastnosti obousměrně spojeného seznamu, ovšem s možností otočit směr (a to i uvnitř tohoto seznamu).

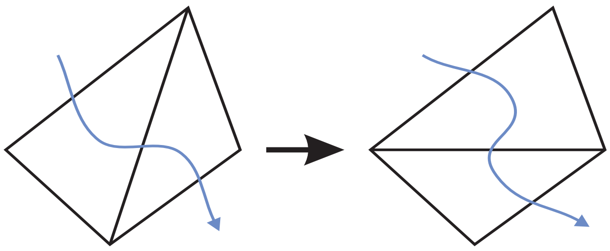
3.4 Delaunayovská triangulace a stripy

Protože trojúhelníková síť, vzniklá hamiltonovskou triangulací, obsahuje trojúhelníky, které mají často u některého z vrcholů velmi malý úhel, není tato síť vhodná pro další aplikace – měření, decimace, atp. Zavedením delaunayovské podmínky (pro všechny trojúhelníky platí, že uvnitř kružnice opsané tomuto trojúhelníku neleží žádný další vrchol) dostaneme síť, ve které jsou minimální úhly trojúhelníků maximalizovány. U trojúhelníku, který nesplňuje tuto podmínku, je nalezen vrchol (trojúhelník), který to způsobil, a dojde k prohození společné hrany těchto dvou trojúhelníků (obr.3.4).

Touto výměnou hran může opět dojít k několika různým situacím, které ovlivní průběh stripu. První situace (obr.3.5) nastává v okamžiku, kdy spolu vstupní hrana prvního trojúhelníku a výstupní hrana druhého trojúhelníku nesousedí. Případná změna hran nemá v tomto případě žádný vliv na strip. Vlevo je původní triangulace a strip, který ji pokrývá, vpravo je triangulace splňující Delaunayovu podmínku, pokrytá shodným stripem.

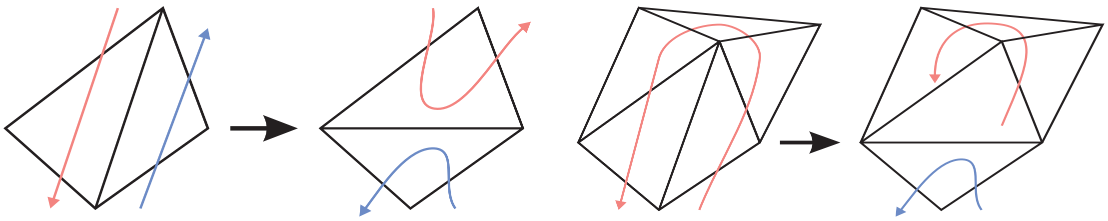


Obr. 3.4: Splnění delaunayovské podmínky



Obr. 3.5: První případ při prohození hran

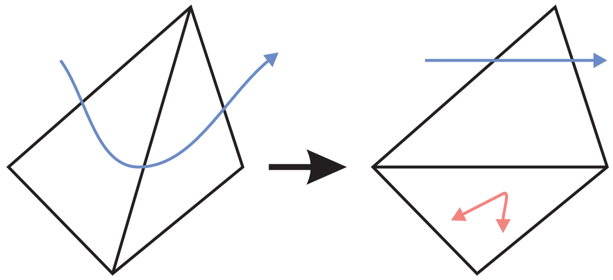
Další situace, při které by počet stripů měl zůstat zachován, je na obrázku 3.6 a). Zde prochází oběma trojúhelníky dva oddělené stripy. Při prohození hrany se pouze napojí zbývající část prvního stripu na strip druhý a opačně (tj. prohodí se části obou stripů). Problém zde nastává v situaci, kdy se jedná o tentýž strip. V tom případě dojde k jeho rozdělení na dva nové stripy (obr.3.6 b)).



a) b)

Obr. 3.6: Druhý případ při prohození hran

Poslední možností, která zvýší počet stripů, je případ, kdy spolu sousedí vstupní hrana prvního a výstupní hrana druhého trojúhelníku (obr.3.7). V tomto případě vzniká izolovaný trojúhelník – tedy strip délky jedna.



Obr. 3.7: Třetí případ při prohození hran

Přidáme-li tyto případy k případům spojeným s vkládáním vrcholu (viz. hamiltonovská triangulace), dostáváme poměrně složitý soubor podmínek, které je třeba dodržovat a na které je třeba v průběhu stripifikace/triangulace reagovat. Protože zde dochází i ke spojování stripů, byl by celý proces poměrně pomalý. Z tohoto důvodu jsem implementoval pouze metodu založenou na využití acyklického grafu (DAG). Samotná stripifikace se tedy vytváří až v závěrečné fázi triangulace (procedura *Extract_triangulation*), kdy se procházením listů DAGu získá výsledná triangulace.

Pro zlepšení výsledků stripifikace jsou navíc upraveny i procedury pro vkládání vrcholu, a to tak, aby byly listy do stromu zařazeny v takovém pořadí, aby neporušily strip.

Závěrečná dekompozice seznamu trojúhelníků na vrcholy stripů pak probíhá podle již známého algoritmu 3.7.

4 Testy a výsledky

Všechny programy uvedené v části 3 byly implementovány ve vývojovém prostředí Borland Delphi. Testy byly prováděny na počítači s procesorem AMD Duron 850MHz s 256MB paměti, grafickou kartou GeForce 2 MX a operačním systémem Windows 2000. Veškeré grafické výstupy byly prováděny pomocí knihovny OpenGL.

4.1 Výhodnost stripu

První prováděné testy se netýkají samotných stripifikačních algoritmů, ale výhodnosti použití trojúhelníkových stripů, rozšíření knihovny OpenGL a kombinace těchto metod. Tyto testy byly prováděny na uměle vytvořených datech, aby bylo možné zajistit pokrytí všech trojúhelníků jediným stripem, a to i pro velká data. Jednalo se o rovinnou čtvercovou mřížku, kterou strip esovitě procházel (za cenu 3 swapů při každé otočce).Výsledky těchto měření jsou ukázány v tabulkách 4.1 a 4.2. Tabulka 4.1 ukazuje počet snímku za vteřinu a tabulka 4.2 poměr počtu snímků u testované metody k počtu snímků u klasické metody.

počet trojúhelníků	5k	19k	44k	79k	124k	179k	244k	318k	403k
klasická	138	56.2	28.7	18.7	12	8.5	6.2	5.2	4.2
rozšíření	124	97.7	53.5	33	21	15.5	11.5	9.7	7.7
strip	111	75.5	72.5	45.5	30.5	22.7	17	14.2	11.5
rozšíření + strip	99.5	142	82.5	66.7	45.2	39.7	31.2	26.3	22

Tab. 4.1: Porovnání rychlosti jednotlivých zobrazovacích metod
(Počet zobrazených snímků za vteřinu).

počet trojúhelníků	5k	19k	44k	79k	124k	179k	244k	318k	403k
klasická	1	1	1	1	1	1	1	1	1
rozšíření	0.90	1.74	1.86	1.76	1.75	1.82	1.86	1.86	1.84
strip	0.80	1.34	2.53	2.43	2.54	2.67	2.76	2.73	2.76
rozšíření + strip	0.72	2.53	2.87	3.57	3.77	4.67	5.06	5.06	5.28

Tab. 4.2: Porovnání rychlosti jednotlivých zobrazovacích metod
(poměr počtu zobrazovaných snímků měřenou a klasickou metodou).

Jak již bylo napsáno v kapitole 3, klasickou metodou se rozumí přenášení a zobrazování jednotlivých trojúhelníků. Rozšířená metoda využívá možnost zobrazování trojúhelníků pomocí indexů do pole vrcholů. Zobrazování pomocí stripů je prováděno podobně jako u klasické metody a kombinace rozšíření a stripů je obdobné jako u algoritmu 3.3. Z tabulek je patrné, že práce vložená do procesu stripifikace se vrátí v podobě rychlejšího zobrazování. Urychlení při použití pole vrcholů se blíží teoretické hranici 2.0 (pro velká data je urychlení přibližně 1.85). Rozdíl je pravděpodobně způsoben trojúhelníky umístěnými na hranici sítě, kde neplatí skutečnost, že každý bod je použit šestkrát. Při použití trojúhelníkových stripů se vykreslování urychlí přibližně 2.7 krát. Tato hodnota je o něco nižší než očekávané teoretické maximum (< 3). Kombinace obou metod pak přináší urychlení, které je blízké očekávanému výsledku (tedy urychlení stripů * urychlení rozšíření). Toto měření také potvrzuje teorii, že rychlost je více omezována propustností sběrnice než samotnými výpočty.

4.2 SGI metoda

Implementace SGI metody byla prověřena řadou měření, a to především na množině asi 25 modelů z [STA02, LAR02, ZCU02]. Tyto modely byly vybrány také z toho důvodu, že jsou používány i jinými autory, a je tedy možné porovnávat s nimi dosažené výsledky. V této sekci budou též pro srovnání uváděny výsledky dosažené metodou $STRIPE$. Časy vstupně/výstupních operací byly ze zřejmých důvodů vyřazeny z měření.

Tabulka 4.3 uvádí informace o jednotlivých modelech, které byly při testech použity. Některá další měření jsou uveřejněna ve [VAN02].

čís.	model	poč. vrch.	poč. troj.	čís.	model	poč. vrch.	poč. troj.
1	sphere	146	288	6	bell	213373	426572
2	elipsoid	2452	4900	7	hand	327323	654666
3	cow	2905	5804	8	dragon	437645	871414
4	demi	9138	17506	9	happy	543652	1087716
5	bunny	35947	69451		buddha		

Tab. 4.3: Množina testovacích dat

Následující tabulky uvádí porovnání všech pěti modifikací SGI algoritmu, které byly navrženy v kapitole 3, a algoritmu $STRIPE$. Pro připomenutí nejprve uvádím název a stručný popis jednotlivých metod (tab.4.4).

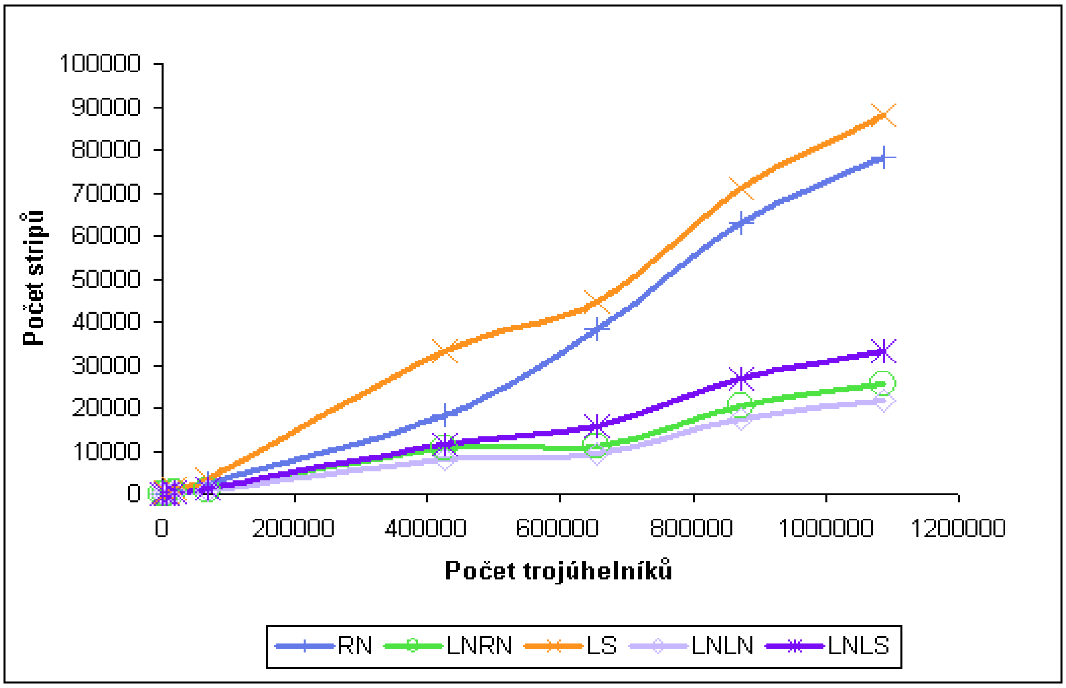
RN	Náhodný výběr
LNRN	Výběr podle počtu sousedů, při shodě náhodně
LS	Výběr podle swapů
LNLN	Výběr podle počtu sousedů, při shodě podle počtu sousedů sousedů
LNLS	Výběr podle počtu sousedů, při shodě podle swapů

Tab. 4.4: Stručný popis jednotlivých modifikací

Tabulka 4.5 ukazuje počet stripů v modelu (graf na obr.4.1). Pro větší modely je chování SGI modifikací takové, jaké jsem očekával při jejich návrhu. Nejméně stripů generuje metoda LNLN. Kromě modelu *elipsoid*(3) je lepší než $STRIPE$. Pro větší data je $STRIPE$ příliš pomalý a dosažené výsledky jsou chybné (podobné chování je zmíněno např. v [STE01]). Snad jediným překvapením je, že metoda LS je co do počtu stripů jasně nejhorší.

model	1	2	3	4	5	6	7	8	9
metoda									
RN	17	26	185	812	2406	18103	38259	62911	78192
LNRN	2	3	110	412	927	10578	10983	20617	25685
LS	12	63	352	1183	3560	33249	44729	71113	88110
LNLN	1	24	98	336	648	8094	9381	17401	21555
LNLS	1	48	158	452	1291	11680	15780	26639	33187
Stripe	2	1	101	385	917				

Tab. 4.5: Počet stripů potřebných pro pokrytí celého modelu.

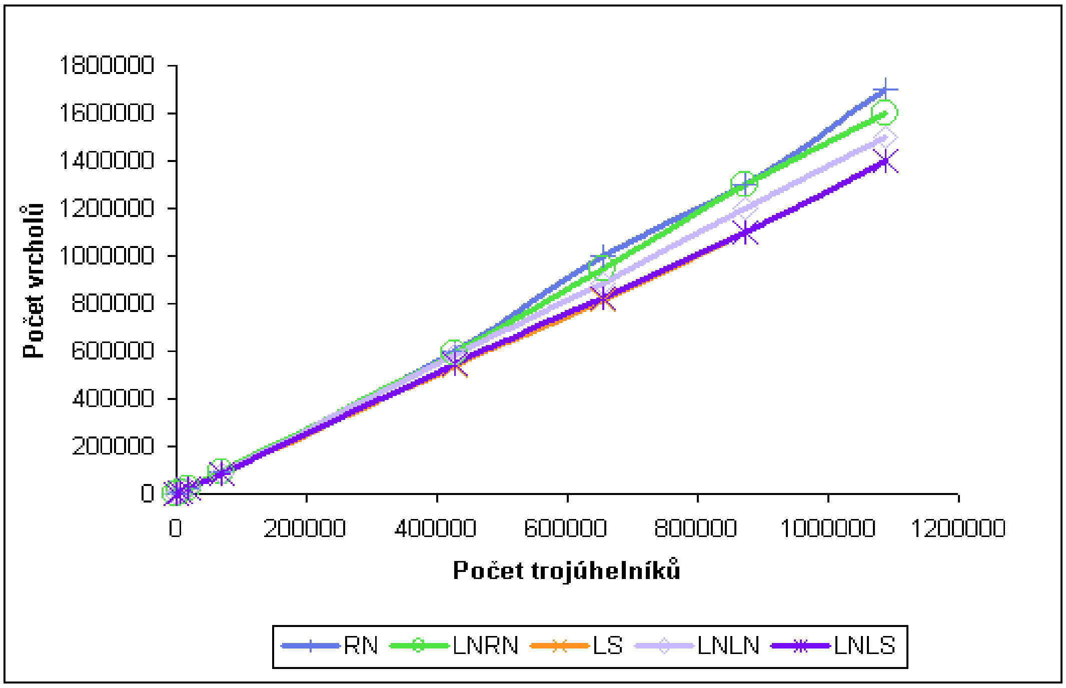


Obr. 4.1: Graf závislosti počtu stripů na počtu trojúhelníků

V tabulce 4.6 je ukázán počet všech vrcholů, které jsou ve stripech použity (graf na obr.4.2). V této tabulce nejsou výsledky příliš překvapivé. Nejhorší metodou je tentokrát podle očekávání RN. Naopak metody LNLS a LS jsou obě lepší než původní LNLN metoda. Zajímavé je, že metoda LS potřebuje, i přes velmi vysoký počet stripů, pro pokrytí velmi málo vrcholů. Výsledky $STRIP$u jsou značně proměnlivé.

model	1	2	3	4	5	6	7	8	9
metoda									
RN	366	6.3k	7.7k	24k	92k	598k	1.0M	1.3M	1.7M
LNRN	363	5.3k	7.7k	23k	92k	591k	944k	1.3M	1.6M
LS	324	5.1k	7.1k	22k	82k	536k	813k	1.1M	1.4M
LNLN	335	5.1k	7.6k	23k	86k	584k	886k	1.2M	1.5M
LNLS	313	5.1k	7.1k	22k	82k	543k	823k	1.1M	1.4M
Stripe	363	5.0k	7.6k	23k	91k				

Tab. 4.6: Počet vrcholů potřebných k pokrytí modelu

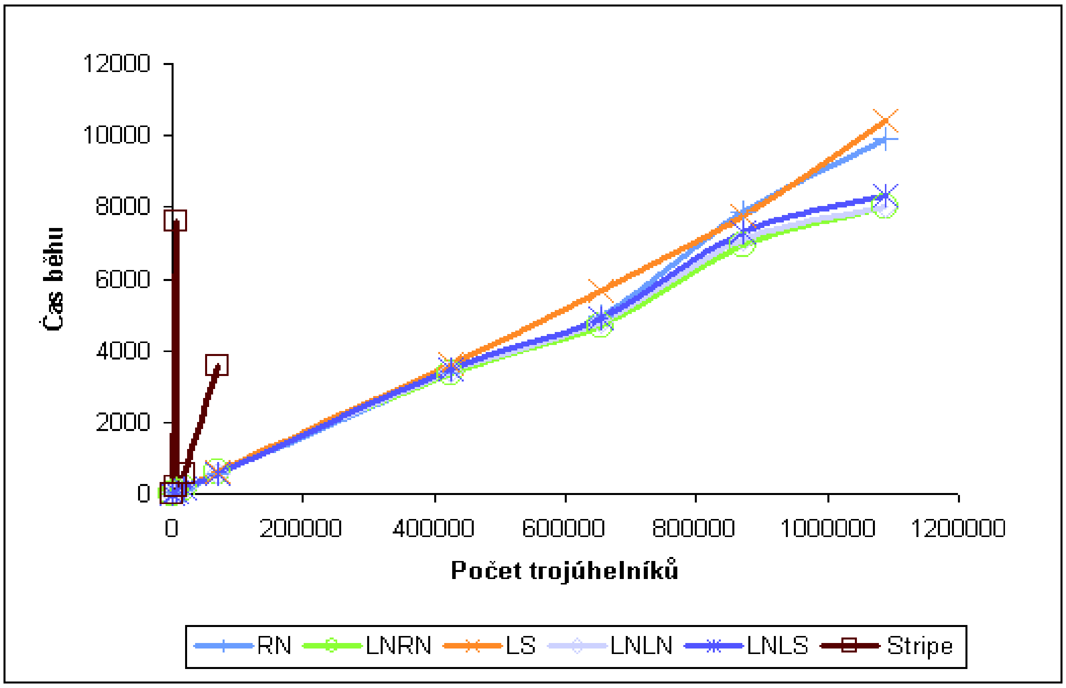


Obr. 4.2: Graf závislosti počtu vrcholů ve stripech
na počtu trojúhelníků

Tabulka 4.7 ukazuje časy potřebné k vytvoření stripifikace (graf na obr.4.3). Zde jsou překvapivé hned dvě věci – nejpomalejší (nepočítám-li *STRIPE*) je algoritmus LS a druhý nejpomalejší je RN. Nízkou rychlost těchto dvou algoritmů lze vysvětlit tím, že doba potřebná k založení nového stripu je delší než samotný výběr následujícího trojúhelníku. Nejrychlejší se zdá být LNRN modifikace. Rychlost algoritmu *STRIPE* je mnohonásobně nižší než u ostatních algoritmů. Tento fakt mě natolik překvapil, že jsem se pokusil nalézt příčinu těchto problémů. Našel jsem ve zdrojovém kódu několik částí, které se zdají být zbytečně složité. Nicméně neměl jsem tolik času a síly, abych mohl s určitostí prohlásit, že lze tyto části zjednodušit (vzhledem k tomu, že $STRIPE$ je schopen pracovat s modely, které nejsou plně trojúhelníkové).

model	1	2	3	4	5	6	7	8	9
metoda									
RN	0	41	40	120	581	3385	4967	7851	9915
LNRN	0	30	50	130	591	3375	4697	6960	7992
LS	0	30	50	130	621	3610	5660	7750	10400
LNLN	0	30	40	130	571	3435	4787	7080	8002
LNLS	0	40	40	131	581	3475	4917	7281	8312
Stripe	0	7621	220	551	3585				

Tab. 4.7: Celkový čas potřebný k vytvoření stripifikace



Obr. 4.3: Graf závislosti doby výpočtu stripfikace na počtu trojúhelníků

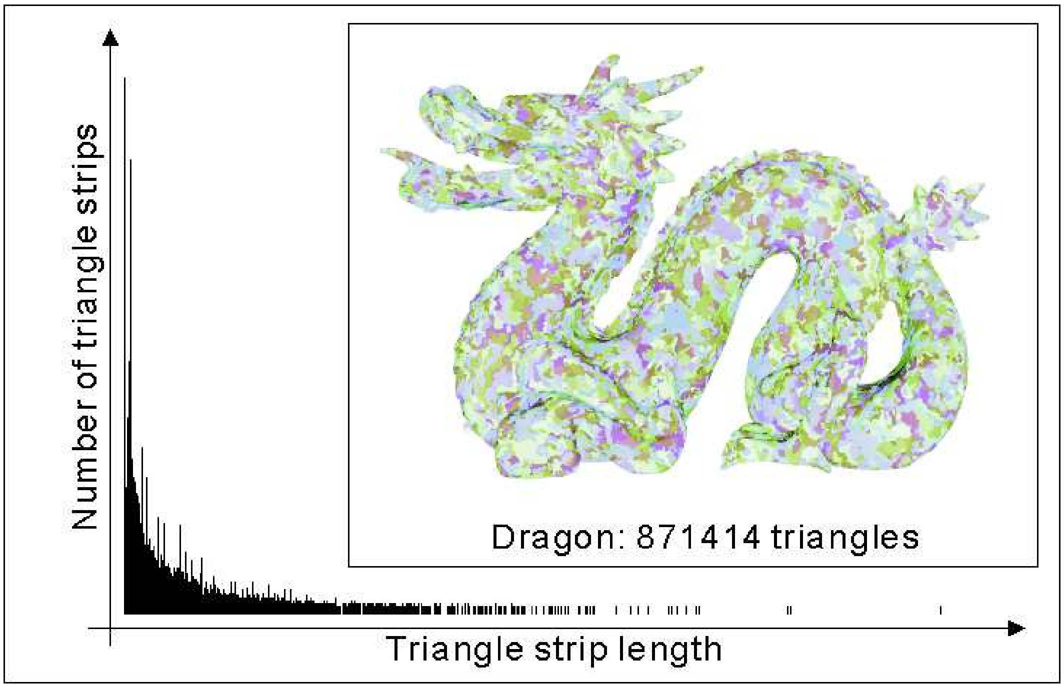
Jedním z důvodů proč $STRIPE$ dopadl ve všech testech špatně, je fakt, že byl primárně určen pro sítě, které nejsou zcela triangulizovány, zatímco testovací data jsou. Není to tedy vůči $STRIP$u zcela spravedlivé, protože v našem případě se uplatnila pouze lokální část algoritmu. Tato skutečnost však neomlouvá rychlost algoritmu, pouze jeho výsledky. Důvodem, proč jsem se rozhodl tento algoritmus použít jako referenční, je fakt, že je používán řadou ostatních autorů, a mohu tedy nepřímo porovnávat výsledky svojí stripifikace s ostatními algoritmy. Další překvapivou věcí ohledně $STRIP$u je, že není schopen zpracovat rozsáhlejší modely. Na toto chování bylo přímo nebo nepřímo ukázáno i v některých článcích ([STE01, SIL02] a dokonce i samotný autor $STRIP$u [EVA96b] používá pouze modely s nízkým počtem trojúhelníků).

V tabulce 4.8 jsou uvedeny frekvence, s jakými byly jednotlivé modely s použitím různých stripifikačních metod vykreslovány (měření bylo provedeno pouze pro větší modely). Přestože jsou modely stripifikované metodou LS tvořeny nejmenším počtem vrcholů, je rychlost zobrazování téměř nejnižší. Nejlépe dopadají modely stripifikované původním algoritmem LNLN. Tento rozdíl je pravděpodobně způsoben tím, že vytvoření nového stripu je časově náročnější operace než přijmutí několika vrcholů navíc.

model	6	7	8	9
metoda				
RN	3.82	2.07	1.50	1.12
LNRN	4.33	2.75	2.02	1.64
LS	3.34	2.18	1.55	1.25
LNLN	4.40	2.96	2.15	1.75
LNLS	4.25	2.80	2.06	1.66

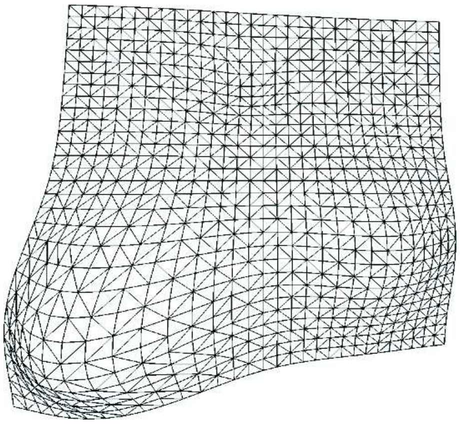
Tab. 4.8: Počet snímků za vteřinu

Dále jsem také zkoumal, jaká je průměrná délka stripu a jak vypadá histogram délky stripů u typického modelu. Jak je patrné z obrázku 4.4, v modelu se nachází pouze několik extrémně dlouhých stripů, zbytek tvoří stripy malé délky. Pokusil jsem se tedy tuto nerovnoměrnost napravit zavedením maximální povolené délky stripu (dlouhé stripy procházející trojúhelníkovou sítí mohou mít za následek rozbití sítě na malé oblasti). Výsledek tohoto experimentu byl ale zcela opačný, než jsem očekával – zvýšil se jako počet stripů, tak počet vrcholů.

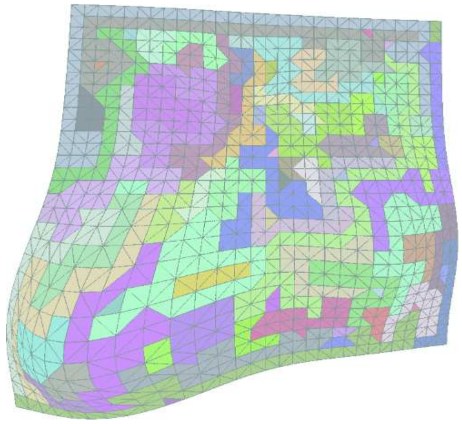


Obr. 4.4: Příklad typického histogramu délek stripů.
Model draka byl stripifikován metodou LNLN a skládá se z 17401 stripů.

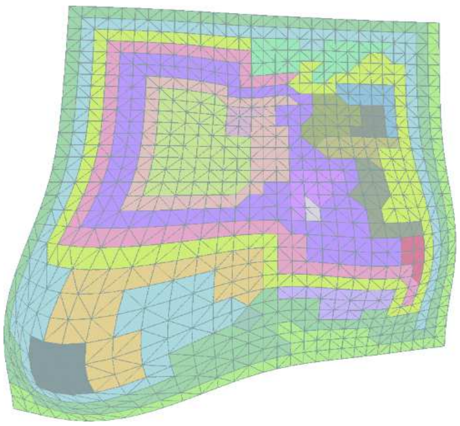
Pro úplnost uvádím i grafické porovnání výstupu jednotlivých algoritmů (obr.4.5). Použitá trojúhelníková síť je tvořena 1800 vrcholy a 960 trojúhelníky. Už u takto malých dat je jasně viditelný rozdíl v počtu stripů při použití rozdílných metod. Obrázek 4.5 a) ukazuje původní trojúhelníkovou síť. Přestože se jedná o pravidelnou čtvercovou mřížku, nejsou hrany, dělící jednotlivé čtverce, shodně orientované. Následek této skutečnosti je jasně patrný na obrázku 4.5 d). V případě shodné orientace hran by byly jednotlivé stripy rovnoběžné se stěnami mřížky. Trojúhelníková síť stejných rozměrů, ale se stejně orientovanými hranami, je na obrázku 4.6 a). Pro tuto síť udávají všechny metody (kromě RN – obr.4.6 b)) shodný výsledek. Proto na obrázku nejsou uvedeny metody LNLS a LNRN.



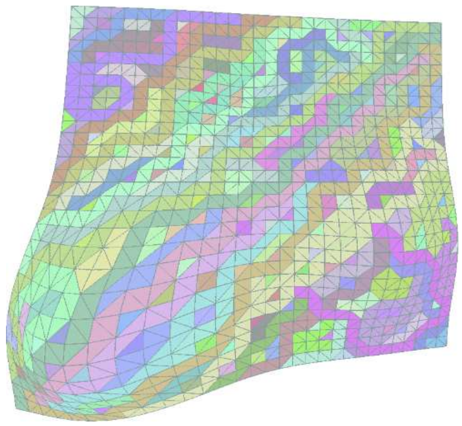
a) původní triangulace



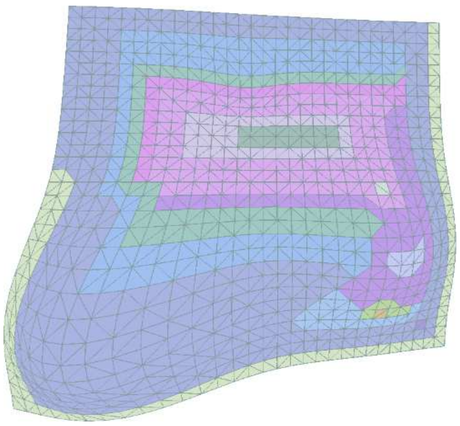
b) metoda RN



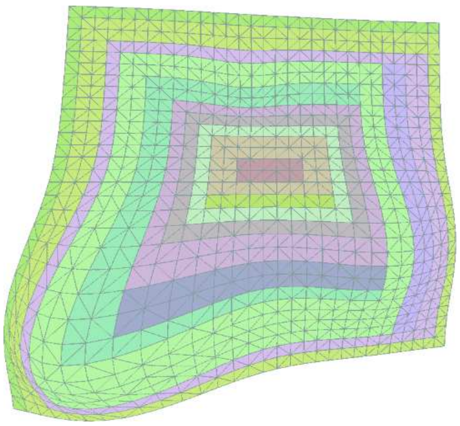
c) metoda LNRN



d) metoda LS

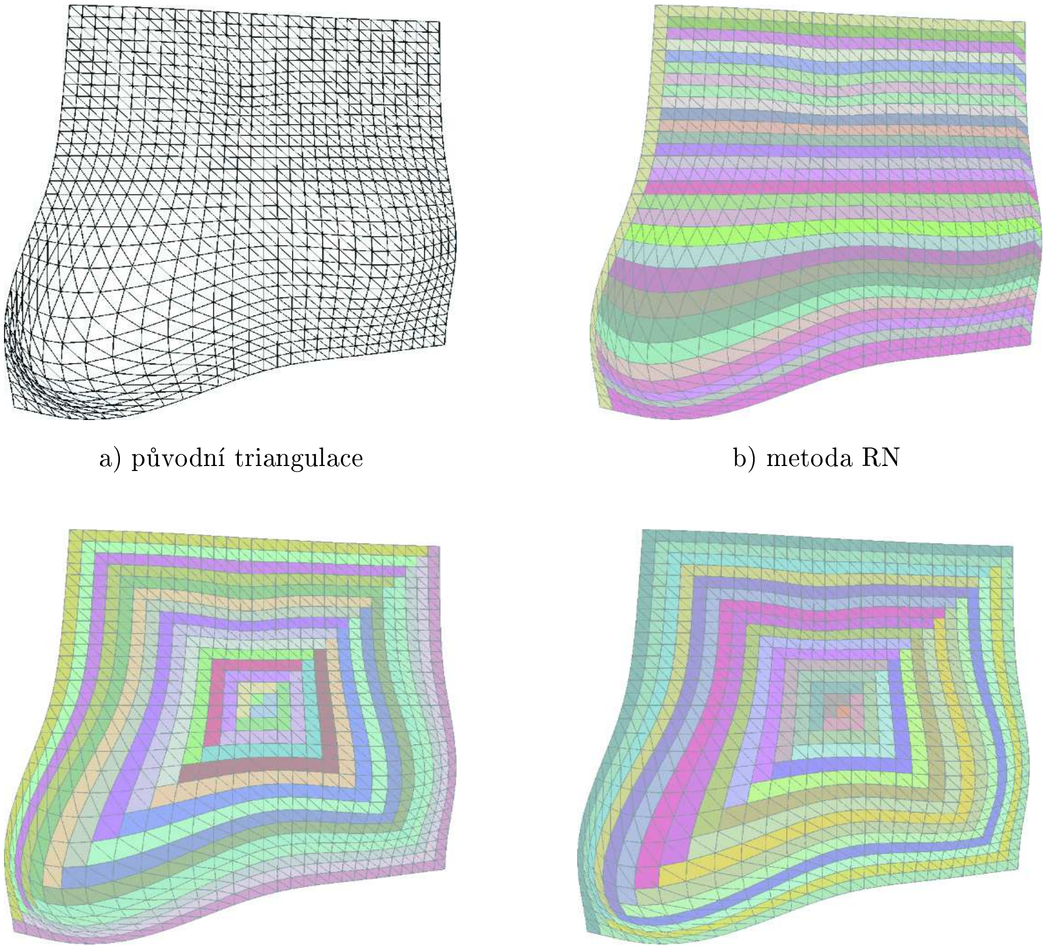


e) metoda LNLN



f) metoda LNLS

Obr. 4.5: Grafické porovnání výstupů jednotlivých metod (1)


a) původní triangulace

b) metoda RN

c) metoda LS

d) metoda LNLN

Obr. 4.6: Grafické porovnání výstupů jednotlivých metod (2)

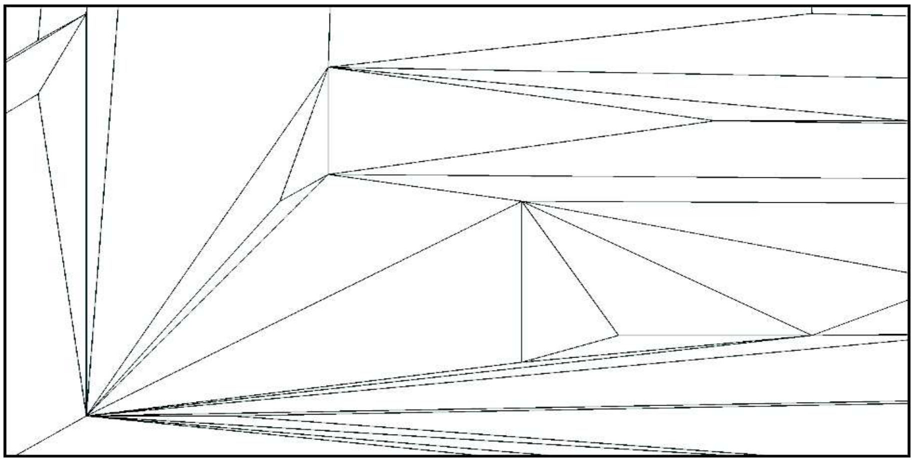
4.3 Hamiltonovská triangulace

Metody pro hamiltonovskou triangulaci jsem testoval pouze na náhodných datech. Jejich velikost se pohybovala od 500 do 500000 vrcholů (počet trojúhelníků je přibližně dvojnásobný, ale může se lišit u jednotlivých metod). Jak je vidět na obr.4.7, výsledná triangulace obsahuje některé trojúhelníky, u kterých je úhel u jednoho z vrcholů velmi blízký nule a triangulace je pro další práci téměř nepoužitelná (povšimněte si především vrcholu v levém dolním rohu). Z toho důvodu jsem neprováděl testování metod na reálných datech. Značení testovaných metod je následující:

HAM On-line metoda. Metoda, při které je stripifikace vytvářena již během triangulace. U této metody nelze rozdělit hranu vkládaným vrcholem.

HAMdag DAG metoda. Metoda, která vytváří stripifikaci až na základě vytvořené triangulace (z listů duálního acyklického grafu). U této varianty není povoleno dělení hrany vkládaným vrcholem.

HAMedge Stejná jako HAMdag, navíc je povoleno dělení hran. Z tohoto důvodu je v tabulce uveden i počet stripů, které jsou vytvořeny (u předešlých metod vzniká pouze jeden strip).



Obr. 4.7: Příklad hamiltonovské triangulace

Jak je vidět v tab.4.9, počet vrcholů ve výsledném stripu (stripech) je u všech metod srovnatelný. Také doba výpočtu je u všech metod téměř shodná (čas u posledního měření není uveden, protože byl ovlivněn nedostatkem paměti). Výsledná stripifikace je velmi dobrá a to jak na počet stripů (většinou jeden), tak na počet vrcholů (dokonce méně než u LSLS metody). To příznivě ovlivňuje i samotný proces vykreslování. Např. model s 300000 vrcholy stripifikovaný metodou LNLN je zobrazován rychlostí 3.4 snímku za vteřinu. Týž model stripifikovaný hamiltonovskou triangulací je zobrazován rychlostí 4.1 snímku za vteřinu. Velkou nevýhodou hamiltonovské stripifikace tedy zůstává tvar některých trojúhelníků. To by bylo možné částečně napravit použitím „cibulové metody“. Další nevýhodou je, že lze tyto metody použít jen na rovinná data.

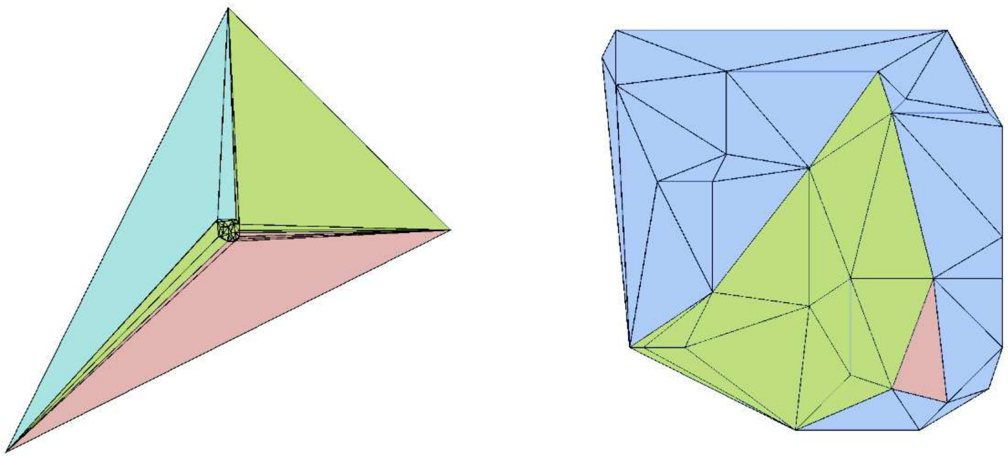
Vrcholy	HAM		HAMdag		HAMedge		
	Vrcholy	Čas [ms]	Vrcholy	Čas [ms]	Vrcholy	Stripy	Čas [ms]
500	1316	20	1322	20	1321	3	10
1000	2629	40	2630	30	2635	3	40
2000	5257	70	5259	70	5280	1	80
3000	7907	120	7896	120	7958	2	120
5000	13155	200	13189	210	13211	5	201
10000	26331	451	26444	451	26294	6	440
20000	52640	951	52725	1001	52653	1	952
30000	79035	1523	79013	1542	79099	3	1522
50000	131581	2623	131867	2624	131755	1	2644
100000	263180	5738	263387	5958	263408	1	5879
200000	527094	12858	526718	13189	526973	4	12468
300000	789937	20329	789891	23504	789924	4	20339
500000	1317164		1316850		1316918	2	

Tab. 4.9: Přehled výsledků metod pro hamiltonovskou triangulaci

4.4 Delaunayovská triangulace a stripy

Poslední skupina testů se týká Delaunayovy triangulace (dále DT). Testování jsem prováděl jak na datech náhodných (podobně jako u hamiltonovské trriangulace), tak na datech reálných. Při testu jsem se kromě jiného zaměřil také na to, jak výslednou stripifikaci ovlivní, jsou-li vrcholy vkládány v náhodném pořadí, či zda jsou seřazeny.

Z důvodů zjednodušení datových struktur se triangulace vytváří uvnitř pomocného trojúhelníku, který se v konečné fázi odstraní. Na obrázku 4.8 a) je uveden příklad triangulace, ve které je ponechán pomocný trojúhelník. Vkládané vrcholy se nachází uprostřed velkého trojúhelníku. Po odebrání vrcholů pomocného trojúhelníku dostaneme výslednou triangulaci 4.8 b). Při odstraňování umělých vrcholů a s nimi incidujících trojúhelníků by mohlo dojít k přerušení stripifikace. Dále jsem tedy zkoumal jaký vliv má odstranění těchto vrcholů na výslednou stripifikaci.



a) před odebráním pomocného trojúhelníku b) po odebrání pomocného trojúhelníku

Obr. 4.8: Vytváření Delaunayovy triangulace

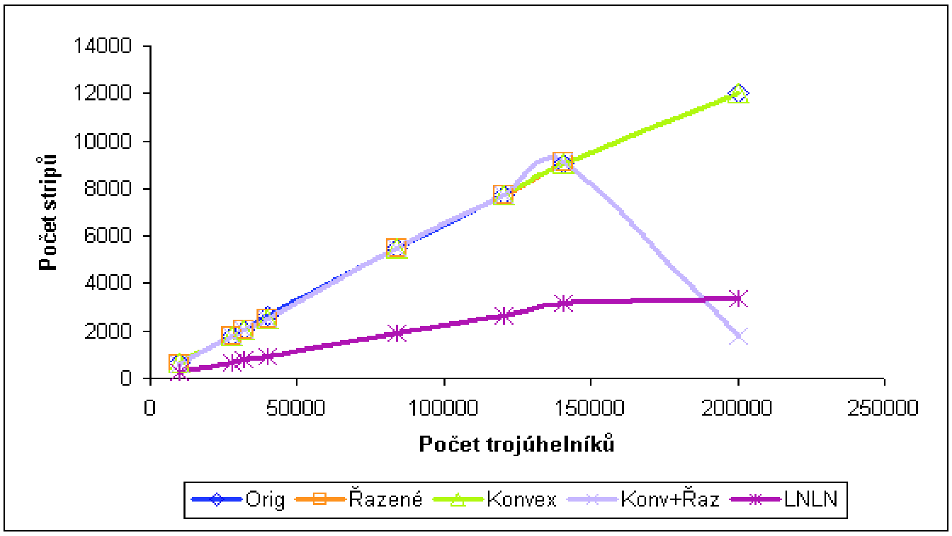
Kombinací těchto dvou měření jsem tedy dostal celkem čtyři modifikace:

Orig	vrcholy se vkládají v náhodném pořadí, umělé vrcholy se odstraňují
Řazené	vrcholy se vkládají ve správném pořadí, umělé vrcholy se odstraňují
Konvex	vrcholy se vkládají v náhodném pořadí, umělé vrcholy jsou součástí triangulace
Konv+Řaz	vrcholy se vkládají ve správném pořadí a umělé vrcholy jsou ponechány v triangulaci

V tab.4.10 jsou uvedeny počty stripů získané jednotlivými metodami pro reálná data (pro porovnání je uvedena i metoda LNLN). Jak je vidět na obr.4.9, všechny metody založené na DT dávají téměř shodné výsledky, které jsou přibližně třikrát horší než u metody LNLN. Pouze u posledního modelu je pro metody, které vkládají vrcholy ve správném pořadí, počet stripů velmi nízký (dokonce nižší než u LNLN). Jak bude ukázáno dále, tento jev je způsoben uspořádáním vrcholů v modelu – pomocný strom (DAG), který při triangulaci vzniká se v tomto případě podobá spíše lineárnímu seznamu. To má za následek několikanásobně vyšší dobu výpočtu.

Trojúhelníky	Orig	Řazené	Konvex	Konv+Řaz	LNLN
9795	638	627	637	627	242
27659	1785	1808	1783	1810	672
31641	2030	2064	2031	2067	769
40016	2625	2533	5455	2536	929
83707	5457	5463	5455	5465	1895
120489	7753	7719	7749	7719	2627
140867	9074	9095	9070	9097	3144
200001	12048		12015	1778	3363

Tab. 4.10: Počet stripů potřebných k pokrytí modelu

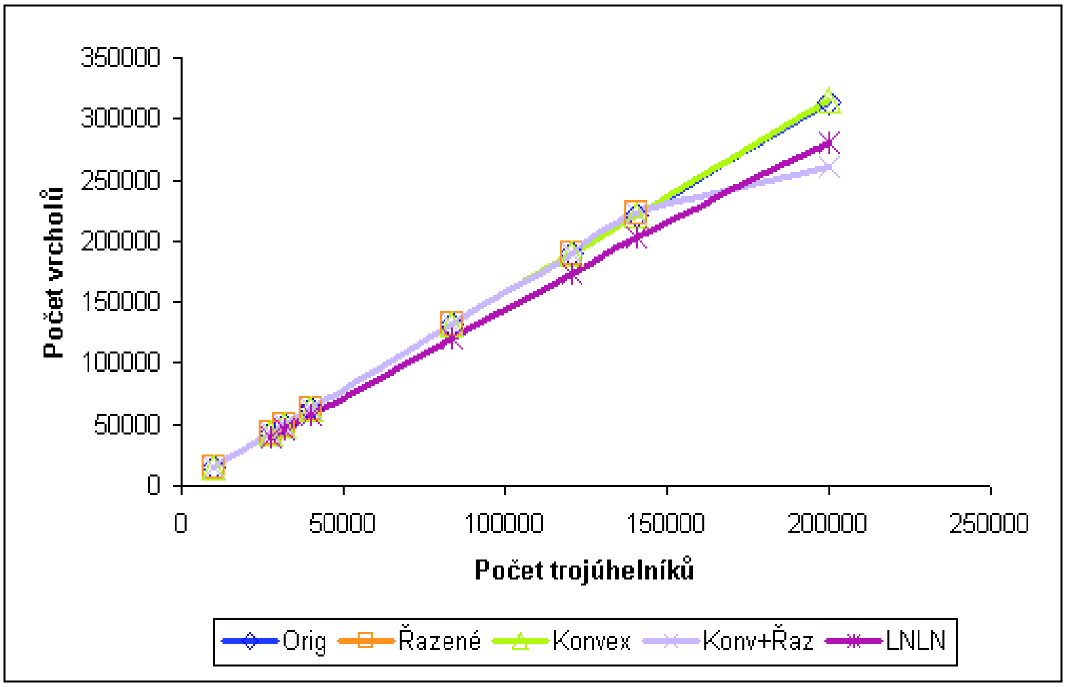


Obr. 4.9: Graf závislosti počtu stripů na počtu trojúhelníků

V další tabulce (tab.4.11) je uveden počet vrcholů potřebných pro stripifikaci (graf na obr.4.10). Rozdíl mezi DT metodami a metodou LNLN už není natolik propastný jako u počtu stripů. Na druhou stranu ani u posledního modelu nedosahují metody se seřazeným vkládáním výrazně lepších výsledků.

Trojúhelníky	Orig	Řazené	Konvex	Konv+Řaz	LNLN
9795	15371	15421	15401	15390	14175
27659	43575	43673	43596	43651	40243
31641	49817	49907	49860	49874	45936
40016	63056	63027	63062	63012	57707
83707	132148	131965	132195	131921	120887
120489	189765	189817	189782	189790	172991
140867	221880	222160	221924	222116	202607
200001	313811		315289	261001	280387

Tab. 4.11: Počet vrcholů k pokrytí modelu



Obr. 4.10: Graf závislosti počtu vrcholů ve stripech
na počtu trojúhelníků

Protože porovnávání časů DT stripifikace s SGI metodou není zcela korektní (SGI vytváří pouze stripy, zatímco DT celou triangulaci), uvádím zde navíc i časy potřebné pro vytvoření DT samotné (tab.4.12). Uvedené DT byly vytvořeny jak s náhodným vkládáním bodů, tak i se seřazeným. Z naměřených časů je vidět, že vytvoření triangulace a její následné stripifikace metodou SGI je přibližně stejně rychlé, jako vytváření stripifikace během triangulace.

Trojúh.	Stripifikace				Bez stripů		
	Orig	Konvex	Řazené	Konv+Řaz	Orig	Řazené	LNLN
9795	481	470	450	440	441	430	70
27659	1783	1502	1432	1422	1443	1362	210
31641	2464	1793	1642	1653	1743	1563	250
40016	2424	2334	2333	2233	2293	2103	300
83707	6189	6770	5097	5077	5158	4867	641
120489	9313	8523	7891	7851	7691	7451	920
140867	10535	10394	9173	9083	9363	8722	1100
200001	15362	15552	»1hod	»1hod	12528	»1hod	1330

Tab. 4.12: Čas [ms] potřebný ke stripifikaci

4.5 Porovnání metod

V následujících dvou tabulkách jsem se pokusil porovnat jednotlivé metody. Protože v některých článcích jsou pro měření použity buď náhodné modely, nebo modely, které mi nebyly dostupné, jsou obě tabulky pouze orientační. První tři sloupce tabulky popisují model. V případě, že měření bylo prováděno na známém modelu, je v prvním sloupci napsáno jeho číslo (viz tab.4.3). Dále následuje přibližný počet vrcholů (v tisících) a přibližný počet trojúhelníků (v tisících). Názvy jednotlivých měřených metod jsou uvedeny v tab.4.13.

Počty vrcholů potřebných k pokrytí modelu se u jednotlivých metod příliš neliší (tab.4.14). Je to dáno také tím, že je tento počet limitován, a to z obou stran. Spodní hranice je $t + 2$, kde t je počet trojúhelníků, a horní hranice je $2 * t + 1$ (viz Úvod).

Poněkud odlišná situace je u počtu stripů, které jednotlivé algoritmy vytvářejí (tab.4.15). Omezení počtu stripů není zdaleka tak přísné jako u vrcholů. Minimální počet stripů je jedna – např. hamiltonovská triangualce (v tabulce není uvedena).

Nejméně stripů (nepočítám-li hamiltonovskou triangulaci) vytváří metoda tunelování [STE01]. Doba potřebná k výpočtu takové stripifikace je však velmi vysoká. Naopak vysoký počet stripů (ovšem za velice krátkou dobu) vytváří algoritmus [SPE97]. Ten je však použitelný pouze pro rovinné modely.

[SGI90]	SGI	[SIL02][b]	Pseudoparalelní strip.
LNLN	viz Implementace		min. počet stripů
LNLS	viz Implementace	[STE01]	Tunelování
[EVA96a]	STRIPE	[SPE97]	TIN
[SIL02][a]	Pseudoparalelní strip.	HAM	viz Implementace
	min. počet vrcholů	DT	viz Implementace

Tab. 4.13: Tabulka měřených metod

Mod.	Vrch.	Troj.	LNLN	LNLS	[EVA96a]	[SIL02][a]	[SIL02][b]	[SPE97]	HAM	DT
	2,8	5,6	8	7,4	6,9/7,5			7,8	7,3	8,7
3	2,9	5,8	7,6	7,1	7,6/7,1	7,1	7,6			
4	9,1	17,5	23	22	23					
		27	38	27	36			37,5	35,6	42,6
		56	80	74	68,6			71	73,9	88,3
5	3,6	69	86	82	91/82	82	86			
		190	272	251	—			281	250	299
7	327	655	886	823	—	867	816			
8	438	871	1238	1152	—	1217	1139			
9	544	1088	1547	1437	—	1421	1520			

Tab. 4.14: Počet vrcholů (v tisících) získaných jednotlivými metodami

Mod.	Vrch.	Troj.	[SGI90]	LNLN	LNLS	[EVA96a]	[SIL02][a]	[SIL02][b]	[STE01]	[SPE97]	DT
	2,8	5,6		124	173	165				724	353
3	2,9	5,8		98	158	101/123	80	137			
4	9,1	17,5		336	452	385					
		19,6	404	435	588	401			82		1267
		27		562	849	821				3430	1770
		56		1183	1714	1455				6280	3624
5	3,6	69	705	648	1291	917/1230	599	1147	158		
		190	3719	3972	5860	—			593	31223	12115
7	327	655		9381	15780	—	8440	14662			
8	438	871	17653	17401	26639	—	16222	23427	1798		
9	544	1088		21555	33187	—	20071	29128			

Tab. 4.15: Počet stripů získaných jednotlivými metodami

5 Závěr

Práce se zabývá problematikou stripifikace trojúhelníkových sítí. Jejím cílem bylo prostudovat možnosti a praktické využití trojúhelníkových stripů pro zobrazování povrchových modelů a navrhnout novou metodu stripifikace.

Prostudoval jsem existující metody stripifikace a pokusil se je porovnat. Zjistil jsem, že většinu stripifikačních metod lze podle přístupu rozdělit do dvou hlavních skupin: přímé metody, využívající přímo topologie trojúhelníkové sítě, a metody duální, které využívají podobnosti mezi problematikou stripifikace a hledání optimální cesty (kostry) grafu. Většina uvedených metod dává velmi podobné výsledky, a to jak kvalitativně, tak časově. Za zmínku stojí zcela jistě metoda tunelování [STE01]. Počet stripů, které generuje, je v porovnání s ostatními metodami mnohonásobně nižší. Těmto výsledkům však odpovídají i časy potřebné pro vytvoření stripifikace.

Z uvedených metod jsem se zabýval především metodami založenými na původní SGI metodě (zřejmě vůbec první stripifikační algoritmus). Navrhl jsem a implementoval několik modifikací původní SGI metody a provedl porovnání s existujícími metodami. Navržené modifikace jsou s ostatními metodami srovnatelné, a to jak po kvalitativní, tak po časové stránce. Tyto metody jsem implementoval též do prostředí MVE[8]. Pro tyto účely musely být navíc implementovány moduly pro začlenění do MVE (modul pro čtení souborů podporujících ukládání trojúhelníkových stripů, konvertor stávajících datových typů do nového typu a vizualizátor, který umí zobrazovat trojúhelníkové stripy).

Dále jsem implementoval metody vytvářející hamiltonovskou stripifikaci. Tyto metody vytvářejí velmi dobrou stripifikaci. Jejich nevýhoda spočívá v tom, že je možné je použít pouze pro rovinná data a triangulace vzniklá těmito metodami obsahuje množství trojúhelníků, které mají u vrcholu velmi malý úhel.

Posledním implementovanou metodou je metoda založená na Delauneyově triangulaci. Výsledky této metody však přinesly jisté zklamání. Počet vzniklých stripů je oproti ostatním metodám přibližně trojnásobný. Časy potřebné k vytvoření triangulace se stripifikací jsou srovnatelné s časy naměřenými při DT triangulaci a následné stripifikaci pomocí SGI algoritmu.

[8]MVE – Modular Visualization Environment systém pro vizualizaci dat. Tento projekt je vyvíjen v Centru počítačové grafiky a vizualizace dat na Západočeké univerzitě. Projekt je ke stažení na stránkách centra: http://herakles.zcu.cz/research.php.

Jedním z cílů této práce také bylo ověřit efektivitu trojúhelníkových stripů (popřípadě dalších optimalizací) v OpenGL. Ověřil jsem, že se při použití trojúhelníkových stripů urychlí zobrazování více než 2.5 krát a blíží se tedy očekávané teoretické hranici (~ 3). Dále jsem prozkoumal možnost využití polí vrcholů ($VertexArrays$) pro urychlení zobrazování. Kombinací obou metod jsem pak dosáhl značného urychlení zobrazování.

Dílčí výsledky této práce byly prezentovány v rámci studentské konference CESCG'02 (Central European Seminar on Computer Graphics). Dokument z této konference ([VAN02]) a další informace o této diplomové práci lze nalézt i na stránkách Centra počítačové grafiky a vizualizace dat (http://herakles.zcu.cz/research.php).

V budoucnosti bych se rád pokusil zvýšit efektivitu jak SGI modifikací, tak „Delaunayovské stripifikace". Na základě zkušeností získaných v této práci bych se také rád pokusil o návrh nové metody pro stripifikaci modelů s proměnným stupněm detailů.

Literatura

[ARK94] E.M.Arkin, M.Held, J.S.B.Mitchell, and S.S.Skiena. Hamiltonian triangulations for fast rendering. In *Second Annual European Symposium on Algorithms*, page 36–47. 1994.

[ARK96] E.M.Arkin, M.Held, J.S.B.Mitchell, and S.S.Skiena. Hamiltonian triangulations for fast rendering. In *Visual Comput.*, 12(9):429–444, 1996.

[BEL01] O.Belmonte, J.Ribelles, I. Remolar, and M. Chover. Searching Triangle Strips Guided by Simplification Criterion. In *Short communications and Posters of 9-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2001)*, pages 68–75, 2001.

[EVA96a] F. Evans, S. Skiena, and A. Varshney. Completing sequential triangulations is hard. Technical report, Department of Computer Science, State University of New York at Stony Brook, 1996.

[EVA96b] F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96*, pages 319–326, 1996.

[EVA98] F. Evans. Stripe, 1998. http://www.cs.sunysb.edu/~stripe/.

[LAR02] Georgia Institute of Technology. Large geometric models archive. http://www.cc.gatech.edu/projects/large_models/.

[SGI90] K. Akeley, P.Haeberli, and D.Burns. tomesh.c. C Program on SGI Develope's Toolbox CD, 1990.

[SIL02] M.V.G. da Silva, O.M. van Kaick, and H. Pedrini. Fast mesh rendering through efficient triangle strip generation. In *WSCG'2002*, pages 127–134, 2002.

[SPE97] B.Speckmann, and J.Snoeyink. Easy triangle strips for TIN terrain models. In *9th Canad. Conf. on Comp. Geom.*, pages 239–244, 1997.

[STA02] Stanford Computer Graphics Laboratory. http://graphics.stanford.edu/data/3Dscanrep/.

[STE01] A. James Stewart. Tunneling for triangle strips in continuous level-of-detail meshes. In *Graphics Interface*, pages 91–100, 2001.

[VAN02] P.Vaněček. Comparison of Stripification Techniques. In *6-th Central European Seminar on Computer Graphics CESCG'02*, pages 65–74,2002.

[VEL99] L.Velho, L.H. de Figueiredo, and J.Gomes. Hierarchical generalized triangle strips. In *The Visual Computer*, vol. 15 pages 21–35, 1999.

[XIA99] Xiang, Held, and Mitchell. Fast and effective stripification of polygonal surface models (short). In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, pages 985–986, 1999.

[ZCU99] University of West Bohemia CCGDV. Specifikace formátu TRI v1.0. http://herakles.zcu.cz/research/mve/developer/tri10.php.

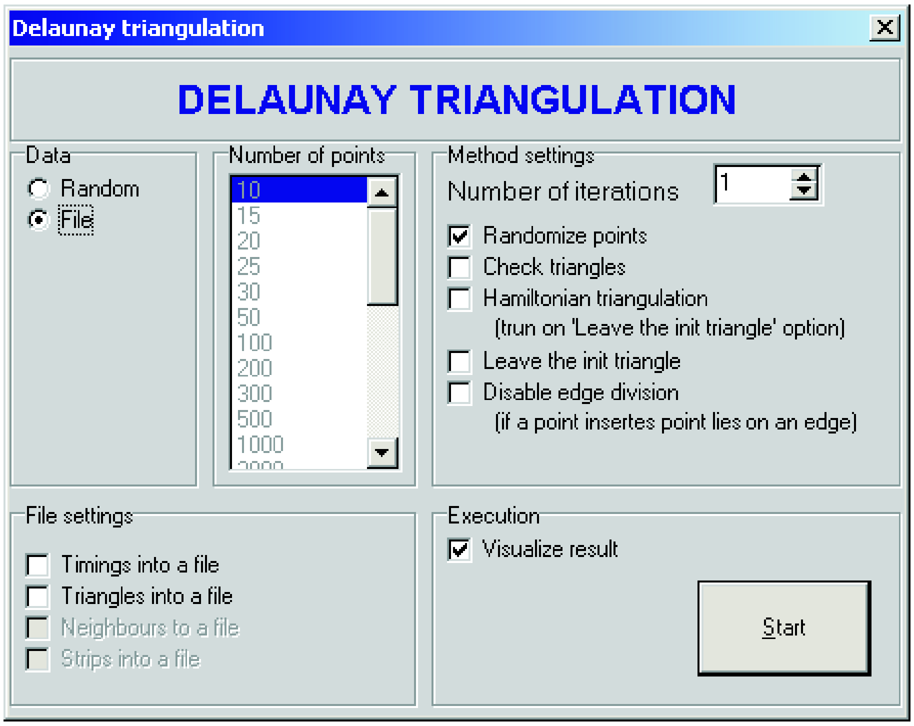
[ZCU02] University of West Bohemia CCGDV. Stránky projektu MVE. http://herakles.zcu.cz/research.php.

Dokumentace k této diplomové práci je k dispozici na internetových stránkách Centra počítačové grafiky a vizualizace dat: http://herakles.zcu.cz/research.php.

A Uživatelský manuál

A.1 Triangulace

Tato část manuálu se týká tří programů pro triangulaci a případnou stripifikaci – DT, DTStrip a HAMStrip. Protože jsou tyto programy pouze rozšířením již existujícího programu pro DT, zachoval jsem původní uživatelské rozhraní. Po spuštění programu se zobrazí okno s hlavním nastavením (obr.A.1).



Obr. A.1: Hlavní okno programu DT

Toto okno je rozděleno do pěti panelů. První panel – Data – obsahuje dva přepínače: 1) Random – program si generuje souřadnice vrcholů náhodně. 2) File – vstupní data se čtou ze souboru. V případě, že je nastaveno náhodné genrování vstupních dat, lze ve druhém panelu – Number of points – vybrat velikost vstupní množiny (vstupních množin se může vybrat i více). Třetí panel – Method settings – nastavuje způsob jakým budou data triangulizována a stripifikována.

Randomize points Před vlastním výpočtem se náhodně pozmění pořadí vrcholů.

U některých vstupních dat se může stát, že data, která nebudou takto upravena, prodlouží dobu výpočtu (pomocný strom pro vyhledávání trojúhelníků degeneruje na seznam).

Check triangles Po triangulaci zkontroluje, zda mají všechny trojúhelníky správnou orientaci a zda splňují Delaunayovu podmínku.

Hamiltonian tri. Při triangulizaci se nekontroluje, zda je splněna Delaunayovská podmínka.

Leave init tri. Ve výsledné triangulaci je zachován pomocný trojúhelník.

Disable edge div. Není povoleno vkládat body na hranu trojúhelníka a dělit jí. Pokud se takový vrchol vyskytne, je místo hrany rozdělen některý z trojúhelníků.

U hamiltonovské triangulace (HAMStrip) jsou přístupné pouze položky „Randomize points" a „Check triangles" (ta navíc neprovádí kontrolu Delaunayovy podmínky).

V panelu „File settings" lze nastavit, která data se mají ukládat do souboru/ů.

Timings ... Ulož jednotlivé časy, které byly zapotřebí pro vytvoření triangulace.

Triangles ... Ulož do souboru výslednou triangulaci.

Neighbors ... Ulož do souboru informace o sousedech.

Strips ... Ulož do souboru výslednou stripifikaci.

Posledním panelem je panel „Execution". Zde lze nastavit, zda se má výsledná triangulace/stripifikace zobrazit (Visualize result). Tlačítkem „Start" se spustí výpočet.

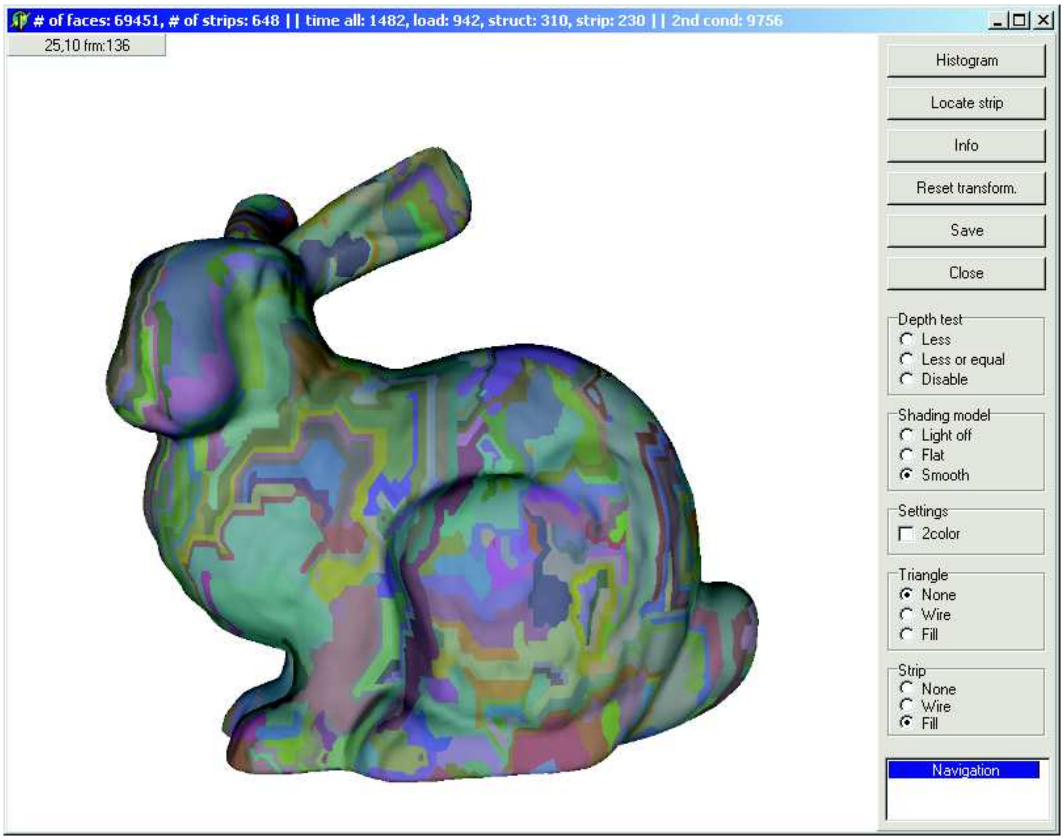
V případě zapnutého zobrazování výsledku se po ukončení výpočtu zobrazí okno s výslednou sítí. Celou síť lze přibližovat/vzdalovat stiskem pravého tlačítka myši a pohybem kurzoru nahoru/dolů. Posouvání sítě se provádí stiskem levého tlačítka a pohybem kurzoru. Posuvníkem „Strip" lze vybrat, který ze stripů má být zobrazen (jsou počítány i stripy, které již neexistují – byly spojeny s jiným stripem). Posuvníkem „Length" lze upravovat zobrazenou délku stripu – vhodné pro zjištění průběhu stripu. Lze zobrazit i všechny stripy současně zaškrtnutím políčka „Show all strips". Dále lze hranice trojúhelníků (Triangles) a jednotlivé vrcholy (Vertices). Zobrazení se ukončí tlačítkem „Close".

A.2 SGI

V následující části je popsáno uživatelské rozhraní samostatného programu pro stripifikaci založeného na SGI metodě. Program byl vytvořen především pro experimentální účely, takže je možnost zadávat parametry i přímo z příkazové řádky.

Interaktivní ovládání

Po spuštění programu se zobrazí dialog pro výběr metody stripifiakce. K dispozici je všech pět modifikací, které byly navrženy a implementovány. Po stisku tlačítka „OK“ program požádá o zadání vstupního souboru. Jako vstupní soubor lze vybrat buď formát .TRI, nebo .STRI. Po otevření souboru se otevře hlavní okno programu (obr.A.2).



Obr. A.2: Hlavní okno programu SGIStrip

V hlavičce tohoto okna jsou informace o modelu: počet trojúhelníků, počet stripů, celkový čas výpočtu, čas potřebný pro nahrání souboru, čas potřebný pro vytvoření pomocných datových struktur (tabulka sousednosti…), čas potřebný pro vlastní stripifikaci a počet, kolikrát byla při stripifikaci využita druhá podmínka. Tyto informace se také zapisují na standardní výstup, takže je lze přesměrovat do souboru (vhodné především pro zpracovávání více modelů z příkazové řádky).

Zobrazovaný model lze při stisku levého tlačítka a pohybu myši otáčet, při stisku pravého tlačítka lze model posouvat a a při současně stisknutém pravém a levém tlačítku či při pohybu kolečka myši lze model přibližovat/oddalovat. Jiná možnost manipulace s objektem je umístění kurzoru do textového pole „Navigation". Kurzorovými šipkami lze model posouvat, šipky na numerické klávesnici slouží k otáčení objektu a klávesy „PgDown" a „PgUp" slouží k přibližování a oddalování modelu.

V panelu „Depth test" lze vybrat jedno ze třech nastavení Z-Bufferu. V panelu „Shading model" lze nastavit způsob stínování objektu (vypnuté světlo, konstantní stínování a vyhlazené stínování). Zaškrtnutím položky „2color" se změní barvy všech stripů na zelenou a barvy trojúhelníků na červenou. to lze využít při ladění programu. V panelech „Triangle" a „Strip" lze vybrat způsob jakým mají být zobrazovány trojúhelníky a stripy (nezobrazovat, zobrazit pouze hrany, zobrazit vyplněné trojúhelníky).

Funkce „Histogram" zobrazí histogram s rozložením délek stripů. Pro zobrazení konkrétního stripu slouží funkce „Locate strip". Zaškrtnutím políčka „Use mask" v dialogu „Locate strip" a následném vybrání jednoho či více stripů ze seznamu se tyto stripy zobrazí. Rovněž lze po zaškrtnutí políčka „Vertices" a následném vybrání stripu zobrazit konkrétní vrcholy tvořící strip.

Funkce „Info" zobrazí dialog s kompletními informacemi o modelu a stripifikaci. Tlačítko „Reset transform." slouží k vynulování transformací.

Pro uložení modelu ve formátu .STRI slouží tlačítko „Save". Po jeho stisknutí se otevře standardní souborový dialog. Posledním tlačítkem je tlačítko „Close", které ukončí aplikaci.

Příkazová řádka

Protože testování velké množiny dat je činnost poměrně zdlouhavá a nezajímavá, je program navržen tak, aby bylo možné pouštět jej z příkazové řádky. Po spuštění programu ve tvaru `SGIStrip <metoda> <model>` provede program výpočet stripifikace zadaného modelu příslušnou metodou. Na standardní výstup pak vrátí informace o průběhu stripifikace a rychlost zobrazování stripifikovaného modelu.

Pro testování většího množství dat lze vyrobit dávkový soubor, který otestuje modely a zapíše výsledky do souboru. Např. pro výpočet stripifikace modelu krávy všemi možnými metodami a zapsání výsledků do souboru by dávkový soubor vypadal následovně:

```
SGIStrip.exe 0 cow.tri >>test.dat
SGIStrip.exe 1 cow.tri >>test.dat
SGIStrip.exe 2 cow.tri >>test.dat
SGIStrip.exe 3 cow.tri >>test.dat
SGIStrip.exe 4 cow.tri >>test.dat
```

A.3 MVE moduly

Pro začlenění začlenění stripifikace do struktury MVE bylo za potřebí několika věcí. Prvním problémem bylo vytvoření nové datové struktury, která by byla schopna nést informaci o stripifikaci. Zavedl jsem tedy strukturu TTriStrips, jejíž struktura vypadá následovně[9]:

```
TTriStrips = record
    Version : array of Char;         // verze souboru
    Name    : array of Char;         // nazev modelu
    Description : array of Char;     // popis modelu
    VertexN : glInt;                 // pocet vrcholu
    TriN    : glInt;                 // pocet trojuhelniku
    BoundBox: TBox;                  // bounding box
    StripN  : glInt;                 // pocet stripu
    bNorm   : boolean;               // existuji normaly?
    Vertex  : AVertex;               // pole vrcholu
    Tri     : ATriangle;             // pole trojuhelniku
    Strip   : AStrip;                // pole stripu
    Norm    : AVertex;               // pole vrcholu
end;
PTriStrip = ^TTriStrips;
```

Datové typy `AVertex`, `ATriangle` a `AStrip` byly již definovány v kapitole 3. Dále bylo nutné vyřešit převod ze stávajících datových struktur do nové struktury. K tomuto účelu byl napsán modul TRI2STRI. Vzhledem k tomu, že pouze převádí data, nemá žádná nastavování. Vstupem modulu jsou data typu Triangles a výstupem jsou data typu TriStrips.

[9] Tato struktura bude v budoucnu pravděpodobně rozšířena (sousednost, mapovací souřadnice,...)

Vstupem Další možností, jak získat data, je načtení ze souboru. K tomuto účelu slouží modul STRILoader. V nastavení modulu je pouze cesta k souboru s daty. Výstupem modulu jsou data typu TriStrips.

O samotnou stripifikaci se pak stará modul Stripify. U něj je možné nastavit metodu, jakou má použít, a případně nastavit, aby se neprováděl výpočet nové stripifikace, pokud ji už model obsahuje (přepínač „Force stripification". Výstupem modulu jsou data typu TriStrips.

Vizualizaci dat obstarává modul STRIViewer. Ovládání je podobné jako u vizualizátoru SGI metod (levé tlačítko rotace, pravé tlačítko posuv, obě tlačítka nebo kolečko přiblížení). V dialogu „Statistic" je jednoduchá statistika modelu (počet vrcholů, počet stripů, atd.). V dialogu „Settings" lze nastavit některé základní vlastnosti vizualizátoru. V panelu „Show" lze nastavit, zda se má pro zobrazování použít pole vrcholů „Use arrays", zda zobrazit ohraničující kvádr (bounding box) a jakým způsobem zobrazovat trojúhelníky a stripy (jen vrcholy, hrany, plné oblasti). V panelu „Rendering" pak lze nastavit tloušťku čar a velikost bodu, způsob osvětlení a Z-Buffer. Vstupem modulu jsou data typu TriStrips.

Posledním implementovaným modulem je modul STRISaver pro ukládání dat do souboru ve formátu .STRI. V nastavení modulu je pouze jméno souboru, do kterého se mají data uložit. Vstupem modulu jsou data typu TriStrips.

B Návrh souboru pro ukládání stripů

Protože stávající formát pro ukládání trojúhelníkových sítí [ZCU99] nepodporuje ukládání informací o stripifikaci, bylo třeba buď navrhnout rozšíření tohoto formátu, nebo navrhnout formát nový. Vzhledem k tomu, že se v původním tri formátu vyskytovaly některé nedostatky (počet vrcholů a trojúhelníků byl udáván v hlavičce, která ovšem nebyla povinná,...), rozhodl jsem se tento formát nahradit formátem novým.

Aby byly změny spojené s přechodem na nový formát minimální, jsou si oba formáty velmi podobné. Struktura formátu, který má pracovní název STRI[10] je následující:

```
[nazev sekce 1]
vlastnost_1=hodnota
...
vlastnost_n=hodnota

[nazev sekce 2]
...
```

Alg. B.1: Struktura STRI formátu

Názvy sekcí mohou být tvořeny libovolnými alfanumerickými znaky a mezerou. Názvy jednotlivých vlastností musí být třípísmenné a tvořené pouze alfanumerickými znaky. Znak „=“ nesmí být oddělený mezerou (celková délka identifikátoru sekce včetně znaku `=` musí být čtyři znaky). Výjimku tvoří sekce `[INFO]`, kde jsou názvy jednotlivých vlastností zarovnány na 10 znaků, a následují znaky `=␣`.

Stávající sekce a vlastnosti

Dále uvádím popis již zavedených sekcí a jejich vlastností. Zavedení nových sekcí by mělo být, při dodržení pravidel, bezproblémové.

Následující dvě sekce jsou povinné a soubor, který je neobsahuje, není považován za STRI:

INFO povinná sekce nacházející se na začátku souboru. V ní mohou být uvedeny informace o objektu (počet vrcholů, stripů, ...), o autorovi, atp. V případě, že tato sekce chybí, považuje se soubor za neplatný.

[10] Název formátu může být chápán buď jako „speciální TRI“ nebo „TRI rozšířený o informace o stripifikaci“

format první povinná vlastnost – udává formát a verzi ve tvaru:

```
format    = stri x.x
```

name jméno modelu

descript krátký popis modelu

vertices počet vrcholů (povinná vlastnost) – pokud není uvedeno, je soubor považován za chybný

triangles počet trojúhelníků – pokud není uvedeno, předpokládá se, že v modelu nejsou trojúhelníky

strips počet stripů – pokud není uvedeno, předpokládá se, že model není stripifikován

VERTICES povinná sekce se souřadnicemi vrcholů.

dim dimenze (povinná vlastnost)– je-li tento rozměr nižší než tři, jsou chybějící souřadnice nastaveny na hodnotu 0. Pokud je dimenze vyšší, jsou načteny pouze první tři souřadnice.

Následuje blok dat ve tvaru `x1␣y1␣z1`.

Dále je možné do souboru uložit následující sekce:

TRIANGLES sekce obsahující informace o trojúhelnících. Každý trojúhelník je definován třemi indexy do tabulky vrcholů ve tvaru `v1␣v2␣v3`.

STRIPS sekce obsahuje informace o stripifikaci

len počet vrcholů konkrétního stripu (povinná vlastnost) – údaj o délce se musí vyskytovat na začátku každého stripu

VERTICES' NORMALS normály ve vrcholech. Jsou definovány stejně jako vrcholy. vlastnost `dim` se v této sekci nevyskytuje.

Budoucí možná rozšíření souboru

VERTEX COLOR barvy vrcholů

VERTEX MAPPING souřadnice pro mapování textur

VERTEX VALUE funkční hodnota (hodnoty) ve vrcholech (definováno podobně jako u souřadnic vrcholů)

POLYGON definice mnohoúhelníků (podobně jako `[TRIANGLE]`)

NEIGHBOR sousednost trojúhelníků

Příklady souboru

Minimální STRI soubor, ve kterém je definován jediný vrchol, tedy vypadá následovně:

```
[INFO]
format    = STRI 0.5
vertices  = 1

[VERTEX]
dim=3
0 0 0
```

V následujícím příkladě jsou definovány vrcholy jednotkové krychle se středem v počátku souřadného systému. Dále jsou zde uloženy informace o jednotlivých trojúhelnících, které tvoří plášť krychle a také informace o stripech (přestože lze krychli pokrýt jediným stripem, je v příkladu každá stěna stripifikována zvlášť).

```
[INFO]
format    = STRI 0.5
name      = krychle
descript  = krychle se stredem v pocatku
vertices  = 1
triangles = 12
strips    = 6

[VERTEX]
dim=3
0.5 0.5 0.5
0.5 -0.5 0.5
...
```

```
[TRIANGLE]
0 3 2
0 2 1
...

[STRIP]
len=4
3
2
0
1
len=4
0
1
...
```

Evidenční list

Souhlasím s tím, aby moje diplomová práce byla půjčována k prezenčnímu studiu v Univerzitní knihovně ZČU v Plzni.

Datum: Jméno:

Uživatel stvrzuje svým čitelným podpisem, že tuto diplomovou práci použil ke studijním účelům a prohlašuje, že ji uvede mezi použitými prameny.

Jméno	**Fakulta/katedra**	**Datum**	**Podpis**